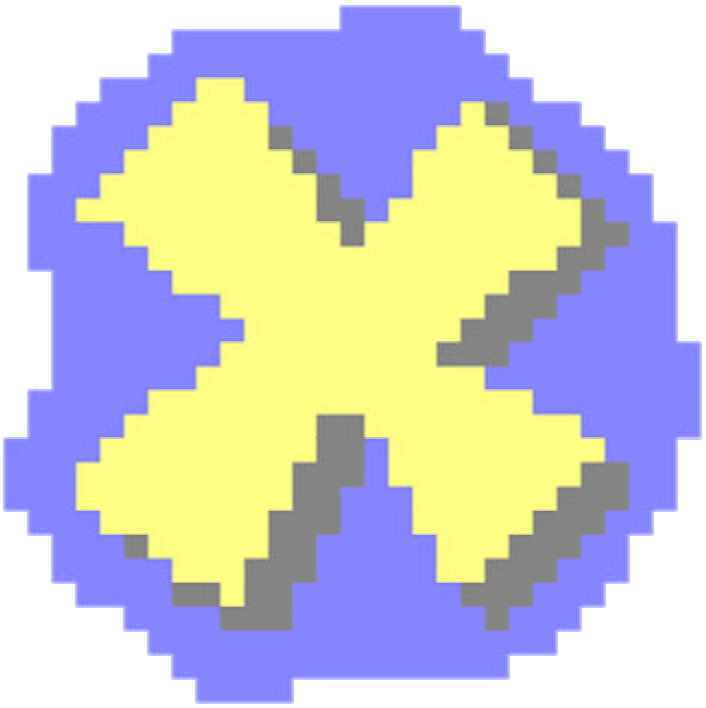
Angewandte DirectX-Programmierung



Sebastian Böhm

Bundesgymnasium und Bundesrealgymnasium Wien 23
Vienna Bilingual Schooling
Draschestraße 90-92

Angewandte DirectX-Programmierung

Fachbereichsarbeit aus Informatik

eingereicht von: Sebastian Böhm, Klasse 8B
bei: Mag. Karin Reinbacher

2001/2002

Wien, 13. Februar 2002

INHALTSVERZEICHNIS

Kapitel 1 Vorwort

Inhalt dieser Arbeit

Diese Arbeit setzt sich mit dem Thema „Angewandte DirectX-Programmierung“ auseinander, was bedeutet, dass die Beschäftigung mit dem Thema „DirectX“ eine sehr praxisnahe sein soll. In je einem eigenen Kapitel werden die Grundlagen der Benutzung der Schnittstellen DirectDraw, DirectSound und DirectInput erläutert, wobei es zu jedem Kapitel auch ein oder mehrere Sourcecode-Beispiel(e) gibt, die die genaue Art der Benutzung demonstrieren. Weiters befinden sich im Anhang die Quellcodes zu weiteren von mir verfassten Anwendungen, die die Benutzung von DirectX erklären bzw. erleichtern sollen, und die ich parallel zur Verfassung dieser Fachbereichsarbeit programmiert habe. Dies wären im Detail: Eine Container-Anwendung für Echtzeit-Multimedia-Anwendungen, eine DirectDraw-Grafikengine, ein DirectSound-Wrapper sowie ein DirectInput Wrapper. Abgesehen davon habe ich ein Tetris-artiges Spiel namens „*snip*“ programmiert, das als Beispiel für die Benutzung meiner Engine bzw. meiner Wrapper dienen soll. Alle erwähnten Utility-Sourcen sind im Anhang des Textes abgedruckt, während der Quellcode meines Spiels nur auf der unten erwähnten Seite abrufbar ist, da er den natürlichen Umfang einer Fachbereichsarbeit sprengen würde.

Diese Fachbereichsarbeit, sowie die Quellcodes meiner Zusatzprogramme liegen diesem Dokument auf CD-ROM bei und sind unter dem URL http://snip.sourceforge.net erhältlich.

Während die Demo-Programme (am Ende jedes Kapitels) Public-Domain sind, habe ich alle anderen Quellcodes unter die *Gnu Public License* (siehe Anhang A) gestellt.

Motivation für die Themenwahl

Da ich mich schon seit geraumer Zeit für Computer interessiere, war es für mich von vornherein klar, dass zu meinen Maturafächern Informatik gehören würde. Hierbei kam mir meine Begeisterung für das Programmieren und den Umgang mit Computern im Allgemeinen zugute. Im Besonderen interessiere ich mich für Multimedia-Anwendungen und daher fasste ich den Entschluss, DirectX als Thema meiner Fachbereichsarbeit zu wählen.

In meiner Arbeit setze ich Grundkenntnisse in den Bereichen C++-Programmierung, Win32-Programmierung sowie allgemeines Grundwissen über Computer und Multimedia-Anwendungen voraus und gehe nicht weiter auf diese Teilaspekte ein.

Kapitel 2 Grundlagen

Was ist DirectX?

DirectX ist eine Sammlung von APIs[1] für Windows-Computer. Diese erlaubt es, Multimedia-Anwendungen zu programmieren, die auf allen Computern, auf denen die DirectX-Laufzeit-Komponenten installiert sind, ausgeführt werden können. Applikationen, die von DirectX Gebrauch machen, zeichnen sich im Vergleich zu Anwendungen, die betriebsystemeigene Schnittstellen verwenden, üblicherweise durch sehr gute Performance aus, was durch die Realisierung von DirectX als Schnittstelle zu den Hardware-Treibern erreicht wurde. Die Umsetzung von DirectX-Funktionsaufrufen erfolgt auf direktem Weg über die jeweiligen Treiber – sofern sie DirectX unterstützen, wofür die Hersteller selbst verantwortlich sind. Programmiert wird DirectX meist mit High- bzw. Mid-Level-Programmiersprachen wie C++ und C, grundsätzlich ist dies aber auch mit Sprachen wie etwa Assembly oder Visual Basic möglich. DirectX wird von Microsoft (weiter)entwickelt und sowohl die Laufzeit-Komponenten, als auch das SDK[2], das benötigt wird, um DirectX-Anwendungen zu programmieren, sowie eine dazugehörige Dokumentation (die DirectX Programmer's Reference) sind kostenlos verfügbar.

Die Geschichte von DirectX

Einer der Anwendungsbereiche, der die Computer-Hardware-Entwicklung der letzten 20 Jahre wohl am meisten vorangetrieben hat, ist zweifellos die Verwendung von Computern als Entertainment-System. Heimcomputer hätten sich nie so lange auf derart erfolgreiche Weise verkaufen können, wenn nicht Computerspiele die Anforderungen, die an neue Rechner gestellt werden ständig erhöhen würden. Auch hätten Hardware-Firmen wie etwa NVidia, ATI, Intel oder auch AMD wohl kaum ihren jetzigen Bekanntheits- und Liquiditätsgrad erreicht, gäbe es keine Computerspiele. Kurz gesagt: Computerspiele stellen einen wesentliches Kriterium für die Durchsetzung einer bestimmten Hardware- oder Software-Plattform dar.

Der erste Boom der Computerspiele in den frühen 80er-Jahren brachte diesbezüglich ganz klare Vorteile für (IBM-) PCs[3] und Apples[4]. Obwohl der Apple dem IBM-PC

[1] ***A**pplication **P**rogramming **I**nterface*
[2] ***S**oftware **D**evelopment **K**it*
[3] **P**ersonal **C**omputer, ursprünglich exklusiv von IBM (International Business Machines) hergestellt.

technologisch um Lichtjahre voraus war, konnte er sich im Privatsektor nur anfangs (siehe Apple II) wirklich durchsetzen.[5] Später gewann das PC-Konzept aufgrund geringerer Kosten und leichterer Erweiterbarkeit zunehmend an Bedeutung. Eine weitere Schlüsselrolle hierfür spielte auch die Tatsache, dass die Bandbreite verfügbarer Spiele beim PC viel höher war.

Das damalige Standard-PC-Betriebssystem MS-DOS bot von sich aus keinerlei Multimedia-Fähigkeiten. Da aber der Bedarf danach bestand, begannen Entwickler bald mit der direkten Programmierung ihrer Hardware. Ein großer Vorteil dieser Technik lag darin, dass man auf diese Weise enorm schnelle Anwendungen die nahezu ohne Overhead von Seiten des Betriebssystems auskamen, erstellen konnte. Der größte Nachteil war hingegen, dass wegen der erstaunlich großen Anzahl verschiedener System-Konfigurationen oft mehr Zeit mit dem Hardware-Troubleshooting als mit der Anwendungsprogrammierung selbst verbracht werden musste. Vor allem bei Soundkarten war die Situation eklatant: Nahezu jeder Soundchipsatz musste auf andere Weise angesprochen werden. An dieser Situation änderte sich auch mit den ersten Releases von Windows nichts, da dieses nicht mehr war, als ein teuer verkaufter DOS-Aufsatz, der DOS einen Grafikserver samt Window-Manager hinzufügte, während der gesamte alte Unterbau ohne jegliche Änderungen oder Erweiterungen übernommen wurde.

Mit dem Erscheinen von Windows 95 war allerdings klar, dass Microsoft nicht damit fortfahren können würde, sich auf so klägliche Problemlösungs-Versuche wie WinG zu beschränken – ein neuer zusätzlicher Kaufanreiz für Windows musste her. Das Resultat dieser Bemühungen war ein Programm namens DirectX, das in seinen ersten Vor-Versionen jedoch geradezu lächerlich war: Im Wesentlichen bestand es aus einigen Routinen, um Primitive auf den Bildschirm zu rendern. Microsoft erkannte relativ schnell, dass ein paar Kreise, Rechtecke und geplottete Pixel noch lange kein Multimedia-Betriebssystem machen und zog die neue „Multimedia“-Schnittstelle wieder vom Markt zurück. Um die Entwicklung eines derartigen Systems zu beschleunigen investierte der Software-Konzern eine größere Summe in die Anwerbung wahrer Heerscharen von Software-Entwicklern, die mit DirectX 1.0 eine durchaus brauchbare aber immer noch unausgereifte Schnittstelle hervorbrachten.

[4] Rechner der Firma Apple, später wurde vor allem die MacIntosh-Reihe bekannt (derzeit verkaufen sich der G4-Mac und der iMac am besten).

[5] Eine Ausnahme hierzu stellt das Mobile-Computing dar – ein Bereich in dem Apple traditionell hohe Marktanteile für sich verbuchen konnte.

Ein Meilenstein war Version 2.0 von DirectX, die eine neue Komponente mit dem Namen Direct3D beinhaltete. Direct3D war allerdings nicht von Microsoft selbst programmiert worden, sondern von einer kleinen Firma namens *RenderMorphics*, die ihr bis dahin als *Reality Lab* bekanntes Programm an den Giganten aus Redmond verkaufen konnte, woraufhin dieser sich bemühte, Reality Lab optimal in DirectX zu integrieren. Ein Teil dieser Integration bestand in der Umbenennung von Reality Lab auf Direct3D.

Ein weiterer großer Sprung fand von Version 2.0 auf 3.0 (bzw. 3.0a/b) statt: Abgesehen von diversen kleineren Detailverbesserungen, hielten die X-Files[6] (das DirectX-eigene Dateiformat), DirectSound 3D und ein virtueller mathematischer Co-Prozessor (um MMX-Unterstützung gewährleisten zu können) Einzug in DirectX. Mit Version 5.0, die nun auch Hardware-Support für 3D Sounds und Force Feedback-Unterstützung bot war bereits abzusehen, dass in Zukunft ein sehr großer Teil alle Windows-Multimedia-Anwendungen mit DirectX programmiert werden würde. Während in Version 6.0 vor allem neue 3D-Techniken zeitgerecht (Bump Mapping etc.) integriert wurden, verspätete sich die ursprünglich für DirectX 6.0 geplante DirectMusic API und wurde erst mit Version 6.1 ausgeliefert.

Zwei andere größere Neuheiten sparte Microsoft für DirectX Version 7.0 auf: Die Implementierung des *TnLHAL*[7], der die Programmierung von Hardware-beschleunigten Transform and Lightning-Anwendungen[8] vereinfachte, sowie die Auslieferung einer Visual Basic DirectX-API mit dem DirectX-SDK.

Mit Version 8.0 von DirectX wurden DirectSound und DirectMusic zu DirectX Audio kombiniert und DirectDraw und Direct3D zu DirectX Graphics verschmolzen. Abgesehen davon wurden in Version 8.0 programmierbare Vertex Shader eingeführt, die von den meisten Grafikkarten ab der GeForce 3-Chipsatz-Generation per Hardware unterstützt werden.

Die Neuerungen in DirectX 8.1 hielten sich hingegen stark in Grenzen; im Wesentlichen wurden einige Bugs ausgebügelt und Details im programmierbaren DirectX-Vertex Shader verbessert.

---

[6] Die so genannten X-Files sind das DirectX-eigene Dateiformat, das aufgrund seiner Endung (.x) zu diesem Namen gekommen ist.

[7] ***T**ransform a**n**d **L**ightning **H**ardware **A**pplication **L**ayer*

[8] Transform and Lightning ist ein Grafikchipsatz-Feature, das den Prozessor bei den namensgebenden Aufgaben entlastet. Der erste Consumer-Chipsatz, der dieses Feature unterstützte, war der GeForce 256 von NVidia.

DirectX-Komponenten

DirectX besteht derzeit (Version 8.1) aus folgenden Komponenten:

- ***DirectX Graphics***: Wie der Name schon vermuten lässt, ist dieser Teil von DirectX für das Rendern von Grafiken auf den Bildschirm zuständig. DirectX Graphics wurde als reine 3D-Schnittstelle mit rudimentären 2D-Rendering-Fähigkeiten designed. In früheren Versionen von DirectX (bis inklusive DirectX 7.0) verwendete man *Direct3D* für 3D-Anwendungen und *DirectDraw* für 2D-Anwendungen. Diese beiden Schnittstellen werden mittlerweile nicht mehr weiterentwickelt, sind aber nach wie vor in DirectX enthalten. DirectX Graphics ist in Architektur und Funktionsweise der freien Schnittstelle *OpenGL* ziemlich ähnlich.

- ***DirectX Audio***: Mit diesem Bestandteil von DirectX werden Sounds jeder Art geladen, modifiziert und wiedergegeben. Man sollte sich allerdings davor hüten, DirectX Audio hierauf zu reduzieren, da es auch dynamische Sounds generieren kann und über ausgezeichnete Synthesizing-Fähigkeiten verfügt. Auch DirectX Audio bestand ursprünglich aus zwei unterschiedlichen Komponenten: *DirectSound* und *DirectMusic*. Während DirectSound lediglich digitale Sounds wiedergab, konnten Dateitypen wie etwa MIDI mit DirectMusic abgespielt werden. DirectX Audio ist definitiv eine der derzeit mächtigsten Audio-APIs.

- ***DirectInput***: DirectInput ist die DirectX-Komponente, die verwendet wird, um Eingabe vom Benutzer zu empfangen. DirectInput unterstützt prinzipiell jedes Eingabegerät, das man an einen PC anschließen kann und für das DirectX-Treiber existieren. Davon abgesehen können mit DirectInput auch Force Feedback-Anwendungen programmiert werden.

- ***DirectPlay***: DirectPlay kommt immer dann zum Einsatz, wenn zwei (oder mehr) vernetzte Computer in einer DirectX-Anwendung miteinander in Verbindung treten sollen, was zum Beispiel in den Netzwerk-Mehrspieler-Modi diverser Spiele der Fall ist.

- ***DirectShow***: DirectShow ist die Streaming-Komponente von DirectX. Mit DirectShow können beispielsweise AVI-Videos wiedergegeben werden. Abgesehen davon ist es auch zum Capturing – also zur direkten Aufnahme – ausgezeichnet geeignet.

- ***DirectSetup***: Das ist die wohl einfachste aller DirectX-Komponenten, die nur dazu verwendet wird, andere Teile nachzuinstallieren. Ein häufiger Einsatzbereich von DirectSetup sind diverse Autorun-Menüs, mit denen aktuellere Versionen von DirectX installiert werden können.

Vor- und Nachteile von DirectX

Zu den unbestrittenen Vorteilen von DirectX gehören:

- **ausgezeichnete Performance**: DirectX-Anwendungen agieren hardwarenahe und sind daher meist sehr schnell.

- **optimale Hardware-Unterstützung**: Für praktisch jede neue Hardware-Komponente sind DirectX-Treiber verfügbar.

- **kostenlose Verfügbarkeit**: Die Programmierung von DirectX erfordert keine zusätzlichen Lizenzierungsgebühren.

- **hervorragende Dokumentation**: DirectX ist eine der bestdokumentierten Multimedia APIs überhaupt.

- **Abwärtskompatibilität**: Durch die Verwendung des Component Object Model (COM) sind Anwendungen, die ursprünglich für eine ältere DirectX-Version erstellt wurden, binärkompatibel zu neueren Releases von DirectX.

Die Nachteile von DirectX sind vor allem:

- **Closed-Source-Entwicklung (Microsoft-proprietär)**: Während Schnittstellen wie OpenGL relativ offen sind, fungiert DirectX als Black Box, über deren genaues Innenleben nur das DirectX-Team bei Microsoft Bescheid weiß.

- **DirectX-Anwendungen sind nicht portabel**: DirectX ist eine API-Sammlung, die Microsoft zu einem einzigen Grund erschaffen hat: Windows sollte ein attraktiver Kauf für Multimedia-Interessierte sein. Daher wird es wohl auch in nächster Zeit keine Portierung von DirectX auf andere Plattformen wie etwa Linux, Unix oder MacOS geben (Emulatoren wie etwa *WineX* ausgenommen).

- **teils schlechte Architektur**: Wenngleich die neuen Interfaces manch vergangenen Fehler ausbügeln, ist stellenweise immer noch deutlich spürbar, dass DirectX

ursprünglich dafür gemacht war, hauptsächlich unter C mit Hilfe von COM programmiert zu werden. Daher sind einige Schnittstellen – zumindest was ihr Design bzw. ihre Implementierung anbelangt – schlichtweg absolut katastrophal.

