Kirill Kranz

# Vergleich von Frameworks zu Initialisierung des OpenGL-Contexts

BGPX 913

SAE Hamburg

22.1.2014

Jan-Friedrich Conrad

# Inhaltsverzeichnis

1. Vorwort ..... 3

2. Das SDL2-Framework in Theorie und Praxis ..... 4
    2.1 Die Übersicht über das Framework ..... 4
    2.2 Der Aufbau eines Programmskeletts ..... 5
    2.3 Die Integration eines Test-Codes in das Framework ..... 6
    2.4 Die Vor- und Nachteile des SDL2-Frameworks ..... 8

3. Das FreeGLUT-Framework in Theorie und Praxis ..... 9
    3.1 Die Übersicht über das Framework ..... 9
    3.2 Der Aufbau eines Programmskeletts ..... 10
    3.3 Die Integration eines Test-Codes in das Framework ..... 11
    3.4 Die Vor- und Nachteile des FreeGLUT-Frameworks ..... 13

4. Das GLfw-Framework in Theorie und Praxis ..... 14
    4.1 Die Übersicht über das Framework ..... 14
    4.2 Der Aufbau eines Programmskeletts ..... 15
    4.3 Die Integration eines Test-Codes in das Framework ..... 16
    4.4 Die Vor- und Nachteile des GLfw-Frameworks ..... 18

5. Resümee ..... 19

6. Literaturverzeichnis ..... 20

# 1.Vorwort

OpenGL ist eine Cross-Plattform-Bibliothek, die die Interaktion zwischen dem Programmierer und der 3D-Beschleuniger Graphik-Karte ermöglicht. Die Befehle von OpenGL sind unter allen Systemen identisch. Allerdings gibt es einen systemabhängigen Teil, und zwar dem Öffnen des OpenGL-Contextes. Dies geschieht für jedes System auf verschiedene Weise. Um sich viel unnötige Arbeit zu ersparen, existieren Frameworks, die diese Aufgabe erledigen. Obwohl jedes System verschieden anzusprechen ist, sind die Frameworks so gebaut, dass man als Benutzer nichts im eigenen Code verändern muss. Das Frameworks enthält alle wichtigen Fragmente, die aufs jeweilige System bezogen sind. Hier werden drei solcher Frameworks (FreeGLUT, SDL2 und GLfw) miteinander verglichen. Es sind die am meisten benutzten Frameworks, die nicht nur ein Rendering-Context[1] erzeugen, sondern auch sich um ankommende systembasierte Ereignisse kümmern und im Falle von SDL2 auch andere Aufgaben managen.

In den Programmbeispielen in folgenden Kapiteln wird ein rotierendes Dreieck in einem Fenster auf dem Bildschirm dargestellt. Dazu wird der selbe OpenGL Code in drei verschieden Programmen, jeweils mit einem anderen Framework, angepasst. Es werden alle drei Programme unter Windows mit Visual Studio kompiliert. Natürlich kann man dasselbe unter Linux oder MAC OS X machen.

1 http://msdn.microsoft.com/en-us/library/windows/desktop/dd369038%28v=vs.85%29.aspx

