



Simple DirectMedia Layer

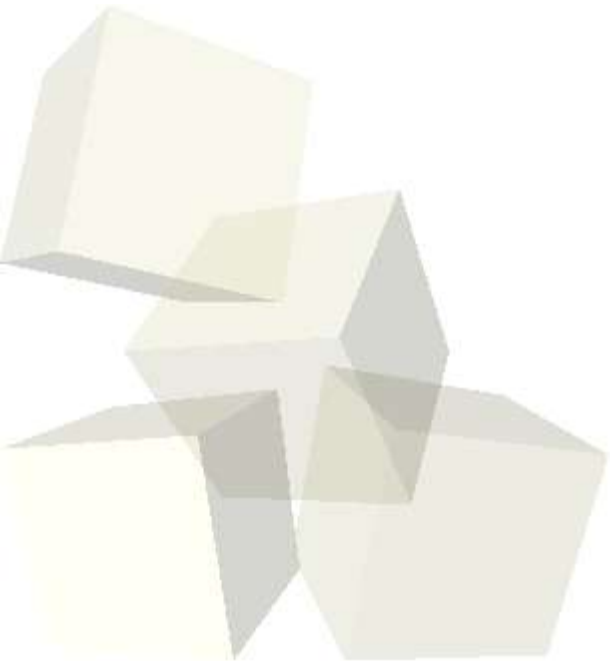


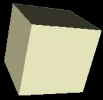
Marco Kraus <marco@libsdl.de>



Kapitel 0:

Intro



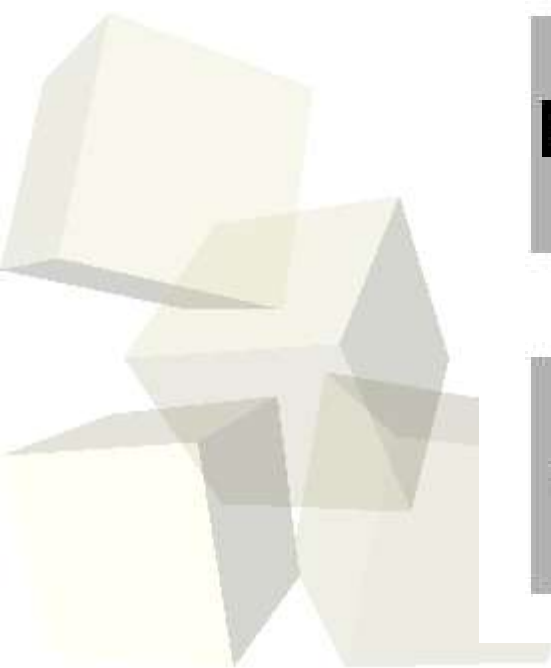
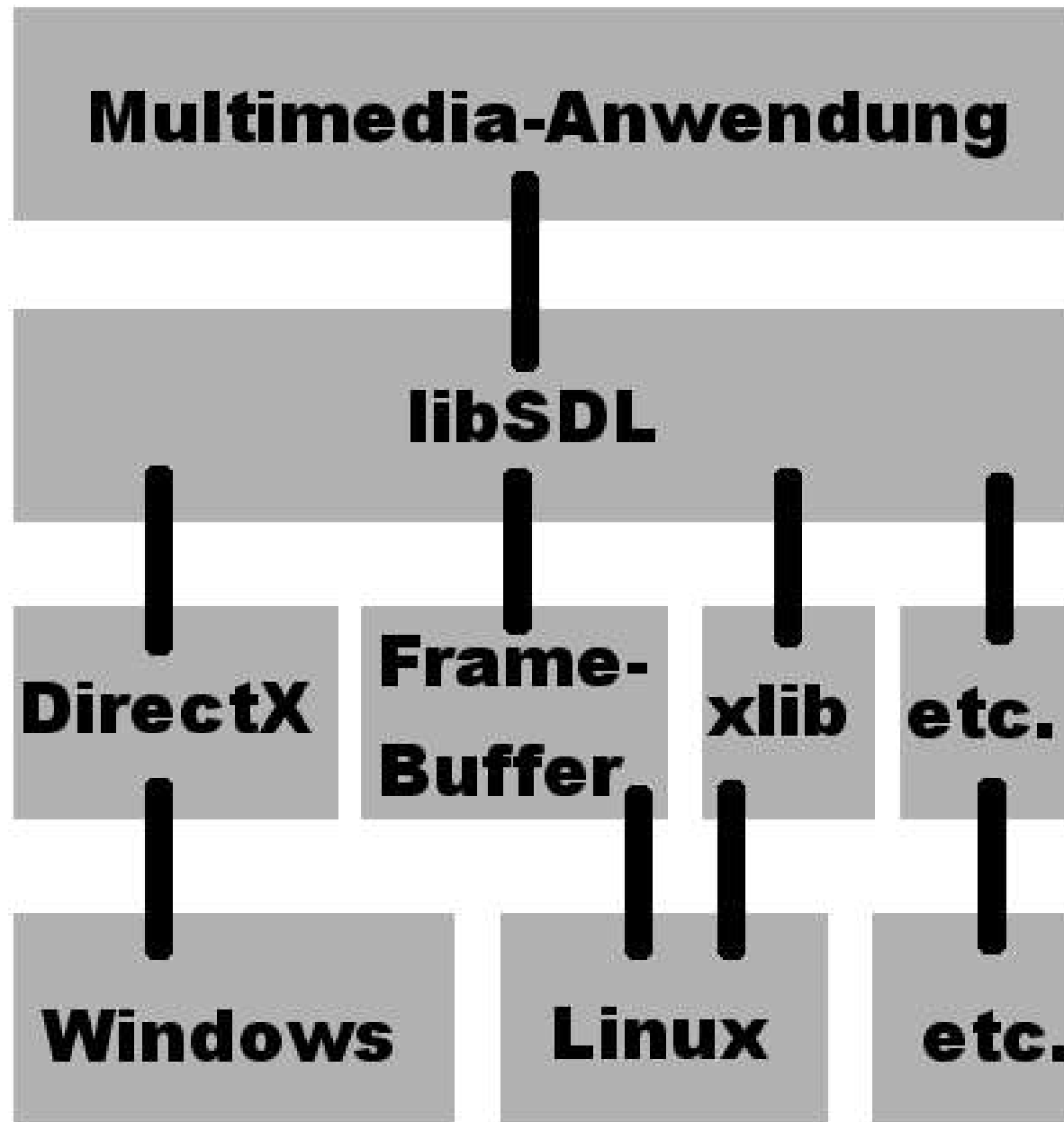