Kapitel 3 DirectDraw

Grafik-Rendering unter Windows

Eine Win32-Anwendung hat im Wesentlichen drei Möglichkeiten, Grafiken zu rendern: Zum einen kann die Windows-native Rendering-Schnittstelle *GDI*[9] verwendet werden, zum anderen kann auf externe Schnittstellen (wie DirectX oder *OpenGL*) zurückgegriffen werden, oder schlichtweg eine direkte Manipulation des VRAM durchgeführt werden. Es gibt mehrere Gründe, aus denen in den meisten Fällen bei Fullscreen-Multimedia-Applikationen (wie etwa Spielen) externe Grafik-Schnittstellen eingesetzt werden: Die Verwendung des GDIs führt zwar zu Programmen, die auf jedem Windows-Computer unabhängig von der Hardware-Ausstattung lauffähig sind und mit stabilen Rendering-Leistungen aufwarten können, aber da das GDI nicht für High-Performance-Anwendungen designed wurde (und das sind die meisten Multimedia-Anwendungen), weist es extrem schlechte Werte auf, was die Geschwindigkeit der Zeichenoperationen betrifft. Zu den größten Problemen, mit denen die oben erwähnten direkten Grafikspeichermanipulationen zu kämpfen haben, zählen die vergleichsweise unelegante Programmierung (meist lowest-level-C und/oder Assembly), die daraus resultierende schlechte Wartbarkeit der Programme und die Frage der Kompatibilität. Der Grund, aus dem dennoch lange Zeit (fast) ausschließlich diese Methode der Grafikprogrammierung gewählt wurde, ist der früher vorherrschende Mangel an Alternativen, der jedoch Hacker wie etwa John Carmack schon damals nicht daran hinderte, technisch brillante Produkte auf den Markt zu bringen. Nachdem mittlerweile bereits technisch durchwegs hochwertige Alternativen wie DirectDraw und OpenGL existieren, deren Kompatibilität zu unterschiedlichen Systemen nahe am Idealmaß ist (im Falle von OpenGL ist auch die höchstmögliche Portabilität

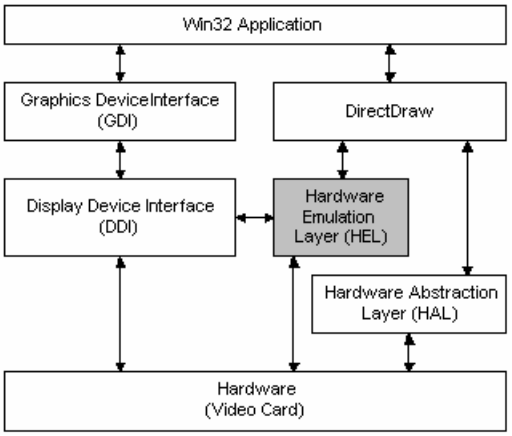


Abb. 3.1: Das Zusammenspiel einer DirectX-Anwendung mit dem Windows Rendering-Subsystem.

[9] *Graphics Device Interface*

gewährleistet) und die, auch die Performance betreffend, in 99,9 Prozent aller Fälle an ihren älteren oder vermeintlich stabileren Kollegen vorbeiziehen, wählen die meisten Entwickler derartige frei verfügbare Lösungen beziehungsweise darauf basierende Engines. Im Falle von DirectX hilft Abb. 3.1, die Funktionsweise des Windows Rendering-Systems besser zu verstehen. Das Windows-interne Rendering – beispielsweise die Darstellung von Schriftarten oder Ressourcen wie Bitmaps und Vektorgrafiken – wird vom GDI übernommen, während DirectDraw zwei verschiedene Subsysteme verwendet: den *HAL*[10] und dessen Software-implementiertes Pendant, den *HEL*[11]. Die Funktionsweise dieser zwei Komponenten lässt sich wohl am besten folgendermaßen erklären: Die Grafikkarte des Zielrechners kann – unabhängig davon, ob eine Operation per Hardware unterstützt wird oder nicht – verwendet werden. Bei der Erstellung des primären DirectDraw-Objekts stellt DirectDraw fest, über welche Features der Grafikchipsatz verfügt[12] und weist jeder Funktion eine Adresse aus der *vtbl*[13] zu. Sofern sie von Grafikchipsatz und Treiber unterstützt wird, übernimmt der HAL das Kommando und die DirectDraw-Funktionen werden direkt an den Treiber (und somit die Grafikhardware) weitergeleitet. Ist das nicht der Fall muss eine Software-implementierte Ersatzfunktion aufgerufen werden, die vom HEL bereitgestellt wird, der nicht direkt unterstütze Funktionen emuliert, und Aufgaben übernimmt, bei denen es vonnöten ist, das GDI zu bemühen, wie etwa Font-Rendering unter Zuhilfenahme von Windows Gerätekontexten. (Trotzdem sollte man es tunlichst vermeiden, zur Laufzeit allzu exotische Funktionen aufzurufen, da die Software-Implementierung von Grafikoperationen fast immer erheblich langsamer als die Hardware-Implementierung ist.)

Der Grund, aus dem ich persönlich bei der Windows-Anwendungsentwicklung DirectDraw allen anderen Schnittstellen vorziehe, ist, abgesehen von der hohen Treiberverfügbarkeit, vor allem die Tatsache, dass DirectDraw – im Gegensatz zu OpenGL etwa – als reines 2D-System entwickelt wurde, man jedoch bei Bedarf auch das *Direct3D/ DirectX Graphics*-API verwenden kann. Ebenfalls sollte man bedenken, dass DirectX mit *DirectSound/ DirectX Audio* mit einer hervorragenden Soundschnittstelle aufwarten kann, deren Programmierung sich stark an die der restlichen DirectX-Komponenten anlehnt.

[10] *Hardware Abstraction Layer*

[11] *Hardware Emulation Layer*

[12] Kann manuell ermittelt werden, indem man in der ***Systemsteuerung*** unter ***DirectX*** den Button ***Caps-Bits*** betätigt.

13 virtual function table

Die Architektur von DirectDraw

Um DirectDraw verwenden zu können, muss man sich zunächst einen Überblick über die verwendeten Datentypen verschaffen. Praktische Bedeutung haben zurzeit die folgenden:

- Das ***DirectDraw-Objekt*** ist der Kern jeder DirectDraw-Applikation. DirectDraw kann nicht verwendet werden, wenn kein DirectDraw-Objekt zur Verfügung steht. Das aktuelle Interface ist `IDirectDraw7`.

- ***DirectDraw Surfaces*** (DirectDraw-Oberflächen) sind Klassen, die Bildinformationen, Metainformationen zum Bild (Pixelformat, Größe, etc.) sowie diverse Schnittstellen-Methoden für Manipulation und Zugriff auf vorher genannte Informationen beinhalten. Derzeit aktuell ist das Interface `IDirectDrawSurface7`.

- ***DirectDraw Clipper*** werden verwendet, um Anwendungen davon abzuhalten, zu versuchen, in Bildschirmbereiche zu rendern, auf die sie eigentlich keinen Zugriff mehr haben sollten (z.B. Pixel 1000/200, obwohl die Auflösung nur 800x600 beträgt). Die Schnittstelle, die Zugriff auf die gesamte Funktionalität eines Clippers bietet, heißt `IDirectDrawClipper`.

Abgesehen von diesen Schnittstellen, wurden vor allem früher, als die Verwendung diverser 8-bit-Modi üblich war,

- ***DirectDraw Palettes*** (DirectDraw-Paletten) verwendet, die man brauchte, um Farben indizieren zu können. Mit Paletten konnte man beispielsweise vereinbaren, dass die Farbe RGB(x, y, z) mit dem Wert n angesprochen werden soll, wobei jede Variable ein Byte groß ist, was letztlich bedeutet, dass in einer Palette exakt 2^8 (=256) verschiedene 24-bit-Farben gespeichert werden können (3*8=24). Nachdem diese Fachbereichsarbeit sich jedoch in erster Linie mit High– und True Color-Modi auseinander setzt, soll auf diesen Objekt-Typus (Interface: `IDirectDrawPalette`) nicht näher eingegangen werden.

DirectDraw initialisieren

Um eine DirectDraw-Anwendung zu programmieren, müssen dem Projekt die Header-Datei `<ddraw.h>`[14] (zumindest per `#include`, besser aber auch zusätzlich über die IDE) sowie die Bibliotheksdateien `<ddraw.lib>` und `<dxguid.lib>`[15] hinzugefügt werden.

Ist das getan, kann das primäre DirectDraw-Objekt mit der Funktion `DirectDrawCreate()` erstellt werden. Hier ist der Funktionsprototyp:

`DirectDrawCreate()`

```
HRESULT WINAPI DirectDrawCreate(GUID FAR *lpGUID, // GUID des Treibers
                    LPDIRECTDRAW FAR *lplpDD, // wird DD-Objekt erhalten
                    IUnknown FAR *pUnkOuter ); // IMMER auf Null setzen!
```

Funktions-Argumente

- **`lpGUID`**: Das ist die GUID des zu verwendenden Grafik-Treibers und wird am besten Null gesetzt, was bewirkt, dass der Default-Treiber verwendet wird.

- **`lplpDD`**: Hier muss eine gültige Variable des Typs `LPDIRECTDRAW*` übergeben werden, in die `DirectDrawCreate()` dann das DirectDraw-Objekt speichert.

- **`pUnkOuter`**: Dieser Parameter wurde von Microsoft primär zu Debugzwecken verwendet, weswegen man immer Null übergeben sollte, da die Funktion andernfalls einen Fehler zurückgibt.

Noch ein Wort zum Rückgabewert dieser Funktion: Variablen vom Typ `HRESULT`, können mit den bool-Makros `FAILED()` und `SUCCEEDED()` interpretiert werden. Alternativ dazu kann auch auf Werte ungleich `DD_OK` getestet werden, was jedoch aufgrund teils sehr detaillierter `return`s nicht immer anzuraten ist. Am günstigsten dürfte es sein, schlichtweg die beiden erwähnten Makros zu verwenden.

Das einzige Problem, das diese Funktion mit sich bringt, ist, dass `lplpDD` nach dem Aufruf lediglich ein `LPDIRECTDRAW`-Objekt enthält, das allerdings keineswegs der neueste Stand der Technik ist: Derzeit ist das Interface `LPDIRECTDRAW7` aktuell

[14] Im Verzeichnis `<DXSDK>\include\` zu finden wobei `<DXSDK>` der Installationspfad des DirectX-SDK ist.

[15] Beide im Verzeichnis `<DXSDK>\lib\` zu finden wobei `<DXSDK>` der Installationspfad des DirectX-SDK ist.

(Einführung: DirectX 7.0). Dieser Missstand kann aber leicht behoben werden: Jedes DirectDraw-Objekt besitzt eine Methode `QueryInterface()`, mit der man die jeweils aktuelle Schnittstellen-Version anfordern kann[16].

`QueryInterface()`

```
HRESULT IUnknown::QueryInterface(REFIID riid, // Referenz-ID d. Interfaces
                                 LPVOID* obp);// Adresse des Zielzeigers
```

Funktions-Argumente

- **`riid`**: Die Referenz-ID (*reference identifier*) der angeforderten Schnittstelle, im Fall von `IDirectDraw7` ist sie `IID_IDirectDraw7`.

- **`obp`**: Hier wird die (zu `LPVOID*` konvertierte) Adresse des Zielzeigers übergeben, der das neue Interface entgegennehmen soll.

Diese Funktion hat auf den ersten Blick eine etwas seltsame Syntax, was vor allem daran liegt, dass sie eigentlich eine Methode von `IUnknown` ist, dem Interface, von dem sich jedes COM-Objekt ableitet. Die Verwendung von `LPVOID*` ist hier ebenfalls nur schwer umgänglich, da hier - einmal mehr - größtmögliche Flexibilität erforderlich ist.

Hat man ein aktuelles `IDirectDraw`-Interface (hier: `IDirectDraw7`), so kann man das **alte** DirectDraw-Objekt freigeben, was mit Hilfe der Methode `IUnknown::Release()` geschieht. Nachdem diese Funktion aufgerufen wurde, sollte man sicherheitshalber noch das alte `LPDIRECTDRAW`-Objekt auf Null setzen.

Mittlerweile sollte ein aktuelles primäres DirectDraw-Objekt vorhanden sein, womit alle Voraussetzungen erfüllt sind, die für den nächsten Schritt zu einer Fullscreen-DirectDraw-Anwendung nötig sind: Das Einstellen des Cooperative Levels.

`SetCooperativeLevel()`

Damit ein DirectDraw-Programm korrekt läuft, muss man den *Cooperative Level* der Anwendung festlegen, d.h. dem Betriebssystem mitteilen, wie die Ressourcenbelegung- bzw. Aufteilung vonstatten gehen soll. Dies geschieht mit `SetCooperativeLevel()`.

16 Alternativ dazu kann man auch die Funktion `DirectDrawCreateEx()` verwenden, mit der man, ohne den Umweg über `QueryInterface()` zu gehen, ein aktuelles `LPDIRECTDRAW` erstellen kann, allerdings funktioniert `DirectDrawCreateEx()` unter Windows 95 / Windows NT (unter Version 5.0) nicht immer wie gewünscht.

```
HRESULT IDirectDraw7::SetCooperativeLevel(HWND hWnd,    // Anwendungs-hWnd
                                          DWORD dwFlags); // Steuerungsflags
```

Funktions-Argumente

- **`hWnd`**: Das ist der Handle auf das Programmfenster der Anwendung.

- **`dwFlags`**: Dieses Argument nimmt sämtliche Steuerungsflags entgegen. Wenn mehrere Flags übergeben werden sollen, müssen sie mit einer bitweisen ODER-Verknüpfung (`|`) verbunden werden. Für Fullscreen-Applikationen die Flags `DDSCL_EXCLUSIVE` (legt fest, dass keine andere Anwendung auf den Bildschirm rendern darf), `DDSCL_FULLSCREEN` (bestimmt, dass die Anwendung im Fullscreen-Modus laufen soll) und `DDSCL_ALLOWREBOOT` ([Ctrl]+[Alt]+[Del] erlauben) setzen!

Um danach die Auflösung des DirectDraw-Programms festzulegen, verwendet man die Funktion `SetDisplayMode()`:

`SetDisplayMode()`

```
HRESULT SetDisplayMode(DWORD dwWidth,    // horizontale Auflösung
                       DWORD dwHeight,      // vertikale Auflösung
                       DWORD dwBPP,         // Farbtiefe
                       DWORD dwRefreshRate, // Bildwiederholrate
                       DWORD dwFlags);      // zusätzliche Steuerungsflags
```

Funktions-Argumente

- **`dwWidth`**: Die horizontale Auflösung (Angabe in px).

- **`dwHeight`**: Die vertikale Auflösung (Angabe in px)

- **`dwBPP`**: **b**its **p**er **p**ixel, die gewünschte Farbtiefe.

- **`dwRefreshRate`**: Die Bildwiederholrate, mit der die Anwendung laufen soll. Null übergeben, damit die Desktop-Refresh-Rate übernommen wird!

- **`dwFlags`**: Zusätzliche Kontrollflags, die nur selten benötigt werden. Null setzen!

Beim Gebrauch dieser Funktion ist auf drei Dinge besonders zu achten: Was das Einstellen der Auflösung angeht, so sollte man sich an die am weitesten verbreiteten Bildschirmmodi halten, um die Nerven (und die Hardware) des Anwenders zu schonen, bei der Bildwiderholrate einfach Null übergeben, womit vermieden wird, dass DirectDraw versucht, Bildwiederholraten einzustellen, die der Monitor gar nicht beherrscht und **NIE** 24 bit Farbtiefe wählen! Viele Grafikkarten (so auch meine) unterstützen den 24-bit-

Modus nicht direkt sondern nur auf dem Umweg über 32 bit (8 dummy-bits, 8 bits rot, 8 bits grün, 8 bits blau). Durch Übergabe von Null bei der Bildwiederholrate sollte theoretisch auch Anwendern mit korrekt konfigurierten Systemen erspart werden, dass sie beispielsweise auf einem 19-Zoll-Bildschirm bei einer Auflösung von 640x480 Bildpunkten mit schaurigen 60-Hertz-Refreshes gequält werden.

Um einen Überblick über die Funktionen, die verwendet werden, um DirectDraw lauffähig zu machen, zu bieten, habe ich eine kleine Demo geschrieben, die DirectDraw im Vollbild-Modus mit der Auflösung 800x600x32 startet, aber sonst nichts tut, außer zu warten, dass der Benutzer [Esc] betätigt, was die Anwendung beendet. Ich habe dafür die oben erwähnte *GameCon* benutzt und ihr einige kleinere Modifikationen zukommen lassen, so dass der Datei-Anfang jetzt aussieht wie folgt[17]:

```
// GameCon.cpp: A simple game console.

#define WIN32_LEAN_AND_MEAN

// INCLUDES
#include <windows.h>
#include <windowsx.h>
#include <string>
#include <ddraw.h> // NEU!!!

using std::string;

// GLOBAL CONSTANTS
const string WinClassName("WinClass01");
const DWORD ScreenWidth = 800; // NEU!!!
const DWORD ScreenHeight = 600; // NEU!!!
const DWORD ScreenBPP = 32; // NEU!!!

// GLOBAL VARIABLES
HWND          g_hWnd = 0;
HINSTANCE     g_hInstance = 0;
LPDIRECTDRAW7 lpDD; // NEU!!!
// snap…
```

Weiters habe ich dem Projekt über die IDE die Dateien `<ddraw.lib>`, `<dxguid.lib>` und `<ddraw.h>` hinzugefügt und die Funktion `GameInit()` folgendermaßen umgeschrieben:

```
// …snip
bool GameInit(void)
{
      LPDIRECTDRAW lpDDTemp;

      if (FAILED(DirectDrawCreate(0, &lpDDTemp, 0)))
```

[17] Um das Lesen der Listings zu erleichtern, habe ich sämtliche Deklarationen neuer Variablen und andere nicht ganz offensichtliche Änderungen mit dem Kommentar `// NEU!!!` versehen.

```
            // error at DirectDrawCreate()
            return false;

      if (FAILED(lpDDTemp->QueryInterface(IID_IDirectDraw7,
                              (LPVOID*)&lpDD)))
            // error at QueryInterface()
            return false;

      lpDDTemp->Release();
      lpDDTemp = 0;

      if (FAILED(lpDD->SetCooperativeLevel(ghWnd, DDSCL_EXCLUSIVE |
                                    DDSCL_FULLSCREEN |
                                    DDSCL_ALLOWREBOOT)))
            // error at SetCooperativeLevel()
            return false;

      if (FAILED(lpDD->SetDisplayMode(ScreenWidth, ScreenHeight,
                              ScreenBPP, 0, 0)))
            // error at SetDisplayMode()
            return false;

      ShowCursor(false); // hide cursor

      return true;
}
// snap…
```

Nachdem ich Ressourcen angefordert habe, muss ich auch mein `GameShutdown()` verändern, damit keine Speicherlecks in meinem Programm entstehen:

```
// …snip
void GameShutdown(void)
{
      if (lpDD)
      {
            lpDD->Release();
            lpDD = 0;
      }
}
// snap…
```

Man sollte beachten, dass man, so mehrere DirectX-Objekte vorhanden sind, diese in umgekehrter Erstellungsreihenfolge wieder freigeben muss. Eigentlich ist es nicht nötig, `lpDD` explizit gleich Null zu setzen, aber da `Release()` nicht zwangsläufig den *Reference Count* auf Null setzt, ist es klüger, das vorsichtshalber zu tun, da so keine Fehler beim Zugriff auf Zeiger auf bereits deallokierte Objekte passieren können.

Verwendung von DirectDraw Surfaces

Ein *DirectDraw Surface* (DirectDraw-Oberfläche) ist nichts Anderes, als eine Klasse, in der Bildinformationen abgelegt werden können, und die darüber hinaus über zahlreiche Methoden verfügt, mit denen man auf den Bildspeicher zugreifen kann, etwa um diesen auszugeben oder zu manipulieren etc. Der Typ einer solchen DirectDraw-Oberfläche ist momentan `IDirectDrawSurface7` (oder auch `LPDIRECTDRAWSURFACE7`).