# 2. Das SDL2-Framework in Theorie und Praxis

## 2.1 Die Übersicht über das Framework

"Die Möglichkeit, ein Spiel unter Windows zu schreiben und dann des Weiteren auf MAC OS X oder Linux zu kompilieren, mit wenigen bis keinen Veränderungen im Code, ist sehr mächtig und perfekt für Entwickler, die so viele Systeme adressieren wollen wie möglich; SDL2 macht solche Crossplattform Entwicklung zu einem Kinderspiel"[2] SDL2 wird verbreitet unter der Zlib-License[3], die es erlaubt, diese Bibliothek in jeder Art von Software zu benutzen. SDL2 wird zur Zeit der Veröffentlichung dieses Textes immer noch weiter entwickelt und trägt die Version 2.0.1. Die SDL2 Bibliothek unterstützt offiziell das Android, das Windows, das MAC OS X, das Linux und das iOS System. Es existiert allerdings nicht offiziell die Unterstützung für FreeBSD, NetBSD, OpenBSD und Solaris Systeme. Veränderung im Quellcode des Benutzers ist nicht nötig. Es ist lediglich eine Neukompilierung unter dem jeweiligen System notwendig.
SDL2 arbeitet auf Basis eines Ereignis-Pools, der die systemabhängigen Ereignisse in universale konvertiert. Es wird ein Main-Loop aufgebaut, in welchem einerseits diese Ereignisse bearbeitet werden und andererseits die Interaktion des Programmierers, z.B. das Zeichnen der Graphik oder das Updaten der Programm-Mechanik, bearbeitet wird.
SDL2 besitzt neben dieser Haupteigenschaft auch Teile, die per Bedarf dazugenommen werden können, falls man das Arbeiten mit verschiedenen Bildspeicher-Formaten, einem Font-System, Musik sowie Soundeffekte und Netzwerk braucht. Diese Teile sind optional zu dem Framework installierbar.

2 Mitchell, 2013. S.6. Übersetzt von Kranz
3 Siehe: http://www.gzip.org/zlib/zlib_license.html

## 2.2 Der Aufbau eines Programmskeletts

Das Einbinden von der Header-Datei SDL.h ist notwendig. Als zweiten Schritt ist die Initialisierung von SDL2 durch *SDL_Init()* notwendig. Als Parameter werden dieser Funktion die zu initialisierenden Subsysteme aufgeführt. In unserem Fall wird nur *SDL_INIT_VIDEO* gebraucht. Als Nächstes wird ein Fenster erzeugt, das später mit dem OpenGL-Context verbunden wird. Das Erzeugen dieses Fensters wird mit dem Befehl *SDL_CreateWindow()* realisiert. Als Parameter werden die Position, die Auflösung, der Name des Fensters sowie eine explizite Anweisung *SDL_WINDOW_OPENGL* übergeben. Nun wird diesem Fenster der OpenGL-Context zugeordnet. Dafür sorgt der Befehl *SDL_GL_CreateContext()*. Ab jetzt ist es möglich, durch die OpenGL Befehle, z.B. *glBegin()* und *glEnd()*[4], zu zeichnen. Falls in der Initialisierungsphase ein Fehler auftritt, sorgt der Befehl *SDL_GetError()*, der eine passende Fehlerbeschreibung zurückgibt, für Klarheit.

Das Programm läuft in einer Schleife, bis die Bedingung zum Beenden erfüllt wird. In dieser Schleife werden die vom System ankommenden Ereignisse bearbeitet. Durch *SDL_PoolEvent()* wird ein Ereignis aus dem Ereignis-Pool geholt und gegebenenfalls von dem Programmierer behandelt, z.B. das Ereignis *SDL_QUIT* tritt auf, falls der Benutzer das SDL2-Fenster zu schließen versucht. So muss man die Beenden-Bedingung der Hauptschleife erfüllen, somit verlässt das Programm diese Schleife und kann nach dem Abschluss-Part geschlossen werden. Als Ereignis kann man z.B. auch die Tastatureingabe betrachten. In dem Codebeispiel wird das Drücken auf die Taste ESC geprüft. In diesem Fall wird die Beenden-Bedingung der Hauptschleife auch erfüllt und diese wird beendet.

Man muss nicht alle ankommenden Ereignisse bearbeiten, es reicht, nur die zu implementieren, die man auch tatsächlich braucht.

Nach dem Code, der für die Ereignisbehandlung zuständig ist, wie es im unten vorgestellten Programmbeispiel dargestellt ist, sollte der Aufruf der Routine für das Zeichnen eines Dreiecks durch OpenGL Befehle folgen.

Als Letztes wird das Zeichnen des Bildes auf dem Bildschirm mit dem Befehl *SDL_GL_SwapWindow()* realisiert.

