
Von: m1smr@hotmail.com
Gesendet: Freitag, 23. November 2018 22:12
An: m1smr@hotmail.com
Betreff: SDL Dokumentation 2 Marcel Römer

Besondere Key inputs erkennen

Dies kann mit `SDL_GetModState` gemacht werden. Diese Funktion returned ein `SDL_Keymod` mit den Informationen zu den einzelnen Tasten.

Alle Tasten, die mit `SDL_Keymod` überprüft werden: https://wiki.libsdl.org/SDL_Keymod

KMOD_NONE	0 (no modifier is applicable)
KMOD_LSHIFT	the left Shift key is down
KMOD_RSHIFT	the right Shift key is down
KMOD_LCTRL	the left Ctrl (Control) key is down
KMOD_RCTRL	the right Ctrl (Control) key is down
KMOD_LALT	the left Alt key is down
KMOD_RALT	the right Alt key is down
KMOD_LGUI	the left GUI key (often the Windows key) is down
KMOD_RGUI	the right GUI key (often the Windows key) is down
KMOD_NUM	the Num Lock key (may be located on an extended keypad) is down
KMOD_CAPS	the Caps Lock key is down
KMOD_MODE	the AltGr key is down
KMOD_CTRL	(KMOD_LCTRL KMOD_RCTRL)
KMOD_SHIFT	(KMOD_LSHIFT KMOD_RSHIFT)
KMOD_ALT	(KMOD_LALT KMOD_RALT)
KMOD_GUI	(KMOD_LGUI KMOD_RGUI)
KMOD_RESERVED	reserved for future use

Beispiel:

```
if( ( SDL_GetModState() & KMOD_CTRL ) == true ){  
    std::cout<<"CTRL wurde gedrückt"<<std::endl;  
}
```

Text mit SDL kopieren und einfügen

Mit `SDL_SetClipboardText(const char* text)` und `char* SDL_GetClipboardText()` kann global text kopiert und eingefügt werden

SDL_SetClipboardText Parameter:

1. *const char** text der kopiert werden soll

Text input verarbeiten ohne jede Taste abzufragen

Mit `SDL_StartTextInput();` und `SDL_StopTextInput();` kann der Textinput aktiviert und deaktiviert werden. Ab jetzt kann ein `SDL_Event` type nach `SDL_TEXTINPUT` abgefragt werden. Jedes event hat nun ein `event.text` element, das den Input text enthält.

Beispiel

```
SDL_StartTextInput();

string input;

if(event.type == SDL_TEXTINPUT){
    input+= event.text.text;
}

SDL_StopTextInput();
```

SDL File Reading and Writing mit multiplattform support

SDL File Reading und Writing funktioniert ähnlich wie `fstream`.


Es wird ein `SDL_RWops*` benutzt, um reading oder writing auszuführen.

Dies wird mit `SDL_RWFromFile()` initialisiert.

Parameter:

1. `const char*` mit dem Pfad des read oder write file
2. `const char*` mit weiteren parametern

r	Open a file for reading. The file must e
w	Create an empty file for writing. If a file
a	Append to a file. Writing operations ap
r+	Open a file for update both reading and
w+	Create an empty file for both reading a
a+	Open a file for reading and appending. created if it does not exist.

 **NOTE:** In order to open a file as a binary file the mixed modes ("rb+", "wb+", "ab+"). Addition

Writing

SDL_RWwrite();

Parameter:

1. SDL_RWops* das vorher erstellt wurde
2. const void* pointer mit data zum writen
3. size_t größe der einzelnen writings
4. size_t mit der maximalen anzahl der writings

Rückgabetyt:

size_t mit der anzahl an writings oder 0 falls die zuvor gesetzte maximalanzahl an writings erreicht wurde.