Üblicherweise verwendet man in 2D-Anwendungen auch DirectDraw Surfaces, deren Aufgabe darin besteht, die am Bildschirm dargestellten Inhalte zwischenzuspeichern, weswegen man dieses Konzept gemeinhin als *buffering* bezeichnet.

Beim ***single buffering*** wird praktischerweise nur **ein** surface mit der Bildschirmdarstellung verbunden, und man kann Grafiken auf ebendiese Oberfläche rendern, woraufhin diese unverzüglich am Bildschirm angezeigt werden. Der Nachteil an dieser Technik ist, dass man Manipulationen des Bildschirminhaltes häufig als unangenehmes Flackern wahrnimmt, und dass Animationssequenzen unter Umständen zu stocken beginnen können, weil das System es nicht immer schafft, die Bildinformationen rechtzeitig zur Verfügung zu stellen. Infolgedessen wurde das *double buffering* erdacht, das diese Nachteile gegen andere austauscht.

Die Idee des ***double buffering***s ist es, bei Rendering-Operationen nicht direkt die Daten, die gerade am Bildschirm sichtbar sind, zu manipulieren, sondern Daten zunächst auf ein *offscreen surface*[18] zu rendern und, wenn die Manipulationen an diesem abgeschlossen sind, dieses auf den Bildschirm zu kopieren, während die andere Oberfläche bearbeitet werden kann. Der Name *double buffering* kommt daher, dass zwei Puffer verwendet werden, von denen jeweils nur einer sichtbar ist. Meistens wird einer dieser Puffer als primärer und einer als sekundärer Puffer bezeichnet, wobei der primäre die Oberfläche ist, die gerade angezeigt wird, und der sekundäre nie sichtbar ist.

Der Nachteil dieser Technik liegt darin, dass die Größe des benutzten Speichers de facto mit der Anzahl der Zwischenspeicher multipliziert wird, da alle Informationen redundant gespeichert werden.

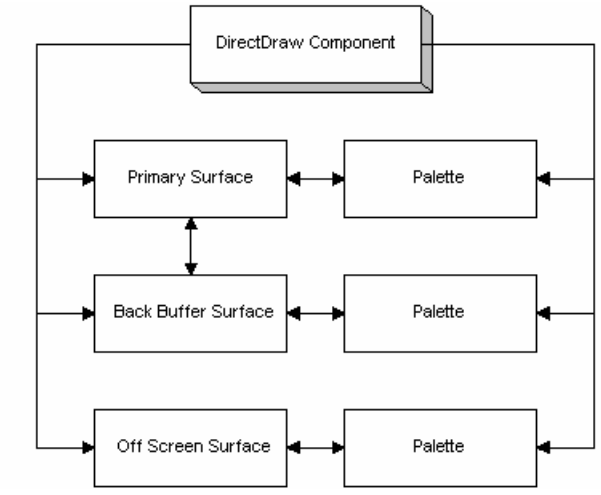


Abb. 3.2: Eine einfache DirectDraw-Anwendung mit surfaces (indizierter Farbmodus).

[18] Oberfläche, die einen Bildschirm voller Bildinformationen beinhaltet aber gerade nicht angezeigt wird.

Davon abgesehen gibt es natürlich auch das ***triple buffering*** bzw. so genannte ***flipping chains***[19], die im Prinzip aber genauso funktionieren, wie die oben erklärten Techniken, mit der Ausnahme, dass drei – oder mehr – unsichtbare back buffers verwendet werden. Die Benutzung von mehr als zwei Puffern bei vergleichsweise simplen 2D-Anwendungen ist allerdings völlig übertrieben, da sie keine weiteren Performance-Vorteile mehr bringen, sondern lediglich eine sinnlose zusätzliche Belastung für den Bus darstellen würde.

Um ein DirectDraw Surface zu erstellen, wird zurzeit die Methode `IDirectDraw7::CreateSurface()` verwendet, die im Folgenden kurz beschrieben ist:

`CreateSurface()`

```
HRESULT CreateSurface(LPDDSURFACEDESC2 lpDDSurfaceDesc2, // descriptor
                      LPDIRECTDRAWSURFACE4 FAR *lplpDDSurface, // Ziel-S.
                      IUnknown FAR *pUnkOuter); // erweiterter COM-Parameter
```

Funktionsargumente

- **`lpDDSurfaceDesc2`**: S*urface descriptor* der Oberfläche. Dies ist eine Struktur vom Typ `LPDDSURFACEDESC2`, die dazu verwendet wird, diverse Angaben über die gewünschten Eigenschaften der Oberfläche zu machen und auf die im Folgenden genauer eingegangen wird.

- **`lplpDDSurface`**: Das ist eine Variable des Typs `LPDIRECTDRAWSURFACE4*`, in die DirectDraw die neu erstellte Oberfläche speichert.

- **`pUnkOuter`**: *(siehe oben)* Auch hier wird einfach Null übergeben!

Damit die `CreateSurface()`-Argumente etwas verständlicher werden, möchte ich im nächsten Abschnitt die Untiefen eines `DDSURFACEDESC2` näher beleuchten:

`DDSURFACEDESC2`

Ein *surface descriptor* ist eine Struktur, die diverse Beschreibungen beinhaltet, durch die die Eigenschaften der zu erstellenden Oberfläche festgelegt werden. Das Ausfüllen solcher

Strukturen artet – wenn man die Grundlagen verstanden hat – meistens relativ bald in zwar lästige, aber unkomplizierte Tipparbeit aus.

Um den Umfang dieser Beschreibung wenigstens ein klein wenig zu reduzieren, habe ich beschlossen, nur das zu erwähnen, was im Alltag auch tatsächlich ständig Verwendung findet. Damit eine Orientierung etwas leichter fällt, habe ich diese relevanten Parameter **fett** formatiert. Außerdem ist es an dieser Stelle wichtig, zu wissen, dass diese Struktur nach ihrer Erstellung nicht zwangsläufig clean, d.h. bereit, korrekt zu funktionieren, ist. Microsofts Struktur-Design legt dummerweise nicht fest, dass derartige `structs` nach ihrer Erstellung mit Nullwerten gefüllt werden, weswegen einige MS-Strukturen vor der Verwendung mit einer Funktion wie etwa `ZeroMemory()`[20] geräumt werden sollten, damit es nicht zu unerklärlichen Abstürzen kommt.

```
typedef struct _DDSURFACEDESC2 {
    DWORD          dwSize; // Größe der Struktur
    DWORD          dwFlags; // Kontrollflags
    DWORD          dwHeight; // Höhe der Oberfläche in px
    DWORD          dwWidth; // Breite der Oberfläche in px
    union
    {
        LONG       lPitch; // (line pitch)
        DWORD      dwLinearSize; // (Größe d. Puffers – wenn komprimiert!)
    } DUMMYUNIONNAMEN(1);
    DWORD          dwBackBufferCount; // Anzahl der back buffers
    union
    {
        DWORD      dwMipMapCount; // (Anzahl der mipmap-levels)
        DWORD      dwRefreshRate; // (gewünschte Bildwiederhol-Rate)
    } DUMMYUNIONNAMEN(2);
    DWORD          dwAlphaBitDepth; // (alpha-Anteil im RGB-Tripel)
    DWORD          dwReserved; // (für DX-interne Zwecke reserviert)
    LPVOID         lpSurface; // (Zeiger auf den verwendeten Speicher)
    DDCOLORKEY     ddckCKDestOverlay; // (overlay destination color key)
    DDCOLORKEY     ddckCKDestBlt; // (destination color key für blits)
    DDCOLORKEY     ddckCKSrcOverlay; // (source color key für overlays)
    DDCOLORKEY     ddckCKSrcBlt; // source color key für blits
    DDPIXELFORMAT ddpfPixelFormat; // (Pixel-Format)
    DDSCAPS2       ddsCaps; // surface caps
    DWORD          dwTextureStage; // (texture-stage - nur für 3D!)
} DDSURFACEDESC2, FAR* LPDDSURFACEDESC2;
```

Struktur-Komponenten

- **`dwSize`**: Diese Komponente gibt die Größe der Struktur an und sollte `sizeof(DDSURFACEDESC2)` gesetzt werden.

- **`dwFlags`**: An dieser Stelle werden diverse Steuerungsflags übergeben, mit denen man angibt, welche Felder der Struktur auch tatsächlich gültige Werte enthalten

[19] Normalerweise verwendet man für buffering mit mehr als drei Zwischenspeichern den – allgemeineren – Begriff *flipping chain*.

[20] Das erste Argument für `ZeroMemory()` ist die Adresse, das zweite die Größe einer Struktur!

und daher von DirectDraw interpretiert werden sollen. Eine genaue Auflistung aller möglichen Werte würde den vorliegenden Text vermutlich zu sehr in die Länge ziehen, daher sei hier auf die *DirectX Programmer's Reference* verwiesen, in der diese selbstverständlich genau dokumentiert sind. Für die Erstellung vergleichsweise einfacher 2D-Anwendungen, dürfte das nächste Listing wohl ausreichen, da darin auch demonstriert wird, wie DirectDraw Surfaces verwendet werden können.

- **`dwHeight`**: Das ist die Höhe, die die Oberfläche letztlich haben soll. Sie wird, wie üblich, in px angegeben.
- **`dwWidth`**: Die Breite der Oberfläche (siehe oben).
- **`dwBackBufferCount`**: Bei Deskriptoren von primären Oberflächen wird hier die Anzahl der back buffers angegeben.
- **`ddckCKSrcBlt`**: Das ist der Color Key fürs *Source Color Keying*.
- **`ddsCaps`**: `ddsCaps` ist eine Struktur, die die *surface capabilities* festlegt.

DDCOLORKEY

Color Keying ist ein äußerst nützliches Konzept. Jeder, der schon einmal ein halbwegs aktuelles Computerspiel gespielt hat, oder auch nur einen etwas fortschrittlicheren Window-Manager wie etwa *Enlightenment* in Aktion gesehen hat, kennt die Auswirkungen dieser Technik. Vermutlich lässt auch sie sich am besten anhand eines konkreten Beispiels erklären:

Wenn ein Bild (in eine Oberfläche) geladen wird, um dieses anschließend auf eine (andere) Oberfläche zu blitten[21], so lädt der Computer es tatsächlich Pixel für Pixel. Die Übertragung erfolgt ebenso, was beim Blitten ohne Color Keying folgendermaßen aussieht:

[21] Das Wort *Blit* ist eine Abkürzung für „bit block transfer" und beschreibt eine besonders schnelle Möglichkeit des Datentransfers, bei dem eine große Datenmenge sequenziell gelesen und in einen anderen Speicherbereich übertragen wird. Aufgrund der Geschwindigkeit eines Blits ist er vor allem beim Rendern enorm nützlich.

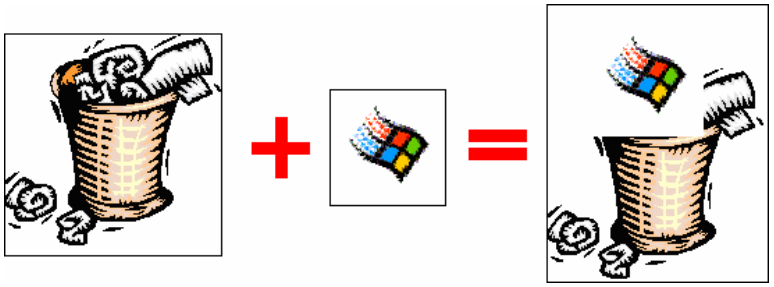


Abb. 3.3: Blitten ohne Color Keying.

Wie man unschwer erkennen kann, produziert diese Art des Blits verhältnismäßig hässliche Bilder, was primär daran liegt, dass das zweite Bild einfach Pixel für Pixel über das erste gelegt wird. Das hat bei Formaten die keine nichtindizierte Transparenz unterstützen[22] zur Folge, dass in der resultierenden Grafik Stellen, an denen eigentlich der Papierkorb zu sehen sein sollte, mit dem weißen Teil des Windows-Logos überdeckt sind, der aber nicht notwendigerweise weiß sein müsste, sondern eigentlich nur transparent sein sollte.

Um dieses Problem zu beheben, kann man sich des *Color Keying*s bedienen, wobei hier eindeutig das *Source Color Keying* am besten geeignet ist. Color Keying bedeutet, bestimmte Farben oder Farbräume als transparent zu definieren, so dass sie beim Blitten nicht mit übertragen werden. Derzeit unterstützt DirectDraw mehrere Arten des Color Keyings:

- *Destination Color Keying beim Blitten*
- *Source Color Keying beim Blitten*
- *Destination Color Keying für overlays*
- *Source Color Keying für overlays*

In der Praxis werden allerdings nahezu ausschließlich das Source Color Keying beim Blitten und das Destination Color Keying für overlays gebraucht, wobei jedoch die – von mir im Code dieses Kapitels demonstrierte – erste Methode bei weitem die wichtigste darstellt.

Source Color Keying beim Blitten heißt, dass eine Farbe (oder ein Farbraum) in der Quelloberfläche als transparent bestimmt wird und infolgedessen nicht auf die Zieloberfläche gerendert wird.

[22] PNG ausgenommen unterstützt dieses Feature fast kein brauchbares non-proprietäres Format...

Destination Color Keying für overlays hingegen wird verwendet, um eine Farbe in der Zieloberfläche zu bestimmen, die auf keinen Fall überschrieben werden darf, was beispielsweise dann getan werden kann, wenn eine Oberfläche von einer anderen partiell verdeckt werden soll. Der Grund dafür, dass diese Art des Color Keyings auch verhältnismäßig selten benutzt wird, ist, dass man es auch dadurch emulieren kann, dass man schlichtweg Source Color Keying verwendet und die Objekte der x-Koordinate nach absteigend sortiert und dann in exakt dieser Reihenfolge zeichnet, was normalerweise als overdraw bezeichnet wird und (teils verbunden mit *Hidden Surface Removing*) in diversen Bitmap-basierten 3D-Engines zum Einsatz kommt bzw. kam.

Verwendet man nun etwa beispielsweise Source Color Keying, sieht das Endresultat des Blits gleich eine Spur ansprechender aus:

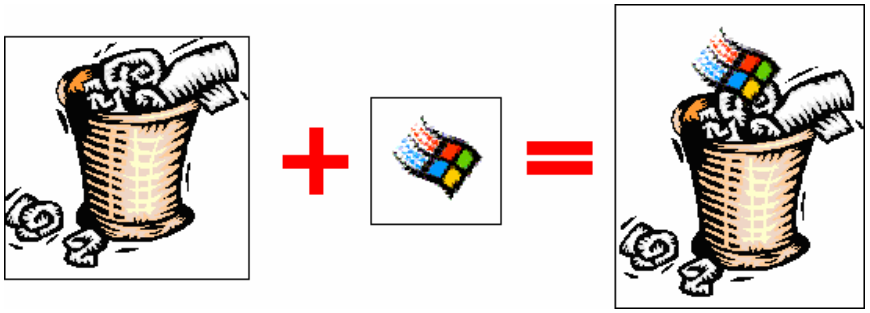


Abb. 3.4: Blitten mit Source Color Keying.

Nun endlich zur Struktur selbst, die bei weitem nicht so kompliziert wie die Beschreibung ihres Verwendungszwecks ist:

```
typedef struct _DDCOLORKEY{
       DWORD dwColorSpaceLowValue; // Anfang des Farbraumes
       DWORD dwColorSpaceHighValue; // Ende des Farbraumes
       }
DDCOLORKEY,FAR* LPDDCOLORKEY;
```

Struktur-Komponenten

- **`dwColorSpaceLowValue`**: Der Wert der hier eingesetzt wird entspricht dem Beginn des Farbraumes, der als Color Key verwendet werden soll.

- **`dwColorSpaceHighValue`**: Das ist das Ende des Farbraumes. Um nur eine einzige Farbe als Color Key zu verwenden kann man bei beiden Funktionsargumenten denselben Wert angeben.

DDSCAPS2

DDSCAPS steht für ***D****irect****D****raw* ***S****urface* ***Cap****abilities* – Eigenschaften einer DirectDraw-Oberfläche. Der Typ `DDSCAPS2` im Detail sieht folgendermaßen aus:

```
typedef struct _DDSCAPS2 {
    DWORD    dwCaps;   // surface capabilities
    DWORD    dwCaps2;  // noch mehr surface capabilities
    DWORD    dwCaps3;  // derzeit nicht in Verwendung
    DWORD    dwCaps4;  // derzeit nicht in Verwendung
} DDSCAPS2, FAR* LPDDSCAPS2;
```

In den meisten Fällen kommt man ganz gut zu Recht, wenn man sich auf die Verwendung von `dwCaps` beschränkt, daher werde ich diesmal nicht einzeln auf die Komponenten eingehen. (Für eine genaue Erklärung der einzelnen Flags, die übergeben werden können, sei hiermit auf die DirectX-Referenz verwiesen!)

Anfangs ist immer sehr schwer, Surfaces zu verstehen, ohne sie je praktisch angewandt zu haben, vor allem weil so viele Strukturen verwendet werden, die ihrerseits weitere Strukturen enthalten, die ausgefüllt werden müssen. Wenn man den Ablauf einmal kennt, ist es wirklich nur mehr pure Schreibarbeit. (Um diese zu umgehen besteht natürlich die Möglichkeit – mit relativ wenig Arbeitsaufwand – einen Wrapper zu coden!) Da mittlerweile bereits einige neue Konzepte erläutert wurden, folgt an dieser Stelle ein weiteres Programmbeispiel.

