NeHE Tutorials
(http://nehe.gamedev.net)

Übersetzung der NeHE Tutorials in das Deutsche von: Joachim Rohde
(http://www.joachimrohde.com)

Archivierung in diesem E-Book von: Hager Harald
(http://dim.harrys-net.at) (http://www.harrys-net.at)

Inhaltsverzeichnis

Inhaltsverzeichnis 2
Lektion 01 - Ein OpenGL-Fenster 3
Lektion 02 - Ihr erstes Polygon 25
Lektion 03 - Farbe hinzufügen 29
Lektion 04 - Rotation 33
Lektion 05 - 3D Objekte 39
Lektion 06 - Textur Mapping 45
Lektion 07 - Texturfilter, Beleuchtung und Tastatureingaben 54

Lektion 01 - Ein OpenGL-Fenster

Willkommen bei meinen OpenGL Tutorials. Ich bin ein durchschnittlicher Typ mit einer Passion für OpenGL! Das erste Mal, dass ich was über OpenGL gehört habe, war, als 3Dfx ihren Hardwarebeschleunigten OpenGL Treiber für die Voodoo 1 Karte veröffentlichte. Ich wusste sofort, dass OpenGL etwas war, dass ich lernen musste. Unglücklicherweise war es recht hart, irgendwelche Informationen über OpenGL zu finden, weder in Büchern noch im Netz. Ich verbrachte Stunden damit, Code zum laufen zu kriegen und noch mehr Zeit damit, Leute um Hilfe per E-Mail oder im IRC anzubetteln. Ich stellte fest, dass die Leute, die OpenGL verstanden, sich selbst als Elite betrachteten und kein Interesse hatten, ihr Wissen zu teilen. SEHR frustrierend!

Ich habe diese Webseite erstellt damit Leute die OpenGL lernen wollen einen Ort haben, wo sie hinkommen können, wenn sie Hilfe benötigen. In jedem meiner Tutorials versuche ich so detailiert wie menschlich möglich zu erklären, was jede CodeZeile macht. Ich versuche meinen Code simpel zu halten (kein MFC zu lernen)! Ein absoluter Neuling sowohl in Visual C++ als auch OpenGL sollte fähig sein, durch den Code zu gehen und eine ziemlich genaue Vorstellung zu bekommen, was da vor geht. Meine Seite ist nur eine von vielen Seiten, die OpenGL Tutorials anbieten. Wenn Sie ein Hardcore-OpenGL-Programmierer sind, mag meine Seite zu einfach sein, aber wenn Sie gerade erst anfangen, denke ich, dass meine Seite recht viel zu bieten hat!

Dieses Tutorial wurde im Januar 2000 komplett neu geschrieben. Dieses Tutorial wird Ihnen beibringen, wir man ein OpenGL-Fenster initialisiert. Das Fenster kann im Fenster-Modus oder als Fullscreen laufen, jede Größe die Sie wollen, jeder Auflösung die Sie wollen und jede Farbtiefe die Sie wollen. Der Code ist sehr flexibel und kann für all Ihre OpenGL Projekte benutzt werden. Alle meine Tutorials werden auf diesem Code basieren! Ich habe den Code geschrieben, so dass er felxibel und mächtig zur selben Zeit ist. Alle Fehler werden berichtet. Es sollten keine Speicherlecks existieren und der Code ist einfach zu lesen und einfach zu modifizieren. Danke an Frederic Echols für die Modifikationen an dem Code!

Ich fange mit diesem Tutorial an, indem wir direkt mit dem Code anfangen. Das erste was wir machen müssen, ist ein Projekt in Visual C++ zu erstellen. Wenn Sie nicht wissen, wie Sie das machen müssen, sollten Sie nicht mit OpenGL anfangen, sondern erst einmal Visual C++ lernen. Der Code zum herunterladen ist Visual C++ 6.0 Code. Bei einigen Versionen von VC++ muss bool zu BOOL geändert werden, true in TRUE und false in FALSE. Mit den erwähnten Änderungen konnte ich den Code ohne Probleme auch unter Visual C++ 4.0 und 5.0 kompilieren.

Nachdem Sie eine neue Win32 Applikation (NICHT eine Konsolen-Applikation) in Visual C++ erstellt haben, müssen Sie die OpenGL Librariers hinzu linken. In Visual C++ gehen Sie auf Projekt / Einstellungen und klicken dann auf den LINK Reiter. Unter "Objekt/Bibliotheks Module" am Anfang der Zeile (noch vor kernel32.lib) fügen Sie OpenGL32.lib, GLu32.lib und GLaux.lib ein. Wenn Sie das getan haben, klicken Sie auf OK. Nun sind Sie bereit, ein OpenGL Windows-Programm zu schreiben.

Die ersten 4 Zeilen inkludieren die Header-Dateien für die jeweiligen Libraries, die wir benutzen. Die Zeilen sehen wie folgt aus:

```
#include < windows.h>        // Header Datei für Windows
#include < gl\gl.h>          // Header Datei für die OpenGL32 Library
#include < gl\glu.h>         // Header Datei für die GLu32 Library
#include < gl\glaux.h>       // Header Datei für die GLaux Library
```

Als nächstes müssen Sie alle Variablen deklarieren, die Sie in Ihrem Programm benutzen wollen. Dieses Programm wird ein leeres OpenGL Fenster erzeugen, weshalbt wir noch nicht allzu viele Variable deklarieren müssen. Die paar Variablen die wir deklarieren sind sehr wichtig und werden in jedem OpenGL Programm, welches Sie mit Hilfe dieses Codes schreiben, benutzt werden.

Die erste Zeile ist für unseren Rendering Context. Jedes OpenGL Programm ist mit einem Rendering Context verbunden. Ein Rendering Context ist das, was die OpenGL Aufrufe mit unserem Device Context (=Geräte-Kontext) verbindet. Der OpenGL Rendering Context ist als hRC definiert. Um mit Ihrem Programm in einem Fenster zu zeichnen, müssen Sie einen Device Context erstellen, was in der zweiten Zeile gemacht wird. Der Fenster Device Context ist als hDC definiert. Der DC verbindet das Fenster mit der GDI (Graphics Device Interface). Der RC verbindet OpenGL mit dem DC.

In der dritten Zeilen ist die Variable hWnd welche das Handle, das unserem Fenster von Fenster zugeordnet wird, enthält und letztendlich ist in der vierten Zeile unsere Instanz für unser Programm.

```
HGLRC hRC=NULL;           // Permanenter Rendering Context
HDC hDC=NULL;             // Privater GDI Device Context
HWND hWnd=NULL;           // Enthält unser Fenster-Handle
HINSTANCE hInstance;      // Enthält die Instanz der Applikation
```

Die folgende erste Zeile definiert ein Array, das wird benutzen werden, um Tastenanschläge der Tastatur zu überwachen. Es gibt viele Wege zu überprüfen, ob eine Taste gedrückt wurde, aber ich mache es auf diesen Weg. Es ist verlässlich und außerdem kann man so mehr als eine Taste gleichzeitig überprüfen.

Die Variable active wird benutzt, um unser Programm mitzuteilen, ob unser Fenster in die Taskbar minimiert wurde oder nicht. Wenn das Fenster minimiert wurde, können wir alles machen, vom still legen bis zum beenden des Programmes. I lege das Programm still. Auf diese Art läuft es nicht im Hintergrund weiter, wenn es minimiert ist.

Die Variable fullscreen ist ziemlich selbsterklärend. Wenn unser Programm im Fullscreen-Modus läuft, wird fullscreen auf TRUE gesetzt, wenn unser Programm im Fenster-Modus läuft, wird fullscreen auf FALSE gesetzt. Es ist wichtig diese Variable global zu machen, so dass jede Prozedur weiß, ob das Programm im Fullscreen-Modus läuft oder nicht.

```
bool keys[256];          // Array das für die Tastatur Routine verwendet wird bool
bool active=TRUE;        // Fenster Aktiv Flag standardmäßig auf TRUE gesetzt
bool fullscreen=TRUE;    // Fullscreen Flag ist standardmäßig auf TRUE gesetzt
```

Nun müssen wir WndProc() definieren. Der Grund dafür ist, dass CreateGLWindow() WndProc() benutzt, aber WndProc() erst nach CreateGLWindow() kommt. In C müssen wir, wenn wir Zugriff auf eine Prozedur oder eine Code-Sektion haben wollen, die nach der Code-Sektion kommt, in der wir uns gerade befinden, die Code-Sektion auf die wir zugreifen wollen, am Anfang unseres Programms deklarieren. Deshalb definieren wir WndProc() mit der folgenden Zeile, so dass CreateGLWindow() gebrauch von WndProc() machen kann.

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);          // Deklaration für WndProc
```

Die Aufgabe des nächsten Code-Abschnitts ist es, die Größe der OpenGL Szene neu zu setzen, wann immer sich das Fenster (angenommen, Sie verwenden ein Fenster statt des Fullscreen-Modus) in seiner Größe ändert. Selbst wenn Sie die Größe des Fensters nicht änderen können (wenn Sie sich zum Beispiel im Fullscreen-Modus befinden), wird diese Routine zumindest einmal am Anfang des Programms aufgerufen, wenn unsere Perspektiven-Ansicht das erste Mal gesetzt wird. Die Größe der OpenGL Szene wird basierend auf der Breite und Höhe des Fensters angezeigt, indem die Szene angezeigt wird.

```
// verändert die Größe und initialisiert das GL-Fenster
//
GLvoid ReSizeGLScene(GLsizei width, GLsizei height)
{
        if (height==0)                  // Verhindere eine Division durch 0, indem
        {
                height=1;               // die Höhe auf 1 gesetzt wird
        }

        glViewport(0, 0, width, height); // Resette die aktuelle Ansicht (Viewport)
```

Die folgenden Zeilen intialisieren den Screen für unsere Perspektive. Das bedeutet, dass Dinge in der Ferne kleiner werden. Das erzeugt ein realistisches Aussehen der Szene. Die Perspektive wird mit einem 45 Grad Sichtwinkel, basierend auf der Fenster Breite und Höhe berechnet. Die 0.1f, 100.0f ist der Start- und Endpunkt für die Tiefe der Szene, wie weit wir in den Bildschirm hinein zeichnen können.

glMatrixMode(GL_PROJECTION) indiziert, dass die nächsten beiden 2 Zeilen Code die Projektionsmatrix beeinflussen. Die Projektionsmatrix is verantwortlich für eine zusätzliche Perspektive unserer Szene. glLoadIdentity() gleicht einem Reset. Der Befehl stellt den original Status der ausgewählte Matrix wieder her. Nachdem glLoadIdentity() aufgerufen wurde, setzen wir unsere Perspektive für unsere Szene. glMatrixMode(GL_MODELVIEW) indiziert, dass jede Transformation die Modelview-Matrix beeinflusst. Die Modelview-Matrix enthält Informationen über unser Objekt. Als letztes resetten wir die Modelview-Matrix. Machen Sie sich nichts draus, wenn Sie das nicht verstanden haben, ich werde es in späteren Tutorials weiter erklären. Nur damit man weiß, dass es gemacht werden MUSS, wenn Sie eine schöne perspektivische Szene haben wollen.

```
        glMatrixMode(GL_PROJECTION);                    // wähle die Projektions-Matrix aus
        glLoadIdentity();                               // Resette die Projektions-Matrix

        // berechne das Seitenverhältnis des Fensters
        //
        gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100.0f);
        glMatrixMode(GL_MODELVIEW);                     // Wähle die Modelview Matrix aus
        glLoadIdentity();                               // Resette die Modelview Matrix
}
```

Im nächsten Code-Abschnitt machen wir alles für die Initialisierung für OpenGL. Wir geben an, zu welcher Farbe der Screen gelöscht werden soll, wir schalten den Depth-Buffer an, aktivieren weiches Shading (smooth shading), etc. Diese Routine wird nicht aufgerufen, bis das OpenGL-Fenster erzeugt wurde. Diese Prozedur liefert zwar einen Wert zurück, da unsere Initialisierung nicht so komplex ist, kümmern wir uns um diesen Wert erst mal nicht.

```
int InitGL(GLvoid)                          // Der ganze Setup Kram für OpenGL kommt hier rein
{
```

Die nächste Zeile aktiviert Smooth Shading. Smooth Shading blendet Farben sehr schön über ein Polygon und erzeugt auch 'weiches' Licht. Ich werde Smooth Shading in einem anderen Tutorial näher erklären.

```
        glShadeModel(GL_SMOOTH);            // aktiviert weiches Shading (Smooth Shading)
```

Die folgende Zeile setzt die Farbe des Screens, wenn er gelöscht wird. Wenn Sie nicht wissen, wie Farben funktionieren, erkläre ich es hier kurz. Die Farbwerte bewegen sich zwischen 0.0f und 1.0f. 0.0f ist das Dunkelste und 1.0f das Hellste. Der erste Parameter von glClearColor ist die Rot-Intensität, der zweite Parameter ist für Grün und der dritte für Blau. Je höher die Zahl ist, desto heller wird die entsprechende Farbe sein. Die letzte Zahl ist der Alpha-Wert. Wenn wir den Bildschirm löschen, müssen wir uns nicht um den 4ten Parameter kümmern. Wir lassen ihn erstmal auf 0.0f. Ich werde seinen Gebrauch in einem anderen Tutorial beschreiben.

Sie erzeugen verschiedene Farben, indem Sie die drei Gruindfarben mischen (rot, grün, blau). Ich hoffe, Sie haben die Grundfarben in der Schule gelernt. So, wenn Sie jetzt glClearColor(0.0f,0.0f,1.0f,0.0f) hätten, würden Sie den Bildschirm auf ein helles Blau löschen. Wenn Sie glClearColor(0.5f,0.0f,0.0f,0.0f) hätten, würden Sie auf ein mittleres Rot löschen. Nicht hell (1.0f) und auch nicht dunkel (0.0f). Um einen weißen Hintergrund zu erhalten, würden Sie alle Farben auf den höchst möglichen Wert setzen (1.0f). Um einen schwarzen Hintergrund zu erhalten, würden Sie alle Farben auf den niedrigst möglichen Wert setzen (0.0f).

```
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);       // Schwarzer Hintergrund
```

Die nächsten drei Zeilen haben etwas mit dem Depth Buffer zu tun. Stellen Sie sich den Depth Buffer als Schichten im Bildschirm vor. Der Depth Buffer gibt an, wie weit das Objekt in den Bildschirm 'hinein' kann. Wir werden den Depth Buffer in diesem Programm nicht wirklich gebrauchen, aber jedes OpenGL Programm, dass etwas in 3D auf den Schirm bringt, benutzt den Depth Buffer. Er sortiert aus, welche Objekte als erstes gezeichnet werden, so dass ein Quadrat, dass Sie hinter einem Kreis zeichnen wollen, nicht vor dem Kreis erscheint. Der Depth Buffer ist ein sehr wichtiger Teil von OpenGL.

```
glClearDepth(1.0f);                     // Depth Buffer Setup
glEnable(GL_DEPTH_TEST);                // aktiviert Depth Test
glDepthFunc(GL_LEQUAL);                 // Die Art des auszuführenden Depth Test
```

Als nächstes teilen wir OpenGL mit, dass wir die beste Perspektiven Korrektur wünschen, die möglich ist. Dadurch erleiden wir zwar einen winzigen Performanceverlust, erhalten aber eine besser aussehende Perspektive.

```
// wirklich nette Perspektiven Berechnungen
//
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
```

Letztendlich geben wir TRUE zurück. Wenn wir überprüfen wollen, ob die Initialisierung fehlerfrei war, können wir überprüfen ob TRUE oder FALSE zurückgegeben wurde. Sie können Ihren eigenen Code einfügen und FALSE zurückgeben, wenn ein Fehler auftritt. Zur Zeit müssen wir uns darüber aber keine Gedanken machen.

```
return TRUE;                                    // Initialisierung war OK
```

In diesem Abschnitt geschieht unser ganzer Zeichnen-Code. Alles was Sie planen, auf dem Bildschirm anzuzeigen, wird in diesem Abschnitt des Codes untergebracht. Jedes Tutorial nach diesem, wird etwas Code in diesem Abschnitt des Programms hinzufügen. Wenn Sie bereits eine Vorstellung von OpenGL haben, können Sie versuchen, einfache Figuren zu erzeugen, indem Sie zwischen glLoadIdentity() und vor return FALSE OpenGL-Code einfügen. Wenn Ihnen OpenGL neu ist, warten Sie bis zum nächsten Tutorial. Bisher haben wir die Bildschirm auf die Farbe gelöscht, die wir vorher ausgewählt haben, löschen den Depth Buffer und resetten die Szene. Wir werden noch nichts zeichnen.

Das return TRUE teilt unserem Programm mit, dass keine Probleme aufgetreten sind. Wenn Sie das Programm aus irgendwelchen Gründen stoppen wolle, fügen Sie eine return FALSE Zeile irgendwo vor return TRUE ein, um den Programm mitzuteilen, dass der Zeichnen-Code fehlgeschlagen ist. Das Programm wird dann beendet.

```
int DrawGLScene(GLvoid) // Hier kommt der ganze Zeichnen-Kram hin
{
        // Lösche den Bildschirm und den Depth-Buffer
        //
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();                                   // Resettet die aktuelle Modelview Matrix
        return TRUE;                                        // Alles war OK
}
```

Der nächste Code-Abschnitt wird aufgerufen, kurz bevor das Programm beendet wird. Die Aufgabe von KillGLWindow() ist es, den Rendering Kontext, den Device Kontext und letzlich das Fenster-Handle freizugeben. Ich habe eine Menge Fehlerüberprüfungen eingefügt. Wenn das Programm irgendeinen Teil des Fensters nicht zerstören konnte, wird eine MessageBox mit einer Fehlernachricht angezeigt, welche Ihnen mitteilt, was fehlgeschlagen ist. Das macht es wesentlich einfacher Probleme in Ihrem Code zu finden.

```
GLvoid KillGLWindow(GLvoid)                             // Entferne das Fenster korrekt
{
```

Als erstes was wir in KillGLWindow() überprüfen sollten, ist, ob wir uns im Fullscreen-Modus befinden. Wenn wir es sind, wechseln wir zurück zum Desktop. Wir sollten das Fenster zerstören, bevor wir den Fullscreen-Modus deaktivieren, allerdings ist es auf einigen Grafikkarten der Fall, dass wenn wir das Fenster zerstören BEVOR wir den Fullscreen-Modus deaktivieren, der Desktop nicht richtig angezeigt wird. Deshalb deaktivieren wir erst den Fullscreen-Modus. Das verhindert, dass der Desktop korrupt dargestellt wird und arbeitet sowohl für Nvidia als auch 3dfx Grafikkarten!

```
        if (fullscreen)             // Sind wir im Fullscreen Modus?
        {
```

Wir benutzen ChangeDisplaySettings(NULL,0) um uns den original Desktop zurückgeben zu lassen. Indem wir NULL als ersten Parameter und 0 als zweiten Parameter übergeben, zwingen wir Windows dazu, die aktuell in der Registry gespeicherten Werte (die Standard Auflösung, Bit-Tiefe, Frequenz, etc.) zu benutzen, um den original Desktop wieder herzustellen. Nachdem wir zurück zum Desktop gewechselt haben, machen wir den Cursor wieder sichtbar.

```
                ChangeDisplaySettings(NULL,0);          // Wenn ja, wechsle zurück zum Desktop
                ShowCursor(TRUE);                       // Zeige den Maus-Zeiger
        }
```

Der folgende Code überprüft, ob wir einen Rendering Context (hRC) haben. Falls nicht, springt das Programm in den Code-Abschnitt, wo überprüft wird, ob wir einen Device Context haben.

```
        if (hRC)            // Haben wir einen Rendering Context?
        {
```

Wenn wir einen Rendering Context haben, überprüft der folgende Code, ob wir ihn freigeben können (den hRC vom hDC lösen können). Beachten Sie den Weg, wie ich auf Fehler überprüfe. Grundsätzlich teile ich unserem Programm mit, dass es versuchen soll ihn freizugeben (mit wglMakeCurrent(NULL,NULL)), dann überprüfe ich, ob das freigeben erfolgreich war oder nicht. Somit werden einige Zeilen zu einer Codezeile zusammengefasst.

```
        if (!wglMakeCurrent(NULL,NULL))    // Können wir den DC und RC Kontext freigeben?
        {
```

Wenn wir die DC und RC Kontexte nicht freigeben konnten, wird eine MessageBox angezeigt, die uns wissen lässt, dass DC und RC nicht freigegeben werden konnten. NULL bedeutet hier, dass die MessageBox kein Parent-Fenster besitzt. Der Text direkt nach NULL, ist der Text, der in der MessageBox erscheint. "SHUTDOWN ERROR" ist der Text, der über der MessageBox erscheint (im Titel). Als nächstes haben wir MB_OK, was bedeutet, dass die MessageBox einen Button namens "OK" haben soll. MB_ICONINFORMATION lässt ein kleines i in einem Kreis innerhalb der Messagebox erscheinen (um es auffälliger zu machen).