4 Siehe Reminder, 2009. S 82.

## 2.3 Die Integration eines Test-Codes in das Framework

```
#include <string>
#include <iostream>
#include <Windows.h>
#include <gl/GL.h>
#include <gl/GLU.h>

#include <SDL.h>

using namespace std;


unsigned int screenwidth = 800;
unsigned int screenheight = 600;


void error(string errstr)
{
	cout << errstr << endl;
	system("pause");
	exit(-1);
}


void RenderFunction(void)
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glRotatef(1,0,1,0);

	glBegin(GL_TRIANGLES);
		glColor3f(1,0,0);
		glVertex3f( 0.0f, 1.0f, 0.0f);		// Top
		glColor3f(0,1,0);
		glVertex3f(-1.0f,-1.0f, 0.0f);		// Bottom Left
		glColor3f(0,0,1);
		glVertex3f( 1.0f,-1.0f, 0.0f);		// Bottom Right
	glEnd();
}



int main(int argc, char** argv)
{
     if ( SDL_Init(SDL_INIT_VIDEO) != 0 )
		error("Unable to initialize SDL2: " + (string) SDL_GetError());


	SDL_Window *SDLwindow = SDL_CreateWindow("SDL2 test window",
					SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
					screenwidth,screenheight, SDL_WINDOW_OPENGL);
	if (!SDLwindow) error("SDL_CreateWindow: "+ (string) SDL_GetError());


	SDL_GLContext SDLopenGLcontext = SDL_GL_CreateContext(SDLwindow);
	if(!SDLopenGLcontext)
		error("Unable to initialize SDL2 OpenGL context: " + (string) SDL_GetError());
```

```
	bool quit=false;
	SDL_Event event;
	while (!quit)
	{

			//Event polling
			while (SDL_PollEvent(&event))
			{
				switch(event.type)
				{
					case SDL_QUIT: { quit = true; } break;
					case SDL_KEYDOWN:
						{
							switch (event.key.keysym.sym)
							{
								case SDLK_ESCAPE: {quit = true;}break;
								default: break;
							}
						} break;
					default: break;
				}
			 }



			RenderFunction();

			SDL_GL_SwapWindow(SDLwindow);

	}



	SDL_GL_DeleteContext(SDLopenGLcontext);
	SDL_DestroyWindow(SDLwindow);
	SDL_Quit();

	return 0;
}
```

### 2.4 Die Vor- und Nachteile des SDL2-Frameworks

Zu den Vorteilen Des SDL2-Frameworks gehören seine Aktualität sowie seine Portabilität, sogar auf Geräten die nicht als PC eingestuft werden, wie Android oder iOS Geräte. Das Umwandeln der Benachrichtigungen, die vom System kommen, in einheitliche Ereignisse sorgt dafür, dass man ohne das Umschreiben des Quellcodes auf jedem System, auf dem SDL2 kompilierbar ist, operieren kann. Durch das Konzept der Übergabe einer Event-Struktur, aus der der Programmierer nur gebrauchte Ereignisse prüft und bearbeitet, wird durch den Einsatz einer Schleife die Mechanik der Applikation ungemein erleichtert.
Es ist möglich, die Eigenschaften des erzeugten Fensters sowie auch des OpenGL-Contextes durch Übergabe von States an SDL2 zu verändern. Dies erlaubt zusätzliche Flexibilität. Es ist z.B. Möglich, die Core-Version[5] von OpenGL zu setzen oder im Kompatibilitätsmodus zu arbeiten.
Ein Nachteil ist die relative große Bibliothek, die das Basis-Framework beherbergt, die immer mitgeliefert sein muss. Diese wiegt über 850Kb und ist unter Windows, bei Benutzung von Visual Studio, nur dynamisch erhältlich.