Beispiel:

```
char* c = "HELLO";
```

```
SDL_RWops* writeStream = SDL_RWFromFile("PFADZUMFILE", "w");
```

```
SDL_RWwrite(writeStream, c, std::strlen(c), 1);
```

Reading

```
SDL_RWread();
```

Parameter:

1. SDL_RWops* das vorher erstellt wurde
2. const void* pointer zum Ziel des Lesevorgangs
3. size_t gröÙe der einzelnen readings
4. size_t mit der maximalen anzahl der readings

Rückgabetyt:

size_t mit der anzahl an readings oder 0 falls das Ende des files erreicht wurde.

Beispiel:

```
char c[256];

SDL_RWops* readStream = SDL_RWFromFile("PFADZUMFILE", "r");

int i = 0;

while ( SDL_RWwrite(readStream, c[i], sizeof(char), 256) != 0 )
{
    ++i;
}
```

Window Events

Mit Pollevent kann abgefragt werden, ob ein SDL_Event::type SDL_WINDOWEVENT ist.
Mit SDL_Event::window::event kann dann unter den verschiedenen window events unterschieden werden.

Die verschiedenen window events:

SDL_WINDOWEVENT_NONE	(never used)
SDL_WINDOWEVENT_SHOWN	window has been s
SDL_WINDOWEVENT_HIDDEN	window has been h
SDL_WINDOWEVENT_EXPOSED	window has been e
SDL_WINDOWEVENT_MOVED	window has been m
SDL_WINDOWEVENT_RESIZED	window has been r
SDL_WINDOWEVENT_SIZE_CHANGED	window size has ch
SDL_WINDOWEVENT_MINIMIZED	window has been m
SDL_WINDOWEVENT_MAXIMIZED	window has been m
SDL_WINDOWEVENT_RESTORED	window has been r
SDL_WINDOWEVENT_ENTER	window has gained
SDL_WINDOWEVENT_LEAVE	window has lost m
SDL_WINDOWEVENT_FOCUS_GAINED	window has gained
SDL_WINDOWEVENT_FOCUS_LOST	window has lost ke
SDL_WINDOWEVENT_CLOSE	the window manag
SDL_WINDOWEVENT_TAKE_FOCUS	window is being off
SDL_WINDOWEVENT_HIT_TEST	window had a hit te

Beispiel:

```
SDL_Event e;

while( SDL_PollEvent(&e) )
{
    if(e.type == SDL_WINDOWEVENT)
    {
        if(e.window.event == SDL_WINDOWEVENT_MINIMIZED)
        {
            std::cout<<"WINDOW WURDE MINIMIERT"<<std::endl;
        }
    }
}
```

```
}  
  
}
```

Window titel ändern, auslesen

Mit `SDL_SetWindowTitle()` kann der titel des windows geändert werden.

Parameter:

1. `SDL_Window*` window mit dem window, bei dem der Titel geändert werden soll
2. `const char*` mit dem neuen Titel

Mit `SDL_GetWindowTitle()` kann der aktuelle Titel eines windows ausgelesen werden.
Der Rückgabetyt ist ein `const char*` mit dem Titel.

Parameter:

1. `SDL_Window*` window des windows, dessen titel ausgelesen werden soll

Window Fullscreen aktivieren oder deaktivieren

Mit `SDL_SetWindowFullscreen` kann der fullscreen eines windows aktiviert oder deaktiviert werden.

Parameter:

1. `SDL_Window*` das window, bei dem fullscreen aktiviert oder deaktiviert werden soll
2. `uint32 flags` (`SDL_WINDOW_FULLSCREEN` = fullscreen, `SDL_WINDOW_FULLSCREEN_DESKTOP` = fullscreen mit der größe des desktops, 0 = windowmode)

Texturen Manipulieren

Pixel können von einem `SDL_Surface` durch `SDL_Surface::pixel` in einem void pointer gespeichert werden.

Um zum beispiel einzelne pixel einer textur zu bearbeiten, müssen die Pixel der textur an einen void pointer (`void*`) gebunden werden. Dieser `void*` enthält dann die Pixel, die bearbeitet werden können.