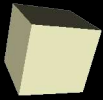
Was ist der „Simple DirectMedia Layer“ ?

- "Simple DirectMedia Layer", kurz SDL bzw. libSDL
- Programmierschnittstelle für Multimediaanwendungen.
- setzt auf plattformeigenen Bibliotheken auf
- für duzende Systeme verfügbar
- portabler Code
- OpenGL Framework
- Open Source (GPL)



Simple DirectMedia Layer

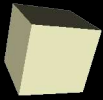




Historie:

- Entwickelt durch Sam Lantinga
- Ende der 90er bei Loki Entertainment
- Entwickelt zur Spieleportierung (siehe Beispiele)
- Loki machte 2001 pleite und Sam wechselte zu Blizzard
- Heute ist es eine komplett freie Entwicklung von vielen Leuten
- De-Facto Standard zur 2D-Spiele-Entwicklung unter Linux
- Neben QT wichtigstes OpenGL Framework unter Linux





Beispiele:

kommerzielle Spiele:

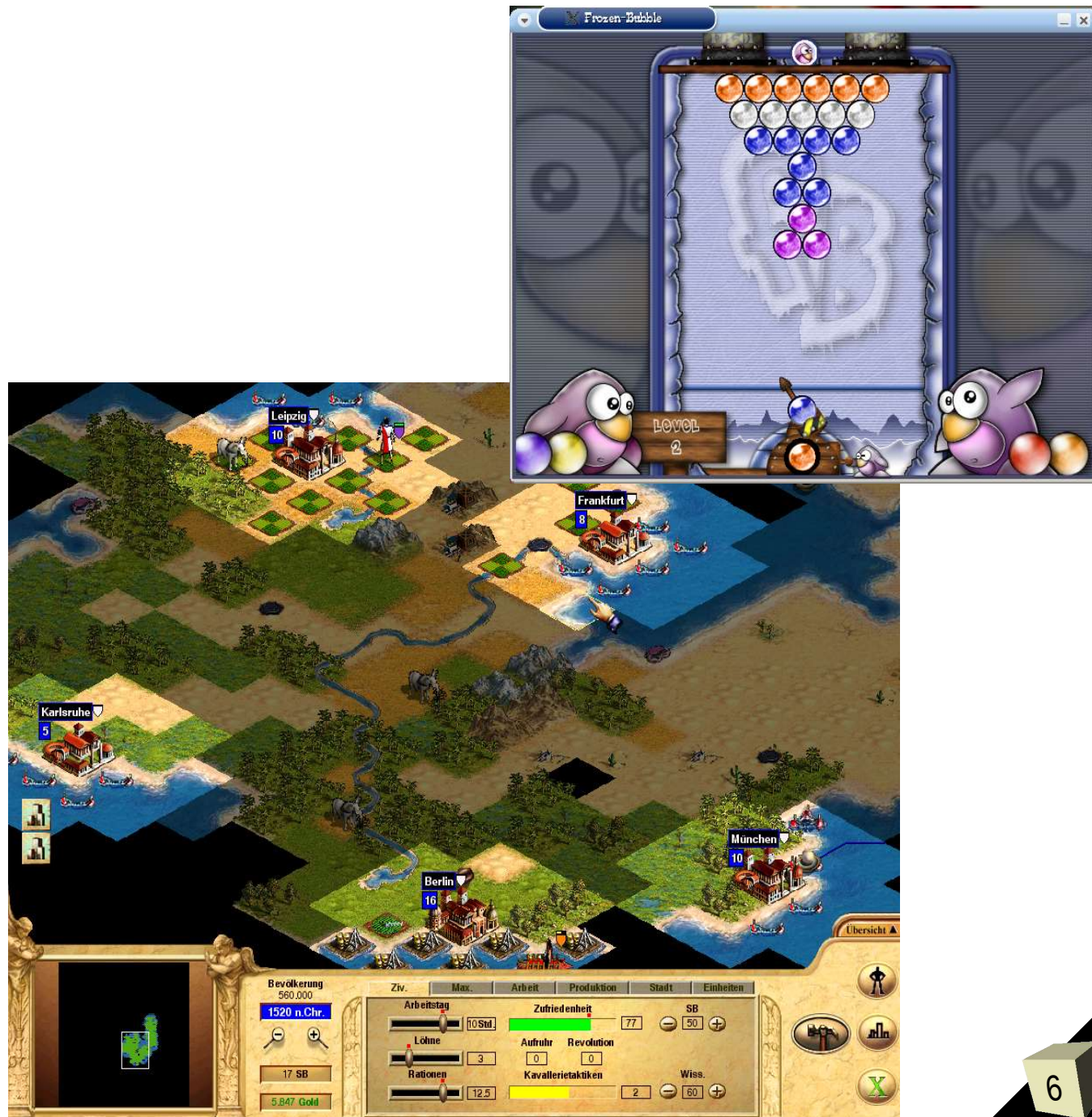
- Civilization CTP
- Descent
- SimCity 3
- Alpha Centauri
- ...