So sehen die Quellcode-Modifikationen aus:

Am Datei-Anfang:

```
// …snip
// GLOBAL VARIABLES
HWND                g_hWnd = 0;
HINSTANCE           g_hInstance = 0;
LPDIRECTDRAW7       lpDD;
LPDIRECTDRAWSURFACE7        lpDDSPrimary; // NEU!!!
LPDIRECTDRAWSURFACE7        lpDDSSecondary; // NEU!!!
// snap…
```

Die neue `GameInit()`:

```
bool GameInit(void)
{
      DDSURFACEDESC2      ddsd; // NEU!!!
      LPDIRECTDRAW lpDDTemp;

      if (FAILED(DirectDrawCreate(0, &lpDDTemp, 0)))
            // error at DirectDrawCreate()
            return false;
      if (FAILED(lpDDTemp->QueryInterface(IID_IDirectDraw7,
                                    (LPVOID*)&lpDD)))
            // error at QueryInterface()
            return false;
      lpDDTemp->Release();
      lpDDTemp = 0;

      if (FAILED(lpDD->SetCooperativeLevel(ghWnd, DDSCL_EXCLUSIVE |
                        DDSCL_FULLSCREEN | DDSCL_ALLOWREBOOT)))
            // error at SetCooperativeLevel()
            return false;
```

```
	if (FAILED(lpDD->SetDisplayMode(ScreenWidth, ScreenHeight,
				ScreenBPP, 0, 0)))
		// error at SetDisplayMode()
		return false;

	ShowCursor(false); // hide cursor

	ZeroMemory(&ddsd, sizeof(ddsd));

	ddsd.dwSize = sizeof(ddsd);
	ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
	ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP
					| DDSCAPS_COMPLEX;

	ddsd.dwBackBufferCount = 1;

	if (FAILED(lpDD->CreateSurface(&ddsd, &lpDDSPrimary, 0)))
		// error at CreateSurface()
		return false;

	// set up DDSCaps for querying for a back buffer
	ddsd.ddsCaps.dwCaps = DDSCAPS_BACKBUFFER;
	if (FAILED(lpDDSPrimary->GetAttachedSurface(&ddsd.ddsCaps,

	&lpDDSSecondary)))
		// error at GetAttachedSurface()
		return false;

	return true;
}
// snap…
```

Und natürlich muss auch `GameShutdown()` ergänzt werden:

```
// …snip
void GameShutdown(void)
{
	if (lpDDSSecondary)
	{
		lpDDSSecondary->Release();
		lpDDSSecondary = 0;
	}
	if (lpDDSPrimary)
	{
		lpDDSPrimary->Release();
		lpDDSPrimary = 0;
	}
	if (lpDD)
	{
		lpDD->Release();
		lpDD = 0;
	}
}
// snap…
```

Damit existiert auch schon ein lauffähiges Programm, das genau das Gleiche tut, wie die letzte Version, darüber hinaus auch einen primären und einen sekundären Puffer erstellt, und diese in `GameMain()` flippt. Aber da keiner der beiden Puffer zu irgendeinem Zeitpunkt auch tatsächlich Daten, enthält, die angezeigt werden könnten, werde ich im

nächsten Abschnitt noch etwas genauer darauf eingehen, wie man Grafik in einen Puffer rendern kann.

DirectDraw Offscreen Surfaces

Wie ich bereits früher erwähnte, sind Oberflächen nicht nur dazu geeignet, sie in einer flipping chain zu verwenden; man kann auch sogenannte *Offscreen Surfaces* erstellen. Diese Art von Grafikpuffern wird immer dann eingesetzt, wenn man ein Bild kapseln möchte – und wer möchte das nicht? Um es auf den Punkt zu bringen: Immer, wenn auf dem Bildschirm irgendeine Art von Grafik zu sehen ist, muss man ein Objekt verwenden, das zumindest konzeptionell einer Oberfläche gleichkommt, denn sonst würde jede einfache Schriftdarstellung zu einer nur sehr schwer zu bewältigenden Aufgabe: Das System müsste für jedes dargestellte Objekt genau berechnen, welche Bildpunkte auf dem Display eingefärbt werden sollen. Die Tatsache, dass so etwas wie offscreen buffering existiert, erleichtert derartige Vorhaben enorm; für jede Einheit, oder – um beim Beispiel von vorher zu bleiben – für jeden Buchstaben, wird ein eigener Speicherbereich allokiert und mit Informationen gefüllt, die sich dann ganz einfach wiedergeben lassen – Einheit für Einheit, Buchstabe für Buchstabe.

Die Erstellung einer Offscreen-Oberfläche funktioniert im Prinzip genauso, wie es vorher beim double buffering der Fall war, mit der Ausnahme, dass (im surface descriptor) bei `dwFlags` alle Flags angegeben werden müssen, die DirectDraw interpretieren soll und `ddsCaps.dwCaps` den Wert `DDSCAPS_OFFSCREENPLAIN` erhält. Danach kann man genauso verfahren wie bei der primären Oberfläche.

Somit wäre also ein Offscreen Surface erstellt, das man dann auch mit Daten füllen kann. Nur wie? Ich möchte nicht versuchen, auf diese Frage eine allgemeingültige Antwort zu geben. Tatsache ist, dass jedes Grafik-Dateiformat einen eigenen Loader benötigt. Das Programmieren eines ebensolchen ist in erster Linie Schreibarbeit, da im Wesentlichen zahlreiche Felder und Strukturen eingelesen und kopiert werden müssen und man danach einfach die Rohdaten in eine Oberfläche extrahiert. Es gbt zahlreiche frei verfügbare Klassen, die einem diese Arbeit sehr erleichtern können und abgesehen davon liefert Microsoft bereits in den DirectX SDK-Samples einen funktionsfähigen .bmp-Loader, der in den meisten Fällen zumindest für erste Gehversuch im Bereich der Grafikprogrammierung ausreichen dürfte. Außerdem werde ich für die Grafik-Engine, die

ich am Ende dieser Arbeit vorstellen werde, einen eigenen kleinen Loader programmieren[23].

Um grundlegende Techniken wie das Blitten zu verstehen, kann wohl nicht darauf verzichtet werden, Grafik auf Oberflächen zu rendern. Daher werde ich mir folgende Tatsache zunutzen machen: Jede DirectDraw-Oberfläche besitzt eine Methode `GetDC()` mit der ein Handle auf einen GDI-kompatiblen Gerätekontext angefordert werden kann, mit dem man auf ein `IDirectDrawSurface7` genauso zugreifen kann wie auf einen gewöhnlichen `HDC`.

`GetDC()`

```
HRESULT IDirectDrawSurface7::GetDC(HDC FAR *lphDC); //Adresse d. Ziel-HDCs
```

Funktions-Argumente

- **`lphDC`**: Das ist die Adresse des `HDC`s über den auf die Oberfläche zugegriffen können werden soll.

Das Einzige worauf bei dieser Funktion geachtet werden muss, ist, die Oberfläche wieder freizugeben, sobald man sämtliche Zeichenoperationen abgeschlossen hat und die Oberfläche auf den Bildschirm bringen möchte. Das geschieht mittels `ReleaseDC()`:

`ReleaseDC()`

```
HRESULT IDirectDrawSurface::ReleaseDC(HDC hDC); // freizugebender HDC
```

Funktions-Argumente

- **`hDC`**: Das ist der `HDC`, der freigegeben werden soll.

Nachdem mittlerweile auch die Möglichkeit besteht, auf Offscreen Surfaces zu rendern, wäre es natürlich interessant, zu wissen, wie man diese am Bildschirm anzeigen kann. Auch hierzu gibt es eine eigene Methode, die selbstverständlich auch verwendet werden kann, um zwischen zwei Offscreen Surfaces zu blitten. Hier ist der Prototyp:

[23] Genaue Spezifikationen unterschiedlicher Dateiformate (wie man sie als Programmierer zur Implementierung von Loadern benötigt) sind unter http://www.wotsit.org zu finden.

`Blt()`

Achtung: Diese Methode wird nicht von der Quelloberfläche, sondern von der Zieloberfläche aus aufgerufen!

```
HRESULT IDirectDrawSurface7::Blt(LPRECT lpDestRect, // Ziel-Rechteck
             LPDIRECTDRAWSURFACE7 lpDDSrcSurface, // Quelloberfläche
             LPRECT lpSrcRect, // Quellrechteck
             DWORD dwFlags, // Steuerungsflags
             LPDDBLTFX lpDDBltFx); // Adresse einer sfx-Struktur
```

Funktions-Argumente

- **`lpDestRect`**: Das ist der Zielbereich, in den die Oberfläche geblittet wird. Er wird als Zeiger auf ein Rechteck angegeben.

- **`lpDDSrcSurface`**: Das ist die Quelloberfläche, die übertragen werden soll.

- **`lpSrcRect`**: Das ist ein Zeiger auf eine `RECT`-Struktur die angibt, aus welchem Bereich der Quelloberfläche die Daten geholt werden sollen. Hier kann Null übergeben werden, um die gesamte Oberfläche zu blitten!

- **`dwFlags`**: Hier können diverse Steuerungsflags übergeben werden. Wichtig: Immer (auch) `DBLT_WAIT` übergeben. Dieser Wert gibt an, dass DirectDraw `DDERR_WASSTILLDRAWING`-Fehlermeldungen unterdrücken und stattdessen warten soll, bis der Blitter wieder korrekt funktioniert, was meistens gewünscht ist.

- **`lpDDBltFx`**: Das ist ein Zeiger auf eine Struktur, in der man diverse Angaben zu gewünschten Spezialeffekten beim Blitten machen kann. Die Verfügbarkeit derartiger Effekte ist allerdings stark hardwareabhängig, weshalb auf allzu exzessive Verwendung dieses Features verzichtet werden sollte.

Es gibt auch eine Funktion namens `IDirectDrawSurface7::BltFast()`, die etwas weniger Features enthält als `Blt()`, so kann sie zum Beispiel keine Oberflächen skalieren. Wenn man allerdings nichts Derartiges im Sinn hat, ist es klüger, `BltFast()` zu verwenden, da diese Methode eine etwas bessere Performance hat, als ihr etwas umfangreicheres Pendant. Die Parameter sehen bei beiden Versionen nahezu gleich aus. Vorsicht: Bei Verwendung von `BltFast()` kann keine *clip list* (siehe unten) gesetzt werden!

Nun ergibt sich aber ein neues Problem: Man kann auch verhältnismäßig leicht versuchen, in einen Bereich zu rendern, der eigentlich gar nicht mehr existiert. Wenn beispielsweise die Auflösung 640x480 Bildpunkte beträgt, ist nicht festgelegt, was passiert, wenn man als

Zielrechteck einen Bereich zwischen (650/490) und (700/540) wählt! Derart simple Fälle können leicht mit einem primitiven Algorithmus, der auf völlige Unsichtbarkeit testet und etwa so formuliert sein könnte, vermieden werden:

```
if (x1 > ScreenWidth || x2 <= 0 || y1 > ScreenHeight || y2 <= 0)
      // Objekt außerhalb des Sichtbereiches; nicht darstellen
      ;
else
      // Objekt sichtbar; zeichnen
      drawObject();
```

Natürlich wäre das viel zu einfach, als dass es hierfür Verwendung gäbe: Diese vier Zeilen Code brächten rein gar nichts, wenn ein Objekt etwa nur teilweise verdeckt ist. Selbstverständlich könnte man auf ähnliche Weise selbst einen vollständigen *Clipper*[24] implementieren, aber um der Realität ins Auge zu sehen: Das wäre enorm viel Arbeit und selbst mit massivem Assembly-Einsatz könnte die Performance wohl nicht überzeugen. Da in DirectDraw ein Clipping-Mechanismus enthalten ist, dessen Leistung äußerst gut ist, kann Gebrauch von ebendiesem gemacht werden. Um einen solchen Clipper zu verwenden, erstellt man zunächst ein Objekt vom Typ `LPDIRECTDRAWCLIPPER` und weist diesem dann mit der Funktion `CreateClipper()` (die übrigens eine Methode von `IDirectDraw7` ist) einen gültigen Clipper zu. Dann wird eine clip list gesetzt, die mehrere Clipping-Rechtecke[25] beinhaltet.

`CreateClipper()`

```
HRESULT IDirectDraw7::CreateClipper(DWORD dwFlags, // flags
      LPDIRECTDRAWCLIPPER FAR *lplpDDClipper, // Adresse d. Ziel-Clippers
      IUnknown FAR *pUnkOuter); // Null setzen!
```

Funktions-Argumente

- **`dwFlags`**: Steuerungsflags, die aber nicht verwendet werden und daher den Wert Null bekommen müssen!

- **`lplpDDClipper`**: Das ist die Adresse der `LPDIRECTDRAWCLIPPER`-Struktur, die den gültigen Clipper erhalten soll.

- **`pUnkOuter`**: Null setzen (*siehe oben*)!

Jetzt gibt es einen Clipper, mit dem DirectDraw eine clip list zugeordnet werden kann, und zwar mit folgender Methode:

[24] Mechanismus, der auf Sichtbarkeit von Oberflächen testet.
[25] Rechteckiges Areal, dessen Inhalt immer sichtbar sein soll.

SetClipList()

```
HRESULT IDirectDrawClipper::SetClipList(LPRGNDATA lpClipList,//region-data
                                        DWORD dwFlags); // Steuerungsflags
```

Funktions-Argumente

- **lpClipList**: Das ist die Adresse einer Struktur, die im Abschnitt RGNDATA genauer erklärt werden wird.

- **dwFlags**: Hier sollte man eines Tages einmal Steuerungsflags übergeben können, was aber bis heute nicht möglich ist; Null setzen!

RGNDATA

RGNDATA-Strukturen beinhalten genaue Beschreibungen zu Anzahl und Dimension verwendeter Clipping-Rechtecke. Ihre Architektur ist – gelinde gesagt – nicht unbedingt sonderlich fortschrittlich. Microsoft verwendet keinerlei echte dynamische Datenstrukturen, sondern hat das Rad stattdessen einmal mehr (schlecht) neu erfunden und emuliert elegante Speicherallokierung zur Laufzeit einfach mittels der vorliegenden Struktur, die einen Puffer von der Größe eines chars beinhaltet. Wenn die Struktur verwendet wird, greifen sämtliche Funktionen nur mehr auf den Puffer-Bereich zu, beziehen Informationen über den tatsächlich belegten Speicher aber aus dem Feld rdh::dwSize. Dass diese Art der Programmierung nicht unbedingt zu stabilen Anwendungen führt, ist wohl offensichtlich. Ihr Prototyp sieht so aus:

```
typedef struct _RGNDATA {  RGNDATAHEADER rdh; // Header
                           char Buffer[1]; // dummy char
                        } RGNDATA;
```

Struktur-Komponenten

Die einzige Komponente dieser Struktur, die direkt bearbeitet werden darf, ist rdh vom Typ RGNDATAHEADER, der sofort beschrieben wird.

RGNDATAHEADER

```
typedef struct _RGNDATAHEADER {
      DWORD dwSize; // Größe des Headers
      DWORD iType; // Typ der Regionsdaten
      DWORD nCount; // Anzahl der Rechtecke in Buffer[]
      DWORD nRgnSize; // wahre Größe von Buffer[]
      RECT rcBound; // Umrahmung aller Rechtecke
} RGNDATAHEADER;
```

Struktur-Komponenten

- **`dwSize`**: Das ist die Größe des Headers. Sie sollte `sizeof(RGNDATAHEADER)` gesetzt werden!
- **`iType`**: Das ist die Art der Regionsdaten. Hier muss `RDH_RECTANGLES` übergeben werden.
- **`nCount`**: Hier wird die Anzahl der Clipping Rechtecke, die in die clip list aufgenommen werden sollen, eingesetzt.
- **`nRgnSize`**: Das ist die wahre (=im Speicher belegte) Größe von `Buffer[]`. Sie beträgt `nCount*sizeof(RECT)`.
- **`rcBound`**: Das ist ein Rechteck, das alle Clipping-Rechtecke umschließt.

Nachdem nun ein funktionstüchtiger Clipper inklusive clip list erstellt wurde, kann er auch einer x-beliebigen Oberfläche zugeordnet werden, in den allermeisten Fällen dürfte das der sekundäre Bildpuffer sein. Da üblicherweise auf genau diesen Puffer auch gerendert wird, muss dieser irgendwann mit dem primären Puffer vertauscht werden, damit man auch tatsächlich ein Bild zu sehen bekommt. Dies geschieht mit der Funktion Flip:

`Flip()`

```
HRESULT IDirectDrawSurface7::Flip(
      LPDIRECTDRAWSURFACE4 lpDDSurfaceTargetOverride, // womit flippen?
      DWORD dwFlags); // Kontrollflags
```

Funktionsargumente

- **`lpDDSurfaceTargetOverride`**: Wenn der primäre Bildpuffer (von dem aus die Funktion aufgerufen wird) nicht mit dem dazugehörigen sekundären Puffer ausgetauscht werden soll, muss hier die Alternativ-Oberfläche spezifiziert werden. Da man in den meisten Fällen aber einfach den Inhalt des sekundären Puffers auf den Bildschirm bringen möchte, sollte man hier normalerweise Null übergeben.
- **`dwFlags`**: An dieser Stelle werden die obligaten Steuerungsflags übergeben. Normalerweise reicht es aus, hier `DDFLIP_WAIT` zu übergeben.

Hier noch ein kleines Beispiel, das die in diesem Abschnitt besprochenen Techniken in Aktion zeigt:

Der Dateianfang sieht jetzt so aus:

```
// DDSurf2GameCon.cpp: A simple game console (extended).

#define WIN32_LEAN_AND_MEAN

// INCLUDES
#include <windows.h>
#include <windowsx.h>
#include <string>
#include <ddraw.h>

using std::string;

// GLOBAL CONSTANTS
const string WinClassName("WinClass01");
const DWORD ScreenWidth = 800;
const DWORD ScreenHeight = 600;
const DWORD ScreenBPP = 32;
const string message("Hello world!"); // NEU!!!

// GLOBAL VARIABLES
HWND                    g_hWnd = 0;
HINSTANCE               g_hInstance = 0;
LPDIRECTDRAW7           lpDD;
LPDIRECTDRAWSURFACE7    lpDDSPrimary;
LPDIRECTDRAWSURFACE7    lpDDSSecondary;
LPDIRECTDRAWSURFACE7    surf01; // NEU!!!
LPDIRECTDRAWCLIPPER     lpDDC; // NEU!!!
// snap…
```

Die neue `GameInit()`:

```
bool GameInit(void)
{
      DDSURFACEDESC2      ddsd;
      LPRGNDATA           lpRgnData; // NEU!!!
      RECT                visible; // NEU!!!
      HDC                 xDC; // NEU!!!
      DDBLTFX             ddBltFx; // NEU!!!
      LPDIRECTDRAW        lpDDTemp;

      // random seed
      srand(GetTickCount());

      if (FAILED(DirectDrawCreate(0, &lpDDTemp, 0)))
            // error at DirectDrawCreate()
            return false;

      if (FAILED(lpDDTemp->QueryInterface(IID_IDirectDraw7,
                                    (LPVOID*)&lpDD)))
            // error at QueryInterface()
            return false;

      lpDDTemp->Release();
      lpDDTemp = 0;

      if (FAILED(lpDD->SetCooperativeLevel(g_hWnd, DDSCL_EXCLUSIVE
                        | DDSCL_FULLSCREEN | DDSCL_ALLOWREBOOT)))
            // error at SetCooperativeLevel()
            return false;

      if (FAILED(lpDD->SetDisplayMode(ScreenWidth, ScreenHeight,
                                    ScreenBPP, 0, 0)))
            // error at SetDisplayMode()
            return false;

      ShowCursor(false);
```

```
ZeroMemory(&ddsd, sizeof(ddsd));

ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP
                          | DDSCAPS_COMPLEX;

ddsd.dwBackBufferCount = 1;

if (FAILED(lpDD->CreateSurface(&ddsd, &lpDDSPrimary, 0)))
      // error at CreateSurface()
      return false;

// set up DDSCaps for querying for a back buffer
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.ddsCaps.dwCaps = DDSCAPS_BACKBUFFER;
if (FAILED(lpDDSPrimary->GetAttachedSurface(&ddsd.ddsCaps,
                                        &lpDDSSecondary)))
      // error at GetAttachedSurface()
      return false;

// create clipper
if (FAILED(lpDD->CreateClipper(0, &lpDDC, 0)))
      return false;

visible.left = 0;
visible.right = ScreenWidth;
visible.top = 0;
visible.bottom = ScreenHeight;

// size = sizeof(RGNDATAHEADER) + nRects * sizeof(RECT)
lpRgnData = reinterpret_cast<LPRGNDATA> (operator new(
                  sizeof(RGNDATAHEADER) + sizeof(RECT)));

// fill in header
lpRgnData->rdh.dwSize = sizeof(RGNDATAHEADER);
lpRgnData->rdh.iType = RDH_RECTANGLES;
lpRgnData->rdh.nCount = 1;
lpRgnData->rdh.nRgnSize = sizeof(RECT);
lpRgnData->rdh.rcBound.left = 0;
lpRgnData->rdh.rcBound.right = ScreenWidth;
lpRgnData->rdh.rcBound.top = 0;
lpRgnData->rdh.rcBound.bottom = ScreenHeight;

// copy rectangle into Buffer[]
memcpy(lpRgnData->Buffer, &visible, sizeof(RECT));

if (FAILED(lpDDC->SetClipList(lpRgnData, 0)))
{
      delete lpRgnData;
      return false;
}

if (FAILED(lpDDSSecondary->SetClipper(lpDDC)))
{
      delete lpRgnData;
      return false;
}

// new clipper was created
delete lpRgnData;

// create an offscreen surface
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH;
```

```
	ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
	ddsd.dwHeight = 50;
	ddsd.dwWidth = 200;

	if (FAILED(lpDD->CreateSurface(&ddsd, &surf01, 0)))
		return false;

	// fill surf01 & render some text to it
	ZeroMemory(&ddBltFx, sizeof(ddBltFx));
	ddBltFx.dwSize = sizeof(ddBltFx);
	ddBltFx.dwFillColor = RGB(0, 0, 0);
	surf01->Blt(0, 0, 0, DDBLT_COLORFILL | DDBLT_WAIT, &ddBltFx);
	if (FAILED(surf01->GetDC(&xDC)))
		return false;
	SetTextColor(xDC, RGB(40, 119, 215));
	SetBkColor(xDC, RGB(0, 0, 0));
	SetBkMode(xDC, TRANSPARENT);
	TextOut(xDC, 0, 0, message.c_str(), message.size());
	surf01->ReleaseDC(xDC);

	return true;
}
```