```
            MessageBox(NULL,"Release Of DC And RC Failed.","SHUTDOWN
            ERROR",MB_OK | MB_ICONINFORMATION);
        }
```

Als nächstes versuchen wir den Rendering Kontext zu löschen. Wenn wir damit nicht erfolgreich sind, geben wir eine Fehlermeldung aus.

```
        if (!wglDeleteContext(hRC))            // Können wir den RC löschen?
        {
```

Wenn wir den Rendering Context nicht löschen konnten, wird der folgende Code eine Message-Box anzeigen, die uns wissen lässt, dass das Löschen des RC nicht erfolgreich war. hRC wird auf NULL gesetzt.

```
            MessageBox(NULL,"Release Rendering Context Failed.","SHUTDOWN
            ERROR",MB_OK | MB_ICONINFORMATION);
        }

        hRC=NULL;                          // Setze RC auf NULL
    }
```

Nun überprüfen wir, ob unser Programm ein Device Context hat, und falls ja, versuchen wir diesen freizugeben. Wenn wir ihn nicht freigeben können, wird eine Fehlermeldung angezeigt und hDC auf NULL gesetzt.

```
        if (hDC && !ReleaseDC(hWnd,hDC))                    // Können wir DC freigeben?
        {
                MessageBox(NULL,"Release Device Context Failed.","SHUTDOWN ERROR",MB_OK |
                MB_ICONINFORMATION);
                hDC=NULL;                                   // Setze DC auf NULL
        }
```

Als letztes de-registrieren wir unsere Fenster-Klasse. Das erlaubt es uns, dass Fenster ordentlich zu kellen und dann ein anderes Fenster zu öffnen, ohne die Fehlernachricht "Fenster-Klasse ist bereits registriert" zu erhalten.

```
        if (!UnregisterClass("OpenGL",hInstance))       // Können wir die Klasse de-registrieren?
        {
                MessageBox(NULL,"Could Not Unregister Class.","SHUTDOWN ERROR",MB_OK |
                MB_ICONINFORMATION);
                 hInstance=NULL;                        // Setze hInstance auf NULL
        }
}
```

Der nächste Code-Abschnitt erzeugt unser OpenGL-Fenster. Ich habe ziemlich lange gebraucht, um zu entscheiden, ob ich ein fixes Fullscreen-Fenster erzeugen soll, welches nicht viel extra Code benötigt hätte oder ein einfaches anzuwendendes, benutzerfreundliches Fenster, dass mehr Code benötigt. Ich habe mich für das benutzerfreundliche Fenster mit viel mehr Code entschieden, weil ich es für die beste Wahl hielt. Ich wurde die folgende Fragen immer wieder per E-Mail gefragt: Wie kann ich ein Fenster erzeugen, anstatt Fullscreen zu benutzen? Wie ändere ich den Fenster-Titel? Wie ändere ich die Auflösung oder das Pixel Format des Fensters? Der folgende Code macht all das! Demnach ist er besser zum lernen geeignet und wird Ihnen das schreiben eigener OpenGL Programm um ein vielfaches erleichtern!

Wie Sie sehen können, gibt die Prozedur BOOL zurück (TRUE oder FALSE) und übernimmt 5 Parameter: Titel des Fensters, Breite des Fensters, Höhe des Fensters, Bits (16/24/32) und letzlich fullscrrenflag gleich TRUE für Fullscreen oder FALSE für Fenster-Modus. Wir geben einen bolleschen Wert zurück, der uns mitteilt, ob das Fenster erfolgreich erzeugt wurde.

```
BOOL CreateGLWindow(char* title, int width, int height, int bits, bool fullscreenflag)
{
```

Wenn wir Windows auffordern uns ein passendes Pixel Format zu finden, wird die Nummer des Modus, mit dem Windows dann am Ende aufkommt, in der Variable PixelFormat gespeichert.

```
        GLuint PixelFormat;  // Enthält die Ergebnisse nachdem nach was passendem gesucht wurde
```

wc wird benutzt, um unsere Fenster-Klassen-Struktur aufzunehmen. Die Fenster-Klassen-Struktur enthält Informationen über unser Fenster. Indem wir verschiedene Felder in der Klasse ändern, können wir das Aussehen und Verhalten des Fensters ändern. Jedes Fenster gehört zu einer Fenster-Klasse. Bevor Sie ein Fenster erzeugen, MÜSSEN Sie eine Klasse für das Fenster registrieren.

```
WNDCLASS wc;                    // Fenster Klassen Struktur
```

dwExStyle und dwStyle nehmen die erweiterten und normalen Fenster-Stil-Informationen auf. Ich benutze Variablen um die Stile zu speichern, so dass ich die Stile abhängig von der Art des Fensters, dass ich erzeugen will, ändern kann (ein Popup Fenster für Fullscreen oder ein Fenster mit Rahmen für den Fenster-Modus).

```
DWORD dwExStyle;                // erweiterter Fenster-Stil
DWORD dwStyle;                  // Fenster-Stil
```

Die folgenden 5 Zeilen Code ermitteln die obere linke und untere rechte Koordinaten eines Rechtecks. Wir werden diese Werte verwenden, um unser Fenster so zu justieren, dass die Fläche auf die wir zeichnen exakt der Auflösung ist, die wir wünschen. Normalerweise, wenn wir ein 640x480 Fenster erzeugen würden, würden die Kanten unseres Fensters etwas von dieser Fläche beanspruchen.

```
RECT WindowRect;      // Enthält die obere linke / untere rechte Eckwerte des Rechtecks

WindowRect.left=(long)0;            // Setze linken Wert auf 0
WindowRect.right=(long)width;       // Setze rechten Wert auf die gewünschte Breite
WindowRect.top=(long)0;             // Setze den oberen Wert auf 0
WindowRect.bottom=(long)height;     // Setze unteren Wert auf die gewünschte Höhe
```

In der nächsten Codezeile setzen wir die globale Variable Fullscreen gleich fullscreenflag.

```
fullscreen=fullscreenflag;      // Setze das globale Fullscreen Flag
```

Im nächsten Code-Abschnitt ermitteln wir eine Instanz für unser Fenster, dann defnieren wir die Fenster-Klasse.

Die Stile CS_HREDRAW und CS_VREDRAW zwingt das Fenster neu zu zeichnen, wenn dessen Größe geändert wird. CS_OWNDC erzeugt ein privaten DC für das Fenster. Das bedeutet, dass der DC nicht zwischen anderen Applikationen geteilt wird. WndProc ist die Prozedur, welche nach Nachrichten in unserem Programm schaut. Es werden keine extra Fenster-Daten benutzt, so dass wir die beiden Felder auf 0 setzen. Dann setzen wir die Instanz. Als nächstes setzen wir das hIcon auf NULL, was bedeutet, dass wir kein Icon im Fenster verwenden möchten und für den Mauszeiger verwenden wir den Standard-Pfeil. Die Hintergrundfarbe ist egal (wir setzen diese in GL). Wir wollen kein Menü in diesem Fenster, deswegen setzen wir es auf NULL und der Klassen-Name kann irgend ein von Ihnen gewählter Name sein. Ich benutze "OpenGL" wegen seiner Einfachheit.

```
        hInstance = GetModuleHandle(NULL); // Ermittle die Instanz für unser Fenster

        // Zeichne neu beim bewegen und eigener DC für's Fenster
        wc.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
        wc.lpfnWndProc = (WNDPROC) WndProc;         // WndProc behandelt die Nachrichten
        wc.cbClsExtra = 0;                          // Keine extra Fenster-Daten
        wc.cbWndExtra = 0;                          // Keine extra Fenster-Daten
        wc.hInstance = hInstance;                   // Setze die Instanz
        wc.hIcon = LoadIcon(NULL, IDI_WINLOGO);     // Lade das Standard-Icon
        wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Lade den Pfeil-Zeiger
        wc.hbrBackground = NULL;                    // es wird kein Hintergrund für GL benötigt
        wc.lpszMenuName = NULL;                     // Wir wollen kein Menü
        wc.lpszClassName = "OpenGL";                // Setze den Klassen-Namen
```

Nun registrieren wir die Klasse. Wenn etwas schief geht, wird eine Fehler-Nachricht angezeigt. Ein Klick auf OK der Fehler-Meldung beendet das Programm.

```
        if (!RegisterClass(&wc))                    // Versuche die Fenster-Klasse zu registrieren
        {
                MessageBox(NULL,"Failed To Register The Window Class.",
                "ERROR",MB_OK|MB_ICONEXCLAMATION);

                return FALSE;                       // beende und gebe FALSE zurück
        }
```

Nun überprüfen wir, ob das Programm im Fullscrenn-Modus oder im Fenster-Modus laufen soll. Wenn es im Fullscreen-Modus laufen soll, versuchen wir, den Fullscreen-Modus zu setzen.

```
        if (fullscreen)    // Fullscreen Modus?
        {
```

Der nächste Codeabschnitt ist etwas, wo die Leute scheinbar Probleme mit haben... in den Fullscreen-Modus zu wechseln. Es gibt einige wichtige Dinge, die Sie im Hinterkopf behalten sollten, wenn Sie in den Fullscreen-Modus wechseln. Stellen Sie sicher, dass Breite und Höhe, welche Sie im Fullscreen-Modus verwenden, die selbe Breite und Höhe ist, die Sie für ihr Fenster verwenden wollen und am wichtigsten: setzen Sie den Fullscreen-Modus BEVOR Sie ihr Fenster erzeugen. In diesem Code müssen Sie sich nicht um Breite und Höhe kümmern, da Fullscreen und Fenster-Größe jeweils auf die erforderliche Größe gesetzt sind.

```
        if (fullscreen)    // Fullscreen Modus?
        {
            DEVMODE dmScreenSettings;                           // Device Modus
            // Stelle sicher, dass der Speicher geleert ist
            memset(&dmScreenSettings,0,sizeof(dmScreenSettings));
            dmScreenSettings.dmSize=sizeof(dmScreenSettings); // Größe der Devmode Struktur
            dmScreenSettings.dmPelsWidth = width;               // ausgewählte Screen Breite
            dmScreenSettings.dmPelsHeight = height;             // ausgewählte Screen Höhe
            dmScreenSettings.dmBitsPerPel = bits;               // ausgewählte Bits Per Pixel
            dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
```

Im oberen Code machen wir Platz um unsere Video-Einstellungen zu speichern. Wir setzen die Breite, Höhe und Bits, zu der der Bildschirm wechseln soll. Im unteren Code versuchen wir den angeforderten

Fullscreen-Modus zu setzen. Wir speichern alle Informationen über Breite, Höhe und Bits in dmScreenSettings. In der Zeile die auf ChangeDisplaySettings folgt, versuchen wir in einen Modus zu wechseln, der dem entspricht, welcher in dmScreenSettings gespeichert ist. Ich benutze den Parameter CDS_FULLSCREEN, wenn ich in einen anderen Modus wechsle, weil dann die Start-Leiste entfernt werden soll und die Fenster auf dem Desktop werden weder in ihrer Größe noch in ihrer Position verändert, wenn Sie in den Fullscreen-Modus und zurück wechseln.

```
// Versuche gewählten Modus zu setzen und Ergebnisse zurückliefern.
// ANMERKUNG: CDS_FULLSCREEN lässt die Start-Leiste nicht anzeigen.
if(ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=
    DISP_CHANGE_SUCCES SFUL)
 {
```

Wenn der Modus nicht gesetzt werden konnte, wird der untere Code ausgeführt. Wenn kein passender Fullscreen-Modus existiert, wird eine Message-Box angezeigt und bietet zwei Optionen an... Die Option, im Fenster-Modus zu laufen oder die Option zum Beenden.

```
// Wenn der Modus fehl schlägt, biete zwei Optionen an.
// Beenden oder im Fenster-Modus laufen lassen.
if (MessageBox(NULL,"The Requested Fullscreen Mode Is Not Supported By\nYour
Video Card. Use Windowed Mode Instead?","NeHe GL",
MB_YESNO|MB_ICONEXCLAMATION)==IDYES)
{
```

Wenn der Benutzer sich entscheided den Fenster-Modus benutzen will, wird die Variable fullscreen auf FALSE gesetzt und das Programm läuft weiter.

```
                    fullscreen=FALSE;             // wähle Fenster-Modus aus
                    (fullscreen=FALSE);
                }
                else
                {
```

Wenn sich der Benutzer entscheided zu beenden, wird eine MessageBox angezeigt, die mitteilt, dass das Programm dabei ist, geschlossen zu werden. Es wird FALSE zurückgegeben, um dem Programm mitzuteilen, dass das Fenster nicht erfolgreich erzeugt wurde. Das Programm wird beendet.

```
                    // Message Box anzeigen, die den Benutzer wissen lässt,
                    dass das Programm geschlossen wird.
                    MessageBox(NULL,"Program Will Now Close.",
                    "ERROR",MB_OK|MB_ICONSTOP);

                    return FALSE;          // beende und gebe FALSE zurück
                }
            }
        }
```

Weil der obere Fullscreen Code vielleicht fehlgeschlagen ist und der Benutzer entschieden hat, dass Programm statt dessen im Fenster-Modus laufen zu lassen, überprüfen wir erneut, ob Fullscreen gleich TRUE oder FALSE ist, bevor wir den Screen / die Fenster-Art neu initialisieren.

```
		if (fullscreen)		// Sind wir immer noch im Fullscreen Modus?
		{
```

Wenn wir immer noch im Fullscreen-Modus sind, setzen wir den erweiterten Stil auf WS_EX_APPWINDOW, was ein Top-Level-Fenster in die Taskbar verbannt, sobald unser Fenster sichtbar ist. Für den Fenster-Stil erzeugen wir ein WS_POPUP Fenster. Diese Art von Fenster hat keinen Rahmen, was es perfekt für den Fullscreen-Modus macht.

Zu letzt deaktivieren wir den Mauszeiger. Wenn ihr Programm nicht interaktiv ist, ist es in der Regel netter, den Mauszeiger zu verstecken, wenn man sich im Fullscreen-Modus befindet. Das liegt aber an Ihnen.

```
		if (fullscreen)		// Sind wir immer noch im Fullscreen Modus?
		{
			dwExStyle=WS_EX_APPWINDOW;	// erweiterter Fenster-Stil
			dwStyle=WS_POPUP;		// Fenster-Stil
			ShowCursor(FALSE);		// verstecke Maus-Zeiger
		}
		else
		{
```

Wenn wir statt eines Fullscreen-Modus ein Fenster verwenden, fügen wir WS_EX_WINDOWEDGE zum erweiterten Stil zu. Das gibt dem Fenster einen mehr 3 dimensionales Aussehen. Für den Stil benutzen wir WS_OVERLAPPEDWINDOW statt WS_POPUP. WS_OVERLAPPEDWINDOW erzeugt ein Fenster mit einer Titelleiste, Fenstermenü, Minimierungs- und Maximierungs-Buttons.

```
			// erweiterter Fenster-Stil
			dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;
			dwStyle=WS_OVERLAPPEDWINDOW;		// Fenster Stil
		}
```

Die folgende Zeile richtet unser Fenster abhängig vom Stil des Fensters aus. Durch die Ausrichtung wird unser Fenster exakt der geforderten Auflösung angepasst. Normalerweise überlappen die Ränder teilen von unserem Fenster. Indem wir aber AdjustWindowRectEx verwenden, wird nichts unserer OpenGL Szene von den Rändern verdeckt, sondern das Fenster wird größer gemacht, um entsprechend Platz für die Pixel zu machen, die am Fenster-Rand gezeichnet werden. Im Fullscreen-Modus hat dieser Befehl keinen Effekt.

```
	// justiere das Fenster der angeforderten Größe entsprechend
	AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);
```

Im nächsten Code-Abschnitt erzeugen wir unser Fenster und überprüfen, ob es auch wirklich korrekt erzeugt wurde. Wir übergeben CreateWindowEX() alle benötigten Parameter. Wir entscheiden uns, den erweiterten Stil zu verwenden. Der Klassenname (welcher identisch mit dem sein muss, den Sie bei der Registrierung der Fenster-Klasse verwendet haben). Der Fenster-Titel. Der Fenster-Stil. Die obere linke Ecke Ihres Fenster (0,0 ist immer gut). Die Breite und Höhe des Fensters. Wir wollen kein Parent-Fenster und wir brauchen kein Menü, weswegen wir diese beiden Parameter auf NULL setzen. Wir übergeben unsere Fenster-Instanz und zu letzt NULL als letzten Parameter.

Beachten Sie, dass wir die Stile WS_CLIPSIBLINGS und WS_CLIPCHILDREN den anderen Stilen hinzufügen, die wir benutzen wollen. WS_CLIPSIBLINGS und WS_CLIPCHILDREN werden bei BENÖTIGT damit OpenGL korrekt läuft. Diese Stile verhindern, dass andere Fenster über oder in unser OpenGL Fenster zeichnet.

```
if (!(hWnd=CreateWindowEx( dwExStyle,        // erweiterter Stil für das Fenster
                           "OpenGL",         // Klassen Name
                           title,            // Fenster Titel
                           WS_CLIPSIBLINGS |     // benötigter Fenster-Stil
                           WS_CLIPCHILDREN |     // benötigter Fenster-Stil
                           dwStyle,          // ausgewählter Fenster Stil
                           0, 0,             // Fenster Position

                           // berechne die justierte Fenster-Breite
                           WindowRect.right-WindowRect.left,
                           // berechne die justierte Fenster-Höhe
                           WindowRect.bottom-WindowRect.top,

                           NULL,             // Kein Parent-Fenster
                           NULL,             // Kein Menü
                           hInstance,        // Instanz
                           NULL)))           // Leite nichts an WM_CREATE weiter
```

Als nächstes überprüfen wir, ob unser Fenster korrekt erzeugt wurde. Wenn unser Fenster erzeugt wurde, enthält hWnd unser Fenster-Handle. Wenn das Fenster nicht erzeugt wurde, wird eine Fehlernachricht angezeigt und das Programm beendet (return FALSE).

```
        {
                KillGLWindow();         // Resette die Ansicht
                MessageBox(NULL,"Window Creation Error.","ERROR",MB_OK|
                MB_ICONEXCLAMATION);
                return FALSE;           // Gebe FALSE zurück
        }
```

Der nächste Code-Abschnitt beschreibt ein Pixel-Format. Wir wählen ein Format aus, das OpenGL und Double-Buffering unterstützt, sowie RGBA (rot, grün, blau und alpha-Kanal). Wir versuchen ein Pixel-Format zu finden, dass den Bits entspricht, die wir verwenden wollen (16 Bit, 24 Bit, 32 Bit). Letztendlich benötigen wir noch ein 16 Bit Z-Buffer. Die restlichen Parameter werden entweder nicht benutzt oder sind nicht wichtig (vom Stencil Buffer und dem (langsamen) Accumulation Buffer abgesehen).

```
	// pfd teilt Windows mit, wie wie die Dinge haben wollen
	static PIXELFORMATDESCRIPTOR pfd=
	{
		sizeof(PIXELFORMATDESCRIPTOR),	// Größe des Pixel Format Descriptors
		1,				// Versions Nummer
		PFD_DRAW_TO_WINDOW |		// Format muss Fenster unterstützen
		PFD_SUPPORT_OPENGL |		// Format muss OpenGL unterstützen
		PFD_DOUBLEBUFFER,		// Muss Double Buffering unterstützen
		PFD_TYPE_RGBA,			// Fordere ein RGBA Format an
		bits,				// wähle unsere Farbtiefe aus
		0, 0, 0, 0, 0, 0,		// Color Bits werden ignoriert
		0,				// Kein Alpha Buffer
		0,				// Shift Bit wird ignoriert
		0,				// Kein Accumulation Buffer
		0, 0, 0, 0,			// Accumulation Bits werden ignoriert
		16,				// 16Bit Z-Buffer (Depth Buffer)
		0,				// Kein Stencil Buffer
		0,				// Kein Auxiliary Buffer
		PFD_MAIN_PLANE,			// Haupt-Zeichen-Schicht
		0,				// Reserviert
		0, 0, 0				// Layer Masken werden ignoriert
	};
```

Wenn keine Fehler während der Fenster-Erstellung auftreten, versuchen wir ein OpenGL Device Context zu bekommen. Wenn wir kein DC kriegen können, wird eine Fehler-Meldung angezeigt und das Programm beendet (return FALSE).