5 Sehe: http://www.opengl.org/wiki/Core_And_Compatibility_in_Contexts

# 3. Das FreeGLUT-Framework in Theorie und Praxis

## 3.1 Die Übersicht über das Framework

"Die Handhabung von Systembefehlen auf der untersten Ebene, besonders für grafische Applikationen, kann ein langweilige und schmerzhafte Arbeit sein, doch die FreeGLUT Bibliothek wurde erschaffen, um Menschen wie uns zu helfen, OpenGL-basierte Programme ohne solche Belastungen zu schreiben"[6] Das FreeGLUT-Framework ist eine Open Source Weiterentwicklung des GLUT, das seit 1998 sich nicht verändert hat. Dies ist einer der Gründe, warum das Standard-GLUT-Paket als Framework nicht in Vergleich gestellt wird und es sich eher auf FreeGLUT, dessen letzte Version 2.8.1 am 5 April 2013 veröffentlicht wurde, konzentriert wird. Das FreeGLUT wird unter der X-Consortium license[7] veröffentlicht, was das Benutzen dieses Programmpakets im kommerziellen, dessen Quellcode nicht frei zu Verfügung steht, sowie auch im nicht-kommerziellen Sinne erlaubt. Die FreeGLUT Bibliothek erlaubt die Entwicklung unter WINDOWS, LINUX sowie auch unter MAC OS X Systemen. Veränderung im Quellcode des Benutzers ist nicht nötig. Es ist lediglich eine Neukompilierung unter dem jeweiligen System notwendig. FreeGLUT arbeitet auf der Basis der sogenannten Callbacks. Dies bedeutet, dass pro Aktion, wie z.B. dem Druck einer Taste auf der Tastatur, ein Aufruf einer für diesen Zweck zugeordneten Funktion stattfindet. Man muss verschiedene Callbacks mit Funktionen belegen, um die Kontrolle über das Programm zu haben, das sich in einer Schleife nach dem Aufruf von *glutMainLoop()* befinden wird. Die oben genannte Schleife interagiert mit dem jeweiligen Betriebssystem und behandelt dessen Ereignisse so, dass nur vom User gewünschte Callbacks ausgeführt werden und die nicht belegten Ereignisse eine Standardbehandlung erfahren. Zum Zeichnen auf dem Bildschirm bzw. das Bearbeiten der Programmmechanik wird durch den Callback der Render-Methode realisiert.

6 Dickinson, 2013. S. 11. Übersetzt von Kranz.
7 http://opensource.org/licenses/mit-license.html

### 3.2 Der Aufbau eines Programmskeletts

Das Einbinden von der Header-Datei freeglut.h ist notwendig. Als zweiter Schritt ist die Initialisierung von FreeGLUT durch die *glutInit()* Funktion notwendig. Als Parameter erhält diese in der *main()* übergebenen Kommandozeilenparameter, mit denen das Programm gestartet wurde. Danach folgt der *glutInitWindowSize()* Befehl, der als Parameter die gewünschte Größe des zu eröffnenden Fensters annimmt. Gefolgt von *glutInitDisplayMode()*, welche im Normalfall drei, durch ein logisches *ODER* getrennte Parameter annimmt: *GLUT_DEPTH* für das Initialisieren des Depth-Buffers, *GLUT_DOUBLE* für das Initialisieren des Double-Buffers und *GLUT_RGBA* für das Setzen des Pixelformats. Mit dem Befehl *glutCreateWindow()*, der als Parameter den Titel des Fensters annimmt und ein *int* als Window-Handler zurückgibt, wird ein OpenGL-fähiges Fenster erzeugt, falls der Window-Handler nicht weniger als Eins ist.
Jetzt ist es an der Zeit, die Callback Funktionen zu initialisieren. In dieser Abhandlung werde ich nur drei von ihnen anpassen: *glutReshapeFunc(),* die als Parameter eine *void* Funktion, mit zwei Parametern, die von FreeGLUT auf die neue Größe des Fensters gesetzt werden, erwartet. Diese Funktion ist leider notwendig, da FreeGLUT nur ein Fenster, das in der Lage ist, seine Größe zu verändern, erzeugt. Doch wir können, nachdem man das Ändern der Fenstergröße mit der Maus abgeschlossen hat, auf ihre vorgegebene Größe wieder zurücksetzen. Dies geschieht mit der Funktion *glutReshapeWindow(),* die die Größe des Fensters als Parameter übernimmt.
Die Funktion *glutKeyboardFunc(),* die den Callback für die Tastaturereignisse einrichtet muss unbedingt auch eingestellt werden. Diese gibt ein *void* zurück und erwartet als Parameter einen u*nsigned char* und folglich zwei *ints*. Der *char* Parameter wird von FreeGLUT so gesetzt, dass er die Nummer der gedrückten Taste der PC-Tastatur widerspiegelt. In dieser Funktion wird geprüft, ob die ESC-Taste gedrückt worden war. Falls das der Fall ist, wird ein Programmabbruch initiieren.
Als Letztes ist die der Callback der Funktion zu setzten, die, wenn die Zeit dafür gekommen ist, auf den Bildschirm zeichnet. Dafür ist die Funktion *glutDisplayFunc()* zuständig. Als Parameter dieses Befehls ist eine *void* Funktion ohne Parameter zu setzten.
Nun ist die Zeit gekommen, einfach den Main-Loop zu starten. g*lutMainLoop()* tut dies. Das Programm wird nun komplett durch die Callbacks steuerbar.
Natürlich ist es möglich, Callbacks für z.B. die Behandlung der Maus-Aktivitäten, sowie eine ganze Reihe anderer Callbacks, die ich hier nicht angesprochen werden, zu setzen.