Und hier der Source-Code der neuen `GameMain()`:

```
bool GameMain(void)
{
	HRESULT ddrval;
	RECT dest; // NEU!!!
	dest.left = rand()%ScreenWidth;
	dest.right = dest.left + 200;
	dest.top = rand()%ScreenHeight;
	dest.bottom = dest.top + 50;

	if (KeyDown(VK_ESCAPE))
		return false;

	if (FAILED(lpDDSSecondary->Blt(&dest, surf01, 0, DDBLT_WAIT, 0)))
		return false;


	while (true)
	{
		// flip surfaces at next vertical refresh
		ddrval = lpDDSPrimary->Flip(0, DDFLIP_WAIT);
		if (ddrval == DD_OK)
			// everything ok: leave flipping-loop
			break;
		else if (ddrval == DDERR_SURFACELOST)
			// try to restore lost surface
			if(FAILED(lpDDSPrimary->Restore()))
				break;
		else
			// error occured: leave flipping-loop
			break;
	}

	return true;
}
```

Zu guter Letzt noch `GameShutdown()`:

```
void GameShutdown(void)
{
	if (surf01)
```

```
	{
		surf01->Release();
		surf01 = 0;
	}
	if (lpDDC)
	{
		lpDDC->Release();
		lpDDC = 0;
	}
	if (lpDDSSecondary)
	{
		lpDDSSecondary->Release();
		lpDDSSecondary = 0;
	}
	if (lpDDSPrimary)
	{
		lpDDSPrimary->Release();
		lpDDSPrimary = 0;
	}
	if (lpDD)
	{
		lpDD->Release();
		lpDD = 0;
	}
}
```

Natürlich würde kein (guter) Programmierer derartigen Spaghetti-Code verfassen, vor allem dann nicht, wenn er vorhätte, ein umfangreicheres Projekt zu erstellen, aber zu Lehrzwecken eignen sich diese Listings trotzdem gut.

Nachdem in diesem Kapitel über DirectDraw, den umfangreichsten klassischen Teil von DirectX, zahlreiche allgemeingültige Grundlagen sowohl theoretisch als auch praktisch eingehend demonstriert wurden, wird in den folgenden Ausführungen auf diese Aspekte nicht mehr so detailliert eingegangen, wie das in den bisherigen Abschnitten der Fall war.

Kapitel 4 DirectX Audio

Digitale Sounds Vs. Sythesizersounds

Traditionell wird in der Sound Engineering-Welt zwischen zwei verschiedenen Arten der Soundspeicherung unterschieden: Einerseits finden Synthesizer-basierende Sound-Reproduktionstechniken wie etwa MIDI (teils mit DLS[26]-Techniken) nach wie vor Verwendung, andererseits ist die digitale Soundspeicherung im Zeitalter der DVDs und CDs beliebter denn je.

Ein paar Worte zu den jeweiligen Vor- und Nachteilen dieser Techniken:

Zu den wohl größten Vorteilen eines Synthesizers zählt, dass man Computer verhältnismäßig leicht so programmieren kann, dass sie in Echtzeit skriptgesteuert neue Musik „komponieren", denn die Verwendung deskriptiver melodischer Sequenzen erleichtert dieses Vorhaben enorm. Davon abgesehen, verbrauchen Synthesizer-Sounds nur sehr wenig Speicherplatz, weil im Prinzip nur Beschreibungen verschiedener Töne übermittelt werden müssen, und die Sounds selbst modular verwendet werden können. Leider bedingt das aber auch, dass längere Melodien oft sehr künstlich (eben synthetisch) klingen, was vor allem darauf zurückzuführen ist, dass ein analoges Instrument nie genau gleiche Töne produziert. (Egal wie oft man es auch versucht, man wird nie denselben Ton zweimal ganz exakt wiedergeben können.) Für einen Computer stellt das kein Problem dar; er hätte im Gegensatz dazu eher Probleme, sollte er „fehlerbehaftete", ungenaue Geräusche machen. Dadurch hören sich viele Synthesizer-generierte Songs stark nach 80er-Jahre-Anachronismen an…

Der größte Vorteil der digitalisierten Tonaufzeichnung hingegen ist die absolute Authentizität. Geräusche werden nicht vom Computer erstellt, oder aus mehreren Teilen zusammengesetzt, sondern aufgenommen und vom Computer nahezu genauso wiedergegeben, wie sie in natura geklungen haben. Diese Methode ist auch die derzeit beste, wenn es um Effekte geht! Eine Explosion beispielsweise kann durch nichts so gut wiedergegeben werden, wie mit der Aufnahme einer ebensolchen… Der Nachteil dürfte ebenso klar ersichtlich sein: Der Speicherverbrauch ist enorm! Um ein alltägliches Beispiel anzuführen: Auf eine CD, die immerhin eine Speicherkapazität von etwa 650 MB hat, passen lediglich 74 Minuten Audio-Daten im Rohformat! (Bei einer Samplingfrequenz von

[26] ***D**own**l**oadable **S**ounds*

44,1 kHz bei 16 Bit Stereo, wie sie bei CD-As üblich ist.) Daher ist es natürlich verständlich, dass sich Synthesizer-Sounds in manchen Anwendungsbereichen nach wie vor größter Beliebtheit erfreuen.

Wie bereits früher erwähnt, unterschied auch DirectX in früheren Versionen strikt zwischen diesen beiden Kategorien und stellte ihnen unterschiedliche APIs zur Verfügung, was sich allerdings mit Version 8 änderte. So wie DirectDraw und Direct3D zu *DirectX Graphics* wurden, entschied Microsoft sich dazu, mit Version 8 DirectSound und DirectMusic in eine einzige hochintegrierte Schnittstelle überzuführen – DirectX Audio. Abgesehen davon lieferte MS mit dem SDK der Version 8 erstmals eine Reihe von äußerst praktischen Audio-Loadern mit, die man vor nicht allzu langer Zeit noch selbst schreiben musste!

Die Architektur von DirectX Audio

Die folgenden DirectX Audio-Datentypen werden benötigt, um Waves oder MIDI-Dateien wiederzugeben:

- Der zentrale Datentyp einer DirectX Audio-Anwendung ist wohl die ***DirectMusic Performance***. Eine Performance verwaltet sämtliche abgespielte Sounds einer Anwendung und stellt diverse Methoden bereit, mit denen grundlegende Verhaltensweisen von DirectX Audio gesteuert und analysiert werden können. Eine DirectMusic Performance ist im Prinzip das Audio-Pendant zum primären DirectDraw Objekt. Die derzeit aktuelle Version ist `IDirectMusicPerformance8`.

- Zu den wohl wichtigsten Typen gehört auch das ***DirectMusic Loader-Objekt***. Es stellt die Klasse dar, mit der man diverse Datentypen in Rohform konvertieren und in einen Zwischenspeicher laden kann. Aktuell ist momentan das Interface `IDirectMusicLoader8`.

- Davon abgesehen, benötigt man natürlich auch eine Struktur die als Zwischenspeicher fungiert und die oben angesprochenen Rohdaten aufnehmen kann. Diese Aufgabe übernimmt das ***DirectMusic Segment***, dessen Rolle sich sehr gut mit der einer DirectDraw-Oberfläche vergleichen lässt. Die neueste Schnittstelle hierfür ist `IDirectMusicSegment8`.

Selbstverständlich gibt es noch zahlreiche andere Interfaces, mit denen sich weit kompliziertere Aufgaben bewerkstelligen lassen, als das bloße Wiedergeben von Sounddateien, aber alleine die Aufzählung aller beteiligten Datentypen würde den Rahmen dieser Arbeit sprengen…

DirectX Audio-Programmierung

Wie schon bei DirectDraw, so muss man auch bei DirectX Audio penibel darauf achten, die passenden DirectX-Header- bzw. Bibliotheksdateien korrekt in das Programm einzubinden. Konkret wären das `<dmusici.h>`[27] sowie `<dxguid.lib>`[28].

Leider werden für die Erstellung der wichtigsten DirectX Audio-Komponenten von Microsoft keinerlei Hilfsfunktionen bereitgestellt, wie das bei DirectDraw beispielsweise mit `DirectDrawCreate()` der Fall war, weswegen man sich hierbei wohl oder übel auf die COM-Programmierung einlassen muss. Nachdem aber schnell ein paar Zeilen Code gefunden sind, die diese Aufgabe zuverlässig lösen, dürfte es wohl das Klügste sein, sich in dieser Hinsicht auf die – in der DirectX Programmer's Reference demonstrierte – einfachste Methode zu verlassen:[29]

```
CoInitialize(0);

CoCreateInstance(CLSID_DirectMusicLoader, 0, CLSCTX_INPROC,
             IID_IDirectMusicLoader8, (void**)&g_pLoader);

CoCreateInstance(CLSID_DirectMusicPerformance, 0, CLSCTX_INPROC,
             IID_IDirectMusicPerformance8,
      (void**)&g_pPerformance);
```

Für DirectX Audio 8 funktioniert der Aufruf dieser drei Funktionen ganz ausgezeichnet, so zuvor ein `IDirectMusicLoader8*` und eine `IDirectMusicPerformance8*` erstellt (und mit Null initialisiert) wurden.

Natürlich müssen danach noch die Performance und der Synthesizer initialisiert werden, was mit der Methode `IDirectMusicPerformance8::InitAudio()` geschieht:

`InitAudio()`

```
HRESULT IDirectMusicPerformance8::InitAudio(
             IDirectMusic** ppDirectMusic, // Ziel-Zeiger für DM
             IDirectSound** ppDirectSound, // Ziel-Zeiger für DS
```

[27] Im Verzeichnis `include` des DirectX-SDK.
[28] Im Verzeichnis `lib` des DirectX-SDK.
[29] Vgl.: *DirectX Programmer's Reference*

```
    HWND hWnd, // Window-Handle der Anwendung
    DWORD dwDefaultPathType, // Default-Pfadtyp
    DWORD dwPChannelCount, // Anzahl der benötigten Kanäle
    DWORD dwFlags, // Synthesizer-Eigenschaften
    DMUS_AUDIOPARAMS *pParams); // Audio-Wiedergabe-Eigenschaften
```

Funktionsargumente

- **`ppDirectMusic`**: So man auf die intern erstellte DirectMusic-Schnittstelle explizit zugreifen möchte, kann man hier einen Zeiger auf einen Zeiger auf ein `IDirectMusic`-Objekt übergeben, andernfalls einfach Null übergeben, was wohl in den allermeisten Fällen angebracht ist.

- **`ppDirectSound`**: Natürlich wird DirectX-intern auch ein primäres DirectSound-Objekt verwendet. Sollte direkter Zugriff auf ebendieses vonnöten sein, kann hier ein Zeiger auf einen Zeiger auf ein `IDirectSound` angegeben werden (siehe oben).

- **`hWnd`**: Das ist (einmal mehr) ein Handle auf das aktuelle Anwendungsfenster. Wenn hier Null übergeben wird, verwendet DirectX einfach den Handle auf das aktuell aktive Fenster bzw. den Desktop.

- **`dwDefaultPathType`**: Dieses Argument ist der Default-Pfadtyp, den DirectX Audio verwenden soll. `DMUS_APATH_SHARED_STEREOPLUSREVERB` ist hier für die meisten Applikationen gut geeignet.

- **`dwPChannelCount`**: Hiermit lässt sich die Anzahl der Kanäle festlegen, die DirectX Audio für den Audiopfad allokieren soll. Pro Wave-Datei reicht üblicherweise ein Kanal während Datentypen, die den Synthesizer beanspruchen in der Regel mehrere Kanäle brauchen. So belegen beispielsweise MIDI-Dateien bis zu 16 Kanäle und vom DirectMusic Producer generierte Segmente unter Umständen sogar mehr.

- **`dwFlags`**: Dieser Parameter bestimmt die verwendeten Synthesizer-Features. Nachdem man oft im Vorhinein nicht allzu genau sagen kann, welche Eigenschaften man benötigen wird, halte ich es für am besten, in Fällen, in denen die Performance nicht extrem kritisch ist, `DMUS_AUDIOF_ALL` zu übergeben.

- **`pParams`**: Mit diesem Parameter können diverse erweiterte Eigenschaften angefordert werden, die meistens jedoch sowieso nicht gebraucht werden. Am besten Null übergeben!

Nun kann ein `IDirectMusicLoader8` erstellt werden, mit dem es dann möglich sein wird, Sound-Dateien zu laden. Damit das reibungslos funktioniert, muss zunächst mit der Methode `SetSearchDirectory()` das Suchverzeichnis für diese Dateien gesetzt werden:

SetSearchDirectory()

```
HRESULT IDirectMusicLoader8::SetSearchDirectory(
      REFGUID rguidClass, // Referenz auf Zielobjekttyp
      WCHAR* pwszPath, // Suchverzeichnis
      BOOL fClear); // Gebrauchsinformationen löschen
```

Funktionsargumente

- **`rguidClass`**: Dieser Parameter empfängt eine Referenz auf den DirectMusic-Objekttyp, der bei Ladevorgängen berücksichtigt werden soll. `GUID_DirectMusicAllTypes` stellt hierbei in nahezu jedem Fall eine gute Wahl dar; es weist den DirectX Audio Loader an, jeden ladbaren Objekttypus auch tatsächlich zu berücksichtigen.

- **`pwszPath`**: Das ist das wichtigste Argument dieser Funktion: Es bestimmt, welches Verzeichnis als Suchverzeichnis gesetzt wird.

- **`fClear`**: `fClear` gibt an, ob die Loader-Informationen vom letzten Funktionsaufruf gelöscht werden sollen, oder nicht. Meistens ist das nicht nötig, weswegen hier `false` übergeben werden kann. (Dieser Parameter muss nur dann `true` gesetzt werden, wenn die Gefahr besteht, dass zufällig gleich benannte Dateien aus unterschiedlichen Verzeichnissen geladen werden!)

Wenn diese Funktion korrekt ausgeführt wurde, ist das Suchverzeichnis für zu ladende Dateien eingestellt und Dateien können mit der Loader-Methode `LoadObjectFromFile()` geladen werden.

LoadObjectFromFile()

```
HRESULT IDirectMusicLoader8::LoadObjectFromFile(
                        REFGUID rguidClassID, // Klasse
                        REFIID iidInterfaceID, // Schnittstellen-ID
                        WCHAR *pwzFilePath, // Dateiname
                        void ** ppObject); // Zeiger auf Ziel-Interface
```

Funktionsargumente

- **`rguidClassID`**: Hierbei handelt es sich um einen Identifikator der DirectX Audio Klasse, die verwendet wird. In den meisten Fällen ist das `CLSID_DirectMusicSegment`.

- **`iidInterfaceID`**: Das ist die ID der Schnittstelle, die verwendet wird, hier wäre das `IID_IDirectMusicSegment8`.

- **`pwzFilePath`**: Dieses Argument ist ein Zeiger auf ein `WCHAR` und erhält den Namen der Datei, die geladen werden soll.

- **`ppObject`**: `ppObject` ist der Zeiger, der angibt, in welchem Objekt die Audiodaten gespeichert werden sollen. Allerdings ist zu beachten, dass die Adresse zu `LPVOID*` konvertiert werden muss, damit die Methode korrekt funktioniert!

Bevor man nun den geladenen Sound abspielen kann, muss noch die Band-Download-Funktion `Download()` aufgerufen werden, damit der Synthesizer auch wirklich Daten bekommt:

```
HRESULT IDirectMusicSegment::Download(IUnknown *pAudioPath); //performance
```

Funktionsargumente

- **`pAudioPath`**: Das ist der Audiopfad oder die Performance, der (die) als Download-Ziel angegeben werden soll.

Nun sind alle Vorbereitungen getroffen, um die Datei abspielen zu können. Dies lässt sich mittels eines Aufrufs von `PlaySegment()` bewerkstelligen.

`PlaySegment()`

```
HRESULT IDirectMusicPerformance8::PlaySegment(
      IDirectMusicSegment* pSegment, // Segment
      DWORD dwFlags, // Steuerungs-Flags
      __int64 i64StartTime, // Start-Zeit
      IDirectMusicSegmentState** ppSegmentState); // selten verwendet
```

Funktionsargumente

- **`pSegment`**: Das ist das Segment, das wiedergegeben werden soll.

- **`dwFlags`**: Auch beim Aufruf dieser Funktion gibt es die Möglichkeit, Kontrollflags zu übergeben, die das Verhalten genauer spezifizieren. Eine

detailliertere Erläuterung einzelner Werte ist in der DirectX Programmer's Reference zu finden, hier muss aber an und für sich überhaupt nichts übergeben, da die Wiedergabe auch ohne zusätzliche Angaben erfolgen kann.

- **`i64StartTime`**: An dieser Stelle wird die Start-Zeit festgelegt. Am häufigsten dürfte wohl der Wert Null verwendet werden, der DirectX Audio anweist, mit dem Playback so bald wie möglich zu beginnen.