```
	if (!(hDC=GetDC(hWnd)))		// Haben wir einen Device Kontext erhalten?
	{
		KillGLWindow();		// Resette die Anzeige
		MessageBox(NULL,"Can't Create A GL Device Context.","ERROR",
		MB_OK|MB_ICONEXCLAMATION);
		return FALSE;		// Gebe FALSE zurück
	}
```

Wenn wir ein Device Context für unser OpenGL Fenster bekommen konnten, versuchen wir ein Pixel Format zu finden, welches mit dem oben beschriebenen übereinstimmt. Wenn Windows kein passendes Pixel Format finden konnte, wird eine Fehlernachricht angezeigt und das Programm beendet (return FALSE).

```
        // Hat Windows ein passendes Pixel Format gefunden?
        if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))
        {
                KillGLWindow();         // Resette die Anzeige
                MessageBox(NULL,"Can't Find A Suitable PixelFormat.",
                ROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;           // Gebe FALSE zurück
        }
```

Wenn Windows ein passendes Pixel Format gefunden hat, versuchen wir das Pixel Format zu setzen. Wenn das Pixel Format nicht gesetzt wurden konnte, wird eine Fehlernachricht angezeigt und das Programm beendet (return FALSE).

```
        if(!SetPixelFormat(hDC,PixelFormat,&pfd))       // Können wir das Pixel Format setzen?
        {
                KillGLWindow();         // Resette die Anzeige
                MessageBox(NULL,"Can't Set The PixelFormat.",
                "ERROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;           // Gebe FALSE zurück
        }
```

Wenn das Pixel Format richtig gesetzt werden konnte, versuchen wir einen Rendering Context zu erhalten. Wenn wir kein Rendering Context erhalten konnten, wird eine Fehlernachricht angezeigt und das Programm beendet (return FALSE).

```
        if (!(hRC=wglCreateContext(hDC)))      // Können wir einenRendering Kontext kriegen?
        {
                KillGLWindow(); // Resette die Anzeige
                MessageBox(NULL,"Can't Create A GL Rendering Context.",
                "ERROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;   // Gebe FALSE zurück
        }
```

Wenn bis hier her keine Fehler aufgetreten sind und wir es geschafft haben sowohl Device Context als auch Rendering Context zu erzeugen, müssen wir den Rendering Context nur noch aktivieren. Wenn wir den Rendering Context nicht aktivieren konnten, wird eine Fehlernachricht angezeigt und das Programm beendet (return FALSE).

```
        if(!wglMakeCurrent(hDC,hRC))            // Versuche den Rendering Kontext zu aktivieren
        {
                KillGLWindow(); // Resette die Anzeige
                MessageBox(NULL,"Can't Activate The GL Rendering Context.",
                "ERROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;   // Gebe FALSE zurück
        }
```

Wenn alles glatt gelaufen ist und unser OpenGL Fenster erzeugt wurde, zeigen wir das Fenster als Vordergrundsfenster (um ihm eine höhere Priorität zu geben) und setzen den Fokus auf das Fenster. Dann rufen wir ReSizeGLScene auf und übergeben die Bildschirm Breite und Höhe, um die Perspektive unseres OpenGL Screens zu setzen.

```
        ShowWindow(hWnd,SW_SHOW);       // Zeige das Fenster
        SetForegroundWindow(hWnd);      // Etwas höhere Priorität
        SetFocus(hWnd);                 // Setze den Tastatur-Fokus auf das Fenster
        ReSizeGLScene(width, height);   // Initialisiere unseren perspektivischen GL-Screen
```

Letztendlich springen wir zu InitGL(), wo wir Beleuchtung, Texturen und alles andere vorbereiten, was wir benötigen. Sie können ihre eigene Fehlerbehandlung in InitGL() machen und TRUE (wenn alles OK war) oder FALSE (wenn was falsch gelaufen ist) zurückgeben. Wenn Sie zum Beispiel Texturen in InitGL() laden wollen und ein Fehler auftritt, wollen Sie das Programm vielleicht anhalten. Wenn Sie FALSE für InitGL() zurückgeben, sehen die folgende Zeilen das FALSE als eine Fehler-Nachricht an und beenden das Programm.

```
        if (!InitGL())    // Initialisiere unser neu erzeugtes GL Fenster
        {
                KillGLWindow();         // Resette die Anzeige
                MessageBox(NULL,"Initialization Failed.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;           // Gebe FALSE zurück
        }
```

Wenn wir es bishierher geschafft haben, können wir uns sicher sein, dass das Erzeugen des Fensters erfolgreich war. Wir geben TRUE an WinMain() zurück und teilen WinMain() damit mit, dass keine Fehler aufgetreten sind. Damit verhindern wir, dass das Programm beendet wird.

```
        return TRUE;    // Erfolg
}
```

Hier werden alle Fenster-Nachrichten bearbeitet. Wenn wir die Fenster-Klasse registrieren, teilen wir ihr mit, in diesen Code-Abschnitt zu springen, um die Fenster-Nachrichten zu bearbeiten.

```
LRESULT CALLBACK WndProc ( HWND hWnd,           // Handle für dieses Fenster
                           UINT uMsg,           // Nachricht für dieses Fenster
                           WPARAM wParam,       // Weitere Nachrichten Informationen
                           LPARAM lParam)       // Weitere Nachrichten Informationen
{
```

Der folgende Code vergleich uMsg mit den Case-Statements. uMsg enthält den Namen der Nachricht, die wir bearbeiten werden.

```
        switch (uMsg)           // Überprüfe auf Fenster-Nachrichten
        {
```

Wenn uMsg gleich WM_ACTIVATE ist, überprüfen wir, ob unser Fenster immer noch aktiv ist. Wenn unser Fenster minimiert wurde, wird die Variable active auf FALSE gesetzt. Wenn unser Fenster aktiv ist, wird die Variable active auf TRUE gesetzt.

```
		case WM_ACTIVATE:		// Wenn Aktivierungs-Nachricht reinkommt
			{
				if (!HIWORD(wParam))		// Überprüfe Minimierungs-Status
				{
					active=TRUE;		// Programm ist aktiv
				}
				else
				{
					active=FALSE;		// Programm ist nicht länger aktiv
				}
			return 0;				// Kehre zurück zur Nachrichten-Schleife
			}
```

Wenn die Nachricht gleich WM_SYSCOMMAND ist (System Befehl), vergleichen wir wParam mit den Case-Statements. Wenn wParam gleich SC_SCREENSAVE oder SC_MONITORPOWER ist, also ein Screensaver versucht sich zu starten oder der Monitor versucht in den Energiesparmodus überzutreten. In dem wir 0 zurückgeben, verhindern wir diese beiden Sachen.

```
		case WM_SYSCOMMAND:		// hereinkommende System Befehle
			{
				switch (wParam)		// Überprüfe auf System Aufrufe
				{
				case SC_SCREENSAVE: // versucht der Screensaver sich zu starten?

				// versucht der Monitor in den Energiesparmodus zu gehen?
				case SC_MONITORPOWER:
					return 0;	// verhindere das
				}

				break;		// Ende
			}
```

Wenn uMsg gleich WM_CLOSE ist, wird das Fenster geschlossen. Wir senden eine Beenden-Benachrichtung, dass die Haupt-Schleife beendet wird. Die Variable done wir auf TRUE gesetzt, die Haupt-Schleife in WinMain() wird beendet und das Programm wird geschlossen.

```
		case WM_CLOSE:		// Haben wir eine Nachricht zum Schließen erhalten?
			{
				PostQuitMessage(0);		// Sende eine Beenden-Nachricht
				return 0;			// Springe zurück
			}
```

Wenn eine Taste gedrückt wird, können wir das herausfinden, indem wir wParam auslesen. Ich setze dann das Feld der Taste in dem Array keys[] auf TRUE. Auf diesem Weg kann ich das Array später auslesen um herauszufinden, welche Taste gerdrückt wurde. Das erlaubt uns, dass mehr als eine Taste zur selben Zeit gedrückt wird.

```
		case WM_KEYDOWN:			// Wird eine Taste gedrückt?
			{
				keys[wParam] = TRUE;		// Wenn ja, markiere sie mit TRUE
				return 0;			// Springe zurück
			}
```

Wenn eine Taste losgelassen wurde, finden wir die Taste heraus, indem wir wParam auslesen. Wir markieren dann die Felder in dem Array keys[] mit FALSE. Auf diesem Weg weiß ich, wenn ich das Feld auslese, ob die Taste noch gedrückt ist oder ob sie losgelassen wurde. Jede Taste der Tastatur kann durch eine Zahl zwischen 0 und 255 repräsentiert werden. Wenn ich eine Taste drücke, die zum Beispiel durch die Nummer 40 repräsentiert wird, keys[40] wird auf TRUE gesetzt. Wenn ich wieder los lasse, wird es FALSE. So benutzen wir das Feld, um Tastendrücke zu speichern.

```
		case WM_KEYUP:		// Wurde eine Taste losgelassen?
			{
				keys[wParam] = FALSE;		// Wenn ja, markiere sie mit FALSE
				return 0;			// Springe zurück
			}
```

Jedes mal, wenn wir die Größe unseres Fensters verändern, wird uMsg die WM_SIZE Nachricht enthalten. Wir lesen die Werte LOWORD und HIWORD des lParam aus, um die neue Breite und Höhe des Fensters zu erhalten. Wir übergeben die neue Breite und Höhe an ReSizeGLScene(). Die OpenGL Szene wird dann der neue Breite und Höhe angepasst.

```
		case WM_SIZE: // ändere die Größe des Fenster
			{
				// LoWord=Breite, HiWord=Höhe
				ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));
				return 0;		// Springe zurück
			}
	}
```

Jede Nachricht, die uns nicht interessiert, übergeben wir DefWindowProc, so dass Windows diese abarbeiten kann.

```
	// Übergebe alle nicht bearbeiteten Nachrichten an DefWindowProc
	return DefWindowProc(hWnd,uMsg,wParam,lParam);
}
```

Das ist der Einstiegspunkt unserer Windows-Applikation. Hier rufen wir unsere Fenster-Erzeuguns-Routine auf, behandeln Fenster-Nachrichten und überwachen die Interaktion mit dem Benutzer.

```
int WINAPI WinMain( HINSTANCE hInstance,		// Instanz
		    HINSTANCE hPrevInstance,	// vorherige Instanz
		    LPSTR lpCmdLine,		// Kommandozeilen Parameter
		    int nCmdShow)		// Fenster Anzeige Status
{
```

Wir führen zwei Variablen ein. msg wird benutzt um zu überprüfen, ob eine Nachricht gerade wartet, die behandelt werden sollte. Die Variable done ist anfangs auf FALSE gesetzt. Das bedeutet, dass unser Programm noch nicht zu Ende ist. So lange wie done auf FALSE bleibt, läuft das Programm weiter. Sobald done von FALSE auf TRUE wechselt, wird unser Programm beendet.

```
	MSG msg;		// Windows Nachrichten Struktur
	BOOL done=FALSE;	// Bool Variable um die Schleife zu beenden
```

Dieser Code-Abschnitt ist komplett optional. Es wird eine MessageBox angezeigt, die frage, ob im Fullscreen-Modus gestartet werden soll oder nicht. Wenn der Benutzer den NEIN-Button anklickt, wird die fullscreen-Variable von TRUE (dem Standardwert) auf FALSE gesetzt und das Programm im Fenster-Modus, anstatt des Fullscreen-Modus gestartet.

```
	// Frage den Benutzer, in welchen Modus er starten will
	if (MessageBox(NULL,"Would You Like To Run In Fullscreen Mode?", "Start FullScreen?",
	MB_YESNO|MB_ICONQUESTION)==IDNO)
	{
		fullscreen=FALSE;	// Fenster-Modus
	}
```

So erzeugen wir unser OpenGL Fenster. Wir übergeben den Titel, die Breite, die Höhe, die Farbtiefe und TRUE (Fullscreen) oder FALSE (Fenster-Modus) der CreateGLWindow-Funktion. Das war's! Ich bin ziemlich glücklich mit der Einfachheit dieses Codes. Wurde das Fenster aus irgend einem Grund nicht erzeugt, wird FALSE zurückgegeben und unser Programm augenblicklich gestoppt (return 0).

```
	// erzeuge unser OpenGL Fenster
	if (!CreateGLWindow("NeHe's OpenGL Framework",640,480,16,fullscreen))
	{
		return 0;	// Beende, wenn Fenster nicht erzeugt wurde
	}
```

Das ist der Anfang unserer Schleife. Solange done gleich FALSE ist, wird die Schleife durchlaufen.

```
	while(!done)		// Schleife die so lange läuft, bis done=TRUE
	{
```

Als erstes müssen wir überprüfen, ob irgendwelche Nachrichten warten. Indem wir PeekMessage() verwenden, können wir auf Nachrichten überprüfen ohne das Programm anzuhalten. Viele Programme verwenden GetMessage(). Das läuft zwar, aber wenn Sie GetMessage() verwenden, macht Ihr Programm so lange nichts, bis es eine Zeichnen-Nachricht oder eine andere Fenster-Nachricht erhält.

```
		if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))	// Wartet eine Nachricht?
		{
```

Im nächsten Codeabschnitt überprüfen wir, ob eine Beendigungs-Nachricht vorliegt. Wenn die aktuelle Nachricht eine WM_QUIT Nachricht ist, die von PostQuitMessage(0) erzeugt wurde, dann wird die Variable done auf TRUE gesetzt, damit das Programm beendet wird.

```
                if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))        // Wartet eine Nachricht?
                {
                        // Haben wir eine Nachricht zum beenden erhalten?
                        if (msg.message==WM_QUIT)                {
                                done=TRUE;                       // Wenn ja done=TRUE
                        }
                        else    // Wenn nicht, bearbeite die Fenster-Nachrichten
                        {
```

Wenn die Nachricht keine Beendigungs-Nachricht ist, übersetzen wir die Nachricht und bearbeiten sie dann, so dass WndProc() oder Windows mit der Nachricht umgehen kann.

```
                                TranslateMessage(&msg);     // Übersetze die Nachricht
                                DispatchMessage(&msg);      // bearbeite die Nachricht
                        }
                }
                else    // Wenn da keine Nachrichten sind
                {
```

Wenn keine Nachrichten vorhanden waren, zeichnen wir unsere OpenGL Szene. Die erste Zeile Code überprüft, ob das Fenster aktiv ist. Wenn die ESC-Taste gedrückt wurde, ist die Variable done auf TRUE gesetzt und das Programm wird beendet.

```
                        // Zeichne die Szene. Schau nach der ESC-Taste und
                        // Beendigungs-Nachrichten von DrawGLScene()

                        if (active)       // Programm aktiv?
                        {
                                if (keys[VK_ESCAPE])  // Wurde ESC gedrückt?
                                {
                                        done=TRUE;    // ESC Signalisiert, dass Beendet werden soll
                                }
                                else    // Es ist noch nicht Zeit zum beenden, zeichne Screen neu
                                {
```

Wenn das Programm aktiv ist und ESC nicht gedrückt wurde, rendern wir die Szene und tauschen den Buffer (indem wir Double Buffering verwenden, erreichen wir weiche Flackerfreie Animationen). Indem wir Double Buffering verwenden, zeichnen wir alles in einen 'versteckten' Screen, den wir nicht sehen können. Wenn wir die Buffer tauschen, wird der angezeigte Screen, den wir sehen können, der versteckte Screen und der versteckte Screen wird sichtbar. Dadurch sehen wir nicht, wie unsere Szene nach und nach gezeichnet wird. Sie erscheint einfach komplett.

```
                                        DrawGLScene();      // Zeichne die Szene
                                        SwapBuffers(hDC);   // Swap Buffers (Double Buffering)
                                }
                        }
```

Das nächste Stück Code ist neu und wurde gerade erst eingefügt (05-01-00). Es erlaubt uns, die F1-Taste zu drücken, um zwischen Fullscreen und Fenster-Modusm, beziehungsweise andersherum, zu wechseln

```
                        if (keys[VK_F1])        // Wurde F1 gedrückt?
                        {
                                keys[VK_F1]=FALSE;   // Wenn ja, setze Taste auf FALSE
                                KillGLWindow();         // Kill unser aktuelles Fenster

                                // Wechsel zwischen Fullscreen und Fester-Modus
                                fullscreen=!fullscreen;
                                // Erzeuge unser OpenGL Fenster erneut
                                if (!CreateGLWindow("NeHe's OpenGL Framework",
                                                    640,480,16,fullscreen))
                                {
                                        return 0;  // Beenden, wenn das Fenster nicht erzeugt wurde
                                }
                        }
                }
        }
```

Wenn die done Variable nicht länger auf FALSE gesetzt ist, wird das Programm beendet. Wir de-initialisieren das OpenGL-Fenster korrekt, so das alles freigegeben wird und wir beenden das Programm.

```
        // Shutdown
        KillGLWindow();          // Kill das Fenster
        return (msg.wParam);   // Beende das Programm
}
```

In diesem Tutorial habe ich so detailiert wie möglich versucht zu erklären, welche Schritte notwendig sind, um ein OpenGL Programm zu initialisieren und im Fullscreen-Modus laufen zu lassen, welches mit ESC beendet werden kann und das überwacht, ob das Fenster aktiv ist oder nicht. Ich habe ungefähr 2 Wochen gebraucht, um den Code zu schreiben, eine Woche, um die Fehler zu beheben & mit Programmier-Gurus mich zu unterhalten und 2 Tage (ungefähr 22 Stunden, um diese HTML-Datei zu schreiben). Wenn Sie Anmerkungen oder Fragen haben, schreiben Sie mir bitte eine E-Mail. Wenn Sie denken, ich habe etwas nicht korrekt kommentiert oder der Code könnte an einigen Stellen besser gemacht werden, lassen Sie es micht wissen. Ich möchte das beste OpenGL Tutorial machen. Ich bin an Ihrem Feedback interessiert.

Jeff Molofee (NeHe)

* DOWNLOAD Visual C++ Code für diese Lektion.

* DOWNLOAD ASM Code für diese Lektion. (Conversion by Foolman)
* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD BeOS Code für diese Lektion. (Conversion by Rene Manqueros)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Joachim Rohde)
* DOWNLOAD VB.Net CsGL Code für diese Lektion. (Conversion by X)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Java/SWT Code für diese Lektion. (Conversion by Victor Gonzalez)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Perl Code für diese Lektion. (Conversion by Cora Hussey)
* DOWNLOAD Python Code für diese Lektion. (Conversion by John Ferguson)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Jon DuBois)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Ross Dawson)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.

Lektion 02 - Ihr erstes Polygon

Im ersten Tutorial habe ich Ihnen beigebracht, wie man ein OpenGL-Fenster erzeugt. In diesem Tutorial bringe ich Ihnen bei, wie man Dreiecke und Quadrate zeichnet. Wir werden ein Dreieck mittels GL_TRIANGLES und ein Qudrat mittels GL_QUADS erzeugen.

Wir werden den Code aus dem ersten Tutorial benutzen und die DrawGLScene() Prozedur erweitern. Ich werde die komplette Prozedur neu schreiben. Wenn Sie die letzte Lektion erweitern wollen, können Sie die DrawGLScene() Prozedur mit dem unteren Code ersetzen oder einfach die fehlenden Zeilen einfügen.

```
int DrawGLScene(GLvoid)        // Hier kommt der ganze Zeichnen-Kram hin
{
// Löscht den Bildschirm und den Depth-Buffer
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glLoadIdentity();              // Resettet die Ansicht (View)
```

Wenn Sie glLoadIdentity() aufrufen, bewegen Sie sich zurück ins Zentrum des Bildschirms mit der X-Achse von links nach rechts und der Y-Achse von oben nach unten und der Z-Achse die in den Bildschirm rein und raus geht.

Das Zentrum eines OpenGL-Bilschirms ist 0.0f auf der X und Y-Achse. Links vom Zentrum ist der negative Zahlrenbereich. Rechts davon der positve. Bewegt man sich am Bildschirm nach oben, befindet man sich im positiven Zahlenbereich und wenn man sich nach unten bewegt im negativen. Bewegt man sich tiefer in den Bildschirm hinein bewegt man sich in den negativen Zahlenbereich und wenn man sich dem Betrachter nähert, in den positiven.

glTranslatef(x, y, z) bewegt sich entlängst der X, Y und Z-Achse, in dieser Reihenfolge. Die untere Code-Zeile bewegt sich 1.5 Einheiten auf der X-Achse. Auf der Y-Achse wird sich überhaupt nicht bewegt (0.0) und 6.0 Einheiten in den Bildschirm hinein. Dabei bewegen Sie sich nicht vom Zentrum des Bildschirms aus, sondern von da aus, wo Sie sich gerade befinden.