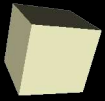
freie Spiele:

- LBreakout
- Matchball
- Frozen-Bubble
-

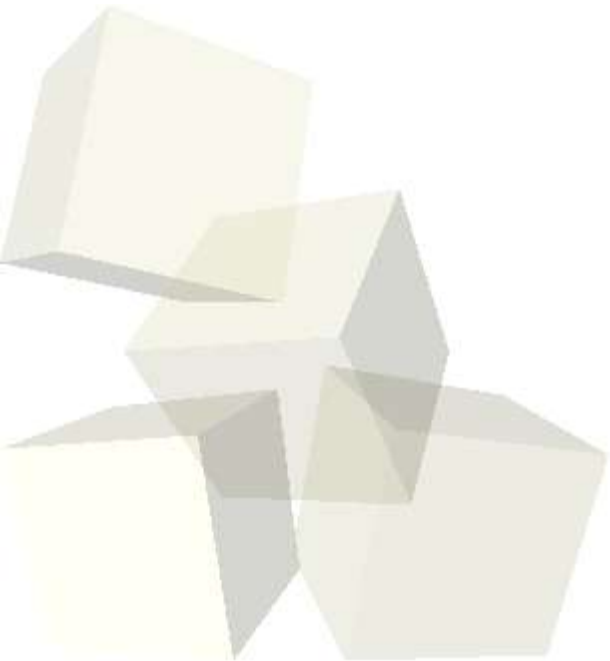
Multimedia:

- mplayer
- smpeg





Kapitel 1: Grundlagen





SDL - Installation:

Am besten die Pakete aus der Distribution installieren oder Bibliotheken von <http://www.libsdl.org/download-1.2.php> ziehen (Sourcen oder precompiled binaries für die Standard-Betriebssysteme).

Compiler und Linker – Einstellungen

Beispiel gcc:

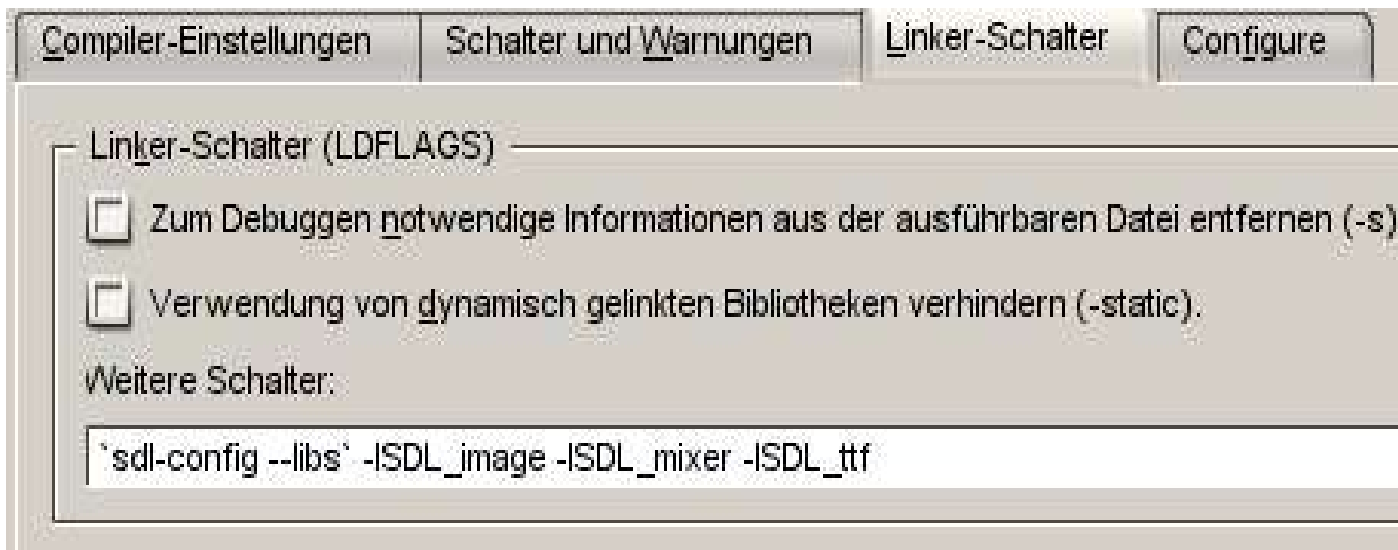
Includes dem Compiler mitgeben: *-I/usr/include/SDL*

Bibliotheken mitlinken: *-lSDL*

*Unter UNIX heißen Bibliotheken z.B. libSDL, libSDL_image, etc.
Bei der Linkeroption -l wird das Prefix „lib“ weggelassen.*



Beispiel: KDevelop Einstellungen: *(unter Compileroptionen)*



The screenshot shows the 'Linker-Schalter' tab of the KDevelop configuration dialog. The 'Compiler-Einstellungen' tab is also visible. The 'Linker-Schalter (LDFLAGS)' section contains two unchecked checkboxes: 'Zum Debuggen notwendige Informationen aus der ausführbaren Datei entfernen (-s)' and 'Verwendung von dynamisch gelinkten Bibliotheken verhindern (-static)'. Below these, a text box labeled 'Weitere Schalter:' contains the command: ``sdl-config --libs` -ISDL_image -ISDL_mixer -ISDL_ttf`.