### 3.3 Die Integration eines Test-Codes in das Framework

```
#include <iostream>
#include <string>
#include <Windows.h>

#include <gl/GL.h>
#include <gl/GLU.h>
#include "freeglut.h"

using namespace std;


int    CurrentWidth = 800,
       CurrentHeight = 600,
       WindowHandle = 0;

void processKeys(unsigned char key, int x, int y)
{
    if(key==27) exit(0);
}

void ResizeFunction(int Width, int Height)
{
    glutReshapeWindow(CurrentWidth, CurrentHeight);
}

void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

       glRotatef(1,0,1,0);

       glBegin(GL_TRIANGLES);
              glColor3f(1,0,0);
              glVertex3f( 0.0f, 1.0f, 0.0f);              // Top
              glColor3f(0,1,0);
              glVertex3f(-1.0f,-1.0f, 0.0f);              // Bottom Left
              glColor3f(0,0,1);
              glVertex3f( 1.0f,-1.0f, 0.0f);              // Bottom Right
       glEnd();


    glutSwapBuffers();
}


int main(int argc, char* argv[])
{
       glutInit(&argc, argv);

       glutInitWindowSize(CurrentWidth, CurrentHeight);
       glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
       WindowHandle = glutCreateWindow("FreeGLUT test window");
        if(WindowHandle < 1)
        {
              cout << "ERROR: Could not create a new rendering window." << endl;
              system("pause");
              exit(-1);
        }
```

```
        glutReshapeFunc(ResizeFunction);
        glutKeyboardFunc(processKeys);
        glutDisplayFunc(RenderFunction);


    glutMainLoop();


        return 0;

}
```

### 3.4 Die Vor- und Nachteile des FreeGLUT-Frameworks

Zu den Vorteilen des FreeGLUT-Frameworks gehört das Callback-Ereignisverarbeitung-System. Es ist möglich, fast für jedes Ereignis eine Callbackroutine einzurichten. "Die wiederholte Ausführung der bereits definierten Hauptschleifenfunktionen wird schließlich durch *glutMainLoop()* eingeleitet"[8]

Nachteilhaft ist das Aufrufen der Funktion, die die Bildschirmpuffer vertauscht und somit das Doublebuffering organisiert, mit in die Callback-Funktion zum Zeichen eingebaut werden muss.

Ein weiterer Nachteil ist, dass es unmöglich ist, in diesem Framework die variable Größe des Fensters auszustellen. Man kann zwar ein Callback auf das Bildgrößenveränderung-Ereignis setzen, in dem man das Fenster einfach wieder auf die gewünschte Größe zurücksetzt, doch der Benutzer kann, bevor das Callback aufgerufen wird, die Größe des Fensters verändern. Erst nach dem "Loslassen" des Fensters wird dieses Callback aufgerufen.