```
// 1.5 Einheiten nach Links dann 6.0 Einheiten in den Bildschirm hinein
glTranslatef(-1.5f,0.0f,-6.0f);
```

Nun, da wir uns auf die linke Hälfte des Bildschirms bewegt haben und wir die Ansicht tief genug in den Bildschirm (-6.0) gesetzt haben, so dass wir die gesamte Szene sehen können, erzeugen wir ein Dreieck. glBegin(GL_TRIANGLES) bedeutet, dass wir mit dem Zeichnen eines Dreiecks beginnen wollen und glEnd() teilt OpenGL mit, dass wir fertig mit dem Erzeugen des Dreiecks sind. Normalerweise, wenn Sie 3 Punkte haben, benutzen Sie GL_TRIANGLES. Dreiecke zu zeichnen ist auf den meisten Grafikkarten ziemlich schnell. Wenn Sie 4 Punkte haben, benutzen Sie GL_QUADS, um sich Ihr Leben zu erleichtern. Von dem, was ich gehört habe, rendern die meisten Videokarten Objekte sowieso als Dreiecke. Letztendlich, wenn Sie mehr als 4 Punkte haben, benutzen Sie GL_POLYGON.

In unserem einfachen Programm zeichnen wir nur ein Dreieck. Wenn wir ein zweites Dreieck zeichnen wollten, könnten wir 3 weitere Zeilen Code (3 Punkte) einfügen, direkt nach den ersten drei. Alle sechs Zeilen Code würden zwischen glBegin(GL_TRIANGLES) und glEnd() liegen. Es macht keinen Unterschied, wenn Sie ein glBegin(GL_TRIANGLES) und ein glEnd() um jede Gruppe von 3 Punkten setzen. Das gilt für Quadrate genauso. Wenn Sie wissen, dass Sie nur Quadrate zeichnen, können Sie die zweite Gruppe an 4 Punkten direkt hinter den ersten vier Zeilen einfügen. Ein Polygon (GL_POLYGON) kann allerdings aus vielen Punkten bestehen, so dass es kein Unterschied macht, wieviele Zeilen Sie zwischen glBegin(GL_POLYGON) und glEnd() haben.

Die erste Zeile hinter glBegin, setzt den ersten Punkt unseres Polygons. Die erste Zahl bei glVertex ist für die x-Achse, die zweite für die Y-Achse und die dritte für die Z-Achse. In der ersten Zeile, bewegen wir uns also nicht auf der X-Achse. Wir bewegen uns eine Einheit auf der Y-Achse und wir bewegen uns nicht auf der Z-Achse. Dadurch erhalten wir den obersten Punkt des Dreiecks. Das zweite glVertex bewegt eine Einheit nach links auf der X-Achse und eine Einheit auf der Y-Achse nach unten. Das ergibt den unteren linken Punkt des Dreiecks. Das dritte glVertex bewegt eine Einheit nach rechts und eine runter. Dadurch erhalten wir den unteren rechten Punkt des Dreiecks. glEnd() teilt OpenGL mit, dass keine weiteren Punkte folgen. Das gefüllte Dreieck wird dargestellt.

```
glBegin(GL_TRIANGLES);          // Zeichne Dreiecke
glVertex3f( 0.0f, 1.0f, 0.0f);  // Oben
glVertex3f(-1.0f,-1.0f, 0.0f);  // Unten Links
glVertex3f( 1.0f,-1.0f, 0.0f);  // Unten Rechts
glEnd();                        // Fertig gezeichnet
```

Nun, da wir das Dreieck auf der linken Seite des Bildschirms angezeigt haben, müssen wir uns auf die rechte Seite bewegen, um das Quadrat anzuzeigen. Um das zu machen, benutzen wir erneut glTranslate. Diesmal müssen wir uns nach rechts bewegen, weswegen X ein positiver Wert sein muss. Da wir uns schon 1.5 Einheiten nach links bewegt haben; um ins Zentrum zu gelangen, müssen wir uns nun 1.5 Einheiten nach rechts bewegen. Nachdem wir das Zentrum erreicht haben, bewegen wir uns weitere 1.5 Einheiten nach rechts. Insgesamt bewegen wir uns also 3.0 Einheiten nach rechts.

```
glTranslatef(3.0f,0.0f,0.0f);       // Bewege 3 Einheiten nach rechts
```

Nun erzeugen wir das Quadrat. Wir machen das, indem wir GL_QUADS benutzen. Ein Quad ist prinzipiell ein 4-seitiges Polygon. Perfekt, um ein Quadrat zu machen. Der Code, um das Quadrat zu machen, ist sehr ähnlich dem Code, den wir für das Dreieck verwendet haben. Der einzige Unterschied ist, dass wir GL_QUADS statt GL_TRIANGLES benutzen und ein extra glVertex3f für den 4ten Punkt des Quadrats. Wir zeichnen das Quadrat oben links, oben rechts, unten rechts, unten links (Uhrzeigersinn). Dadurch, dass wir das Quadrat im Uhrzeigersinn zeichnen, wird das Quadrat als hintere Seite gezeichnet. Das bedeutet, dass die Seite des Quad, die wir sehen, eigentlich die hintere Seite ist. Objekte die gegen den Uhrzeigersinn gezeichnet werden, sind uns zugewendet. Zur Zeit ist das nicht wichtig, aber für später müssen wir das wissen.

```
        glBegin(GL_QUADS);              // Zeichne eine Quadrat
        glVertex3f(-1.0f, 1.0f, 0.0f);  // Oben Links
        glVertex3f( 1.0f, 1.0f, 0.0f);  // Oben Rechts
        glVertex3f( 1.0f,-1.0f, 0.0f);  // Unten Rechts
        glVertex3f(-1.0f,-1.0f, 0.0f);  // Unten Links
        glEnd();                        // Fertig mit Quadrat zeichnen
        return TRUE;                    // Weiter geht's
}
```

Zu guter letzt ändern Sie den Code um zwischen Fenster- und Fullscreen-Modus zu wechseln, so dass der Titel des Fensters korrekt ist.

```
if (keys[VK_F1])                        // Wurde F1 gedrückt?
{
        keys[VK_F1]=FALSE;              // Wenn ja, setze Taste auf FALSE
        KillGLWindow();                 // Kill unser aktuelles Fenster
        fullscreen=!fullscreen;         // Wechsel zwischen Fullscreen und Fester-Modus

        // Erzeuge unser OpenGL-Fenster neu ( Modifiziert )
        if (!CreateGLWindow("NeHe's First Polygon Tutorial", 640,480,16,fullscreen))
        {
                return 0;               // Beenden, wenn das Fenster nicht erzeugt wurde
        }
}
```

Markus Knauer fügt hinzu: In dem Buch ("OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1", J. Neider, T. Davis, M. Woo, Addison-Wesley, 1993) erklärt folgender Paragraph genau, was NeHe meint, wenn er die Bewegung von Einheiten in OpenGL anspricht:

"[Ich erwähnte] Inches und Millimeter - haben diese irgend etwas mit OpenGL zu tun? Die Antwort, in einem Wort, ist nein. Die Projektion und andere Transformationen sind Einheitenlos. Wenn Sie wollen, können Sie sich die nahe und ferne Clipping-Ebene in der Entfernung 1.0 und 20.0 Meter, Inches, Kilometers oder Ligen vorstellen, das liegt ganz an Ihnen. Die einzige Regel ist, dass Sie eine konsistente Einheit als Maß nehmen."

In diesem Tutorial habe ich so detailiert wie möglich versucht zu erklären, welche Schritte notwendig sind, um Polygone und Quadrate auf den Bildschirm mittels OpenGL zu zeichnen. Wenn Sie Anmerkungen oder Fragen haben, schreiben Sie mir bitte eine E-Mail. Wenn Sie denken, ich habe etwas nicht korrekt kommentiert oder der Code könnte an einigen Stellen besser gemacht werden, lassen Sie es micht wissen. Ich möchte das beste OpenGL Tutorial machen. Ich bin an Ihrem Feedback interessiert.

Jeff Molofee (**NeHe**)

* DOWNLOAD Visual C++ Code For This Lesson.
* DOWNLOAD ASM Code für diese Lektion. (Conversion by Foolman)
* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD BeOS Code für diese Lektion. (Conversion by Rene Manqueros)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Joachim Rohde)
* DOWNLOAD VB.Net CsGL Code für diese Lektion. (Conversion by X)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD GLUT Code für diese Lektion. (Conversion by Andy Restad)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Java/SWT Code für diese Lektion. (Conversion by Victor Gonzalez)
* DOWNLOAD Jedi-SDL Code für diese Lektion. (Conversion by Dominique Louis)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Perl Code für diese Lektion. (Conversion by Cora Hussey)
* DOWNLOAD Python Code für diese Lektion. (Conversion by Travis Wells)
* DOWNLOAD QT/C++ Code für diese Lektion. (Conversion by Popeanga Marian)
* DOWNLOAD REALbasic Code für diese Lektion. (Conversion by Thomas J. Cunningham)
* DOWNLOAD Ruby Code für diese Lektion. (Conversion by Ben Goodspeed)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Jon DuBois)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Ross Dawson)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.

Lektion 03 - Farbe hinzufügen

Im letztem Tutorial habe ich Ihnen beigebracht, wie man Dreiecke und Quadrate auf dem Bildschirm anzeigt. In diesem Tutorial werde ich Ihnen beibringen, wie man das Dreieck und das Quadrat auf zwei verschiedene Arten färben kann. Flat Coloring füllt das Quadrat mit einer Farbe. Smooth Coloring blendet die 3 Farben, die an jedem Punkt (Vertex) des Dreiecks angegeben sind, ineinander über, so dass eine nette Vermischung der Farben entsteht.

Wir benutzen den Code aus dem letzten Tutorial und erweitern die DrawGLScene Prozedur. Ich werde folgend nochmal die komplette Prozedur aufschreiben, so dass Sie, wenn Sie den Code der letzten Lektion verändern wollen, die DrawGLScene Prozedur mit dem folgenden Code ersetzen können oder einfach den fehlenden Code hinzufügen, der im letzten Tutorial noch nicht enthalten war.

```
int DrawGLScene(GLvoid)       // Hier kommt der ganze Zeichnen-Kram hin
        {
                // Löscht den Bildschirm und den Depth-Buffer
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                glLoadIdentity();                 // Resettet die aktuelle Modelview Matrix
                glTranslatef(-1.5f,0.0f,-6.0f);   // Links 1.5 dann 6 Einheiten in den Bildschirm hinein
                glBegin(GL_TRIANGLES);            // Fange an, Dreiecke zu zeichnen
```

Wenn Sie sich an das letzte Tutorial erinnern, ist das die Sektion Code, wo das Rechteck auf der linken Hälfte des Bildschirms gezeichnet wird. In der nächsten Zeile Code benutzen wir das erste Mal glColor3f(r,g,b). Die drei Parameter in den Klammern sind die Werte für die Rot, Grün und Blau Intensität. Die Werte können zwischen 0.0f und 1.0f liegen. Das ganze arbeitet in der selben Weise, wie die Farbwerte, die wir zum löschen des Hintergrunds verwenden.

Wir setzen die Farbe Rot (volle Rot-Intensität, kein Grün, kein Blau). Die Codezeile direkt danach ist der erste Vertex (die Spitze des Dreiecks) und wird mit der Farbe gezeichnet ist, die gerade gesetzt ist, also rot. Alles was wir von nun an zeichnen, wird in rot sein, bis wir die Farbe auf was anderes als Rot setzen.

```
        glColor3f(1.0f,0.0f,0.0f);        // Setzt die Farbe auf Rot
        glVertex3f( 0.0f, 1.0f, 0.0f);    // Eine Einheit aus dem Zentrum nach oben (Oberste Punkt)
```

Wir haben den ersten Vertex auf den Bildschirm gebracht und seine Farbe auf Rot gesetzt. Nun, bevor wir den zweiten Vertex setzen, ändern wir die Farbe auf Grün. Auf diese Weise wird der zweite Vertex, welcher die linke Ecke des Dreiecks ist, auf grün gesetzt.

```
        glColor3f(0.0f,1.0f,0.0f);        // Setzt die Farbe auf Grün
        glVertex3f(-1.0f,-1.0f, 0.0f);    // Eine Einheit nach links und unten (unten links)
```

Nun kommen wir zum dritten und letzten Vertex. Bevor wir ihn aber zeichnen, setzen wir die Farbe auf Blau. Dieser wird die rechte Ecke des Dreiecks. Sobald der glEnd() Befehl ausgeführt wird, wird das

Polygon gefüllt. Da es aber an jedem Vertex eine andere Farbe hat, statt einer einheitlichen Farbe, werden die Farben jeweils aus jeder Ecke gezeichnet und treffen ungefähr in der Mitte zusammen, wo sie ineinander übergehen. Das nennt man Smoot-Coloring.

```
        glColor3f(0.0f,0.0f,1.0f);          // Setzt die Farbe auf Blau
        glVertex3f( 1.0f,-1.0f, 0.0f);      // Eine Einheit nach rechts und unten (unten rechts)
glEnd();          // Fertig mit dem Zeichnen von Dreiecken

        glTranslatef(3.0f,0.0f,0.0f);       // Vom rechten Punkt 3 Einheiten nach rechts bewegen
```

Nun werden wir ein solides blaues Rechteck zeichnen. Es ist wichtig, daran zu denken, dass alles was gezeichnet wird, nachdem die Farbe gesetzt wurde, in dieser Farbe gezeichnet wird. Jedes Projekt das Sie erzeugen werden, wir in irgend einer Weise Färbungen vornehmen. Selbst in Szenen, wo alles texturiert ist, kann glColor3f dazu benutzt werden, um die Farbe der Textur zu tönen. Mehr dazu später.

So, um nun unser Rechteck in einer Farbe zu zeichnen, müssen wir lediglich die Farbe auf eine Farbe setzen, die wir haben wollen (blau in unserem Beispiel) und das Rechteck zeichnen. Die Farbe Blau wird für jeden Vertex verwendet, da wird OpenGL nicht mitteilen, dass wir die Farbe für jeden Vertex ändern wollen. Das Ergebnis... ein solides blaues Rechteck. Erneut wird das Rechteck (Viereck) im Uhrzeigersinn gezeichnet, was heißt das wir wieder auf dir Rückseite des Quadrats sehen.

```
glColor3f(0.5f,0.5f,1.0f); // Setzt die Farbe auf Blau

glBegin(GL_QUADS);    // Fange an Quadrate zu zeichnen
        glVertex3f(-1.0f, 1.0f, 0.0f); // links und eine Einheit nach oben (oben links)
        glVertex3f( 1.0f, 1.0f, 0.0f); // rechts und eine Einheit nach oben (oben rechts)
        glVertex3f( 1.0f,-1.0f, 0.0f); // rechts und eine Einheit runter (unten rechts)
        glVertex3f(-1.0f,-1.0f, 0.0f); // links und eine Einheit nach unten (unten links)
glEnd();                // Fertig mit Quadraten zeichnen

return TRUE;            // Weiter machen
}
```

Zu guter letzt ändern Sie den Code um zwischen Fenster- und Fullscreen-Modus zu wechseln, so dass der Titel des Fensters korrekt ist.

```
if (keys[VK_F1]) // Wurde F1 gedrückt?
 {
        keys[VK_F1]=FALSE;   // Wenn ja, setze Taste auf FALSE
        KillGLWindow();      // Kill unser aktuelles Fenster
        fullscreen=!fullscreen;  // Wechsel zwischen Fullscreen und Fester-Modus

        // Erzeuge unser OpenGL-Fenster neu ( Modifiziert )
        if (!CreateGLWindow("NeHe's Color Tutorial",640,480,16,fullscreen))
        {
                return 0; // Beenden, wenn das Fenster nicht erzeugt wurde
        }
 }
```

In diesem Tutorial habe ich versucht so detailiert wie möglich zu beschreiben, wie Sie Ihre OpenGL Polygone mittels Flat und Smooth Coloring färben können. Spielen Sie ein wenig mit dem Code herum, versuchen Sie die Rot, Grün und Blau Werte durch andere Zahlen zu ersetzen. Schauen Sie, welche Farbe dabei herauskommen. Wenn Sie Kommentare oder Fragen haben, schreiben Sie mir bitte eine E-Mail. Wenn Sie denken, ich habe etwas unkorrekt kommentiert oder das der Code irgendwo besser gemacht werden können, lassen Sie mich es bitte wissen. Ich möchte das beste OpenGL-Tutorial machen. Ich bin an Ihrem Feedback interessiert.

Jeff Molofee (NeHe)

* DOWNLOAD Visual C++ Code für diese Lektion.
* DOWNLOAD ASM Code für diese Lektion. (Conversion by Foolman)
* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD BeOS Code für diese Lektion. (Conversion by Rene Manqueros)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Joachim Rohde)
* DOWNLOAD VB.Net CsGL Code für diese Lektion. (Conversion by X)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Euphoria Code für diese Lektion. (Conversion by Evan Marshall)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD GLUT Code für diese Lektion. (Conversion by Andy Restad)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Java/SWT Code für diese Lektion. (Conversion by Victor Gonzalez)
* DOWNLOAD Jedi-SDL Code für diese Lektion. (Conversion by Dominique Louis)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Perl Code für diese Lektion. (Conversion by Cora Hussey)
* DOWNLOAD Python Code für diese Lektion. (Conversion by John Ferguson)
* DOWNLOAD REALbasic Code für diese Lektion. (Conversion by Thomas J. Cunningham)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Jon DuBois)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Ross Dawson)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.

Lektion 04 - Rotation

Im letzten Tutorial habe ich Ihnen beigebracht, wie man Dreiecken und Quadraten Farbe hinzufügen kann. In diesem Tutorial werde ich Ihnen beibringen, wie man diese farbigen Objekte um eine Achse rotieren lassen kann.

Wie benutzen den Code aus dem letzten Tutorial und fügen an einigen Stellen Code ein. Ich werde die kompletten Code-Abschnitte noch mal aufschreiben, damit Sie es einfacher nachzuvollziehen können, wo was hinzugefügt wurde und was ersetzt werden muss.

Wir fangen mit dem Hinzufügen von zwei Variablen an, die die Rotatation des jeweiligen Objektes enthält. Das geschieht am Anfang des Programms, unter den anderen Variablen. Sie werden zwei neue Zeilen hinter 'bool fullscreen=TRUE;' sehen. Diese Zeile deklarieren zwei Fließkomma Variable, die wir benutzen können, um die Objekte mit sehr feiner Genauigkeit rotieren zu lassen. Fließkommazahlen können Dezimale Zahlen aufnehmen. Das bedeutet, dass wir nicht nur 1, 2, 3 für die Winkel benutzen können, wir können 1.1, 1.7, 2.3 oder sogar 1.015 für feine Genauigkeit verwenden. Sie werden feststellen, dass Fließkommazahlen notwendig bei der OpenGL Programmierung sind. Die neuen Variablen heißen rtri um das Dreieck rotieren zu lassen und rquad welche das Quadrat rotieren lässt.

```
#include < windows.h>          // Header Datei für Windows
#include < gl\gl.h>            // Header Datei für die OpenGL32 Library
#include < gl\glu.h>           // Header Datei für die GLu32 Library
#include < gl\glaux.h>         // Header File For The GLaux Library

HDC hDC=NULL;                  // Privater GDI Device Context HGLRC
hRC=NULL;                      // Permanenter Rendering Context
HWND hWnd=NULL;                // Enthält unser Fenster-Handle
HINSTANCE hInstance;           // Enthält die Instanz der Applikation

bool keys[256];                // Array das für die Tastatur Routine verwendet wird
bool active=TRUE;              // Fenster Aktiv Flag
bool fullscreen=TRUE;          // Fullscreen Flag ist standardmäßig auf TRUE gesetzt

GLfloat rtri;                  // Winkel für das Dreieck ( NEU )
GLfloat rquad;                 // Winkel für das Quadrat ( NEU )
```

Nun müssen wir den DrawGLScene() Code modifizieren. Ich werde die komplette Prozedur neu schreiben. Das sollte Ihnen es einfacher machen, zu sehen, was für Änderungen ich am original Code vorgenommen habe. Ich werde erklären warum Zeilen modifiziert wurden und was die neuen Zeilen genau machen. Der folgende Code ist genau der selbe wie im letzten Tutorial.