Die Pixel, die die Textur bekommen soll, können von einem `SDL_Surface*` bekommen werden.
Dieses Surface muss zuerst wie gewohnt mit `IMG_Load` erstellt und danach in ein neues `SDL_Surface*` mit `SDL_ConvertSurface` in das window format umgewandelt werden.

Jetzt muss eine neue Textur mit `SDL_CreateTexture` erstellt werden und im access parameter `SDL_TEXTUREACCESS_STREAMING` angegeben werden.

`SDL_CreateTexture` Parameter:

1. `SDL_Renderer*` renderer der benutzt werden soll
2. `Uint32 format` das format das benutzt werden soll (kann mit `SDL_GetWindowPixelFormat(SDL_Window*)` vom window bekommen werden) Ansonsten alle formate: https://wiki.libsdl.org/SDL_PixelFormatEnum
3. `int access` mit dem access enum z.B. `SDL_TEXTUREACCESS_STREAMING` alle möglichkeiten: https://wiki.libsdl.org/SDL_TextureAccess

Jetzt wird `SDL_LockTexture()` verwendet, um die pixel der textur an einen `void*` zu binden.

`SDL_LockTexture` Parameter:

1. `SDL_Texture*` die Textur, die gelockt wird
2. `const SDL_Rect*` rect des bereiches, der gelockt wird
3. `void**` pointer zu den pixeln die gelockt werden
4. `int*` länge einer reihe der textur in pixel

Jetzt können die Pixel mit `memcpy` auf die Textur übertragen werden.

`memcpy` Parameter:

1. `void*` mit dem Ziel der übertragung
2. `void*` mit der Quelle der übertragung
3. `size_t` mit der größe in bytes, die gebunden werden sollen

Als Letztes wird mit `SDL_UnlockTexture()` die Bindung aufgehoben und die Textur gespeichert.

`SDL_UnlockTexture()` Parameter:

1. `SDL_Texture*`

Beispiel:

```
//Surface von einem Bild laden und in das window format umwandeln
SDL_Surface* loadedSurface = IMG_Load( path.c_str() );
SDL_Surface* formattedSurface = SDL_ConvertSurface( loadedSurface, SDL_GetWindowSurface( gWindow )-
>format, NULL );

//Neue textur mit format des windows, streaming access und länge, breite des surfaces erstellen
SDL_Texture* newTexture = SDL_CreateTexture( gRenderer, SDL_GetWindowPixelFormat( gWindow ),
SDL_TEXTUREACCESS_STREAMING, formattedSurface->w, formattedSurface->h );

//Pixel der textur an den void pointer mPixels binden
SDL_LockTexture( newTexture, NULL, &mPixels, &mPitch );

//Den speicher des surface pointers in den speicher des mPixels pointers schreiben
memcpy( mPixels, formattedSurface->pixels, formattedSurface->pitch * formattedSurface->h );

//Bindung aufheben und textur updaten
SDL_UnlockTexture( newTexture );
mPixels = NULL;
```

Textur als render target verwenden

Um eine Textur als render target verwenden zu können, muss bei der `SDL_CreateTexture` funktion `SDL_TEXTUREACCESS_TARGET` als access parameter benutzt werden.

Jetzt muss noch die Textur mit `SDL_SetRenderTarget()` als target bestimmt werden.

`SDL_SetRenderTarget` Parameter:

1. `SDL_Renderer*` renderer, der verwendet werden soll
2. `SDL_Texture*` die als target verwendet werden soll (0 = standard render target)

Nun wird alles auf das texture gerendert.

Wenn danach das render target zurückgesetzt wird und das texture gerendert wird, wird nun die textur gerendert, auf die davor gerendert wurde.

Multithreading

Um Multithreading in SDL zu verwenden muss `SDL_thread.h` bzw. `SDL2/SDL_thread.h` inkludiert werden.