Außerdem wiegt die unter Visual Studio kompilierte FreeGLUT Bibliothek, als eine dynamisch linkbare Datei, über 330KB. und muss immer mitgeliefert werden.

8 Apetri, 2010. S. 821

# 4. Das GLfw-Framework in Theorie und Praxis

## 4.1 Die Übersicht über das Framework

Das GLfw-Framework ist ein Open Source Projekt, das unter der zlib/libpng-license[9] für kommerzielle sowie nichtkommerzielle Zwecke zu Verfügung steht. Die aktuelle Version ist 3.0.4 und wurde am 31.12.2013 veröffentlicht, was eine ständige Weiterentwicklung beweist. Diese Bibliothek umfasst die Systeme Windows, Linux und MAC OS X. Veränderung des Quellcodes des Benutzers ist nicht notwendig, lediglich eine Neukompilierung unter dem gewünstem System ist notwendig.
GLfw arbeitet teils mit der Callback Architektur, teils mit einer Hauptschleife, die so konzipiert ist, dass in ihr die Mechanik der Applikation stattfindet. GLfw behandelt vom Benutzer einkommende Ereignisse, wie z.B. von der Tastatur oder der Maus, mit Callbacks jedoch erst durch den Aufruf einer speziellen Funktion in der Hauptschleife des Programms. Es ist sogar möglich, unabhängig von der Hauptschleife ein Callback für die Fehlerbehandlung einzurichten. Es ist möglich, mit GLfw bis zu 10 Fenster zu erzeugen und darauf zu rendern.

9 Sehe: http://opensource.org/licenses/zlib-license.php

### 4.2 Der Aufbau eines Programmskeletts

Die Einbindung der Header-Datei glfw3.h ist notwendig. Des Weiteren ist das Setzen des Fehlerbehandlung-Callbacks ratsam. Dies geschieht mit der Funktion *glfwSetErrorCallback(),* die als Parameter eine Funktion des Typen *void,* die wiederum die Parameter: *int error,* für die Fehlernummeridentifikation und einen konstanten *char pointer* für die Beschreibung des Fehlers als einen Text, erwartet. Nach der Einrichtung eines Fehlermanagers kann nun das Framework durch den Befehl *glfwInit()* initialisiert werden. Die Funktion *glfwWindowHint(),* die mit dem ersten Parameter *GLFW_RESIZEBLE* und dem zweiten *GL_FALSE* aufgerufen wird, sorgt dafür, dass das mit dem Befehl *glfwCreateWindow()* erzeugte Fenster nicht vom Benutzer skaliert werden kann. Der Befehl *glfwCreateWindow()*, der als Parameter die Größe des Fensters sowie dessen Namen und einige weitere Parameter erwartet, gibt einen Zeiger auf den Typ *GLFWwindow* zurück. Falls beim Erstellen des Fensters ein Fehler aufgetreten ist, ist der Zeiger, den die Funktion *glfwCreateWindow()* zurückliefert, *NULL*. Somit ist das Erschaffen des Fensters fehlgeschlagen. In diesem Falle muss *glfwTerminate()* aufgerufen werden, gefolgt von der Fehlerbehandlung-Callback-Routine. Der Aufruf von *glfwMakeContextCurrent()* mit dem Zeiger, den die Funktion *glfwCreateWindow()* zurückgegeben hat, als Parameter übergeben, sorgt für das Binden des OpenGL-Contextes mit dem erzeugten Fenster. Jetzt ist an der Zeit, alle anderen Callbacks zu setzen. In dem unterem Code Beispiel wird nur der Callback, der für die Tastaturabfrage zuständig ist, gesetzt. Dies erlaubt die Funktion *glfwSetKeyCallback()*, die als ersten Parameter das Fenster der Applikation braucht und als zweiten Parameter die Funktion, die das Bearbeiten des Callbacks erwartet. Nun wird die Hauptschleife beschrieben. Sie läuft solange, bis die Funktion *glfwWindowShouldClose()* einen wahren Wert einnimmt. In der Schleife wird als Erstes normalerweise die Funktion *glfwPoolEvent()* stehen. Diese bearbeitet den Ereignis-Pool und ruft die gesetzten Callbacks auf. Danach kann man die Grafik mittels OpenGL-Befehlen auf den erzeugten Context zeichnen. Als Letztes sollte der Befehl zu dem Vertauschen des Framebuffers[10] stehen.
Wenn die Bedingung des Ausgangs aus der Hauptschleife gegeben ist, sollte das Deinitialisieren des Frameworks mittels den Befehlen *glfwDestroyWindow()* und *glfwTerminate()* geschehen.