```
int DrawGLScene(GLvoid) // Hier kommt der ganze Zeichnen-Kram hin
{
        // Löscht den Bildschirm und den Depth-Buffer
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity(); // Resettet die Ansicht (View)

        // In den Bildschirm hineinbewegen und dann links
        glTranslatef(-1.5f,0.0f,-6.0f);
```

Die nächste Code-Zeile ist neu. glRotatef(Angle,Xvector,Yvector,Zvector) ist verantwortlich, um ein Objekt um eine Achse rotieren zu lassen. Sie werden diesen Befehl ziemlich häufig verwenden. Angle (=Winkel) ist eine Zahl (in der Regel in einer Variable gespeichert), die repräsentiert, wie weit das Objekt gedreht werden soll. Die Parameter Xvector, Yvector und Zvector präsentieren zusammen den Vektor um welchen die Rotation statt findet. Wenn Sie die Werte (1,0,0) verwenden, beschreiben Sie einen Vektor, der sich 1 Einheit entlängs der X-Achse nach rechts bewegt. Die Werte (-1,0,0) beschreiben einen Vektor der in die Richtung einer Einheit entlängs der X-Achse bewegt, aber diesmal nach links.

D. Michael Traub: hat die obige Erklärung der Xvector, Yvector und Zvector Parameter geliefert.

Um X, Y und Z-Rotation besser zu verstehen, werde ich sie anhand von Beispielen erklären...

X-Achse - Sie arbeiten an einer Kreissäge. Der Stab der direkt durch die Mitte des Sägeblatts geht, verläuft von links nach rechts (wie die X-Achse in OpenGl). Die scharfen Zähne rotieren um die X-Achse (den Stabe, der durch die Mitte des Sägeblatts verläuft) und scheint sich auf Sie zu oder weg von Ihnen zu bewegen, je nachdem in welche Richtung das Sägeblatt rotiert. Wenn Sie etwas in OpenGL auf der X-Achse rotieren lassen, wird es genauso rotieren.

Y-Achse - Stellen Sie sich vor, dass Sie inmitten eines Feldes stehen. Ein gewaltiger Tornado kommt direkt auf Sie zu. Das Zentrum des Tornados verläuft vom Himmel zum Boden (rauf und runter, genauso wie die Y-Achse in OpenGL). Der Schmutz und alles andere in dem Tornado rotiert um die Y-Achse (das Zentrum des Tornados) von links nach rechts oder von Rechts nach links. Wenn Sie etwas in OpenGL auf der Y-Achse rotieren lassen, wird es genauso rotieren.

Z-Achse - Sie schauen direkt auf einen Fächer. Der Mittelpunkt des Fächers zeigt auf Sie und weg von Ihnen (wie die z-Achse in OpenGL). Die einzelnen Fächer des Fächers rotieren um die Z-Achse (Zentrum des Fächers) im Uhrzeigersinn oder gegen den Uhrzeigersinn. Wenn Sie etwas in OpenGL auf der Z-Achse rotieren lassen, wird es genauso rotieren.

Demnach würde die folgende Code-Zeile, wenn rtri gleich 7 ist, eine Rotation um 7 auf der Y-Achse (links nach recht) tätigen. Sie können ein wenig mit dem Code herum experimentieren. Ändern Sie die 0.0f in 1.0f und die 1.0f in 0.0f um das Dreieck auf der X und Y Achse zur selben Zeit rotieren zu lassen.

Es ist wichtig zu beachten, dass die Rotation in Grad vorgenommen wird. Wenn rtri einen Wert von 10 hat, würden wir um 10 Grad auf der Y-Achse rotieren.

```
glRotatef(rtri,0.0f,1.0f,0.0f);                    // Rotier das Dreieck auf der Y-Achse ( NEU )
```

Der nächste Code-Ausschnitt wurde nicht geändert. Er zeichnet ein farbverlaufenes Dreieck. Das Dreieck wird auf der rechten Seite des Bildschirms gezeichnet und wird auf seiner Y-Achse rotieren, was eine Rotation von links nach rechts bedeutet.

```
glBegin(GL_TRIANGLES);          // Fange an ein Dreieck zu zeichnen
        glColor3f(1.0f,0.0f,0.0f);      // Setze den obersten Punkt des Dreeicks auf Rot
        glVertex3f( 0.0f, 1.0f, 0.0f);  // Erster Punkt des Dreiecks
        glColor3f(0.0f,1.0f,0.0f);      // Setze den linken Punkt des Dreiecks auf Grün
        glVertex3f(-1.0f,-1.0f, 0.0f);  // Zweiter Punkt des Dreiecks
        glColor3f(0.0f,0.0f,1.0f);      // Setze den rechten Punkt des Dreiecks auf Blau
        glVertex3f( 1.0f,-1.0f, 0.0f);  // Dritter Punkt des Dreiecks
glEnd();                        // Fertig mit dem Zeichnen des Dreiecks
```

Sie werden im folgenden Code bemerken, dass wir ein weiteres glLoadIdentity() hinzugefügt haben. Wir machen das, um die Ansicht (View) zu resetten. Wenn wir die Ansicht nicht resetten und wir danach die Translation des Objekt nach dessen Rotation ausführen, würden Sie ziemlich unerwartete Ergebnisse erhalten. Da die Achsen rotiert wurden, zeigen sie wohl nicht mehr in die Richtung, in die Sie denken. Wenn wir also eine Verschiebung nach links auf der X-Achse haben, könnten wir statt dessen nach oben oder unten bewegen, abhängig davon, wie weit wir die jeweiligen Achsen rotiert haben. Versuchen Sie, die glLoadIdentity() Zeile herauszunehmen und schauen Sie, was passiert.

Wurde die Szene resettet, verläuft X von links nach rechts, Y von oben nach unten und Z von innen nach aussen und wir führen die Verschiebung aus. Sie werden bemerken, dass wir nur 1.5 nach rechts bewegen, statt 3.0, wie wir es in der letzten Lektion gemacht haben. Wenn wir den Screen resetten, bewegt sich unser Fokus in den Mittelpunkt des Bildschirms. Das bedeutet, wir befinden uns nicht 1.5 Einheiten nach links, sondern wir sind wieder bei 0.0. Um also 1.5 Einheiten nach Rechts (vom Ursprung) zu kommen, müssen wir nicht erst 1.5 Einheiten nach links zu bewegen, um zum Mittelpunkt zu gelangen und dann erst 1.5 Einheiten nach rechts (insgesamt also 3.0), sondern wir müssen einfach nur vom Mittelpunkt aus nach rechts bewegen, was nur 1.5 Einheiten sind.

Nachdem wir uns zu unserem neuen Ziel auf der rechten Seite des Bildschirms bewegt haben, rotieren wir das Quadrat auf der X-Achse. Das hat zur Folge, dass das Qudrat auf und ab rotiert.

```
glLoadIdentity();               // Resette die aktuelle Modelview Matrix

// 1.5 Einheiten nach links und dann 6 Einheiten in den Bildschirm hinein
glTranslatef(1.5f,0.0f,-6.0f);
glRotatef(rquad,1.0f,0.0f,0.0f); // Rotiere das Rechteck auf der X-Achse ( NEU )
```

Dieser Code-Ausschnitt bleibt der selbe. Er zeichnet ein blaues Quadrat. Das Qudrat wird auf der rechten Seite des Bildschirm gezeichnet.

```
        glColor3f(0.5f,0.5f,1.0f); // Setzt die Farbe auf einen netten Blauton

        glBegin(GL_QUADS);    // Fange an ein Quadrat zu zeichnen
                glVertex3f(-1.0f, 1.0f, 0.0f); // Oben links des Quadrats
                glVertex3f( 1.0f, 1.0f, 0.0f); // Oben rechts des Quadrats
                glVertex3f( 1.0f,-1.0f, 0.0f); // Unten rechts des Quadrats
                glVertex3f(-1.0f,-1.0f, 0.0f); // Unten links des Quadrats
        glEnd();                // Quadrat fertig gezeichnet
```

Die nächsten beiden Zeilen sind neu. Stellen Sie sich rtri und rquad als Container vor. Am Anfang unseres Programms haben wir die Containter gemacht (GLfloat rtri, and GLfloat rquad). Als wir die Container erzeugt haben, haben Sie noch nichts enthalten. Die erste folgende Zeile ADDIERT 0.2 in diesen Container. Demnach enthält der Wert in dem rtri Container jedesmal nach diesem Code-Ausschnitt einen um 0.2 erhöhten Wert. Der rquad Container wird um 0.15 vermindert. Demnach enthält der rquad Container jedesmal einen um 0.15 verminderten Wert. Das Herunterzählen hat zur Folge, dass das Objekt in die entgegengesetzte Richtung rotiert, als wenn wir raufzählen würden.

Versuchen Sie das + in ein + in der folgenden Zeile zu ändern, um zu sehen, wie das Objekt in die andere Richtung rotiert. Versuchen Sie die Werte von 0.2 auf 1.0 zu ändern. Je höher die Zahl ist, um so schneller rotiert das Objekt. Je kleiner die Zahl, desto langsamer wird rotiert.

```
        rtri+=0.2f;      // Inkrementiere die Rotations Variable für das Dreieck ( NEU )
        rquad-=0.15f;  // Dekrementiere die Rotations Variable für das Quadrat ( NEU )

        return TRUE;    // Weiter geht's
}
```

Zu guter letzt ändern Sie den Code um zwischen Fenster- und Fullscreen-Modus zu wechseln, so dass der Titel des Fensters korrekt ist.

```
if (keys[VK_F1]) // Wurde F1 gedrückt?
{
        keys[VK_F1]=FALSE;              // Wenn ja, setze die Taste auf FALSE
        KillGLWindow();                 // Kill unser aktuelles Fenster
        fullscreen=!fullscreen;         // Wechsel zwischen Fullscreen und Fester-Modus

        // Erzeuge unser OpenGL-Fenster neu ( Modifiziert )
        if (!CreateGLWindow("NeHe's Rotation Tutorial",640,480,16,fullscreen))
        {
                return 0; // Beenden, wenn das Fenster nicht erzeugt wurde
        }
}
```

In diesem Tutorial habe ich so detailiert wie möglich versucht zu erklären, welche Schritte notwendig sind, um Objekte um eine Achse rotieren zu lassen. Spielen Sie ein wenig mit dem Code herum, versuchen Sie Objekte um die Z-Achse, die x & Y Achse oder alle drei rotieren zu lassen :) Wenn Sie Anmerkungen oder Fragen haben, schreiben Sie mir bitte eine E-Mail. Wenn Sie denken, ich habe etwas nicht korrekt kommentiert oder der Code könnte an einigen Stellen besser gemacht werden, lassen Sie es micht wissen. Ich möchte das beste OpenGL Tutorial machen. Ich bin an Ihrem Feedback interessiert.

Jeff Molofee (NeHe)

* DOWNLOAD Visual C++ Code für diese Lektion.
* DOWNLOAD ASM Code für diese Lektion. (Conversion by Foolman)
* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD BeOS Code für diese Lektion. (Conversion by Rene Manqueros)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Sabine Felsinger)
* DOWNLOAD VB.Net CsGL Code für diese Lektion. (Conversion by X)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Euphoria Code für diese Lektion. (Conversion by Evan Marshall)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD Genu Code für diese Lektion. (Conversion by Louis-Charles Dumais)
* DOWNLOAD GLUT Code für diese Lektion. (Conversion by Andy Restad)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Java/SWT Code für diese Lektion. (Conversion by Victor Gonzalez)
* DOWNLOAD Jedi-SDL Code für diese Lektion. (Conversion by Dominique Louis)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Perl Code für diese Lektion. (Conversion by Cora Hussey)
* DOWNLOAD Python Code für diese Lektion. (Conversion by John Ferguson)
* DOWNLOAD REALbasic Code für diese Lektion. (Conversion by Thomas J. Cunningham)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Jon DuBois)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Ross Dawson)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.

Lektion 05 - 3D Objekte

Aufbauend auf dem letzten Tutorial, machen wir aus unseren Objekten nun ECHTE 3D Objekte, statt nur 2D Objekte in einer 3D Welt. Wir erreichen dsa, indem wir links, hinten und rechts dem Dreieck eine Seite hinzufügen und dem Quadrat eine Seite links, rechts, hinten, oben und unten. Dadurch machen wir aus dem Dreieck eine Pyramide und aus dem Qudrat ein Würfel.

Die Farbe der Pyramide lassen wir verlaufen und der Würfel bekommt für jede Seite eine eigene verschiedene Farbe.

```
int DrawGLScene(GLvoid)                 // Hier kommt der ganze Zeichnen-Kram hin
{
        // Löscht den Bildschirm und den Depth-Buffer
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glLoadIdentity();               // Resettet die Ansicht (View)
        glTranslatef(-1.5f,0.0f,-6.0f); // In den Bildschirm hineinbewegen und dann links

        glRotatef(rtri,0.0f,1.0f,0.0f); // Rotiere die Pyramide um seine Y-Achse

        glBegin(GL_TRIANGLES);          // Fange an die Pyramide zu zeichnen
```

Einige von Ihnen haben den Code aus dem letzten Tutorial genommen und eigene 3D-Objekte gemacht. Eine Sache, die ich ziemlich häufig gefragt wurde, war "wie kommt es, dass mein Objekt sich nicht um die eigene Achse dreht? Es sieht so aus, als ob sie über den ganzen Bildschirm rotieren würden." Damit Ihr Objekt sich um eine Achse drehen kann, muss es UM diese Achse HERUM aufgebaut sein. Sie müssen daran denken, dass das Zentrum eines Objekts 0 auf der X-Achse, 0 auf der Y-Achse und 0 auf der Z-Achse sein sollte.

Der folgende Code erzeugt eine Pyramide um eine zentrale Achse. Die Spitze der Pyramide liegt eine Einheit über dem Zentrum, der Boden der Pyramide liegt eine unter dem Zentrum. Die Spitze liegt genau in der Mitte (null) und die unteren Punkte liegen eine Einheit links und eine rechts vom Zentrum.

Beachten Sie, dass alle Dreiecke in einer gegen den Uhrzeigersinn verlaufenden Rotation gezeichnet werden. Das ist wichtig und wird in einem späteren Tutorial erklärt, aber zum jetzigen Zeitpunkt müssen Sie nur wissen, dass es guter Stil ist, Objekte entweder im Uhrzeigersinn oder gegen den Uhrzeigersinn zu erzeugen, aber Sie sollte nicht beides mischen, es sei denn, Sie haben einen driftigen Grund dafür.

Wir beginnen mit dem Zeichnen der vorderen Seite. Da all Seiten den obersten Punkt gemeinsam haben, machen wir ihn jeweils Rot für jedes Dreieck. Die Farbe der unteren beiden Punkt des Dreiecks werden andere Farben haben. Die vordere Seite wird einen grünen linken Punkt und einen blauen rechten Punkt haben. Dann wird das Dreieck auf der rechte Seite einen blauen linken Punkt und einen grünen rechten Punkt haben. Dadurch haben wir am Boden jeweils gleichfarbige Eckpunkte.

```
glColor3f(1.0f,0.0f,0.0f);          // Rot
glVertex3f( 0.0f, 1.0f, 0.0f);      // Spitze des Dreiecks (vorne)
glColor3f(0.0f,1.0f,0.0f);          // Grün
glVertex3f(-1.0f,-1.0f, 1.0f);      // Linke Seite des Dreiecks (vorne)
glColor3f(0.0f,0.0f,1.0f);          // Blau
glVertex3f( 1.0f,-1.0f, 1.0f);      // Rechte Seite des Dreiecks (vorne)
```

Nun zeichnen wir die rechte Seite. Beachten Sie, dass die beiden unteren Punkte eine Einheit rechts vom Zentrum gezeichnet und die Spitze eine Einheit nach oben auf der Y-Achse und rechts in die Mitte auf der X-Achse gezeichnet wird. Dadurch neigt sich die Seite vom Mittelpunkt von der Spitze auf die rechte Seite des Bildschirms hinunter zum Boden.

Beachten Sie, dass der linke Punkt diesmal blau ist. Dadurch, dass wir ihn blau zeichnen, wird er die selbe Farbe haben, wir die rechte untere Ecke der Vorderseite.

Beachten Sie auch, wie die restlichen drei Seite innerhalb des selben glBegin(GL_TRIANGLES) und glEnd() Paars eingefügt werden, genauso, wie die erste Seite. Da wir das komplette Objekt aus Dreiecken erstellen, weiß OpenGL, dass alle drei Punkte ein Dreeick gezeichnet wird. Wenn die ersten drei Punkte gezeichnet wurden und drei weiter Punkte warten, wird angenommen, dass ein weiteres Dreieck gezeichnet werden soll. Wenn Sie vier statt drei Punkten angeben, würde OpenGL die ersten drei Punkte zeichnen und annehmen, dass der vierte Punkt der Anfang eines neuen Dreiecks ist. Es würde kein Quadrat gezeichnet werden. Stellen Sie also sicher, dass Sie nicht zufällig zuviele Punkte einfügen.

```
glColor3f(1.0f,0.0f,0.0f);          // Rot
glVertex3f( 0.0f, 1.0f, 0.0f);      // Spitze des Dreiecks (rechts)
glColor3f(0.0f,0.0f,1.0f);          // Blau
glVertex3f( 1.0f,-1.0f, 1.0f);      // Linke Seite des Dreiecks (rechts)
glColor3f(0.0f,1.0f,0.0f);          // Grün
glVertex3f( 1.0f,-1.0f, -1.0f);     // Rechte Seite des Dreiecks (rechts)
```

Nun für die Rückseite. Erneut Farben wechseln. Der linke Punkt ist nun wieder grün, da die gemeinsame Ecke mit der rechten Seite auch grün ist.

```
glColor3f(1.0f,0.0f,0.0f);          // Rot
glVertex3f( 0.0f, 1.0f, 0.0f);      // Spitze des Dreiecks (hinten)
glColor3f(0.0f,1.0f,0.0f);          // Grün
glVertex3f( 1.0f,-1.0f, -1.0f);     // Linke Seite des Dreiecks (hinten)
glColor3f(0.0f,0.0f,1.0f);          // Blau
glVertex3f(-1.0f,-1.0f, -1.0f);     // Rechte Seite des Dreiecks (hinten)
```

Letztendlich zeichnen wir die linke Seite. Die Farben wechseln ein letztes Mal. Der linke Punkt ist blau und verläuft mit dem rechten Punkt der Hinterseite. Der rechte Punkt ist grün und verläuft mit dem linken Punkt der Vorderseite.

Wir sind fertig, mit dem Zeichnen der Pyramide. Da die Pyramide nur um die Y-Achse rotiert, werden wir niemals den Boden sehen, weshalb auch kein Grund besteht, einen zu zeichnen. Wenn Sie etwas

herumexperimentieren wollen, versuchen Sie, einen Boden hinzuzufügen, indem Sie ein Quadrat benutzen und rotieren dann um die X-Achse, um zu sehen, ob Sie es auch richtig gemacht haben. Stellen Sie sicher, dass die für die einzelnen Ecken verwendeten Farben des Quadrats mit den Farben der vier Ecken der Pyramide übereinstimmen.

```
        glColor3f(1.0f,0.0f,0.0f);          // Rot
        glVertex3f( 0.0f, 1.0f, 0.0f);      // Spitze des Dreiecks (links)
        glColor3f(0.0f,0.0f,1.0f);          // Blau
        glVertex3f(-1.0f,-1.0f,-1.0f);      // Linke Seite des Dreiecks (Links)
        glColor3f(0.0f,1.0f,0.0f);          // Grün
        glVertex3f(-1.0f,-1.0f, 1.0f);      // Rechte Seite des Dreiecks (links)
    glEnd();    // Fertig mit zeichnen der Pyramide
```

Nun werden wir den Würfel zeichnen. Er besteht aus sechs Quadraten. All diese Quadrate werden gegen den Uhrzeigersinn gezeichnet. Das bedeutet, dass der erste Punkt oben recht, der zweite Punkt oben links, der dritte Punkt unten links und letztendlich der letzte Punk unten rechts gezeichnet wird. Wenn wir die Hinterseit zeichnen, wird es so aussehen, als ob wir mit dem Uhrzeigersinn zeichnen würden, aber Sie müssen im Hinterkopf behalten, dass, wenn wir hinter dem Würfel wären und auf ihn schauen würden, die linke Seite des Bildschirms tatsächlich die rechte Seite des Quadrats ist und die rechte Seite des Bildschirms würde tatsächlich die linke Seite des Quadrats sein.

Beachten Sie, dass wir den Würfel ein klein wenig weiter in den Bildschirm hineinschieben werden, in dieser Lektion. Damit sieht der Würfel etwa genauso groß aus wie die Pyramide. Wenn Sie nur 6 Einheiten in den Bildschirm hineingehen würde, würde der Würfel wesentlich größer aussehen, als die Pyramide und Teile könnten am Rande des Bildschirms abgeschnitten werden. Sie können mit den Einstellungen ein wenig herumspielen und sich anschauen, wir der Würfel kleiner wird, wenn Sie ihn weiter in den Bildschirm verschieben und größer, wenn Sie ihn näher heran holen. Der Grund dafür ist die Perpektive. Objekte sollte in der Entfernung kleiner erscheinen :)

```
glLoadIdentity();
glTranslatef(1.5f,0.0f,-7.0f);              // nach rechts und in den Bildschirm hinein bewegen

glRotatef(rquad,1.0f,1.0f,1.0f);            // Rotiere den Würfel um X, Y & Z

glBegin(GL_QUADS);                          // Fange an den Würfel zu zeichnen
```

Wir werden mit der Oberseite des Würfels anfangen. Wir bewegen uns eine Einheit nach oben, vom Zentrum des Würfels aus gesehen. Beachten Sie, dass die Y-Achse immer gleich eins ist. Dann zeichnen wir das Qudrat auf der Z-Ebenen. Das bedeutet in den Bildschirm hinein. Wir beginnen beim Zeichnen mit dem oberen rechten Punkt der Oberseite des Würfels. Der obere rechte Punkt würde eine Einheit rechts und eine Einheit in den Bildschirm hineingehen. Der zweite Punkt würde eine Einheit nach links und eine Einheit in den Bildschirm hinein sein. Nun müssen wir die Unterkante des Quadrats, zum Betrachter hin, zeichnen. Um das zu machen, bewegen wir uns nicht in den Bildschirm hinein, sondern wir bewegen uns eine Einheit aus dem Bildschirm heraus? Verstanden?