Eine Funktion, die mit Multithreading ausgeführt werden soll, muss einen void pointer als parameter besitzen.

Diese kann dann mit einem `SDL_Thread*` ausgeführt werden, indem das `SDL_Thead*` mit `SDL_CreateThread()` erstellt wird.

Der Thread wird sofort nach der erstellung ausgeführt.

`SDL_CreateThread` Parameter:

1. `SDL_ThreadFunction` funktion die ausgeführt werden soll
2. `const char*` name des threads
3. `void*` mit den parametern der thread function

Beispiel:

```
int ThreadFunction(void* data)
{
    std::cout<<"Thread data: "<<(int)data<<std::endl;
    return 0;
}
```

```
int data = 11;
SDL_Thread* thread1 = SDL_CreateThread( ThreadFunction, "thread1", (void*)data );
```

Auf noch nicht abgeschlossenen Thread warten

Falls die main funktion abgeschlossen ist, obwohl ein thread noch läuft, muss darauf gewartet werden.

Dies geht mit der funktion `SDL_WaitThread()`;

`SDL_WaitThread` Parameter:

1. `SDL_Thread*` auf den gewartet werden soll
2. `int*` das return value der thread funktion wird hier übergeben

Mit threads auf speicher zugreifen (Semaphores, Atomic Operations)

Wenn mehrere threads auf speicher zugreifen, muss verhindert werden, dass beide gleichzeitig auf den gleichen speicher zugreifen.

Dies funktioniert mit Semaphores. Semaphores haben einen anfangswert der incrementiert oder decrementiert werden kann. Diese sind bei SDL `SDL_sem*` Objekte.

Dann gibt es noch eine funktion die solange wartet, bis ein semaphore einen positiven wert hat.

Somit kann man z.B. den wert auf 1 setzen und vor einem speicherzugriff den semaphore decrementieren, damit jeder andere thread wartet, bis der semaphore wieder positiv ist.

`SDL_sem*` initialisieren mit `SDL_CreateSemaphore()`

Parameter:

1. `Uint32` anfangswert

`SDL_sem*` löschen mit `SDL_DestroySemaphore()`

Parameter:

1. `SDL_sem*` der gelöscht werden soll

`SDL_sem*` incrementieren mit `SDL_SemPost()`

Parameter:

1. `SDL_sem*` das zu incrementierende semaphore

Auf positiven semaphore wert warten und decrementieren mit `SDL_SemWait()`

Parameter:

1. `SDL_sem*` semaphore bei dem auf einen positiven wert gewartet und decrementiert wird

Beispiel:

```
int data;
SDL_sem* dataLock = SDL_CreateSemaphore(1);
```



```

int ThreadFunc(void* data)
{
    SDL_SemWait(dataLock);

    data++;

    SDL_SemPost(dataLock);
}

SDL_Thread* thread1 = SDL_CreateThread( ThreadFunc, "Thread1", "Thread1" );
SDL_Thread* thread2 = SDL_CreateThread( ThreadFunc, "Thread2", "Thread2");

SDL_WaitThread(thread1, NULL);
SDL_WaitThread(thread2, NULL);

SDL_DestroySemaphore(dataLock);

```

Zugriff anpassen mit Atomic Operations

Wichtig: The atomic locks are efficient spinlocks using CPU instructions, but are vulnerable to starvation and can spin forever if a thread holding a lock has been terminated. For this reason you should minimize the code executed inside an atomic lock and never do expensive things like API or system calls while holding them. - SDL WIKI

Mit Atomic operations kann ein speicherzugriff auf an oder aus geschaltet werden. Im gegensatz zu Semaphores kann man damit nur zwischen einem globalen zugriff oder nicht zugriff unterscheiden. Ansonsen funktionieren Atomic Operations wie Semaphores.

Dazu ist allerdings ein `SDL_Spinlock*` anstatt ein `SDL_sem*` notwendig.