10 Sehe: Apetri, 2008. S. 241

## 4.3 Die Integration eines Test-Codes in das Framework

```
#include <string>
#include <iostream>
#include <Windows.h>
#include <gl/GL.h>
#include <gl/GLU.h>

#include "glfw3.h"
#include <stdlib.h>
#include <stdio.h>

using namespace std;


unsigned int screenwidth = 800;
unsigned int screenheight = 600;

void error_callback(int error, const char* description)
{
	cout << description << endl;
	system("pause");
	exit(-1);
}

void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
	if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
					glfwSetWindowShouldClose(window, GL_TRUE);
}


void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glRotatef(1,0,1,0);

	glBegin(GL_TRIANGLES);
		glColor3f(1,0,0);
		glVertex3f( 0.0f, 1.0f, 0.0f);		// Top
		glColor3f(0,1,0);
		glVertex3f(-1.0f,-1.0f, 0.0f);		// Bottom Left
		glColor3f(0,0,1);
		glVertex3f( 1.0f,-1.0f, 0.0f);		// Bottom Right
	glEnd();
}

int main(void)
{
	glfwSetErrorCallback(error_callback);

	if (!glfwInit()) error_callback(0,"Error: glfwInit()");

	glfwWindowHint( GLFW_RESIZABLE, GL_FALSE );

	GLFWwindow* window;

	window = glfwCreateWindow(screenwidth, screenheight, "GLfw example", NULL, NULL);
	if (!window)
```

```
	{
		glfwTerminate();
		error_callback(0,"Error: glfwCreateWindow()");
	}

	glfwMakeContextCurrent(window);

	glfwSetKeyCallback(window, key_callback);

	while (!glfwWindowShouldClose(window))
	{

		glfwPollEvents();

		RenderFunction();

		glfwSwapBuffers(window);

	}

	glfwDestroyWindow(window);
	glfwTerminate();
}
```

### 4.4 Die Vor- und Nachteile des GLfw-Frameworks

Zu den Vorteilen des GLfw-Frameworks gehört einerseits der Vorteil der CallBack basierten Ereignisbehandlung und andererseits, dass es eine Hauptschleife gibt, die mehr Flexibilität ermöglicht. Die Verarbeitung der Fehler durch einen Callback ist auch sehr vorteilhaft, da Fehler manchmal da auftauchen können, wo man nicht mit einer direkten Fehlerprüfung Zugang hat. Die Einstellung des erzeugten Fensters kann explizit geändert werden, somit kann z.B. das Skalieren des Fensters unterdrückt werden. Ein weiterer Vorteil ist es, dass das GLfw-Framework keine dynamische Bibliothek, die noch extra Platz auf der Festplatte raubt, benötigt.