```
glColor3f(0.0f,1.0f,0.0f);          // Setze die Farbe auf Grün
glVertex3f( 1.0f, 1.0f,-1.0f);      // Oben Rechts des Quadrats (oben)
glVertex3f(-1.0f, 1.0f,-1.0f);      // Oben Links des Quadrats (oben)
glVertex3f(-1.0f, 1.0f, 1.0f);      // Unten Links des Quadrats (oben)
glVertex3f( 1.0f, 1.0f, 1.0f);      // Unten Rechts des Quadrats (oben)
```

Die Unterseite wird exakt genauso gezeichnet wie die Oberseite, da es aber die Unterseite ist, zeichnen wir sie eine Einheit unter dem Zentrum des Würfels. Beachten Sie, das die Y-Achse immer gleich -1 ist. Wenn wir uns unter dem Würfel befinden würden und auf das Quadrat schauen würden, dass den Boden darstellt, würden Sie bemerken, dass die obere rechte Ecke die Ecke ist, die am nähsten zum Betrachter ist, demnach, statt die entfernten Punkte zuerst zu zeichnen, zeichnen wir den dichtesten Punkt zum Betrachter als erstes, dann auf der linken Seite dem Betrachter am nächsten Punkt und gehen dann in den Bildschirm hinein, um die unteren beiden Punkte zu zeichnen.

Wenn Sie sich nicht darüm kümmern, in welcher Reihenfolge die Polygone gezeichnet werden (im Uhrzeigersinn oder eben nicht), könnten Sie auch einfach den selben Code für die Oberseite kopieren und die Y-Achse auf -1 setzen und es würde funktionieren, allerdings würde das ignorieren der Reihenfolge seltsame Resultate mit sich bringen, wenn man später Sachen wie Textur-Mapping einsetzen will.

```
glColor3f(1.0f,0.5f,0.0f);          // Setze die Farbe auf Orange
glVertex3f( 1.0f,-1.0f, 1.0f);      // Oben Rechts des Quadrats (unten)
glVertex3f(-1.0f,-1.0f, 1.0f);      // Oben Links des Quadrats (unten)
glVertex3f(-1.0f,-1.0f,-1.0f);      // Unten Links des Quadrats (unten)
glVertex3f( 1.0f,-1.0f,-1.0f);      // Unten Rechts des Quadrats (unten)
```

Nun zeichnen wir die Vorderseite. Dazu kommen wir eine Einheit aus dem Bildschirm heraus, vom Zentrum weg, um die Vorderseite zu zeichnen. Beachten Sie, dass die Z-Achse immer gleich null ist. In der Pyramide war die Z-Achse nicht immer eins. An der Spitze war die Z-Achse gleich null. Wenn Sie versucht haben, die Z-Achse im folgenden Code auf null zu setzen, werden Sie bemerkt haben, dass die Ecke, die Sie verändert haben, sich in den Bildschirm neig. Das ist nicht gerade das, was wir nun wollen :)

```
glColor3f(1.0f,0.0f,0.0f);          // Setze die Farbe auf Rot
glVertex3f( 1.0f, 1.0f, 1.0f);      // Oben rechts des Quadrats (vorne)
glVertex3f(-1.0f, 1.0f, 1.0f);      // Oben links des Quadrats (vorne)
glVertex3f(-1.0f,-1.0f, 1.0f);      // Unten links des Quadrats (vorne)
glVertex3f( 1.0f,-1.0f, 1.0f);      // Unten rechts des Quadrats (vorne)
```

Die hintere Seite ist genauso ein Quadrat wie die vordere Seite, aber sie liegt tiefer im Bildschirm. Beachten Sie, dass die Z-Achse nun -1 für alle Punkte ist.

```
glColor3f(1.0f,1.0f,0.0f);          // Setze die Farbe auf gelb
glVertex3f( 1.0f,-1.0f,-1.0f);      // Unten links des Quadrats (hinten)
glVertex3f(-1.0f,-1.0f,-1.0f);      // Unten rechts des Quadrats (hinten)
glVertex3f(-1.0f, 1.0f,-1.0f);      // Oben rechts des Quadrats (hinten)
glVertex3f( 1.0f, 1.0f,-1.0f);      // Oben links des Quadrats (hinten)
```

Nun müssen wir nur noch zwei weitere Qudrate zeichnen und wir sind fertig. Wie immer, werden Sie bemerken, dass eine Achse für alle Punkte immer gleich ist. In diesem Fall ist die X-Achse immer -1. Das ist so, weil wir immer links vom Zentrum zeichnen, weil es die linke Seite (des Würfels) ist.

```
        glColor3f(0.0f,0.0f,1.0f);          // Setze die Farbe auf Blau
        glVertex3f(-1.0f, 1.0f, 1.0f);      // Oben Rechts des Quadrats (Links)
        glVertex3f(-1.0f, 1.0f,-1.0f);      // Oben Links des Quadrats (Links)
        glVertex3f(-1.0f,-1.0f,-1.0f);      // Unten links des Quadrats (Links)
        glVertex3f(-1.0f,-1.0f, 1.0f);      // Unten Rechts des Quadrats (Links)
```

Das ist die letzte Seite um den Würfel fertig zu stellen. Die x-Achse ist jeweils gleich 1. Gezeichnet wird gegen den Uhrzeigersinn. Wenn Sie wollen, können Sie diese Seite weglassen und eine Kiste draus machen :)

Oder wenn Sie etwas herumexperimentieren wollen, können Sie jeweils versuchen die Farben für jeden Eckpunkt des Würfels zu verändern, um einen Farbverlauf wie bei der Pyramide zu erzeugen. Sie können ein Beispiel dafür sehen, indem Sie Evil's erstes GL demo von meiner Seite herunterladen. Starten Sie es und drücken Sie TAB. Sie werden einen schön gefärbten Würfel sehen mit einem schönen Farbverlauf auf allen Seiten.

```
        glColor3f(1.0f,0.0f,1.0f);          // Setze die Farbe auf violett
        glVertex3f( 1.0f, 1.0f,-1.0f);      // Oben Rechts des Quadrats (Rechts)
        glVertex3f( 1.0f, 1.0f, 1.0f);      // Oben Links des Quadrats (Right)
        glVertex3f( 1.0f,-1.0f, 1.0f);      // Unten Links des Quadrats (Right)
        glVertex3f( 1.0f,-1.0f,-1.0f);      // Unten rechts des Quadrats (Rechts)
 glEnd(); // Fertig mit dem Zeichnen von Quadraten

        rtri+=0.2f;       // Inkrementiere die Rotations-Variable für das Dreieck
        rquad-=0.15f;  // Dekrementiere die Roatations-Variable für das Quadrat

        return TRUE;    // Weiter geht's
}
```

Am Ende dieses Tutorials sollten Sie ein besseres Verständnis haben, wie Objekte im 3D-Raum erzeugt werden. Sie müssen sich den OpenGL Screen als ein gigantisches Stück Graphen-Papier vorstellen, mit vielen durchsichtigen Schichten dahinter. Fast wie ein gigantischer Würfel, der aus Punkten besteht. Einige dieser Punkte verlaufen von links nach recht, einige von oben nach unten und einige von vorne nach hinten, innerhalb des Würfels. Wenn Sie sich die Tiefe in den Bildschirm hinein vorstellen können, sollten Sie keine Probleme habe, neue 3D Objekte zu erstellen.

Wenn Sie Probleme beim Verstehen des 3D-Raumes haben, seien Sie nicht frustriert. Es kann am Anfang sehr schwierig sein. Ein Objekt wie der Würfel ist ein gutes Beispiel, von dem man lernen kann. Wenn Sie es bemerkt haben, wird die hintere Seite genau gleich gezeichnet wie die vordere Seite, nur einfach tiefer in den Bildschirm hinein. Spielen Sie mit dem Code ein wenig herum und wenn Sie es nicht verstehen, schicken Sie mir eine E-Mail und ich versuchen Ihnen ihre Fragen zu beantworten.

Jeff Molofee (NeHe)

* DOWNLOAD Visual C++ Code für diese Lektion.

* DOWNLOAD ASM Code für diese Lektion. (Conversion by Foolman)
* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Sabine Felsinger)
* DOWNLOAD VB.Net CsGL Code für diese Lektion. (Conversion by X)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Euphoria Code für diese Lektion. (Conversion by Evan Marshall)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD GLUT Code für diese Lektion. (Conversion by Andy Restad)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Java/SWT Code für diese Lektion. (Conversion by Victor Gonzalez)
* DOWNLOAD Jedi-SDL Code für diese Lektion. (Conversion by Dominique Louis)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Perl Code für diese Lektion. (Conversion by Cora Hussey)
* DOWNLOAD Python Code für diese Lektion. (Conversion by Tony Colston)
* DOWNLOAD REALbasic Code für diese Lektion. (Conversion by Thomas J. Cunningham)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Jon DuBois)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Ross Dawson)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.

Lektion 06 - Textur Mapping

Man hat viele Vorteile, wenn man weiß, wie man texturiert. Stellen Sie sich for, Sie wollen eine Raket durch den Screen fliegen lassen. Mit den bisherigen Tutorials würden wir die Rakete wahrscheinlich aus Polygonen und lauter Farben machen. Mit Textur Mapping können Sie ein Foto einer Rakete nehmen und dieses Foto durch den Screen fliegen lassen. Was meinen Sie, was sieht besser aus? Ein Foto oder ein Objekt bestehend aus Dreiecken und Quadraten? Indem Sie Textur Mapping verwenden, wird es nicht nur besser aussehen, Ihr Programm wird auch schneller laufen. Die texturierte Rakete wäre nur ein einzelnes Quadrat, was sich über den Screen bewegt. Eine Rakete aus Polygonen könnte aus hunderten oder tausenden Polygonen bestehen. Das einzelne texturierte Quadrat wird wesentlich weniger Prozessorleistung benötigen.

Fangen wir mit dem hinzufügen einiger weniger Zeilen Code am Anfang von Lektion eins an. Die erste neue Zeile ist #include . Nachdem wir diesen Header hinzugefügt haben, können wir mit Dateien arbeiten. Um später im Code fopen() verwenden zu können, müssen wir diese Zeile einfügen. Dann fügen wir drei neue Fließkommavariablen ein... xrot, yrot und zrot. Diese Variablen werden verwendet, um den Würfel auf der X, Y und Z-Achse zu rotieren. Die letzte Zeile GLuint texture[1] initialisiert den Speicher für eine Textur. Wenn Sie mehr als eine Textur laden wollen, würden Sie die Zahl eins in die Anzahl der zu ladenden Texturen umwandeln.

```
#include < windows.h>        // Header Datei für Windows
#include < stdio.h>          // Header Datei für die Standard Ein-/Ausgabe
#include < gl\gl.h>          // Header Datei für die OpenGL32 Library
#include < gl\glu.h>         // Header Datei für die GLu32 Library
#include < gl\glaux.h>       // Header Datei für die GLaux Library

HDC hDC=NULL;               // Privater GDI Device Context
HGLRC hRC=NULL;             // Permanenter Rendering Context
HWND hWnd=NULL;             // Enthält unser Fenster-Handle
HINSTANCE hInstance;        // Enthält die Instanz der Applikation

bool keys[256];             // Array das für die Tastatur Routine verwendet wird
bool active=TRUE;           // Fenster Aktiv Flag ist standardmäßig auf TRUE gesetzt
bool fullscreen=TRUE;       // Fullscreen Flag

GLfloat xrot;               // X Rotation ( NEU )
GLfloat yrot;               // Y Rotation ( NEU )
GLfloat zrot;               // Z Rotation ( NEU )

GLuint texture[1]; // Speicher für eine Textur ( NEU )

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); // Deklaration für WndProc
```

Nun, direkt nach dem obigen Code und bevor ReSizeGLScene() wollen wir folgenden Codeabschnitt einfügen. Dieser Code lädt eine Bitmap-Datei. Wenn die Datei nicht existiert, wird NULL zurückgegeben, was bedeutet, dass die Textur nicht geladen werden konnte. Bevor ich anfange den Code zu erklären, gibt es ein paar SEHR wichtige Dinge, die Sie über die Bilder wissen müssen, die Sie als Textur verwenden wollen. Die Bild-Höhe MUSS zur Basis 2 sein. Die Breite und Höhe müssen

mindesten 64 Pixel haben und aus kompatibilitäts Gründen sollten sie nicht mehr als 256 Pixel haben. Wenn das Bild, dass Sie verwenden wollen, nicht 64, 128 oder 256 Pixel in der Breite oder Höhe hat, ändern Sie die Größe mit einem Grafikprogramm. Es gibt zwar Wege, um diese Beschränkungen zu umgehen, aber wir bleiben erst einmal bei den Standard Textur Größen.

Als erstes erzeugen wir ein Datei Handle. Ein Handle ist ein Wert, der eine Ressource identifiziert, so dass unser Programm zugriff darauf hat. Wir setzen das Handle anfangs auf NULL.

```
AUX_RGBImageRec *LoadBMP(char *Filename)        // Lädt ein Bitmap
{
        FILE *File=NULL;                        // Datei Handle
```

Als nächstes überprüfen wir, dass ein Dateiname übergeben wurde. Die Person kann LoadBMP() vielleicht ohne die zu ladende Datei aufgerufen haben, weshalb wir das lieber überprüfen. Wir wollen ja nicht versuchen, nichts zu laden :)

```
        if (!Filename)          // gehe sicher, dass ein Dateiname übergeben wurde
        {
                return NULL;    // wenn nicht, gebe NULL zurück
        }
```

Wenn ein Dateiname übergeben wurde, überprüfen wir, ob die Datei existiert. Die folgende Zeile versucht die Datei zu öffnen.

```
        File=fopen(Filename,"r");       // überprüfe, ob die Datei existiert.
```

Wenn wir die Datei öffnen konnten wird diese wohl auch existieren. Wir schließen die Datei mit fclose(File) und geben die Bilddaten zurück. auxDIBImageLoad(Filename) liest die Daten ein.

```
        if (File) // Existiert die Datei?
        {
                fclose(File);           // Schließe das Handle

                // Lädt das Bitmap und gibt einen Zeiger zurück
                return auxDIBImageLoad(Filename);
        }
```

Wenn wir die Datei nicht öffnen konnten, geben wir NULL zurück. Das bedeutet, die Datei konnte nicht geladen werden. Später überprüfen wir im Programm, ob die Datei geladen wurde. Wenn nicht, werden wir das Programm mit einer Fehlermeldung schließen.

```
        return NULL;    // Wenn das Laden fehl schlug, gebe NULL zurück
}
```

Dieser Codeabschnitt lädt das Bitmap (indem der obige Code aufgerufen wird) und konvertiert es in eine Textur.

```
int LoadGLTextures()    // Lade Bitmaps und konvertiere in Texturen
{
```

Wir initialisieren eine Variable namens Status. Wir werden diese Variable dazu verwenden, um zu verfolgen, ob wir das Bitmap laden und in eine Textur verwandeln können. Wir setzen den Standardwert von Status auf FALSE (was bedeutet, dass bisher nichts geladen oder erzeugt wurde).

```
        int Status=FALSE;       // Status Indikator
```

Nun erzeugen wir ein Bild Feld, wo wir unser Bitmap drin speichern können. Das Feld wird die Bitmap Breite, Höhe und Daten enthalten.

```
        AUX_RGBImageRec *TextureImage[1];          // erzeuge Speicherplatz für die Textur
```

Wir löschen das Bild Feld, nur um sicher zu gehen, dass es leer ist. We clear the image record just to make sure it's empty.

```
        memset(TextureImage,0,sizeof(void *)*1);     // Setze den Zeiger auf NULL
```

Nun laden wir das Bitmap und konvertieren es in eine Textur. TextureImage[0]=LoadBMP("Data/NeHe.bmp") ruft unseren LoadBMP() Code auf. Die Datei names NeHe.bmp im Data Verzeichnis wird geladen. Wenn alles glatt läuft, werden die Bild Daten in TextureImage[0] gespeichert, Status auf TRUE gesetzt und wir fangen an, unsere Textur zu erzeugen.

```
// Lade das Bitmap, prüfe auf Fehler, wenn Bitmap nicht gefunden wurde, beende
if (TextureImage[0]=LoadBMP("Data/NeHe.bmp"))
{
        Status=true;            // Setze Status auf TRUE
```

Nun, da wir die Bild-Datei nach TextureImage[0] geladen haben, erzeugen wir eine Textur aus diesen Daten. Die erste Zeile glGenTextures(1, &texture[0]) teilt OpenGL mit, dass wir einen Textur Namen generieren wollen (erhöhen Sie die Nummer, wenn Sie mehr als eine Textur laden wollen). Erinnern Sie sich, dass wir ganz am Anfang des Tutorials Platz für eine Textur erzeugt haben und zwar mit der Zeile GLuint texture[1]. Obwohl Sie sich vielleicht denken, dass die erste Textur in &texture[1] gespeichert wird, anstatt in &texture[0], so ist dem nicht so. Der tatsächliche erste Speicherplatz ist 0. Wenn wir zwei Texturen haben wollen, würden wir GLuint texture[2] benutzen und die zweite Textur würde in texture[1] gespeichert werden.

Die zweite Zeile, glBindTexture(GL_TEXTURE_2D, texture[0]) teilt OpenGL mit, die genannte Textur texture[0] an ein Textur-Ziel zu binden. 2D Texturen haben sowohl beides, Höhe (auf der Y-Achse) und

Breite (auf der X-Achse). Die Hauptfunktion von glBindTexture ist es, einen Textur-Namen mit Textur-Daten zu verbinden. In diesem Fall teilen wir OpenGL mit, dass Speicher in &texture[0] verfügbar ist. Wenn wir eine Textur erzeugen, wird diese in dem Speicher gespeichert, welchen &texture[0] referenziert.

```
                glGenTextures(1, &texture[0]); // erzeuge die Textur

                // typische Textur Erzeugung indem Daten aus dem Bitmap verwendet werden
                glBindTexture(GL_TEXTURE_2D, texture[0]);
```

Als nächste erzeugen wir die aktuelle Textur. Die folgende Zeile teilt OpenGL mit, dass die Textur eine 2D Textur sein wird (GL_TEXTURE_2D). Null repräsentiert das Level an Details des Bildes und wird in der Regel auf Null gesetzt. Drei ist die Anzahl der Daten-Komponenten. Da das Bild aus Rot-Daten, Grün-Daten und Blau-Daten besteht, gibt es drei Komponenten. TextureImage[0]->sizeX ist die Breite der Textur. Wenn Sie die Breite kennen, können Sie sie hier angeben, aber es ist einfacher, den Computer das für Sie herausfinden zu lassen. TextureImage[0]->sizey ist die Höhe der Textur. Null ist die Grenze. Wird normalerweise auf 0 gelassen. GL_RGB teilt OpenGL mit, dass die Bilddaten, die wir verwenden, aus Rot, Grün und Blau Daten bestehen und zwar in dieser Reihenfolge. GL_UNSIGNED_BYTE bedeutet, dass die Daten des Bildes aus vorzeichenlosen (unsigned) Bytes bestehen und letztlich... TextureImage[0]->data teilt OpenGL mit, wo die Textur-Daten liegen.
In diesem Fall zeigt es auf die Daten die in TextureImage[0] gespeichert sind.