Auf den wert 0 warten und danach ändern mit `SDL_AtomicLock()`

Parameter:

1. `SDL_SpinLock*`

Spinlock Wert auf 0 setzen und damit freigeben mit `SDL_AtomicUnlock()`

Parameter:

1. `SDL_SpinLock*`

Zugriff anpassen mit Mutexes and Conditions

Wenn ein Zugriff nicht nur nicht gleichzeitig, sondern auch nur mit bestimmten regeln angepasst werden soll, benutzt man mutexes and conditions.

Wenn z.B. ein thread in eine datei schreiben möchte und ein anderer thread lesen möchte, sollte der lese thread nur lesen, wenn schon geschrieben wurde und der schreibe thread nur schreiben wenn schon gelesen wurde.

Dazu erstellt man ein mutex und zwei conditions.

Ein `SDL_mutex*` intialisiert man mit `SDL_CreateMutex()` (keine parameter)

Ein `SDL_cond*` intialisiert man mit `SDL_CreateCond()` (keine parameter)

Ein `SDL_mutex*` löscht man mit `SDL_DestroyMutex()`

Parameter:

1. SDL_mutex* das gelöscht werden soll

Ein SDL_cond* löscht man mit SDL_DestroyCond()

1. SDL_cond* das gelöscht werden soll

Mutex locken

Dies funktioniert wie Atomic operations

SDL_LockMutex()

Parameter:

1. SDL_mutex* das gelockt werden soll

Mutex unlocken

SDL_UnlockMutex()

Parameter:

1. SDL_mutex* das ungelockt werden soll

Auf condition warten und mutex unlocken

SDL_CondWait()

Parameter:

1. SDL_cond* condition auf die gewartet wird
2. SDL_mutex* das ungelockt wird

Thread neustarten, das auf eine condition wartet

SDL_CondSignal()

Parameter:

1. SDL_cond* condition, auf die gewartet wird

Beispiel: http://lazyfoo.net/tutorials/SDL/49_mutexes_and_conditions/index.php

Bildschirmauflösung auslesen

Mit SDL_GetCurrentDisplayMode() können Informationen über den aktuellen Bildschirm in ein SDL_DisplayMode* gespeichert werden.

SDL_GetCurrentDisplayMode

Parameter:

1. int displayindex 0 = standard
2. SDL_DisplayMode* hier werden die Informationen hineingespeichert

SDL_DisplayMode member:

Uint32	format	one of the SDL_PixelFormatEnum values; see Remarks for d
int	w	width, in screen coordinates
int	h	height, in screen coordinates
int	refresh_rate	refresh rate (in Hz), or 0 for unspecified
void*	driverdata	driver-specific data, initialize to 0

Android entwicklung mit SDL

Zuerst muss alles für die Entwicklung mit Android vorbereitet werden.

Offizielle SDL ANDROID Dokumentation

<https://hg.libsdl.org/SDL/file/default/docs/README-android.md> und
<https://wiki.libsdl.org/Android>

Informationen zu NDK

<https://developer.android.com/ndk/guides/build.html>

Legende:

Dev. Ordner	<dev> Oberordner für <sdk> und <ndk>
Projektordner	<project>
Android SDK ordner	<sdk> tools ordner
NDK ordner	<ndk> ndk ordner

Kompatible Versionen dieser Dokumentation:

Android API 19 = Android 4.4

Android SDK https://dl.google.com/android/repository/tools_r25.2.5-linux.zip

NDK r13b

1. Android SDK downloaden von der offiziellen Seite. Dies in <dev> entpacken und in <sdk> android ausführen und die sdk installieren mit sdk-tools, platform-tools, build-tools, der SDK 19 (android 4.4) und den system images dafür.