Compiler-Einstellungen Schalter und Warnungen **Linker-Schalter** Configure

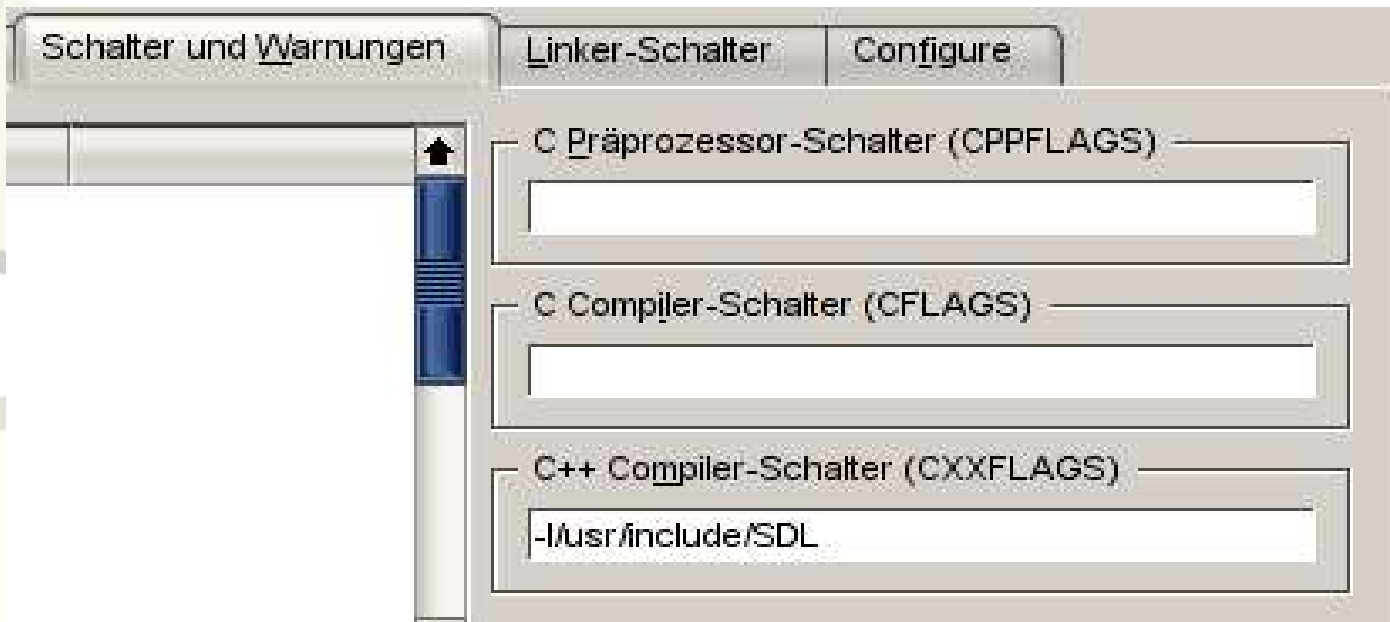
Linker-Schalter (LDFLAGS)

☐ Zum Debuggen notwendige Informationen aus der ausführbaren Datei entfernen (-s).

☐ Verwendung von dynamisch gelinkten Bibliotheken verhindern (-static).

Weitere Schalter:

``sdl-config --libs` -ISDL_image -ISDL_mixer -ISDL_ttf`



The screenshot shows the 'Schalter und Warnungen' tab of the KDevelop configuration dialog. The 'Linker-Schalter' and 'Configure' tabs are also visible. The 'C Präprozessor-Schalter (CPPFLAGS)' section has an empty text box. The 'C Compiler-Schalter (CFLAGS)' section has an empty text box. The 'C++ Compiler-Schalter (CXXFLAGS)' section has a text box containing the command: `-I/usr/include/SDL`.

Schalter und Warnungen Linker-Schalter Configure

C Präprozessor-Schalter (CPPFLAGS)

C Compiler-Schalter (CFLAGS)

C++ Compiler-Schalter (CXXFLAGS)

`-I/usr/include/SDL`

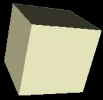


SDL wird in 6 Funktionsbereiche unterteilt:

```
# SDL_INIT_AUDIO  
# SDL_INIT_VIDEO  
# SDL_INIT_CDROM  
# SDL_INIT_TIMER  
# SDL_INIT_JOYSTICK  
# SDL_INIT_EVENTTHREAD
```

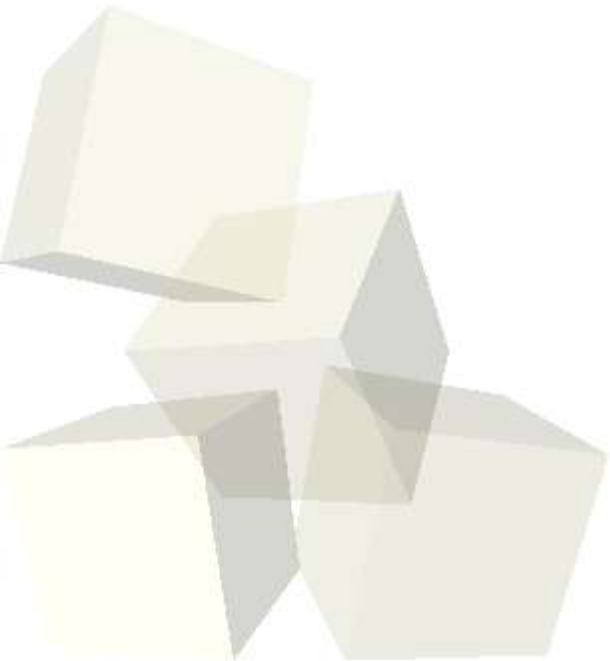
SDL_INIT_EVERYTHING initialisiert alle sechs Teilbereiche

Das Inputhandling muss (bis auf Joystick) nicht initialisiert werden.