```
                // Generiere die Textur
                glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[0]->sizeX,
                                TextureImage[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
                                TextureImage[0]->data);
```

Die nächsten beiden Zeilen teilen OpenGL mit, welche Art von Filter verwendet werden soll, wenn das Bild größer (GL_TEXTURE_MAG_FILTER) oder gestreckter aks die original Textur auf dem Bildschirm erscheint oder wenn es kleiner (GL_TEXTURE_MIN_FILTER) als die eigentliche Textur auf dem Screen wird. Ich benutze in der Regel für beides GL_LINEAR. Das lässt die Textur für beides weich aussehen, sowohl in der Ferne als auch, wenn Sie nahe am Screen ist. Das verwenden von GL_LINEAR benötigt viel Arbeit vom Prozessor/der Grafikkarte, so dass, wenn Ihr System langsam ist, Sie vielleicht GL_NEAREST verwenden wollen. Eine Textur die mit GL_NEAREST gefiltert wird, erscheint klobig, wenn Sie gestreckt wird. Sie können auch eine Kombination der beiden ausprobieren. Dadurch werden Dinge in der Nähe gefiltert, nicht aber in der Distanz.

```
                // Linear Filtering
                glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
                glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        }
```

Nun geben wir jeglichen Speicher frei, den wir verwendet haben, um die Bitmap-Daten zu speichern. Wir überprüfen, ob die Bitmap-Daten in TextureImage[0] gespeichert wurden. Wenn dem so ist,

überprüfen wir, ob Daten gespeichert wurden. Wenn Daten gespeichert wurden, löschen wir diese. Dann geben wir die Bild-Struktur frei, um sicher zu gehen, dass jeglicher benutzter Speicher freigegeben wird.

```
        if (TextureImage[0])     // Wenn Textur existiert
        {
                if (TextureImage[0]->data)     // Wenn Textur-Image existiert
                {
                        free(TextureImage[0]->data);  // gebe Textur-Image Speicher frei
                }

                free(TextureImage[0]);          // gebe die Image-Struktur frei
        }
```

Wir geben den Status zurück. Wenn alles OK ist, wird die Variable Status gleich TRUE sein. Wenn irgendwas schief lieft, wird Status gleich FALSE sein.

```
                return Status;  // gebe Status zurück
        }
```

Ich habe ein paar Codezeilen in InitGL eingefügt. Ich schreibe hier nochmal den ganzen Code-Abschnitt nieder, so dass es einfach zu verstehen ist, welche Zeilen ich hinzugefügt habe und wo sie hinkommen. Die erste Zeile if (!LoadGLTextures()) ruft die obige Routine zum Laden des Bitmaps auf und erzeugt eine Textur daraus. Wenn LoadGLTextures() aus irgend einem Grund fehl schlagen sollte, liefert die nächste Codezeile FALSE zurück. Wenn alls OK war und die Textur erzeugt wurde, aktivieren wir 2D Textur Mapping. Wenn Sie vergessen Textur Mapping zu aktivieren, wird Ihr Objekt komplett in weiß erscheinen, was definitiv nicht gut ist.

```
int InitGL(GLvoid)       // Der ganze Setup Kram für OpenGL kommt hier rein
{
        if (!LoadGLTextures())  // Rufe Textur Lade Routine auf ( NEU )
        {
                return FALSE;  // wenn Textur nicht geladen wurde, gebe FALSE zurück ( NEU )
        }

        glEnable(GL_TEXTURE_2D);            // aktiviere Textur Mapping ( NEU )
        glShadeModel(GL_SMOOTH);            // aktiviere Smooth Shading
        glClearColor(0.0f, 0.0f, 0.0f, 0.5f);   // schwarzer Hintergrund
        glClearDepth(1.0f);                 // Depth Buffer initialisieren
        glEnable(GL_DEPTH_TEST);            // aktiviere Depth Test
        glDepthFunc(GL_LEQUAL);             // Die Art des auszuführenden Depth Test
        // wirklich nette Perspektiven Berechnungen
        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

        return TRUE;                        // Initialisierung war OK
}
```

Nun zeichnen wir den texturierten Würfel. Sie können den DrawGLScene Code mit dem folgenden Code ersetzen oder Sie können die neuen Code Zeilen in den original Code von Lektion 1 einfügen. Dieser Abschnitt wird gut kommentiert sein, so das er einfach zu versetehen ist. Die ersten beiden Codezeilen glClear() und glLoadIdentity() sind aus dem original Code der Lektion eins.

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) löscht den Bildschirm auf die Farbe, die wir in InitGL() ausgewählt haben. In diesem Fall wird der Screen auf schwarz gelöscht. Der Depth Buffer wird ebenso geleert. Die View wird dann mit glLoadIdentity() resettet.

```
int DrawGLScene(GLvoid)        // Hier kommt der ganze Zeichnen-Kram hin
{
        // Lösche den Bildschirm und den Depth-Buffer
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Resette die aktuelle Matrix
        glLoadIdentity();
        glTranslatef(0.0f,0.0f,-5.0f); // Bewege 5 Einheiten in den Screen hinein
```

Die folgenden drei Codezeilen lassen den Würfel auf der X-Achse, dann auf der Y-Achse und letzlich auf der Z-Achse rotieren. Um wieviel auf der jeweiligen Achse rotiert wird, hängt vom Wert ab, der in xrot, yrot und zrot gespeichert ist.

```
        glRotatef(xrot,1.0f,0.0f,0.0f);   // Rotiere auf der X Achse
        glRotatef(yrot,0.0f,1.0f,0.0f);   // Rotiere auf der Y Achse
        glRotatef(zrot,0.0f,0.0f,1.0f);   // Rotiere auf der Z Achse
```

Die nächste Codezeile wählt die Textur aus, die wir verwenden wollen. Wenn Sie mehr als eine Textur verwenden wollen, sollten Sie die Textur mit glBindTexture(GL_TEXTURE_2D, texture[nummer der Textur die sie verwenden wollen]) auswählen. Wenn Sie Texturen ändern wollen, würden Sie neue Texturen binden. Eine Sache, die man erwähnen sollte, ist, dass Sie Texturen NICHT innerhalb von glBegin() und glEnd() binden können, dass müssen Sie davor oder danach machen. Beachten Sie, wie wir glBindTextures verwenden, um anzugeben welche Textur erzeugt werden soll und eine bestimmte Textur ausgewählt werden soll.

```
glBindTexture(GL_TEXTURE_2D, texture[0]);   // wähle unsere Textur aus
```

Um ein Qudrat richtig zu texturieren, müssen Sie sicher stellen, dass die obere rechte Ecke der Textur auf die obere rechte Ecke des Quadrats gemapped wird. Die obere linke Ecke der Textur wird auf die obere linke Ecke des Qudrats gemapped, die untere rechte Ecke der Textur wird auf die untere rechte Ecke des Quadrats gemapped und letztendlich wird die untere linke Ecke der Textur auf die untere linke Ecke des Quadrats gemapped. Wenn die Ecken der Textur nicht mit denen des Quadrats übereinstimmen, erscheint das Bild vielleicht auf dem Kopf, spiegelverkehrt oder gar nicht.

Der erste Wert von glTexCoord2f ist die X Koordinate. 0.0f ist die linke Seite der Textur. 0.5f ist die Mitte der Textur und 1.0f ist die rechte Seite der Textur. Der zweite Wert von glTexCoord2f ist die Y Koordinat. 0.0f ist die untere Seite derTextur. 0.5f ist die Mitte der Textur und 1.0f ist die obere Seite der Textur.

So, nun wissen wir, dass die obere linke Koordinate einer Textur 0.0f auf der X-Achse und 1.0f auf der Y-Achse ist und der obere linke Vertex eines Quadrats ist -1.0f auf der X-Achse und 1.0f auf der Y-Achse. Alles was Sie jetzt noch machen müssen, ist das Anpassen der anderen drei Textur Koordinaten auf die restlichen drei Ecken des Quadrats.

Spielen Sie etwas mit den X und Y Wert von glTexCoord2f. Ändert man 1.0f in 0.5f wird nur die linke Hälfte der Textur von 0.0f (links) bis 0.5f (Mitte der Textur) gezeichnet. Ändert man 0.0f in 0.5f wird nur die rechte Hälfte der Textur von 0.5f (Mitte) bis 1.0f (rechts) gezeichnet.

```
glBegin(GL_QUADS);

// vordere Seite
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);// Unten links der Textur und des Quadrats
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f); // Unten rechts der Textur und des Quadrats
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, 1.0f); // Oben rechts der Textur und des Quadrats
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, 1.0f); // Oben links der Textur und des Quadrats

// hintere Seite
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);// Unten rechts der Textur und des Quadrats
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);// Oben rechts der Textur und des Quadrats
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f,  -1.0f);// Oben links der Textur und des Quadrats
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);// Unten links der Textur und des Quadrats

// obere Seite
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);// Oben links der Textur und des Quadrats
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);  // Unten links der Textur und des Quadrats
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);  // Unten rechts der Textur und des Quadrats
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f); // Oben rechts der Textur und des Quadrats

// untere Seite
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);// Oben rechts der Textur und des Quadrats
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);// Oben links der Textur und des Quadrats
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f); // Unten links der Textur und des Quadrats
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);// Unten rechts der Textur und des Quadrats

// rechte Seite
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);// Unten rechts der Textur und des Quadrats
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f); // Oben rechts der Textur und des Quadrats
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);  // Oben links der Textur und des Quadrats
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f); // Unten links der Textur und des Quadrats

// linke Seite
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);// Unten links der Textur und des Quadrats
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);// Unten rechts der Textur und des Quadrats
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);  // Oben rechts der Textur und des Quadrats
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);// Oben links der Textur und des Quadrats

glEnd();
```

Nun erhöhen wir die Werte von xrot, yrot und zrot. Versuchen Sie die Zahlen jeder Variablen zu verändern, um die die Variablen erhöht werden, um den Würfel schneller oder langsamer rotieren zu lassen oder versuchen Sie aus einem + ein - zu machen, um den Würfel in die andere Richtung rotieren zu lassen.

```
        xrot+=0.3f; // X Achsen Rotation
        yrot+=0.2f; // Y Achsen Rotation
        zrot+=0.4f; // Z Achsen Rotation

        return true; // weiter gehts
}
```

Sie sollten nun ein besseres Verständnis von Textur Mapping haben. Sie sollten in der Lage sein, die Fläche eines jeden Quadrats mit dem Bild Ihrer Wahl zu texturieren. Wenn Sie das Gefühl haben, 2D Textur Mapping verstanden zu haben, versuchen Sie sechs verschiedene Texturen für den Würfel zu verwenden.

Textur Mapping ist nicht so schwer zu verstehen, wenn man erst einmal die Textur Koordinaten verstanden hat. Wenn Sie Probleme haben, irgend einen Teil dieses Tutorials zu verstehen, lassen Sie es mich wissen. Entweder ich schreibe diesen Teil des Tutorials um oder ich werde Ihnen per E-Mail antworten. Viel Spaß dabei, eigene texturierte Szenen zu erzeugen :)

Jeff Molofee (**NeHe**)

* DOWNLOAD Visual C++ Code für diese Lektion.
* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Sabine Felsinger)
* DOWNLOAD VB.Net CsGL Code für diese Lektion. (Conversion by X)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Euphoria Code für diese Lektion. (Conversion by Evan Marshall)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD GLUT Code für diese Lektion. (Conversion by Kyle Gancarz)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Jedi-SDL Code für diese Lektion. (Conversion by Dominique Louis)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Visual C++ / OpenIL Code für diese Lektion. (Conversion by Denton Woods)
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Python Code für diese Lektion. (Conversion by John Ferguson)
* DOWNLOAD REALbasic Code für diese Lektion. (Conversion by Thomas J. Cunningham)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Brendan Burns)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Ross Dawson)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Peter De Tagyos)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.

Lektion 07 - Texturfilter, Beleuchtung und Tastatureingaben

In diesem Tutorial bringe ich Ihnen bei, wie Sie drei verschiedene Textur-Filter verwenden. Ich werde Ihnen beibringen, wie man ein Objekt mittels Tasten auf der Tastatur bewegt und ich werde Ihnen auch beibringen, wie Sie eine einfache Beleuchtung in Ihre OpenGL Szene einfügen. Ziemlich viel Stoff in diesem Tutorial, wenn Sie also noch mit den vorherigen Tutorials Probleme haben, gehen Sie zurück und schauen Sie sie nochmal an. Es ist sehr witchtig ein gutes Verständnis der Grundlagen zu haben, bevor man mit dem folgenden Code weitermacht.

Wir werden den Code aus Lektion eins erneut modizifizieren. Wie immer, werde ich bei grundlegenden Änderungen den kompletten Codeabschnitt, der geändert wurde, hinschreiben. Wir fangen damit an, ein paar neue Variablen zum Programm hinzuzufügen.

```
#include < windows.h>          // Header Datei für Windows
#include < stdio.h>            // Header Datei für Standard Ein-/Ausgabe ( HINZUGEFÜGT )
#include < gl\gl.h>            // Header Datei für die OpenGL32 Library
#include < gl\glu.h>           // Header Datei für die GLu32 Library
#include < gl\glaux.h>         // Header Datei für die GLaux Library

HDC hDC=NULL;                  // Privater GDI Device Context
HGLRC hRC=NULL;                // Permanenter Rendering Context
HWND hWnd=NULL;                // Enthält unser Fenster-Handle
HINSTANCE hInstance;           // Enthält die Instanz der Applikation

bool keys[256];                // Array das für die Tastatur Routine verwendet wird
bool active=TRUE;              // Fenster Aktiv Flag standardmäßig auf TRUE gesetzt
bool fullscreen=TRUE;          // Fullscreen Flag
```

Die folgenden Zeilen sind neu. Wir fügen drei boolean Variablen ein. BOOL bedeutet, dass die Variable nur TRUE (Wahr) oder FALSE (Falsch) sein kann. Wir erzeugen eine Variable names light, um zu verfolgen, ob die Beleuchtung an- oder ausgeschaltet ist. Die Variablen lp und fp werden benutzt, um zu sehen, ob die Taste 'L' oder 'F' gedrückt wurde. Ich werde später im Code erklären, warum wir diese Variablen benötigen. Zum jetzigen Zeitpunkt langt es, zu wissen, dass sie wichtig sind.

```
bool light;        // Beleuchtung AN / AUS
bool lp;           // L gedrückt?
bool fp;           // F gedrückt?
```

Nun führen wir fünf Variablen ein, die den Winkel auf der X-Achse (xrot), den Winkel auf der Y-Achse (yrot), die Geschwindigkeit mit der die Kiste auf der X-Achse rotiert (xspeed) und die Geschwindigkeit mit der die Kiste auf der Y-Achse rotiert (yspeed) kontrollieren. Wir erzeugen dann noch eine Variable namens z welche kontrolliert, wie tief die Kiste im Screen (auf der Z-Achse) ist.

```
GLfloat xrot;           // X Rotation
GLfloat yrot;           // Y Rotation
GLfloat xspeed;         // X Rotationsgeschwindigkeit
GLfloat yspeed;         // Y Rotationsgeschwindigkeit
GLfloat z=-5.0f;        // Tiefe in den Screen hinein
```

Nun erzeugen wir die Arrays, die zur Erzeugung der Beleuchtung verwendet werden. Wir werden zwei verschiedene Arten Licht verwenden. Die erste Art wird ambientes Licht genannt. Ambientes Licht ist Licht, welches nicht aus einer bestimmten Richtung kommt. Alle Objekte in der Szene werden durch das ambiente Licht beleuchtet. Die zweite Art wird diffuses Licht genannt. Diffuses Licht wird durch Ihre Lichtquelle erzeugt und wird von der Oberfläche der Objekte in Ihrer Szene relektiert. Jede Oberfläche eines Objekts, welche direkt von dem Licht beleuchtet wird, wird sehr hell sein und die Flächen, die nicht direkt getroffen werden, dunkler. Das erzeugt einen netten Shading-Effekt an den Seiten unserer Kiste.

Licht wird genauso erzeugt wie Farben erzeugt werden. Wenn die erste Zahl 1.0f ist und die nächste beiden 0.0f, erhalten wir ein helles rotes Licht. Wenn die dritte Zahl 1.0f ist und die ersten beiden 0.0f, erhalten wir ein helles blau. Die letzte Zahl ist der Alpha-Wert. Wir lassen diesen erstmal auf 1.0f.

In der folgenden Zeile speichern wir also die Werte für ein weißes ambientes Licht mit halber Intensität (0.5f). Da alle Zahlen gleich 0.5f sind, bekommen wir einen Wert, der direkt zwischen gar nichts (schwarz) und voller Helligkeit (weiß) liegt. Rot, blau und grün, mit den selben Werten gemischt, erzeugen eine Schattierung von schwarz (0.0f) zu weiß (1.0f). Ohne ein ambientes Licht erscheinen Stellen, wo kein diffuses Licht ist, sehr dunkel.

```
GLfloat LightAmbient[]= { 0.5f, 0.5f, 0.5f, 1.0f };          // Ambiente Lichtwere ( NEU )
```

In der nächsten Zeile speichern wir die Werte für ein super helles, mit voller Intensität, diffuses Licht. Alle Werte sind gleich 1.0f. Das bedeutet, dass das Licht so hell wie möglich ist. Ein diffuses Licht, das so hell ist, beleuchtet die Vorderseite der Kiste, sehr schön.

```
GLfloat LightDiffuse[]= { 1.0f, 1.0f, 1.0f, 1.0f };          // Diffuse Lichtwerte ( NEU )
```

Letztendlich speichern wir die Position des Lichts. Die ersten drei Zahlen sind die selben wie die drei glTranslate Zahlen. Die erste Zahl ist zum links und rechts bewegen auf der X-Ebene, die zweite Zahl ist zum hoch und runter bewegen auf der Y-Ebene und die dritte Zahl ist zum nach vorne und hinten bewegen auf der Z-Ebene. Da wir unser Licht direkt auf die Vorderseite der Kiste strahlen lassen wollen, bewegen wir uns weder nach links noch nach rechts, weshalb der erste Wert gleich 0.0f ist (keine Bewegung auf der X-Achse), wir wollen uns auch nicht nach oben und unten bewegen, weshalb auch der zweite Wert gleich 0.0f ist. Für den dritten Wert wollen wir sicher gehen, dass das Licht immer vor der Kiste ist. Deshalb positionieren wir das Licht außerhalb des Bildschirms zum Betrachter hin. Sagen wir, dass das Glas Ihres Monitors bei 0.0f auf der Z-Ebene ist. Wir positionieren unser Licht bei 2.0f auf der Z-Ebene. Wenn Sie das Licht tatsächlich sehen könnten, würde es sich vor Ihrem Monitor

befinden. Dadurch kann das Licht nicht hinter der Kiste sein, es sei denn, die Kiste wäre ebenfalls vor dem Glas Ihres Monitors. Selbstverständlich würden Sie die Kiste nicht mehr sehen, wenn die Kiste sich nicht mehr hinter dem Glas Ihres Monitors befinden würde, darum ist es dann auch egal wo das Licht ist. Leuchtet das ein?

Es gibt keinen wirklichen einfachen Weg, um den dritten Parameter zu erklären. Sie sollten wissen, dass -2.0f näher zum Betrachter ist, als -5.0f und -100.0f wäre WEIT weg in den Screen hinein. Wenn Sie erst einmal bei 0.0f sind, wird das Bild so groß, dass es den gesamten Bildschirm ausfüllt. Wenn Sie dann erst einmal in den positiven Bereich kommen, wird das Bild nicht mehr auf dem Bildschirm erscheinen, da es "hinter den Bildschirm" ist. Das meinte ich mit, aus dem Screen hinaus. Das Objekt ist immer noch da, Sie können es nur nicht mehr sehen.

Lassen Sie die letzte Zahl auf 1.0f. Damit teilen Sie OpenGL mit, dass die benannten Koordinaten die Position der Lichtquelle ist. Mehr darüber in einem späteren Tutorial.

```
GLfloat LightPosition[]= { 0.0f, 0.0f, 2.0f, 1.0f };                    // Lichtposition ( NEU )
```

Die filter Variable wird benutzt, um zu verfolgen, welche Textur angezeigt wird. Die erste Textur (Textur 0) wird gl_nearest verwenden (keine Weichzeichnung). Die zweite Textur (Textur 1) benutzt gl_linear Filterung, was das Bild etwas weicher zeichnet. Die dritte Textur (Textur 2) verwendet MipMapped Texturen, was eine sehr gut aussehende Textur erzeugt. Die Variable filter wird gleich 0, 1 oder 2 sein, abhänfig von der Textur, die wir verwenden wollen. Wir fangen mit der ersten Textur an.

GLuint texture[3] erzeugt Speicherplatz für die drei verschiedenen Texturen. Die Texturen werden in texture[0], texture[1] und texture[2] gespeichert.