- **`ppSegmentState`**: `ppSegmentState` ist ein eher unwichtiges Argument, dem man die Adresse eines Zeigers auf ein `IDirectMusicSegmentState` übergeben kann, in dem eine Instanz auf das Segment gespeichert werden soll.

Damit wäre die Soundwiedergabe abgeschlossen. Allerdings müssen nach der Wiedergabe einige Ressourcen wieder freigegeben werden, die die Anwendung sonst ewig reserviert halten würde.

DirectX Audio Shutdown

Im Wesentlichen müssen für ein korrektes Beenden von DirectX Audio vier Schritte ausgeführt werden:

- Falls noch Sounds wiedergegeben werden, müssen diese mit der Methode `Stop()` (einer `IDirectMusicPerformance8`) beendet werden. Hierbei kann generell überall das Argument Null übergeben werden, das DirectX Audio anweist, jede noch laufende Wiedergabe zu stoppen.

- Alle aktiven Performances müssen mit `CloseDown()` beendet werden.

- Alle erstellten Schnittstellen-Objekte müssen freigegeben werden.

- Zu guter Letzt muss auch noch COM beendet werden. Das geschieht wie immer mittels `CoUninitialize()`.

Um einen kurzen Überblick über die besprochenen Funktionen zu geben, folgt auch am Ende dieses Kapitels ein Quellcodebeispiel[30]:

[30] Da diesmal keine Spiel-typische Haupt-Schleife nötig ist, wurde die gesamte Datei gelistet!

```
// PlayWave.cpp:    a simple DirectX Audio .WAV-playing demo
//                  (plays [file] in [path])

#define INITGUID
#include <windows.h>
#include <dmusici.h>

char file[] = "test.wav";
char path[MAX_PATH] = "D:\\";

IDirectMusicLoader8*            g_loader = 0;
IDirectMusicPerformance8*       g_performance = 0;
IDirectMusicSegment8*           g_segment = 0;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpCmdLine, int nShowCmd)
{
      // initialize and set up COM
      CoInitialize(0);

      CoCreateInstance(CLSID_DirectMusicLoader, 0, CLSCTX_INPROC,
                        IID_IDirectMusicLoader8, (void**)&g_loader);

      CoCreateInstance(CLSID_DirectMusicPerformance, 0,
                        CLSCTX_INPROC, IID_IDirectMusicPerformance8,
                        (void**)&g_performance);

      // initialize DirectX Audio
      g_performance->InitAudio(0, 0, 0,
                        DMUS_APATH_SHARED_STEREOPLUSREVERB, 4,
                        DMUS_AUDIOF_ALL, 0);

      // convert path to unicode
      WCHAR wPath[MAX_PATH], wFile[MAX_PATH];
      MultiByteToWideChar(CP_ACP, 0, path, -1, wPath, MAX_PATH );
      MultiByteToWideChar(CP_ACP, 0, file, -1, wFile, MAX_PATH );

      // set loader's search directory
      g_loader->SetSearchDirectory(GUID_DirectMusicAllTypes,
                        wPath, false);
      // load file
      if (FAILED(g_loader->LoadObjectFromFile(
                        CLSID_DirectMusicSegment,
                        IID_IDirectMusicSegment8, wFile,
                        (LPVOID*) &g_segment)))
      {
            MessageBox(0, "ERROR: Failed to load file!",
                        "ERROR", MB_ICONSTOP);
            return 0;
      }

      // download band and play file
      g_segment->Download(g_performance);
      g_performance->PlaySegment(g_segment, 0, 0, 0);

      MessageBox(0,
            "The sample should be playing right now. Press OK to quit!",
            "PlayWave-Demo", MB_ICONINFORMATION);

      // shutdown everything
      g_performance->Stop(0, 0, 0, 0);

      g_loader->Release();
      g_loader = 0;

      g_performance->Release();
      g_performance = 0;
```

```
        g_segment->Release();
        g_segment = 0;

        CoUninitialize();

        return 0;
}
```

Kapitel 5 DirectInput

Grundlegendes

Mittlerweile fehlt nur noch ein für interaktive Anwendungen unerlässlicher Faktor: Die Eingabe. Ein wichtiger Bestandteil von DirectX ist DirectInput – die Komponente, die Benutzereingaben entgegennimmt und vorverarbeitet. DirectInput bietet im Wesentlichen die gleichen Vorteile wie alle anderen DirectX-Komponenten: Es ist hardwareunabhängig, ermöglicht die Nutzung vieler Features moderner Hardware (wie etwa Force Feedback) und besticht durch optimale Windows-Kompatibilität. Außerdem ist es sehr leicht, mit DirectInput Anwendungen zu programmieren, die jedes erdenkliche Eingabegerät nutzen können – von Tastatur und Maus bis hin zu Lightgun und ähnlichen Controllern wird alles unterstützt. Diese Vielfältigkeit erreichte Microsoft vor allem dadurch, dass die Anzahl der explizit unterstützten Gerätearten stark eingeschränkt wurde: DirectInput unterscheidet nur zwischen zwei unterschiedlichen Arten von Eingabegeräten: „normale“ Eingabegeräte und Force Feedback Eingabegeräte. Eine Tastatur wird als Eingabegerät mit 102 Tasten angesprochen, während eine Maus zwei bis drei[31] Achsen sowie (in den meisten Fällen) zwei bis fünf Knöpfe hat. Auch wenn das teilweise die Programmierung einfacher Applikationen ein wenig erschwert, ist es letztlich doch ein sehr mächtiges Konzept, denn übertriebene Spezialisierung hätte sich auf Flexibilität und Performance von DirectInput nur negativ ausgewirkt.[32]

Die Architektur von DirectInput

Da Microsoft bei DirectInput ein äußerst sinnvolles Design-Ziel verfolgte, ist auch die Architektur dieser Schnittstelle sehr sauber und klar gehalten. Im Wesentlichen kommt sie mit nur zwei wichtigen Interfaces aus:

- Das ***DirectInput-Objekt*** ist, ähnlich wie bei DirectDraw und DirectX Audio, die Instanz, die die gesamte grundlegende Funktionalität von DirectInput beinhaltet und ohne die keine DirectInput-Anwendung auskommt. Derzeit ist hierbei die Schnittstelle `IDirectInput8` aktuell.

[31] Bei den meisten sauber programmierten Treibern wird das Mausrad als dritte Achse angesprochen.
[32] Man bedenke, dass etwa Unix und seine Derivate lediglich vier (!) grundlegende Operationen besitzen.

- Das ***DirectInput Device*** hingegen repräsentiert ein beliebiges Eingabegerät und lässt sich (nach der Assoziation mit einem ebensolchen) unter anderem verwenden, um den Status des jeweiligen Gerätes zu überprüfen. Momentan wird als Interface `IDirectInputDevice8` verwendet.

- Abgesehen von diesen beiden Schnittstellen existiert noch eine dritte wichtige, die allerdings nur bei Force Feedback Programmen Verwendung findet: Der `IDirectInputEffect` ist genau das, was der Name bereits vermuten lässt – die Repräsentation eines ***Force Feedback Effekt***s. Auf ihn soll allerdings in dieser Arbeit nicht weiter eingegangen werden, da er für Tastaturen und Mäuse nur selten Sinn macht.[33]

DirectInput-Programmierung

Die Funktionalität von DirectInput wird vom Header `<dinput.h>` und der Library `<dinput8.lib>` bereitgestellt. Davon abgesehen benötigt eine DirectInput-Anwendung – wie jedes DirectX-Programm – die Bibliotheksdatei `<dxguid.lib>`. Wie gewohnt sollte man auch hier darauf achten, dem Projekt alle diese Dateien korrekt hinzuzufügen, da sonst Compiler und/ oder Linker den Dienst verweigern.

DirectInput verfügt im Gegensatz zu DirectX Audio über einige Hilfsfunktionen, die das Setup eines DirectInput-Programms zum Kinderspiel machen. Am Anfang jeder Anwendung dürfte dabei wohl die Funktion `DirectInput8Create()` stehen.

`DirectInput8Create()`

```
HRESULT WINAPI DirectInput8Create(HINSTANCE hinst, // Anwendungsinstanz
                                  DWORD dwVersion, // DirectInput-Version
                                  REFIID riidltf, // UID der Schnittstelle
                                  LPVOID* ppvOut, // Interface-Ziel-Zeiger
                                  LPUNKNOWN punkOuter); // selten verwendet
```

Funktionsargumente

- **`hinst`**: Das ist der Handle auf die Instanz der jeweiligen Anwendung. Auch wenn hierfür in den Samples der DirectX Programmer's Reference ein statischer Wert eingesetzt wird (konkret ein gespeicherter Handle auf die Anwendungsinstanz), halte ich es doch für klüger, den passenden Wert mittels `GetModuleHandle()`

[33] Von den neuen *Immersion*-Mäusen von Logitech abgesehen gibt es derzeit keine derartigen Geräte, die Force Feedback unterstützen würden.

zu ermitteln, da man so nicht auf die Instanz eines bestimmten Anwendungsfensters angewiesen ist. Im Alltagsgebrauch dürfte das zwar egal sein, aber letztlich stellt `GetModuleHandle()` eindeutig die elegantere Lösung dar. (Wenn man allerdings die Anwendungsinstanz ohnehin gespeichert hat, so kann man getrost die abgespeicherte Version verwenden.)

- **`dwVersion`**: Dieses `DWORD` legt fest, welche Version von DirectInput verwendet werden soll. Derzeit ist `0x0800` aktuell, aber am besten verwendet man hierbei schlichtweg `DIRECTINPUT_VERSION`, da dieser Wert mittels Präprozessor durch die aktuelle DirectInput-Version ersetzt wird (mittels `#define` im Header `<dinput.h>`).

- **`riidltf`**: Das ist der Unique Identifier der Schnittstelle. In Version 8.0/ 8.1 von DirectX sollte man hier dementsprechend `IID_IDirectInput8` verwenden.

- **`ppvOut`**: Dieser Parameter gibt an, in Variable das Schnittstellen-Objekt gespeichert werden soll, wenn die Funktion erfolgreich ausgeführt werden kann.

- **`pUnkOuter`**: Hier könnte diesmal sogar ein Wert ungleich Null übergeben werden, ohne dass das die Anwendung zum Absturz bringen würde, trotzdem ist es in den allermeisten Fällen nicht angebracht, es sei denn man möchte unbedingt mit COM experimentieren.

Wenn dieser Funktionsaufruf erfolgreich war, kann danach sofort ein IDirectInputDevice8 erstellt werden, was mit `IDirectInput8::CreateDevice()` geschieht.

`CreateDevice()`

```
HRESULT IDirectInput8::CreateDevice(REFGUID rguid, // GUID des Gerätes
            LPDIRECTINPUTDEVICE* lplpDirectInputDevice, // Ziel-Interface
            LPUNKNOWN pUnkOuter); // fortgeschrittenes COM-Feature
```

Funktionsargumente

- **`rguid`**: `rguid` ist die GUID des zu erstellenden Gerätes. Während man für Joysticks und Controller meist selbst eine GUID ermitteln muss, kann man für Tastatur und Maus immer `GUID_SysKeyboard` respektive `GUID_SysMouse` verwenden.

- **`lplpDirectInputDevice`**: Das ist die Adresse des Device-Zeigers, in dem das neu erstellte Gerät gespeichert werden soll.

- **`pUnkOuter`**: Dieses Feature wird normalerweise nicht benutzt.

Wurde `CreateDevice()` erfolgreich ausgeführt, muss das Datenformat des neu erstellten Gerätes festgelegt werden. Um diese Aufgabe durchführen zu können, bedient man sich üblicherweise der Methode `SetDataFormat()`.

`SetDataFormat()`

```
HRESULT IDirectInputDevice8::SetDataFormat(
                                LPCDIDATAFORMAT lpdf); // Datenformat
```

Funktionsargumente

- **`lpdf`**: `lpdf` ist ein Zeiger auf das gewünschte Datenformat des Eingabegeräts. Leider kann es ein recht unangenehme Aufgabe sein, herauszufinden, welches Datenformat für welches Gerät geeignet ist, aber DirectInput bietet einige vordefinierte Datenformate, die für die gängigsten Anwendungen sehr gut geeignet sind. Für Maus und Keyboard beispielsweise gibt es `c_dfDIMouse`[34] sowie `c_dfDIKeyboard`.

Danach sollte natürlich noch das Verhalten des jeweiligen Eingabegerätes festgelegt werden, wofür es auch hier eine Methode `SetCooperativeLevel()` gibt.

`SetCooperativeLevel()`

Im Prinzip ist die Wahl der geeigneten Kontrollflags für diese Funktion reine Geschmackssache. Das Einzige, worauf man achten muss, ist, sowohl eines der beiden Flags `DISCL_BACKGROUND` und `DISCL_FOREGROUND` als auch entweder `DISCL_NONEXCLUSIVE` oder `DISCL_EXCLUSIVE` zu wählen – wenn man beispielsweise `DISCL_BACKGROUND | DISCL_FOREGROUND` übergibt, ist das in zweifacher Hinsicht ein Fehler. Einerseits wäre das nämlich eine widersprüchliche (Vordergrund *und* Hintergrund) und andererseits eine unvollständige Angabe (weder exklusiv noch non-exklusiv). Was genau welches Flag im Detail übergibt, kann in der DirectX Programmer's Reference nachgelesen werden.

```
HRESULT IDirectInputDevice8::SetCooperativeLevel(
                                    HWND hwnd, // Hauptf.-Handle
                                    DWORD dwFlags); // Verhalten
```

[34] Es gibt für Mäuse auch das Datenformat `c_dfDIMouse2`, das bis zu 8 Mouse-Buttons unterstützt (im Gegensatz zu 4 bei `c_dfDIMouse`). Man sollte generell allerdings darauf achten, dieses Format nur gemeinsam mit `DIMOUSESTATE2`-Strukturen zu verwenden.

Funktionsargumente

- **`hwnd`**: Dieser Parameter empfängt einen Handle auf das Hauptfenster der Anwendung.

- **`dwFlags`**: Hier werden Steuerungsflags zur Bestimmung des Eingabegeräteverhaltens übergeben. In den meisten Fällen erreicht man das gewünschte Verhalten dadurch, dass man `DISCL_BACKGROUND | DISCL_NONEXCLUSIVE` übergibt, was festlegt, dass die Anwendung immer Eingabedaten des jeweiligen Gerätes empfängt, selbst wenn sie sich im Hintergrund befindet und dass der Zugriff auf das Gerät nicht exklusiv erfolgt.

Verläuft auch diese Funktion erfolgreich, ist die Anwendung schon fast bereit, den Status des Eingabegerätes zu überprüfen. Der einzige Schritt, der hierfür noch fehlt, ist ein Aufruf von `Acquire()`, mit dem die Anwendung Zugriff auf ein Eingabegerät vorbereitet.

`Acquire()`

```
HRESULT IDirectInputDevice8::Acquire(void);
```

Jetzt kann mit `GetDeviceState()` der aktuelle Status des Eingabegeräts ermittelt werden:

`GetDeviceState()`

Mit dieser Funktion kann der aktuelle Status eines Eingabegerätes überprüft werden. Meistens wird hierbei die *gepufferte Eingabeverarbeitung* verwendet. Das bedeutet, dass man zunächst ein Array erstellt, das als Puffer, den Zustand des jeweiligen Gerätes zum Zeitpunkt der Abfrage speichert und dann – statt das Eingabegerät selbst zu überprüfen – die gepufferten Werte abruft.

```
HRESULT IDirectInputDevice8::GetDeviceState(DWORD cbData, // Puffergröße
                                            LPVOID lpvData); // Pufferadresse
```

Funktionsargumente

- **`cbData`**: An dieser Stelle wird die Größe des Puffers angegeben.

- **`lpvData`**: Diese Variable nimmt die Adresse des Puffers, in den die Zustandsbeschreibung geschrieben werden soll, entgegen.

Sobald auch diese Funktion erfolgreich beendet wurde, kann man folgendermaßen testen, ob etwa die Escape-Taste gedrückt wurde (wenn sich die Zustandsdaten im Array `buffer` befinden):

```
if (buffer[DIK_ESCAPE] & 0x80)
	; // Escape wurde gedrückt
```

Hierbei entspricht `DIK_ESCAPE` der Gerätekonstante der Taste [Esc][35].

Nachdem diese Art der Abfrage aber nicht unbedingt sonderlich gut lesbar ist, wird normalerweise ein Makro eingesetzt, um den Tastenzustand zu überprüfen[36]:

```
#define KEYDOWN(name, key) (name[key] & 0x80)
```

Letztlich stellt das aber keine allzu elegante Lösung dar, denn `#define` ist eine primitive Präprozessor-Direktive, die eigentlich in C++ nichts mehr verloren hat, weswegen ich selbst lieber folgende Funktion verwende:

```
inline bool DIKeyDown(char keystate[], char key)
{
	return (keystate[key] & 0x80) ? true : false;
}
```

Das zweite Argument, `key`, müsste keinesfalls zwangsläufig ein `int` sein, ich habe `int` lediglich aufgrund der geplanten Verwendung dieser Funktion gewählt (auch für Maustasten). Genauso gut könnte man aber auch irgendeinen anderen Datentypen wählen, da in `<dinput.h>` sowieso die verwendetet Text-Konstanten per `#define`s durch Hex-Werte ersetzt werden.

Jeder akzeptable Compiler erkennt das ISO-C++ konforme Schlüsselwort `inline`, was zur Folge hat, dass die Performance meiner Funktion der des Makros um nichts nachstehen sollte. Abgesehen davon erleichtert es die oben demonstrierte Überprüfung ein wenig:

```
if KeyDown(buffer, DIK_ESCAPE)
	; // Escape wurde gedrückt
```

Für Mäuse kann übrigens die Struktur `DIMOUSESTATE` verwendet werden, deren Prototyp folgendermaßen aussieht:

`DIMOUSESTATE`

```
typedef struct _DIMOUSESTATE {
    LONG    lX; // delta(X)
```

[35] Im Anhang C findet sich eine Liste unter DirectInput verfügbarer Gerätekonstanten.
[36] Vgl.: *DirectX Programmer's Reference*

```
    LONG    lY; // delta(Y)
    LONG    lZ; // delta(Z) [Mausrad!]
    BYTE    rgbButtons[4]; // Maustasten
} DIMOUSESTATE, *LPDIMOUSESTATE;
```

Struktur-Komponenten

- **`lX`**: Das ist die Änderung der X-Koordinate seit dem letzten Aufruf von `GetDeviceState()`.

- **`lY`**: Das ist die Änderung der Y-Koordinate seit dem letzten Aufruf von `GetDeviceState()`.

- **`lZ`**: Das ist die Änderung der Z-Koordinate seit dem letzten Aufruf von `GetDeviceState()`. Wie bereits erwähnt repräsentiert die Z-Achse das Mausrad einer Maus (sofern vorhanden), wobei ein positiver Z-Wert bedeutet, dass das Mausrad vom Benutzer wegbewegt wurde und ein negativer Z-Wert, dass es zum Benutzer hin bewegt wurde.

- **`rgbButtons`**: In dieser Matrix sind die Zustände der Maustasten beim letzten Aufruf von `GetDeviceState()` gespeichert.