Kapitel 2:

Video / Grafik



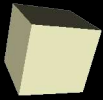


Video-Initialisierung:

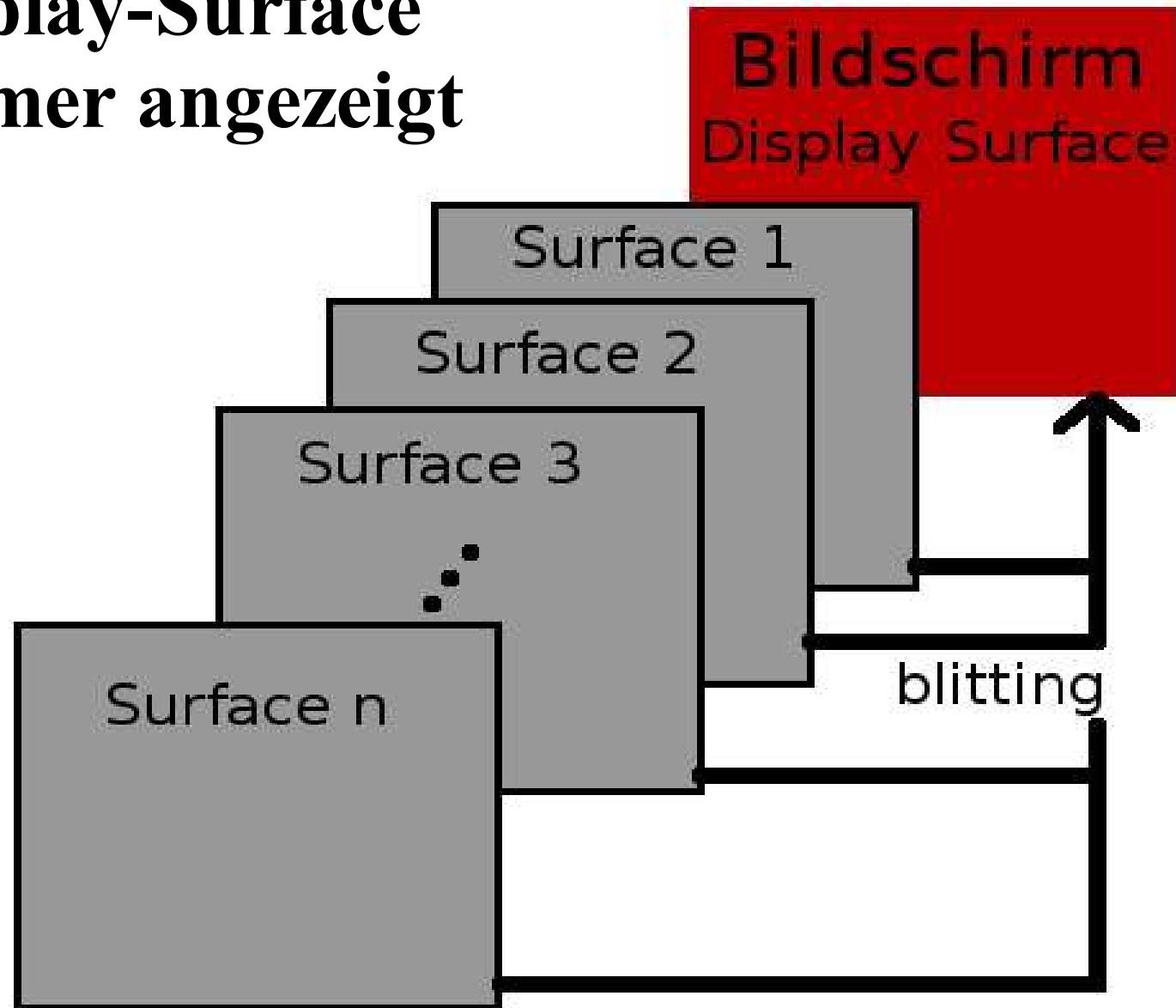
```
#include <iostream>
#include <cstdio>
#include "SDL.h"

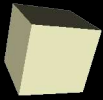
int main()
{
    if ( SDL_Init(SDL_INIT_VIDEO) < 0 )
    {
        std::cerr << "SDL konnte nicht initialisiert werden!";
        exit(-1);
    }

    atexit(SDL_Quit);
}
```



**Nur Display-Surface
wird immer angezeigt**





Der Videomodus:

[...]

```
SDL_Surface *display;
```

```
display = SDL_SetVideoMode(800,600,16,SDL_SWSURFACE);
```

```
if ( display == NULL ){  
    std::cerr << "Konnte kein Fenster 800x600px oeffnen";  
    exit(-1);  
}
```

[...]