2. NDK downloaden und in <dev> entpacken.

3. JDK installieren mit `sudo apt-get install openjdk-7-jdk` (Nach neuster Version suchen)

4. Ant installieren mit `sudo apt-get install ant`

5. Environment variablen erstellen, indem `~/.bash_profile` mit touch erstellt wird und mit mousepad bearbeitet wird. (`touch ~/.bash_profile` und `mousepad ~/.bash_profile`)

Es müssen drei Environment variablen erstellt werden.

1. `PATH="/home/m1smr/android/android-ndk-r13b:$PATH"` Pfad mit pfad von <ndk> ersetzen
2. `PATH="/home/m1smr/android/android-sdk:$PATH"` Pfad mit pfad von <sdk> ersetzen
3. `PATH="/home/m1smr/android/platform-tool:$PATH"` Pfad mit pfad von <dev>/platform-tool ersetzen

Diese können mit `source ~/.bash_profile` angewendet werden.

6. SDL source code downloaden und in <project>/jni entpacken (Der entpackte Ordner muss SDL2 genannt werden bzw. in <projekt>/jni/src/android.mk der pfad von `SDL_PATH` angepasst werden.)

7. <project>/jni/src/Android.mk bearbeiten. Unter LOCAL_SRC_FILES ein tab nach dem anfang der Zeile eine Liste der cpp source files erstellen. (main.cpp abc.cpp ...)

8. <project>/local.properties erstellen und sdk.dir=/home/m1smr/android/android_sdk einfügen. Pfad mit <sdk> ersetzen.

9. ndk-build in <projekt> ausführen, um zu builden

9. ant debug im terminal ausführen um zu debuggen, compilieren

10. ant debug install mit angeschlossenem Handy und angeschaltetem USB DEBUG in Android ausführen.

Um STL mit c++ benutzen zu können, muss in <project>/jni/Application.mk folgendes hinzugefügt werden:

```
APP_STL := stlport_static und  
APP_CPPFLAGS += -std=c++11
```

Um andere SDL libs benutzen zu können, muss in <project>/jni/src/Android.mk bei LOCAL_SHARED_LIBRARIES die libs hinzugefügt werden und die entpackten libs in <project>/jni

"Konsolenausgabe" mit Android

Konsolenausgaben können mit SDL_Log() gemacht werden.

SDL_Log Parameter:

1. const char* ausgabe, die die konsole macht (wird wie printf gehandhabt)

Auf drehen des Bildschirmes reagieren

Wenn der Bildschirm gedreht wird, wird ein SDL_Event::window::event namens SDL_WINDOWEVENT_SIZE_CHANGED ausgelöst, nachdem abgefragt werden kann.

Beim Event kann dann direkt nach der neuen Auflösung gefragt werden, da SDL_Event::window::data1 (= w) und SDL_Event::window::data2 (= h) dann geupdated werden.

Touch erkennen und position auslesen

Ein Touchevent kann standardmäßig mit einem SDL_Event abgefragt werden. Es gibt ein SDL_Event::type namens SDL_FINGERDOWN, SDL_FINGERUP und SDL_FINGERMOTION.

Beispiel:

```
if(e.type == SDL_FINGERDOWN)  
{  
    //Es wurde auf den Bildschirm getoucht  
}
```

Die touchposition ist gespeichert unter SDL_Event::tfinger::x und SDL_Event::tfinger::y. Die positionen sind nicht direkt gespeichert, sondern als float von 0 bis 1 gespeichert. 0 ist 0% der Auflösung und 1 ist 100% der Auflösung.

Multitouch Events: https://wiki.libsdl.org/SDL_MultiGestureEvent

Images laden

Um images zu laden, muss in <project> ein ordner namens assets existieren, indem die images gelagert werden.

Jetzt kann z.B. mit LoadBMP ohne pfadangabe ein bild geladen werden (bsp
LoadBMP(hello.bmp)).

Evernote hilft dir, nichts zu vergessen und alles mühelos zu ordnen und zu organisieren. [Evernote herunterladen](#).