Wenn man bei `SetDataFormat()` `c_dfDIMouse2` gewählt hat, muss statt dieser Struktur `DIMOUSESTATE2` verwendet werden. Im Großen und Ganzen sieht dieser Typ genauso aus, bis auf die Tatsache, dass die Größe von `rgbButtons` von 4 Elementen auf 8 Elemente geändert wurde.

Shutdown von DirectInput

Auch DirectInput muss korrekt beendet werden, wenn es nicht mehr gebraucht wird. Das wird im Wesentlichen mit den gleichen Methoden erledigt, die bereits in den Kapiteln über DirectDraw und DirectX Audio vorgestellt wurden. Nur bei DirectInput-Devices muss eine Kleinigkeit beachtet werden: Bevor man ein derartiges Gerät mittels Release() freigibt, muss man es unbedingt mit `Unacquire()` abmelden, da DirectInput sonst unter Umständen versuchen könnte, auf ein bereits deallokiertes Speicherobjekt zuzugreifen, was wohl einen Bluescreen zur Folge hätte (zumindest unter Win9x).

Um die erläuterten Funktionen auch im Einsatz zeigen zu können, habe ich ein weiteres Code-Beispiel (basierend auf der GameCon) geschrieben:

Der Dateianfang sieht folgendermaßen aus:

```
// DINotify.cpp: A simple DirectInput-Sample-Application

#define WIN32_LEAN_AND_MEAN

// INCLUDES
#include <windows.h>
#include <windowsx.h>
#include <string>
#include <dinput.h> // NEU!!!

using std::string;
typedef unsigned char uchar;

// GLOBAL CONSTANTS
const string WinClassName("WinClass01");

// GLOBAL VARIABLES
HWND         g_hWnd = 0;
HINSTANCE    g_hInstance = 0;
LPDIRECTINPUT8 g_lpDI = 0; // NEU!!!
LPDIRECTINPUTDEVICE8 g_lpKeyboard = 0; // NEU!!!
LPDIRECTINPUTDEVICE8 g_lpMouse = 0; // NEU!!!

// PROTOTYPES
inline bool KeyDown(BYTE VKCode);
inline bool KeyUp(BYTE VKCode);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpCmdLine, int nShowCmd);
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam);
bool CreateNewWindow(HINSTANCE* lphInstance, HINSTANCE* lpg_hInstance,
                     HWND* lpg_hWnd, const string caption,
                     WNDPROC lpfnWndProc);
bool GameInit(void);
bool GameMain(void);
void GameShutdown(void);

// FUNCTIONS
inline bool KeyDown(BYTE VKCode)
{
      return (GetAsyncKeyState(VKCode) & 0x8000) ? true : false;
}

inline bool KeyUp(BYTE VKCode)
{
      return (GetAsyncKeyState(VKCode) & 0x8000) ? false : true;
}

inline bool DIKeyDown(const uchar keystate[], const int key)
{
      return (keystate[key] & 0x80) ? true : false;
}
// …SNAP
```

Hier sind noch die drei Funktionen `GameInit()`, `GameMain()` und `GameShutdown()`:

```
// SNIP…
bool GameInit(void)
{
      if (FAILED(DirectInput8Create(g_hInstance, DIRECTINPUT_VERSION,
                          IID_IDirectInput8, (void**)&g_lpDI, 0)))
            return false;
      if (FAILED(g_lpDI->CreateDevice(GUID_SysKeyboard,
                                      &g_lpKeyboard, 0)))
```

```
            return false;
      if (FAILED(g_lpDI->CreateDevice(GUID_SysMouse, &g_lpMouse, 0)))
            return false;
      if (FAILED(g_lpKeyboard->SetDataFormat(&c_dfDIKeyboard)))
            return false;
      if (FAILED(g_lpMouse->SetDataFormat(&c_dfDIMouse2)))
            return false;
      if (FAILED(g_lpKeyboard->SetCooperativeLevel(g_hWnd,
                        DISCL_BACKGROUND | DISCL_NONEXCLUSIVE)))
            return false;
      if (FAILED(g_lpMouse->SetCooperativeLevel(g_hWnd,
                        DISCL_BACKGROUND | DISCL_NONEXCLUSIVE)))
            return false;
      if (FAILED(g_lpKeyboard->Acquire()))
            return false;
      if (FAILED(g_lpMouse->Acquire()))
            return false;
      return true;
}

bool GameMain(void)
{
      static uchar KeyState[256];
      static DIMOUSESTATE2 MouseState;
      if (FAILED(g_lpKeyboard->GetDeviceState(sizeof(KeyState),
                                    (void*)&KeyState)))
            return false;
      if (FAILED(g_lpMouse->GetDeviceState(sizeof(MouseState),
                                    (void*)&MouseState)))
            return false;

      if (MouseState.lX > 0)
            MessageBox(g_hWnd, "Mouse has been moved right.",
                        "DINotify", MB_ICONINFORMATION);
      else if (MouseState.lX < 0)
            MessageBox(g_hWnd, "Mouse has been moved left.",
                        "DINotify", MB_ICONINFORMATION);
      if (MouseState.lY > 0)
            MessageBox(g_hWnd, "Mouse has been moved down.",
                        "DINotify", MB_ICONINFORMATION);
      else if (MouseState.lY < 0)
            MessageBox(g_hWnd, "Mouse has been moved up.",
                        "DINotify", MB_ICONINFORMATION);
      if (MouseState.lZ > 0)
            MessageBox(g_hWnd, "Mousewheel has been moved up.",
                        "DINotify", MB_ICONINFORMATION);
      else if (MouseState.lZ < 0)
            MessageBox(g_hWnd, "Mousewheel has been moved down.",
                        "DINotify", MB_ICONINFORMATION);
      for (int i = 0; i < sizeof(MouseState.rgbButtons); i++)
            if (DIKeyDown(MouseState.rgbButtons, i))
                  MessageBox(g_hWnd, "A mouse-button has been pushed.",
                        "DINotify", MB_ICONINFORMATION);

      if (DIKeyDown(KeyState, DIK_ESCAPE))
            return false;

      return true;
}

void GameShutdown(void)
{
      if (g_lpMouse)
      {
            g_lpMouse->Unacquire();
            g_lpMouse->Release();
            g_lpMouse = 0;
```

```
        }
        if (g_lpKeyboard)
        {
                g_lpKeyboard->Unacquire();
                g_lpKeyboard->Release();
                g_lpKeyboard = 0;
        }
        if (g_lpDI)
        {
                g_lpDI->Release();
                g_lpDI = 0;
        }
}
// SNAP…
```

Kapitel 6 Schlusswort

Zukunft von DirectX

Einer der Aspekte, der DirectX besonders interessant für Programmierer macht, ist, dass DirectX keine „tote“ Schnittstelle ist. Es wird tatsächlich weiterentwickelt und von Version zu Version besser. Als ich begann, mich intensiver mit DirectX zu beschäftigen, war Version 7.0 gerade der aktuellste Release von DirectX. Mittlerweile ist Version 8.1 erhältlich und erste Feature-Listen von Version 9.0 geistern durch das Netz. Im gleichen Maße, in dem sich die heute erhältliche Hardware weiterentwickelt, ist das auch bei der momentan aktuellen Software der Fall und das bedeutet, dass sich auch die gegebenen Möglichkeiten rasant vermehren werden. Während vor einigen Jahren 3D-Karten teure Luxusgüter waren, wird heutzutage fast jeder neue (nichtmobile) Computer mit einer Grafikkarte mit brauchbarem 3D-Chipsatz ausgeliefert. Abgesehen davon hat die Zukunft von DirectX noch ganz Anderes zu bieten: Die neue Microsoft-Spielekonsole *XBox* verwendet als Betriebssystem eine modifizierte Version von Windows XP und als Haupt-API für Multimedia Inhalte DirectX! Das bedeutet letztlich, dass Anwendungen, die ursprünglich für den PC programmiert wurden mit minimalem Anpassungsaufwand auf die XBox konvertiert werden können!

Kurz gesagt: DirectX sieht zweifellos einer großen Zukunft entgegen…

Kapitel 7 Anhang

Anhang A: GPL[37]

The GNU General Public License (GPL)

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the

[37] Free Software Foundation, Inc.: *The GNU General Public License (GPL)*. Boston: 1991

original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a

notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it

free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C)
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Anhang B: Sourcen

GameCon *(**Game Con**sole)* Echtzeit-Anwendungscontainer

```
// GameCon.cpp: A simple game console.


#define WIN32_LEAN_AND_MEAN

// INCLUDES
#include <windows.h>
#include <windowsx.h>
#include <string>

using std::string;

// GLOBAL CONSTANTS
const string WinClassName("WinClass01");

// GLOBAL VARIABLES
HWND         g_hWnd = 0;
HINSTANCE    g_hInstance = 0;

// PROTOTYPES
inline bool KeyDown(BYTE VKCode);
inline bool KeyUp(BYTE VKCode);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpCmdLine, int nShowCmd);
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam,
                                              LPARAM lParam);
bool CreateNewWindow(HINSTANCE* lphInstance,
                                HINSTANCE* lpg_hInstance, HWND* lpg_hWnd,
                                const string caption, WNDPROC lpfnWndProc);
bool GameInit(void);
bool GameMain(void);
void GameShutdown(void);

// FUNCTIONS
inline bool KeyDown(BYTE VKCode)
{
      return (GetAsyncKeyState(VKCode) & 0x8000) ? true : false;
}

inline bool KeyUp(BYTE VKCode)
{
      return (GetAsyncKeyState(VKCode) & 0x8000) ? false : true;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpCmdLine, int nShowCmd)
{
      MSG msg;
      if (!(CreateNewWindow(&hInstance, &g_hInstance, &g_hWnd,
                                    "GameCon", WndProc)))
      {
            MessageBox(0, "CreateNewWindow() failed.", "error",
                        MB_ICONSTOP);
            PostMessage(g_hWnd, WM_CLOSE, 0, 0);
      }

      if (!(GameInit()))
      {
            MessageBox(0, "GameInit() failed.", "error",
                        MB_ICONSTOP);
```

```
		GameShutdown();
		PostMessage(g_hWnd, WM_CLOSE, 0, 0);
	}

	while (true)
	{
		if (PeekMessage(&msg, 0, 0, 0, PM_REMOVE))
		{
			if (msg.message == WM_QUIT)
				break;
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
		if (!(GameMain()))
			SendMessage(g_hWnd, WM_CLOSE, 0, 0);
	}
	GameShutdown();
	return msg.wParam;
} // end WinMain()

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam,
                                              LPARAM lParam)
{
	switch (msg)
	{
	case WM_CREATE:
		return 0;
	break;

	case WM_PAINT:
		{
			PAINTSTRUCT ps;
			HDC hDC = BeginPaint(hWnd, &ps);
			EndPaint(hWnd, &ps);
			return 0;
		} break;

	case WM_DESTROY:
		{
			PostQuitMessage(0);
			return 0;
		} break;

	default: break;
	} // end switch

	return DefWindowProc(hWnd, msg, wParam, lParam);
} // end WinProc()

bool CreateNewWindow(HINSTANCE* lphInstance,
			HINSTANCE* lpg_hInstance, HWND* lpg_hWnd,
			const string caption, WNDPROC lpfnWndProc)
{
	WNDCLASSEX	winclass;
	winclass.cbSize		= sizeof(WNDCLASSEX);
	winclass.style		= CS_DBLCLKS | CS_OWNDC |
				  CS_HREDRAW | CS_VREDRAW;
	winclass.lpfnWndProc	= lpfnWndProc;
	winclass.cbClsExtra	= 0;
	winclass.cbWndExtra	= 0;
	winclass.hInstance	= *lphInstance;
	winclass.hIcon			= LoadIcon(0, IDI_APPLICATION);
	winclass.hCursor	= LoadCursor(0, IDC_ARROW);
	winclass.hbrBackground	= (HBRUSH)GetStockObject(4);
	winclass.lpszMenuName	= 0;
	winclass.lpszClassName	= WinClassName.c_str();
	winclass.hIconSm	= LoadIcon(0, IDI_APPLICATION);
```

```
	*lpg_hInstance = *lphInstance;

	if (!(RegisterClassEx(&winclass)))
		return false;

	if (!(*lpg_hWnd = CreateWindowEx(0, WinClassName.c_str(),
				caption.c_str(), WS_POPUP |
			WS_VISIBLE, 0, 0, 320, 240,
				0, 0, *lphInstance, 0)))
		return false;

	return true;
}

bool GameInit(void)
{
	// your game's initialization routines here
	return true;
}

bool GameMain(void)
{
	// your game's main loop here
	if (KeyDown(VK_ESCAPE))
		return false;
	return true;
}

void GameShutdown(void)
{
	// your game's shutdown routines here
}
```

blent *(**Bl**itter **Ent**ity)* Grafikengine

```
/*

	blent.hpp: graphics-engine blent 0.7 (using MS DirectDraw)
	Copyright (C) 2002 Sebastian Boehm <s-boehm@users.sourceforge.net>


	This program is free software; you can redistribute it and/or
	modify it under the terms of the GNU General Public License
	as published by the Free Software Foundation; either
	version 2 of the License, or (at your option) any later
	version.

	This program is distributed in the hope that it will be useful,
	but WITHOUT ANY WARRANTY; without even the implied warranty of
	MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
	GNU General Public License for more details.

	You should have received a copy of the GNU General Public
	License along with this program; if not, write to the

		Free Software Foundation, Inc.,
		59 Temple Place, Suite 330,
		Boston, MA 02111-1307
		USA

  Make sure to add <dxguid.lib>, <ddraw.lib> and <ddraw.h> to
  your project in order to be able to compile correctly!
  Check out http://snip.sourceforge.net for details!
```

```
*/


#ifndef BLENT_HPP
#define BLENT_HPP

#include <string>
#include <windows.h>
#include <ddraw.h>

typedef unsigned int uint;

//////////////////////////////////////
// declaration of CDD
//////////////////////////////////////
class CDD
{
public:
	CDD(const DWORD XRes, const DWORD YRes);
	~CDD(void);
	friend class CBlent;
	HRESULT Init(HWND* lphWnd);
	bool Flip(void);
	inline DWORD getXRes(void) const { return xRes; };
	inline DWORD getYRes(void) const { return yRes; };
	inline DWORD getBPP(void) const { return bpp; };
private:
	LPDIRECTDRAW7			lpDD;		// main DD-object
	LPDIRECTDRAWSURFACE7	lpDDSPrimary;// primary surface
	LPDIRECTDRAWSURFACE7	lpDDSSecondary;	// secondary surface
	LPDIRECTDRAWCLIPPER		lpDDC;			// main clipper
	DWORD  xRes;  // horizontal resolution
	DWORD  yRes;  // vertical resolution
	DWORD  bpp;   // bits per pixel
};


//////////////////////////////////////
// declaration of CBlent
//////////////////////////////////////
class CBlent
{
public:
	CBlent(const DWORD XRes, const DWORD YRes, CDD* lpDirectDraw);
	~CBlent(void);
	HRESULT Init(void);
	HRESULT LoadBitmap(const std::string filename, DWORD x = 0,
			DWORD y = 0, DWORD width = 0, DWORD height = 0);
	HRESULT View(DWORD x, DWORD y);
private:
	LPDIRECTDRAWSURFACE7	lpDDS; // surface
	CDD*   DDraw; // associated CDD-object
	DWORD  xRes;  // horizontal resolution
	DWORD  yRes;  // vertical resolution
};

#endif
```

```
// blent.cpp: implementation of blent v0.7


// Read blent.hpp for details!

#include "blent.hpp"

//////////////////////////////////////
// implementation of CDD
```

```
//////////////////////////////////////

CDD::CDD(const DWORD XRes, const DWORD YRes)
: xRes(XRes), yRes(YRes), bpp(16), lpDDSPrimary(0),
       lpDDSSecondary(0), lpDDC(0) {}

CDD::~CDD(void)
{
       if (lpDDC)
       {
              lpDDC->Release();
              lpDDC = 0;
       }
       if (lpDDSSecondary)
       {
              lpDDSSecondary->Release();
              lpDDSSecondary = 0;
       }
       if (lpDDSPrimary)
       {
              lpDDSPrimary->Release();
              lpDDSPrimary = 0;
       }
       if (lpDD)
       {
              lpDD->Release();
              lpDD = 0;
       }
}

HRESULT CDD::Init(HWND* lphWnd)
{
       LPDIRECTDRAW        lpDDTemp;
       HRESULT             tmp;
       DDSURFACEDESC2      ddsd;
       RECT                visible;
       LPRGNDATA           lpRgnData = 0;

       if (FAILED(tmp = (DirectDrawCreate(0, &lpDDTemp, 0))))
              return tmp;

       if (FAILED(tmp = (lpDDTemp->QueryInterface(IID_IDirectDraw7,
                                 (LPVOID*)&lpDD))))
       {
              lpDDTemp->Release();
              lpDDTemp = 0;
              return tmp;
       }

       lpDDTemp->Release();
       lpDDTemp = 0;

       if (FAILED(tmp = (lpDD->SetCooperativeLevel(*lphWnd,
                           DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN |
                           DDSCL_ALLOWREBOOT))))
              return tmp;
       if (FAILED(tmp = (lpDD->SetDisplayMode(xRes, yRes, bpp, 0, 0))))
              return tmp;

       ShowCursor(false);

       ZeroMemory(&ddsd, sizeof(ddsd));
       ddsd.dwSize = sizeof(ddsd);
       ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
       ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP
                                        | DDSCAPS_COMPLEX;
       ddsd.dwBackBufferCount = 1;
```

```
	if (FAILED(tmp = (lpDD->CreateSurface(&ddsd, &lpDDSPrimary, 0))))
		return tmp;

	ZeroMemory(&ddsd, sizeof(ddsd));
	ddsd.dwSize = sizeof(ddsd);
	ddsd.ddsCaps.dwCaps = DDSCAPS_BACKBUFFER;
	if (FAILED(tmp = (lpDDSPrimary->GetAttachedSurface(
				&ddsd.ddsCaps, &lpDDSSecondary))))
		return tmp;

	if (FAILED(tmp = (lpDD->CreateClipper(0, &lpDDC, 0))))
		return tmp;

	visible.left = 0;
	visible.top = 0;
	visible.right = xRes;
	visible.bottom = yRes;

	// size = sizeof(RGNDATAHEADER) + nRects * sizeof(RECT)
	lpRgnData = reinterpret_cast<LPRGNDATA> (operator new(
				sizeof(RGNDATAHEADER) + sizeof(RECT)));

	lpRgnData->rdh.dwSize = sizeof(RGNDATAHEADER);
	lpRgnData->rdh.iType = RDH_RECTANGLES;
	lpRgnData->rdh.nCount = 1;
	lpRgnData->rdh.nRgnSize = sizeof(RECT);
	lpRgnData->rdh.rcBound.left = 0;
	lpRgnData->rdh.rcBound.right = xRes;
	lpRgnData->rdh.rcBound.top = 0;
	lpRgnData->rdh.rcBound.bottom = yRes;

	memcpy(lpRgnData->Buffer, &visible, sizeof(RECT));

	if (FAILED(tmp = (lpDDC->SetClipList(lpRgnData, 0))))
	{
		delete lpRgnData;
		return tmp;
	}

	if (FAILED(tmp = (lpDDSSecondary->SetClipper(lpDDC))))
	{
		delete lpRgnData;
		return tmp;
	}

	delete lpRgnData;

	return DD_OK;
}

bool CDD::Flip(void)
{
	if (!lpDD)
		return false;

	HRESULT tmp;

	while (true)
	{
		tmp = lpDDSPrimary->Flip(0, DDFLIP_WAIT);
		if (SUCCEEDED(tmp))
			break;
		else if (tmp == DDERR_SURFACELOST)
			if (FAILED(lpDDSPrimary->Restore()))
				break;
			else
```

```
                        break;
        }
        return true;
}

//////////////////////////////////////
// implementation of CBlent
//////////////////////////////////////

CBlent::CBlent(const DWORD XRes, const DWORD YRes, CDD* lpDirectDraw)
: xRes(XRes), yRes(YRes), DDraw(lpDirectDraw) {}

CBlent::~CBlent(void)
{
        if (lpDDS)
        {
                lpDDS->Release();
                lpDDS = 0;
        }
}

HRESULT CBlent::Init(void)
{
        if (!DDraw)
                return E_FAIL;
        HRESULT tmp;
        DDSURFACEDESC2 ddsd;

        ZeroMemory(&ddsd, sizeof(ddsd));
        ddsd.dwSize = sizeof(ddsd);
        ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH |DDSD_CKSRCBLT;
        ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
        ddsd.dwWidth = xRes;
        ddsd.dwHeight = yRes;
        ddsd.ddckCKSrcBlt.dwColorSpaceHighValue = 0;
        ddsd.ddckCKSrcBlt.dwColorSpaceLowValue = 0;

        if (FAILED(tmp = (DDraw->lpDD->CreateSurface(&ddsd, &lpDDS, 0))))
                return tmp;

        return DD_OK;
}

HRESULT CBlent::LoadBitmap(const std::string filename, DWORD x,
                           DWORD y, DWORD width, DWORD height)
{
        if (filename == "")
                return E_INVALIDARG;
        if (!lpDDS)
                return E_FAIL;

        HRESULT tmp;
        HBITMAP hBitmap;
        BITMAP bmp;
        HDC hDCImg, hDC;
        DDBLTFX DDBltFx;

        if (!(hBitmap = (HBITMAP)LoadImage(0, filename.c_str(),
                        IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE)))
                return E_FAIL;

        lpDDS->Restore();

        hDCImg = CreateCompatibleDC(0);
        SelectObject(hDCImg, hBitmap);

        GetObject(hBitmap, sizeof(bmp), &bmp);
```

```
	width = !width ? bmp.bmWidth : width;
	height = !height ? bmp.bmHeight : height;

	if(FAILED(tmp = (lpDDS->GetDC(&hDC))))
	{
		DeleteDC(hDCImg);
		return tmp;
	}

	if (!(BitBlt(hDC, 0, 0, width, height, hDCImg, x, y, SRCCOPY)))
	{
		lpDDS->ReleaseDC(hDC);
		DeleteDC(hDCImg);
		return E_FAIL;
	}
	lpDDS->ReleaseDC(hDC);
	DeleteDC(hDCImg);

	return DD_OK;
}

HRESULT CBlent::View(DWORD x, DWORD y)
{
	if (!lpDDS)
		return E_FAIL;

	HRESULT tmp;

	RECT dest;
	dest.left = x;
	dest.top = y;
	dest.right = x + xRes;
	dest.bottom = y + yRes;
	if (FAILED(tmp = (DDraw->lpDDSSecondary->Blt(&dest, lpDDS,
			0, DDBLT_WAIT | DDBLT_KEYSRC, 0))))
		return tmp;
	return DD_OK;
}
```