Durch die Video-Flags, kann man die eigentlichen Videoeigenschaften beeinflussen. Die Flags sind wie folgt definiert (nur die Wichtigsten) :

- # SDL_SWSURFACE - Das Surface wird im Hauptspeicher abgelegt
- # SDL_HWSURFACE - Das Surface wird im Grafikspeicher abgelegt
- # SDL_ANYFORMAT - Erlaubt jedes Pixel-Format
- # SDL_HWPALETTE - Surface nutzt exclusive Farbpalette
- # SDL_DOUBLEBUF - Surface ist "double buffered"
- # SDL_FULLSCREEN - Surface im Full-Screen-Mode initialisieren
- # SDL_OPENGL - Surface nutzt OpenGL
- # SDL_OPENGLBLIT - Surface unterstützt OpenGL blitting
- # SDL_RESIZABLE - Surfacedisplay ist grössenveränderbar
- # SDL_HWACCEL - Surface blit nutzt Hardwarebeschleunigung
- # SDL_SRCCOLORKEY - Surface nutzt colorkey blitting
- # SDL_RLEACCEL - Colorkey blitting ist durch RLE beschleunigt
- # SDL_SRCALPHA - Surface blit nutzt alpha blending



Bild anzeigen

```
#include "SDL_image.h"           // SDL kann nativ nur Bitmaps laden.
                                // Zusatzbibliothek: libSDL_image

[...]
```

SDL_Surface *image;
image = IMG_Load("tux.jpg"); // Bildgröße = Surfacegröße

```
if (image == NULL){
    std::cerr << "Das Bild konnte nicht geladen werden";
    exit(-1);
}
```

```
SDL_BlitSurface(image, NULL, display, NULL);
SDL_Flip(display);
SDL_Delay(3000);

SDL_FreeSurface(image);          // Nur das Display-Surface wird
[...]
```

// automatisch gelöscht



Größenbestimmung von Surfaces und mehr:

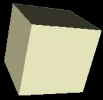
Jedes Surface (auch das Display-Surface) ist pixelgenau in seiner Größe bestimmbar mit :

- `mySurface->w` für die Breite
- `mySurface->h` für die Höhe

Die Surfaces sind so groß wie die Bilder/Objekte, die sie initialisieren.

`SDL_Flip(DisplaySurface)`

aktualisiert den kompletten Bildschirm, auch wenn nur ein kleineres Surface auf das Display geschrieben wurde.



Rectangular Areas :

Ziele:

- nur bestimmte Teilbereiche aus einem Surface blitten
- nur bestimmte Teile des Display Surfaces updaten
(wir haben ja keine ASCII White im Keller)

Lösung: `SDL_Rect`

```
typedef struct{  
    Sint16 x, y;  
    Uint16 w, h;  
} SDL_Rect;
```



optimiertes Bild anzeigen:

```
[...]
image = IMG_Load("tux.jpg");

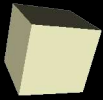
SDL_Rect ZielRect;           // Bereich, der geupdatet werden soll
ZielRect.w = image->w;       // kein scaling der SDL_Rects !
ZielRect.h = image->h;
ZielRect.x = display->w/2;
ZielRect.y = display->h/2;   // zentriert positionieren

SDL_BlitSurface(image, NULL, display, &ZielRect);

SDL_UpdateRects(display, 1, &ZielRect);

SDL_Delay(3000);

SDL_FreeSurface(image);
[...]
```



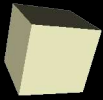
halbes Bild anzeigen:

```
[...]
image = IMG_Load("tux.jpg");

SDL_Rect SourceRect;           // Bereich, der geblittet werden soll
SourceRect.w = image->w/2;     // Bildausschnitt mit halber Breite
SourceRect.h = image->h;

SDL_Rect ZielRect;
ZielRect.w = SourceRect.w;
ZielRect.h = SourceRect.h;
ZielRect.x = display->w/2;     // Ziel zentriert positionieren
ZielRect.y = display->h/2;

SDL_BlitSurface(image, &SourceRect, display, &ZielRect);
SDL_UpdateRects(display, 1, &ZielRect);
SDL_Delay(3000);
SDL_FreeSurface(image);
[...]
```



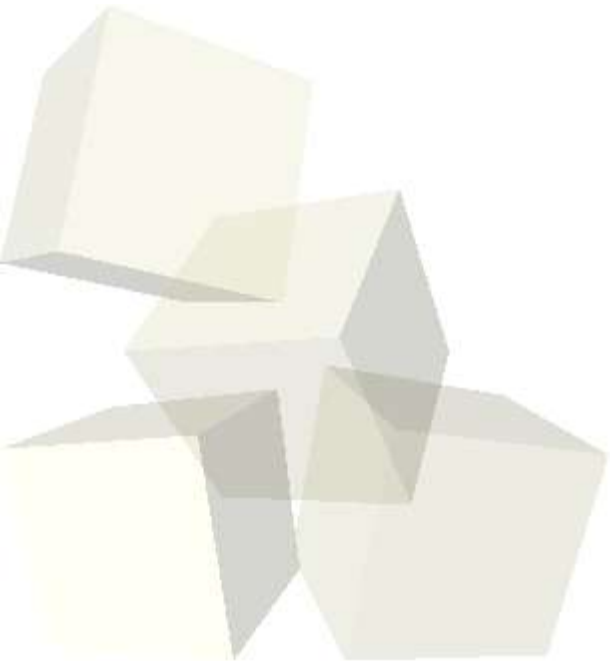
Ergebnis:





Kapitel 3:

Input Handling:





SDL_Event:

```
SDL_Event *myEvent;
```

Steuert die Maus und Tastatur-Events

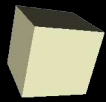
Ereignisabfrage:

Bei Animation wird die Framerate reguliert (wir haben immernoch keine ASCII White, oder ?) und die Eventqueue gepollt :

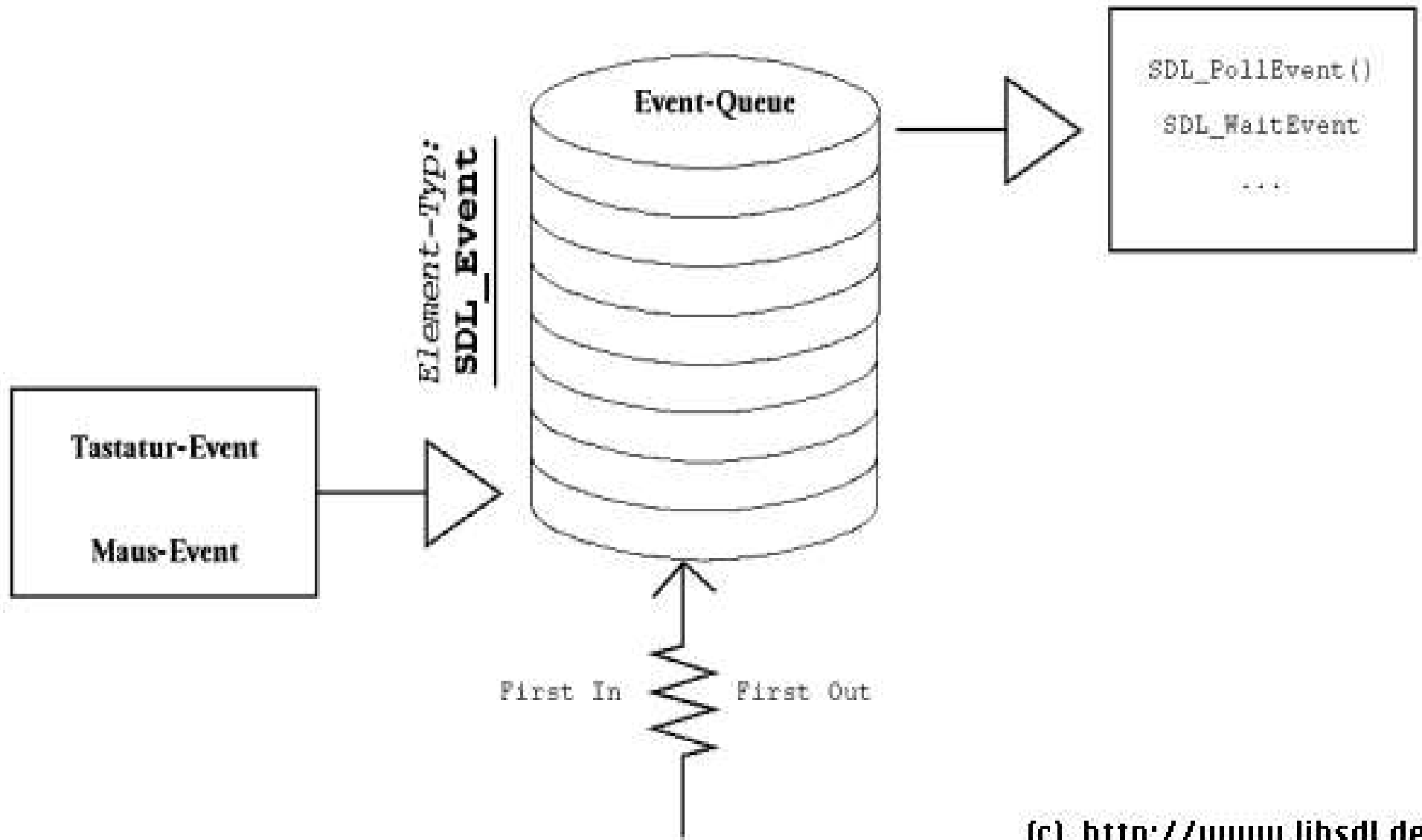
```
SDL_PollEvent(SDL_Event* SDL_Event)  
( Stichwort : Timebased Animation )
```

Bei statischen Elementen (ohne Animation und Framerateregulierung), die auf eine Eventreaktion warten, wird normalerweise :

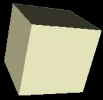
```
SDL_WaitEvent(SDL_Event* SDL_Event)  
verwendet.
```