# 4. Resümee

Das SDL2-Framework ist wohl das mächtigste von den drei verglichenen. Viele kommerzielle Spieletitel nutzen es, um eine Plattformunabhängigkeit zu erreichen. Es ist das Framework mit den meisten Einstellungsmöglichkeiten. Das SDL2-Framework enthält nicht nur Routinen zur Verarbeitung von Systemereignissen, sondern eine ganze Reihe von Befehlen, z.B. Befehle, um mit Dateien oder mit Thereads zu arbeiten, um ein Interface für alle Systeme zu bieten. Dank der optionalen Teile, die für das Laden von Bildern in verschiedenen Formaten, einen Modul, der für Soundeffekte und Musik zuständig ist, einen Plugin für das Laden von TrueType Fonts und nicht zu vergessen eine Bibliothek, die mit dem Netzwerk zu arbeitet, bietet SDL2 ein umfangreiches Paket an Möglichkeiten und lässt einen Programmierer sich nie wieder Sorgen machen, welche Programmteile er zusammenpuzzeln muss, um plattformunabhängig mit den oben genannten Vorteilen eine Applikation zu erschaffen.
Im Gegensatz zum SDL2-Framework ist die FreeGLUT-Framework Alternative sehr viel einfacher gestrickt. Diese Framework enthält weder Plugins für Musik, Soundeffekte, das Laden von verschiedenen Bilddateiformaten oder Netzwerkunterstützung. Es enthält nicht einmal Funktionen zum Behandeln von Dateien oder Threads unter verschiedenen Systemen. Das FreeGLUT-Framework wird jedoch häufig als ein Einsteiger-Framework angesehen. In vielen Büchern wird es für die Beispiele genutzt. Es eignet sich gut, um ein OpenGL-Beispiel-Programm, das nicht mehr als eine Funktionalität in sich darstellen soll. Für größere Projekte ist dieses Framework eher ungeeignet.
Von der Funktionalität ist das GLfw-Framework eine goldene Mitte zwischen den SDL2-Framworkes und des FreeGLUT-Frameworks, es wird zwar hier auch kein Management für Dateien oder Threads gegeben. Auf der einen Seite ist die ziemlich kompakte Ausführung, die sogar ohne eine externe Datei, die dynamisch in das Programm eingebunden sein muss, auskommt. Auf der anderen Seite ist der Weg, das Programm durch eine Hauptschleife zu verwalten, sehr praktisch. Es wird zwar nicht komplett auf Callbacks verzichtet, jedoch wird zum Zeichnen mittels OpenGL-Befehlen oder zum Updaten der Mechanik pro Schleifendurchgang nicht mit Callbacks realisiert. Dieses Framework eignet sich auch für größere Projekte sowie für kleine Beispielprogramme, z.B. aus Fachartikeln.

# 6. Literaturverzeichnis


[1] Microsoft, Rendering Contexts.
http://msdn.microsoft.com/en-us/library/windows/desktop/dd369038%28v=vs.85%29.aspx
2012, abgeruffen Januar 2014

[2] Mitchell, Shaun: SDL Game Development.
Erste Auflage. Packt Publishing: Birmingham, 2013

[3] Jean-loup Gailly und Mark Adler, Z-Lib License.
http://www.gzip.org/zlib/zlib_license.html 2004. abgeruffen Januar 2014

[4] Reminder, Wolfgang: Spieleprogrammierung mit Cocoa und OpenGL.
Erste Auflage. Smart Books Publishing: Pfäffikon, 2009

[5] OpenGL.org, Core And Compatibility in Contexts.
http://www.opengl.org/wiki/Core_And_Compatibility_in_Contexts
2012, abgeruffen Januar 2014

[6] Dickinson, Chris: Learning Game Physics with Bullet Physics and OpenGL.
Erste Auflage. Packt Publishing: Birmingham, 2013

[7] Open Source Initiative, The MIT License.
http://opensource.org/licenses/mit-license.html
abgeruffen Januar 2014

[8] Apetri, Marius: 3D-Grafik mit OpenGL Das umfassende Praxis-Handbuch.
1. Auflage. mitp Verlag: Heidelberg, 2010

[9] Open Source Initiative, The zlib/libpng License.
http://opensource.org/licenses/zlib-license.php
abgeruffen Januar 2014

[10] Apetri, Marius: 3D-Grafik Programmierung.
2. Auflage. mitp Verlag: Heidelberg, 2008