sober (***So**und **B**uff**er***) Soundengine

```
/*

	sober.hpp: sound-engine sober 0.7 (using MS DirectSound)
	Copyright (C) 2002 Sebastian Boehm <s-boehm@users.sourceforge.net>


	This program is free software; you can redistribute it and/or
	modify it under the terms of the GNU General Public License

	as published by the Free Software Foundation; either
	version 2 of the License, or (at your option) any later
	version.

	This program is distributed in the hope that it will be useful,
	but WITHOUT ANY WARRANTY; without even the implied warranty of
	MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
	GNU General Public License for more details.

	You should have received a copy of the GNU General Public
	License along with this program; if not, write to the

		Free Software Foundation, Inc.,
		59 Temple Place, Suite 330,
		Boston, MA 02111-1307
```

```
            USA

  Make sure to add <dmusici.h> and <dxguid.lib> to your project
  in order to be able to compile correctly!
  Check out http://snip.sourceforge.net for details!
*/

#ifndef SOBER_HPP
#define SOBER_HPP

#include <dmusici.h>

////////////////////////////////////////
// declaration of CDXA
////////////////////////////////////////
class CDXA
{
public:
      CDXA(void);
      ~CDXA(void);
      friend class CSober;
      HRESULT Init(const char SoundDir[MAX_PATH]);
private:
      IDirectMusicPerformance8*  lpDMP; // main performance
      IDirectMusicLoader8*       lpDML; // main loader
};


////////////////////////////////////////
// declaration of CSober
////////////////////////////////////////
class CSober
{
public:
      CSober(CDXA* lpDirectXAudio);
      ~CSober(void);
      HRESULT LoadWave(const char filename[]);
      bool Play(void);
      void Stop(void);
private:
      CDXA*  DXAudio;                  // associated CDXA
      IDirectMusicSegment8*      lpDMS; // main segment
};

#endif
```

```
// sober.cpp: implementation of sober v0.7

// Read sober.hpp for details!

#include "sober.hpp"

////////////////////////////////////////
// implementation of CDXA
////////////////////////////////////////

CDXA::CDXA(void)
: lpDMP(0), lpDML(0) {}

CDXA::~CDXA(void)
{
      if (lpDML)
      {
            lpDML->Release();
            lpDML = 0;
      }
      if (lpDMP)
```

```
	{
		lpDMP->Release();
		lpDMP = 0;
	}
}

HRESULT CDXA::Init(const char SoundDir[MAX_PATH])
{
	HRESULT tmp;
	WCHAR wPath[MAX_PATH];

	CoInitialize(0);

	CoCreateInstance(CLSID_DirectMusicLoader, 0, CLSCTX_INPROC,
			IID_IDirectMusicLoader8, (void**)&lpDML);

	CoCreateInstance(CLSID_DirectMusicPerformance, 0, CLSCTX_INPROC,
			IID_IDirectMusicPerformance8, (void**)&lpDMP);

	if (FAILED(tmp = (lpDMP->InitAudio(0, 0, 0,
			DMUS_APATH_SHARED_STEREOPLUSREVERB, 128,
			DMUS_AUDIOF_ALL, 0))))
		return tmp;

	MultiByteToWideChar(CP_ACP, 0, SoundDir, -1, wPath, MAX_PATH );

	if (FAILED(tmp = (lpDML->SetSearchDirectory(
			GUID_DirectMusicAllTypes,
		wPath, false))))
		return tmp;

	return S_OK;
}

////////////////////////////////////
// implementation of CSober
////////////////////////////////////

CSober::CSober(CDXA* lpDirectXAudio)
: DXAudio(lpDirectXAudio), lpDMS(0) {}

CSober::~CSober(void)
{
	if (lpDMS)
	{
		DXAudio->lpDMP->Stop(lpDMS, 0, 0, 0);
		lpDMS->Release();
		lpDMS = 0;
	}
}

HRESULT CSober::LoadWave(const char filename[])
{
	if (!DXAudio)
		return E_FAIL;

	HRESULT tmp;
	WCHAR wFile[MAX_PATH];

	MultiByteToWideChar(CP_ACP, 0, filename, -1, wFile, MAX_PATH );

	if (FAILED(tmp = (DXAudio->lpDML->LoadObjectFromFile(
				CLSID_DirectMusicSegment,
				IID_IDirectMusicSegment8, wFile,
				(LPVOID*) &lpDMS))))
		return tmp;
```

```
	return S_OK;
}

bool CSober::Play(void)
{
	if (!lpDMS)
		return false;

	lpDMS->Download(DXAudio->lpDMP);
	DXAudio->lpDMP->PlaySegment(lpDMS, 0, 0, 0);

	return true;
}

void CSober::Stop(void)
{
	if (lpDMS)
		DXAudio->lpDMP->Stop(lpDMS, 0, 0, 0);
}
```

dice *(**D**irect**I**nput **C**lass **E**ntity)* DirectInput-Wrapper

```
/*

	dice.hpp: DirectInput-wrapper dice 0.7 (using MS DirectInput)
	Copyright (C) 2002 Sebastian Boehm <s-boehm@users.sourceforge.net>


	This program is free software; you can redistribute it and/or
	modify it under the terms of the GNU General Public License
	as published by the Free Software Foundation; either
	version 2 of the License, or (at your option) any later
	version.

	This program is distributed in the hope that it will be useful,
	but WITHOUT ANY WARRANTY; without even the implied warranty of
	MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
	GNU General Public License for more details.

	You should have received a copy of the GNU General Public
	License along with this program; if not, write to the

		Free Software Foundation, Inc.,
		59 Temple Place, Suite 330,
		Boston, MA 02111-1307
		USA

  Make sure to add <dxguid.lib>, <dinput8.lib> and <dinput.h>
  to your project in order to be able to compile correctly!
  Check out http://snip.sourceforge.net for details!
*/

#ifndef DICE_HPP
#define DICE_HPP

#include <dinput.h>

class CDI
{
public:
	CDI(void);
	~CDI(void);
	friend class CDiceKeyboard;
	friend class CDiceMouse;
	HRESULT Init(HINSTANCE* lphInstance);
```

```
private:
	LPDIRECTINPUT8	lpDI;  // main di-object
};

class CDiceKeyboard
{
public:
	CDiceKeyboard(CDI* lpDirectInput);
	~CDiceKeyboard(void);
	HRESULT Init(HWND* lphWnd);
	bool Check(void);
	bool KeyDown(const int key);
	bool KeyUp(const int key);
private:
	CDI*   DInput;                    // associated CDI
	LPDIRECTINPUTDEVICE8 lpKeyboard; // keyboard-device-object
	unsigned char KeyState[256];     // key-buffer
};

class CDiceMouse
{
public:
	CDiceMouse(CDI* lpDirectInput);
	~CDiceMouse(void);
	HRESULT Init(HWND* lphWnd);
	bool Check(void);
	bool KeyDown(const unsigned short key);
	bool KeyUp(const unsigned short key);
	LONG getDeltaX(void) const;      // x-movement
	LONG getDeltaY(void) const;      // y-movement
	LONG getDeltaZ(void) const;      // mouse wheel movement
private:
	CDI*   DInput;                    // associated CDI
	LPDIRECTINPUTDEVICE8 lpMouse;    // mouse-device-object
	DIMOUSESTATE2MouseState;         // mouse-buffer
};

#endif
```

```
// dice.cpp: implementation of dice v0.7

// Read dice.hpp for details!

#include "dice.hpp"


////////////////////////////////////////
// implementation of CDI
////////////////////////////////////////

CDI::CDI(void)
: lpDI(0) {}

CDI::~CDI(void)

{
	if (lpDI)
	{
		lpDI->Release();
		lpDI = 0;
	}
}

HRESULT CDI::Init(HINSTANCE* lphInstance)
{
	HRESULT tmp;
	if (FAILED(tmp = (DirectInput8Create(*lphInstance,
```

```
                    DIRECTINPUT_VERSION, IID_IDirectInput8,
                    (void**)&lpDI, 0))))
                return tmp;

        return DI_OK;
}

//////////////////////////////////////
// implementation of CDiceKeyboard
//////////////////////////////////////

CDiceKeyboard::CDiceKeyboard(CDI* lpDirectInput)
: DInput(lpDirectInput), lpKeyboard(0) {}

CDiceKeyboard::~CDiceKeyboard(void)
{
        if (lpKeyboard)
        {
                lpKeyboard->Unacquire();
                lpKeyboard->Release();
                lpKeyboard = 0;
        }
}

HRESULT CDiceKeyboard::Init(HWND* lphWnd)
{
        HRESULT tmp;

        if (FAILED(tmp = (DInput->lpDI->CreateDevice(GUID_SysKeyboard,
                                                     &lpKeyboard, 0))))
                return tmp;
        if (FAILED(tmp = (lpKeyboard->SetDataFormat(&c_dfDIKeyboard))))
                return tmp;
        if (FAILED(tmp = (lpKeyboard->SetCooperativeLevel(*lphWnd,
                          DISCL_BACKGROUND | DISCL_NONEXCLUSIVE))))
                return tmp;
        if (FAILED(tmp = (lpKeyboard->Acquire())))
                return tmp;

        return DI_OK;
}

bool CDiceKeyboard::Check(void)
{
        if (!lpKeyboard)
                return false;
        if (FAILED(lpKeyboard->GetDeviceState(sizeof(KeyState),
                                              (void*)&KeyState)))
                return false;
        return true;
}

bool CDiceKeyboard::KeyDown(const int key)
{
        return (KeyState[key] & 0x80) ? true : false;
}

bool CDiceKeyboard::KeyUp(const int key)
{
        return (KeyState[key] & 0x80) ? false : true;
}

//////////////////////////////////////
// implementation of CDiceMouse
//////////////////////////////////////

CDiceMouse::CDiceMouse(CDI* lpDirectInput)
```

```
: DInput(lpDirectInput), lpMouse(0) {}

CDiceMouse::~CDiceMouse(void)
{
	if (lpMouse)
	{
		lpMouse->Unacquire();
		lpMouse->Release();
		lpMouse = 0;
	}
}

HRESULT CDiceMouse::Init(HWND* lphWnd)
{
	HRESULT tmp;

	if (FAILED(tmp = (DInput->lpDI->CreateDevice(GUID_SysMouse,
							&lpMouse, 0))))
		return tmp;

	if (FAILED(tmp = (lpMouse->SetDataFormat(&c_dfDIMouse2))))
		return tmp;

	if (FAILED(tmp = (lpMouse->SetCooperativeLevel(*lphWnd,
			DISCL_BACKGROUND | DISCL_NONEXCLUSIVE))))
		return tmp;

	if (FAILED(tmp = (lpMouse->Acquire())))
		return tmp;

	return DI_OK;
}

bool CDiceMouse::Check(void)
{
	if (!lpMouse)
		return false;

	if (FAILED(lpMouse->GetDeviceState(sizeof(MouseState),
					(void*)&MouseState)))
		return false;

	return true;
}

bool CDiceMouse::KeyDown(const unsigned short key)
{
	if (key > 7 || !lpMouse)
		return false;

	return MouseState.rgbButtons[key] & 0x80 ? true : false;
}

bool CDiceMouse::KeyUp(const unsigned short key)
{
	if (key > 7 || !lpMouse)
		return false;

	return MouseState.rgbButtons[key] & 0x80 ? false : true;
}

LONG CDiceMouse::getDeltaX(void) const
{
	if (!lpMouse)
		return 0;

	return MouseState.lX;
```

```
}

LONG CDiceMouse::getDeltaY(void) const
{
	if (!lpMouse)
		return 0;
	return MouseState.lY;
}

LONG CDiceMouse::getDeltaZ(void) const
{
	if (!lpMouse)
		return 0;
	return MouseState.lZ;
}
```

Anhang C: Einige in DirectInput verfügbare DIK-Codes[38]

Gerätekonstante	Taste
DIK_[0-9]	Ziffern auf der Haupttastatur
DIK_[A-Z]	Buchstaben
DIK_APOSTROPHE	Apostroph
DIK_APPS	Anwendungstaste (Kontextmenü-Taste)
DIK_BACK	Backspace
DIK_BACKSLASH	Backslash
DIK_CAPITAL	Caps Lock
DIK_COMMA	Komma
DIK_DECIMAL	Komma am Ziffernblock
DIK_DELETE	Entfernen/ Delete
DIK_DIVIDE	Querstrich/ Slash am Ziffernblock
DIK_DOWN	Pfeil abwärts
DIK_END	Ende/ End
DIK_EQUALS	Gleichzeichen auf der Haupttastatur
DIK_ESCAPE	Escape
DIK_F[1-12]	Funktionstasten
DIK_GRAVE	accent grave (`)
DIK_HOME	Pos1/ Home
DIK_INSERT	Einfg/ Insert
DIK_LBRACKET	eckige Klammer links ([)
DIK_LCONTROL	linke STRG/ CTRL-Taste
DIK_LEFT	Pfeil links
DIK_LMENU	linke ALT-Taste
DIK_LSHIFT	linke Shift-Taste
DIK_LWIN	linke Windows-Taste
DIK_MINUS	Minus auf der Haupttastatur
DIK_MULTIPLY	[*] am Ziffernblock
DIK_NEXT	Bild ab/ Page down
DIK_NUMLOCK	Numlock
DIK_NUMPAD[0-9]	Ziffern am Ziffernblock
DIK_NUMPADENTER	Enter am Ziffernblock
DIK_PAUSE	Pause
DIK_PERIOD	Punkt
DIK_PRIOR	Bild auf/ Page up
DIK_RBRACKET	eckige Klammer rechts (])
DIK_RCONTROL	rechte STRG/ CTRL-Taste
DIK_RETURN	Enter auf der Haupttastatur
DIK_RIGHT	Pfeil rechts
DIK_RMENU	rechte ALT-Taste
DIK_RSHIFT	rechte Shift-Taste
DIK_RWIN	rechte Windows-Taste
DIK_SCROLL	Rollen/ Scroll Lock
DIK_SEMICOLON	Strichpunkt (Semikolon)
DIK_SLASH	Querstrich/ Slash auf der Haupttastatur
DIK_SPACE	Leertaste
DIK_SUBTRACT	Minus am Ziffernblock
DIK_TAB	Tabulator-Taste
DIK_UP	Pfeil aufwärts

[38] Tatsächlich gibt es mehr Konstanten als hier aufgelistet sind, aber da diese teilweise Landes- und/ oder Tastatur-spezifisch sind, ist es guter Programmierstil, sich auf die angeführten zu beschränken.

Anhang D: Literaturverzeichnis

Microsoft Corporation: *DirectX 8.0 Programmer's Reference*.
USA: 2002

LAMOTHE, André: *Tricks of the Windows Game Programming Gurus – Fundamentals of 2D and 3D Game Programming*.
Indianapolis, Indiana: Sams, 1999

STROUSTRUP, Bjarne: *The C++ Programming Language, Special Edition*.
Boston, Massachusetts: Addison-Wesley, 2000

Microsoft Corporation (2001-2002): MSDN-Seite zu DirectX.
URL: http://msdn.microsoft.com/directx [Stand: 4.2.2002]

Game Developer's Network: Homepage des Game Developer's Network.
URL: http://www.gamedev.net [Stand: 4.2.2002]

Marcus Bäckmann: *Die C/C++ Ecke.*
URL: http://www.c-plusplus.de [Stand: 4.2.2002