Simple DirectMedia Layer



(c) <http://www.libsdl.de>



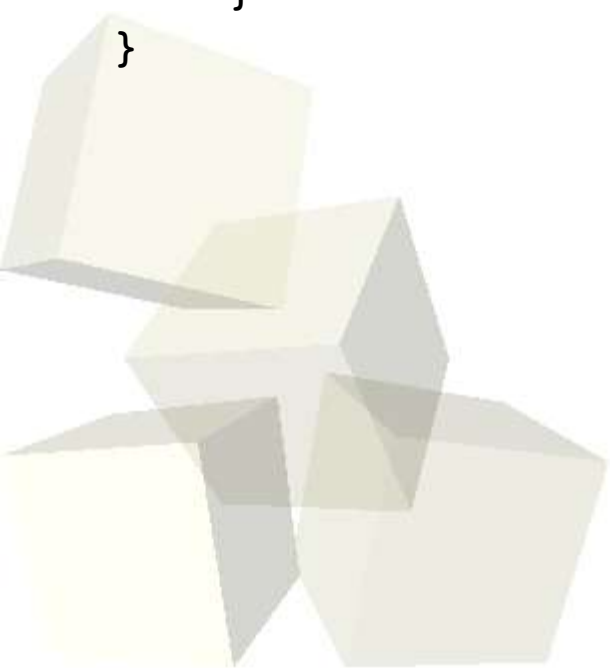
Simple DirectMedia Layer

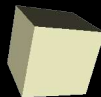
```
static void process_events( SDL_Event *event )
{
    while( SDL_PollEvent( event ) ) {
        switch( event->type ) {
            case SDL_MOUSEBUTTONDOWN:
                std::cout << "Mousebutton down" << std::endl;
                break;
            case SDL_MOUSEBUTTONUP:
                std::cout << "Mousebutton up" << std::endl;
                break;
            case SDL_MOUSEMOTION:
                std::cout << event->motion.x
                    << event->motion.y << std::endl;
                break;
            case SDL_KEYDOWN:
                handle_key_down( event->key.keysym );
                break;
            case SDL_QUIT:
                loop = false;
                break;
        }
    }
}
```



Simple DirectMedia Layer

```
static void handle_key_down( SDL_keysym* keysym )
{
    switch( keysym->sym ) {
        case SDLK_ESCAPE:
            loop = false;
            break;
        case SDLK_F1:
            ShowHelp();
            break;
        default:
            break;
    }
}
```





SDL_keysym.h

SDL_keysym

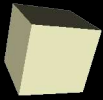
```
typedef struct{
    Uint8 scancode;
    SDLKey sym;
    SDLMod mod;
    Uint16 unicode;
} SDL_keysym;
```

SDL_KeyboardEvent

```
typedef struct{
    Uint8 type;
    Uint8 state;
    SDL_keysym keysym;
} SDL_KeyboardEvent;
```

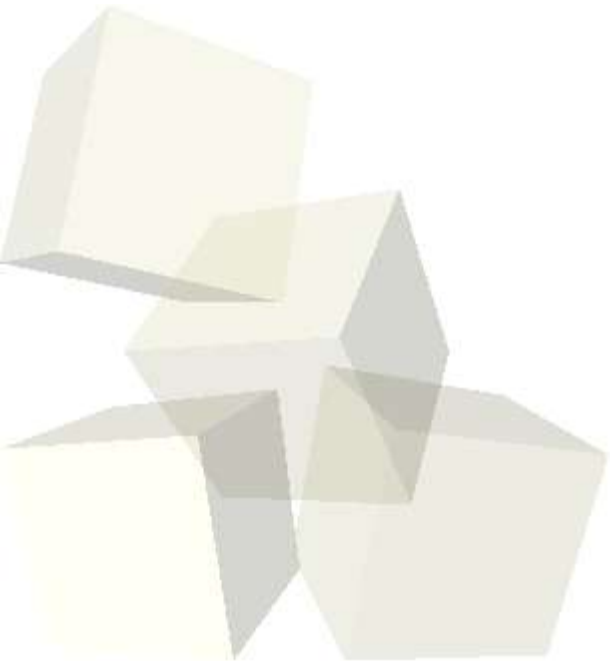
SDL_Event

```
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;
```



Kapitel 4:

Verschiedenes:





Pixel zeichnen:

```
void DrawPixel( SDL_Surface *screen,  
                int x,  
                int y,  
                Uint8 R,  
                Uint8 G,  
                Uint8 B );
```

DrawPixel ist eine Funktion, die wohl JEDER aus der SDL Dokumentation klaut (Code ist GPL). Der Grafikkartenspeicher wird in der Funktion gelockt und die Farbpunktdarstellung entsprechend der eingestellten Farbtiefe berechnet.



Farbflächen füllen:

```
int SDL_FillRect( SDL_Surface *dst,  
                  SDL_Rect *dstrect,  
                  Uint32 color );
```

füllt den angegebene SDL_Rect.

Uint32 color kann einfach mittels

```
Uint32 SDL_MapRGB( SDL_PixelFormat *fmt,  
                   Uint8 r, Uint8 g, Uint8 b);
```

erzeugen. (Pixelformat ist im Display-Surface gespeichert: display->format)

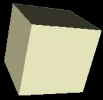


SDL_GetTicks:

```
static int TICK_INTERVAL = 10;
static Uint32 next;
static Uint32 now;

Uint32 time_left(void)
{
    now = SDL_GetTicks();    // millisec. seit SDL-Init

    if(next <= now)
        return 0;
    else
        return (next - now);
}
```



Die Hauptschleife:

[...]

```
while( loop==true ) {
```

```
    DrawStuff();                // zeichnen
```

```
    process_events( );          // Events abfragen
```

```
    SDL_Delay(time_left());     // Wartezeit absitzen
```

```
    next += TICK_INTERVAL;
```

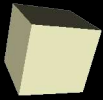
```
}
```

```
[...]
```



Kapitel 5: Zum selbst nachlesen:

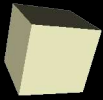
- Colorkey / Alpha-Kanal
- Zusatzbibliotheken (die Wichtigsten):
 - * SDL_gfx
 - * SDL_mixer
 - * SDL_net
 - * SDL_ttf
- SDL als OpenGL Framework



weitere Infos:

<http://www.libSDL.de> (deutsch)

<http://www.libSDL.org> (englisch)



FIN

© 2003 by Marco Kraus <marco@libsdl.de>

Das Original dieses Dokuments ist unter
<http://www.libsdl.de> zu finden.

Nachdruck nur mit Erlaubnis des Autors.