```
GLuint filter;              // welcher Filter benutzt werden soll
GLuint texture[3];          // Speicherplatz für 3 Texturen

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);          // Deklaration für WndProc
```

Nun laden wir ein Bitmap und erzeugen daraus drei verschiedene Texturen. Dieses Tutorial verwendet die glaux Library, um das Bitmap zu laden, stellen Sie also sicher, dass Sie die glaux Library inkludieren, bevor Sie versuchen, den Code zu compilieren. Ich weiß, dass Delphi und Visual C++ jeweils die glaux Library haben. Ich bin mir nicht sicher, was andere Sprachen angeht. Ich werde nur erklären, was die neuen Codezeilen machen, wenn Sie eine Zeile sehen, die ich nicht kommentiert habe und Sie sich wundern, was diese macht, schauen Sie in Tutorial sechs nach. Es erklärt das Laden und erzeugen von Texturen aus Bitmap-Images im Detail.

Direkt nach dem obigen Code und bevor ReSizeGLScene() wollen wir folgenden Codeabschnitt einfügen. Das ist der selbe Code, den wir in Lektion 6 zum Laden einer Bitmap-Datei verwendet haben.

Da hat sich nichts geändert. Wenn Sie bei irgend einer Zeile nicht sicher sind, was diese macht, lesen Sie es in Tutorial sechs nach. Dieses erklärt den folgenden Code im Detail.

```
AUX_RGBImageRec *LoadBMP(char *Filename)          // Lädt ein Bitmap
{
        FILE *File=NULL; // Datei Handle

        if (!Filename)    // gehe sicher, dass ein Dateiname übergeben wurde
        {
                return NULL;    // wenn nicht, gebe NULL zurück
        }

        File=fopen(Filename,"r");       // überprüfe, ob die Datei existiert

        if (File) // Existiert die Datei?
        {
                fclose(File);      // Schließe das Handle
                // Lädt das Bitmap und gebe einen Zeiger zurück
                return auxDIBImageLoad(Filename);
        }

        return NULL;    // Wenn das Laden fehl schlug, gebe NULL zurück
}
```

Dieser Codeabschnitt lädt das Bitmap (indem der obige Code aufgerufen wird) und konvertiert es in drei Texturen. Status wird verwendet, um zu verfolgen ob eine Textur geladen und erzeugt wurde oder nicht.

```
int LoadGLTextures()    // Lade Bitmaps und konvertiere in Texturen
{
        int Status=FALSE;       // Status Indikator

        AUX_RGBImageRec *TextureImage[1];          // erzeuge Speicherplatz für die Textur
        memset(TextureImage,0,sizeof(void *)*1);     // Setze den Zeiger auf NULL
```

Nun laden wir das Bitmap und konvertieren es in eine Textur. TextureImage[0]=LoadBMP("Data/Crate.bmp") ruft unseren LoadBMP() Code auf. Die Datei names Crate.bmp im Data Verzeichnis wird geladen. Wenn alles glatt läuft, werden die Bild Daten in TextureImage[0] gespeichert, Status auf TRUE gesetzt und wir fangen an, unsere Textur zu erzeugen.

```
        // Lade das Bitmap, prüfe auf Fehler, wenn Bitmap nicht gefunden wurde, beende
        if (TextureImage[0]=LoadBMP("Data/Crate.bmp"))
        {
                Status=TRUE; // Setze Status auf TRUE
```

Nun, da wir die Bild-Daten nach TextureImage[0] geladen haben, benutzen wir die Daten um 3 Texturen zu erzeugen. Die folgende Zeile teilt OpenGL mit, dass wir drei Texturen erzeugen wollen und wir wollen die Texturen in texture[0], texture[1] und texture[2] speichern.

```
                glGenTextures(3, &texture[0]);          // erzeuge 3 Texturen
```

In Tutorial sechs haben wir linear gefilterte Textur Maps verwendet. Diese benötigen recht viel Rechnerleistung, dafür sehen sie recht gut aus. Die erste Art an Textur, die wir in diesem Tutorial erzeugen werden, benutzt GL_NEAREST. Grundsätzlich hat diese Art an Textur überhaupt keine Filterung. Sie benötigt recht wenig Rechnerleistung und sieht auch ziemlich schlecht aus. Wenn Sie jemals ein Spiel gespielt haben, wo die Texturen alle klobig aussehen, wird dieses wahrscheinliche dies Art Textur verwenden. Der einzige Vorteil dieser Textur-Art ist, dass Projekte, die diese Textur-Art verwenden in der Regel auch auf langsamen Rechnern recht gut laufen.

Sie werden bemerken, dass wir GL_NEAREST sowohl für MIN als auch MAG verwenden. Sie können GL_NEAREST mit GL_LINEAR mischen und die Textur wird etwas besser aussehen, da wir aber in Geschwindigkeit interessiert sind, werden wir für beides eine niedrige Qualität verwenden. MIN_FILTER ist der Filter, der verwendet wird, wenn ein Bild kleiner gezeichnet wird, als die original Textur-Größe. MAG_FILTER wird benutzt, wenn das Bild größer als die original Textur-Größe ist.

```
// erzeuge Nearest Filtered Textur
glBindTexture(GL_TEXTURE_2D, texture[0]);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST); // ( NEU )
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST); // ( NEU )
glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[0]->sizeX,
             TextureImage[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
             TextureImage[0]->data);
```

Die nächste Textur die wir erzeugen, ist die selbe Art Textur, die wir in Tutorial sechs verwendet haben. Linear gefilterte. Die einzige Sache, die sich geändert hat, ist, dass wir die Textur in texture[1] anstatt in texture[0] speichern, da es unsere zweite Textur ist. Wenn wir sie wie oben in texture[0] speichern, würde es die GL_NEAREST Textur (die erste Textur) überschreiben.

```
// erzeuge Linear Filtered Textur
glBindTexture(GL_TEXTURE_2D, texture[1]);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[0]->sizeX,
                    TextureImage[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
                    TextureImage[0]->data);
```

Nun zu einem anderen Weg Texturen zu erzeugen. Mipmapping! Sie werden bemerkt haben, dass wenn Sie ein Bild sehr winzig auf dem Screen machen, dass viele kleine Details verschwinden. Muster die eigentlich recht gut aussehen, fangen an recht schlecht auszusehen. Wenn Sie OpenGL mitteilen, eine mipmapped Textur zu erzeugen, versucht OpenGL verschieden große hochqualitative Texturen zu erzeugen. Wenn Sie eine mipmapped Textur auf den Screen zeichnen, wird OpenGL die am BESTEN aussehendste Textur aus den erzeugten wählen (Textur mit den meisten Details) und sie auf den Screen zeichnen anstatt das Original in der Größe zu verändern (was ein Detailverlust nach sich zieht).

Ich habe in Tutorial sechs gesagt, dass es einen Weg um die 64,128,256,etc. Grenze gibt, die OpenGL Textur-Breite und Höhe auferlegt. Und zwar mit gluBuild2DMipmaps. Von dem was ich gefunden habe,

können Sie jedes Bitmap-Image verwenden, das sie wollen (jede Breite und Höhe), wenn Sie mipmapped Texturen erzeugen. OpenGL wird diese automatisch der entsprechenden Breite und Höhe anpassen.

Da dies Textur Nummer drei ist, werden wir diese Textur in texture[2] speichern. Nun haben wir texture[0], welche keine Filterung hat, texture[1] welche lineare Filterung verwendet und texture[2] welche mipmapped Texturen verwendet. Damit haben wir die Texturen für dieses Tutorial erzeugt.

```
        // erzeuge MipMapped Textur
        glBindTexture(GL_TEXTURE_2D, texture[2]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                        GL_LINEAR_MIPMAP_NEAREST); // ( NEU )
```

Die folgende Zeile erzeugt die Mipmapped Textur. Wir erzeugen eine 2D Textur, indem wir 3 Farben (rot, grün, blau) verwenden. TextureImage[0]->sizeX ist die Bitmap-Breite, TextureImage[0]->sizeY ist die Bitmap-Höhe, GL_RGB bedeutet, dass wir Rot, Grün, Blau Farben, in dieser Reihenfolge, verwenden. GL_UNSIGNED_BYTE bedeutet, dass die Daten, aus denen die Textur besteht, aus Bytes besteht und TextureImage[0]->data zeigt auf die Bitmap-Daten, aus denen wir die Textur erzeugen.

```
        gluBuild2DMipmaps(GL_TEXTURE_2D, 3,
                          TextureImage[0]->sizeX, TextureImage[0]->sizeY, GL_RGB,
                          GL_UNSIGNED_BYTE, TextureImage[0]->data);   // ( NEU )
}
```

Nun geben wir jeglichen Speicher frei, den wir zum Speichern der Bitmap-Daten verwendet haben. Wir überprüfen, ob die Bitmap-Daten in TextureImage[0] gespeichert wurden. Wenn ja, überprüfen wir, ob Daten gespeichert wurden. Wenn Daten gespeichert wurden, löschen wir diese. Dann geben wir die Image-Struktur frei, um sicherzustellen, dass jeder benutzte Speicher freigegeben wird.

```
if (TextureImage[0])    // Wenn Textur existiert
{
        if (TextureImage[0]->data)     // Wenn Textur-Image existiert
        {
                free(TextureImage[0]->data); // gebe Textur-Image Speicher frei
        }

        free(TextureImage[0]); // gebe die Image-Struktur frei
}
```

Letztendlich geben wir status zurück. Wenn alles OK verlief, wird die Variable Status gleich TRUE sein. Wenn etwas schief ging, wird Status gleich FALSE sein.

```
return Status;  // gebe Status zurück
}
```

Nun laden wir die Texturen und initialisieren die OpenGL Eigenschaften. Die erste Zeile aus InitGL lädt die Texturen, indem der obige Code verwendet wird. Nachdem die Texturen erzeugt wurden, aktivieren wir 2D Textur-Mapping mittels glEnable(GL_TEXTURE_2D). Der Schattierungs-Modus wird auf Smooth-Shading gesetzt. Die Hintergrundfarbe wird auf schwarz gesetzt, wir aktivieren Depth-Testing und dann aktivieren wir hübsche Perspektiven Berechnungen.

```
int InitGL(GLvoid) // Der ganze Setup Kram für OpenGL kommt hier rein
{
        if (!LoadGLTextures()) // Rufe Textur Lade Routine auf
        {
                return FALSE; // wenn Textur nicht geladen wurde, gebe FALSE zurück
        }

        glEnable(GL_TEXTURE_2D);                // aktiviere Textur Mapping
        glShadeModel(GL_SMOOTH);                // aktiviere Smooth Shading
        glClearColor(0.0f, 0.0f, 0.0f, 0.5f);   // schwarzer Hintergrund
        glClearDepth(1.0f);                     // Depth Buffer initialisieren
        glEnable(GL_DEPTH_TEST);                // aktiviere Depth Test
        glDepthFunc(GL_LEQUAL);                 // Die Art des auszuführenden Depth Test
        // wirklich nette Perspektiven Berechnungen
        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
```

Nun erzeugen wir die Beleuchtung. Die folgende Zeile wird die Menge an ambienten Licht bestimmen, die light1 von sich geben wird. Am Anfang dieses Tutorials haben wir die Menge des ambienten Lichts in LightAmbient gespeichert. Es werden die Werte benutzt, die wir im Array gespeichert haben (halb intensives ambientes Licht).

```
        glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);     // initialisiere das ambiente Licht
```

Als nächstes setzen wir die Menge an diffusen Licht, was Licht Nummer eins von sich geben wird. Wir haben die Menge an diffusen Licht in LightDiffuse gespeichert. Es werden die Werte benutzt, die wir in dem Array gespeichert haben (volle Intensität weißes Licht).

```
        glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);     // initialisiere das diffuse Licht
```

Nun setzen wir die Position des Lichts. Wir haben die Position in LightPosition gespeichert. Es werden die Werte aus dem Array benutzt, die wir gespeichert haben (rechts im Zentrum der Vorderseite, 0.0f auf X, 0.0f auf Y und 2 Einheiten zum Betrachter hin {aus dem Screen heraus} auf der Z-Ebene).

```
        glLightfv(GL_LIGHT1, GL_POSITION,LightPosition);     // Position des Lichts
```

Letztendlich aktivieren wir Licht Nummer eins. Wir haben allerdings noch nicht GL_LIGHTING aktiviert, weshalb Sie noch keine Beleuchtung sehen werden. Das Licht wird erzeugt und positioniert und sogar aktiviert, aber solange GL_LIGHTNING nicht aktivieren, wird das Licht nicht funktionieren.

```
        glEnable(GL_LIGHT1);            // aktiviere Licht eins
        return TRUE;                    // Initialisierung war OK
}
```

Im nächsten Codeabschnitt werden wir den texturierten Würfel zeichnen. Ich werde einige der Zeilen kommentieren, weil diese neu sind. Wenn Sie nicht sicher sind, was die unkommentierten Zeilen bedeuten, schauen Sie in Tutorial sechs nach.

```
int DrawGLScene(GLvoid) // Hier kommt der ganze Zeichnen-Kram hin
{
        // Lösche den Bildschirm und den Depth-Buffer
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity(); // Resettet die View
```

Die nächsten drei Codezeilen positionieren und rotiren den texturierten Würfel. glTranslatef(0.0f,0.0f,z) bewegt den Würfel um den Wert z auf der Z-Ebene (vom Betrachter weg und zu ihm hin). glRotatef(xrot,1.0f,0.0f,0.0f) benutzt die Variable xrot, um den Würfel auf der X-Achse zu rotieren. glRotatef(yrot,0.0f,1.0f,0.0f) benutzt die Variable yrot, um den Würfel auf der Y-Achse zu rotieren.

```
        glTranslatef(0.0f,0.0f,z);      // Translatiere in den Screen hinein/heraus um z
        glRotatef(xrot,1.0f,0.0f,0.0f); // Rotiere auf der X-Achse um xrot
        glRotatef(yrot,0.0f,1.0f,0.0f); // Rotiere auf der Y-Achse um yrot
```

Die nächste Zeile ist ähnlich der Zeile, die wir in Tutorial sechs verwendet haben, aber statt texture[0] zu binden, binden wir texture[filter]. Jedes Mal wenn wir die 'F'-Taste drücken, wird der Wert in filter erhöht. Wenn der Wert größer als zwei ist, wird die Variable filter zurück auf null gesetzt. Wenn das Programm startet, ist der filter auf null gesetzt. Das ist das selbe wie glBindTexture(GL_TEXTURE_2D, texture[0]). Wenn wir 'F' ein weiteres Mal drücken, ist die Variable filter gleich eins, was das Selbe wie glBindTexture(GL_TEXTURE_2D, texture[1]) ist. Indem wir die Variable filter verwenden, können wir jede der drei Texturen auswählen, die wir erzeugt haben.

```
        // wähle eine Textur basierend auf dem Filter aus
        glBindTexture(GL_TEXTURE_2D, texture[filter]);

        glBegin(GL_QUADS);    // fange an Quads zu zeichnen
```

glNormal3f ist neu in meinen Tutorials. Ein Normalenvektor (engl.: normal) ist eine Linie die senkrecht (im 90 Grad Winkel) direkt aus der Mitte eines Polygons herauszeigt. Wenn Sie Beleuchtung benutzen, müssen Sie einen Normalenvektor spezifizieren. Der Normalenvektor teilt OpenGL mit, in welche Richtung das Polygon zeigt... was nach oben zeigt. Wenn Sie die Normalenvektoren nicht angeben, passieren die komischtens Sachen. Seiten, die nicht beleuchtet werden sollen, werden beleuchtet, die falsche Seite eines Polygons wird beleuchtet, etc. Der Normalenvektor sollte aus dem Polygon herauszeigen.

Wenn Sie auf die Vorderseite schauen, werden Sie bemerken, dass der Normalenvektor auf der positiven Z-Achse liegt. Das bedeutet, dass der Normalenvektor zum Betrachter zeigt. Exakt die

Richtung, in die er zeigen soll. Auf der Hinterseite zeigt der Normalenvektor vom Betrachter weg, in den Screen hinein. Wieder genau das was wir wollen. Wenn der Würfel entweder auf der X oder Y-Achse um 180 Grad gedreht wurde, zeigt die Vorderseit in den Screen hinein und die Hinterseite zeigt zum Benutzer. Egal welche Seite zum Betrachter zeigt, der Normalenvektor der entsprechenden Seite wird ebenfalls zum Betrachter zeigen. Da sich das Licht beim Betrachter befindet, wird jedes mal, wenn der Normalenvektor zum Betrachter zeigt, ebenfalls zum Licht zeigen. Wenn das passiert, wird die Seite ausgeleuchtet. Je mehr ein Normalenvektor zum Licht zeigt, umso heller wird die Seite. Wenn Sie sich ins Zentrum des Würfels bewegen, werden Sie bemerken, dass es dunkel ist. Die Normalenvektoren zeigen hinaus, nicht hinein, weshalb es kein Licht innerhalb der Kiste gibt, genauso wie es sein soll.

```
                // Vorderseite
                glNormal3f( 0.0f, 0.0f, 1.0f); // Normalenvektor zeigt in Richtung Betrachter
                glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f); // Punkt 1 (vorne)
                glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f); // Punkt 2 (vorne)
                glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);  // Punkt 3 (vorne)
                glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);  // Punkt 4 (vorne)


                // Hinterseite
                glNormal3f( 0.0f, 0.0f,-1.0f); // Normalenvektor zeigt vom Betrachter weg
                glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);  // Punkt 1 (hinten)
                glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);   // Punkt 2 (hinten)
                glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);   // Punkt 3 (hinten)
                glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f); // Punkt 4 (hinten)

                // obere Seite
                glNormal3f( 0.0f, 1.0f, 0.0f); // Normalenvektor zeigt nach oben
                glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f); // Punkt 1 (oben)
                glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f); // Punkt 2 (oben)
                glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f); // Punkt 3 (oben)
                glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f); // Punkt 4 (oben)

                // untere Seite
                glNormal3f( 0.0f,-1.0f, 0.0f); // Normalenvektor zeigt nach unten
                glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f); // Punkt 1 (unten)
                glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f); // Punkt 2 (unten)
                glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f); // Punkt 3 (unten)
                glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f); // Punkt 4 (unten)

                // rechte Seite
                glNormal3f( 1.0f, 0.0f, 0.0f); // Normalenvektor zeigt nach rechts
                glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f); // Punkt 1 (rechts)
                glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f); // Punkt 2 (rechts)
                glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f); // Punkt 3 (rechts)
                glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f); // Punkt 4 (rechts)

                // linke Seite
                glNormal3f(-1.0f, 0.0f, 0.0f); // Normalenvektor zeigt nach links
                glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f); // Punkt 1 (links)
                glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f); // Punkt 2 (links)
                glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f); // Punkt 3 (links)
                glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f); // Punkt 4 (links)

        glEnd(); // fertig mit Zeichnen der Quads
```

Die nächsten beiden Zeilen inkrementieren xrot und yrot um den Wert, der in xspeed und yspeed gespeichert ist. Wenn der Wert in xspeed oder yspeed hoch ist, werden xrot und yrot sehr schnell inkrementiert. Je schneller xrot oder yrot inkrementiert werden, um so schneller dreht sich der Würfel auf der entsprechenden Achse.

```
        xrot+=xspeed; // Addiere xspeed zu xrot
        yrot+=yspeed; // Addiere yspeed zu yrot

        return TRUE; // weiter geht's
}
```

Nun gehen wir runter zur WinMain(). Wir fügen den Code zum ein- und ausschalten der Beleuchtung ein, zum rotieren der Kiste, zum ändern des Filters und um die Kiste in den Screen hinein und hinaus zu bewegen. Zum Ende der WinMain() werden Sie den Befehl SwapBuffers(hDC) finden. Direkt hinter dieser Zeile fügen Sie den folgenden Code ein.

Dieser Code überprüft, ob der Buchstabe 'L' auf der Tastatur gedrückt wurde. Die erste Zeile überprüft, ob 'L' gedrückt wurde. Wenn 'L' gedrückt wurde, aber lp nicht False ist, was bedeutet, dass 'L' bereits einmal gedrückt wurde oder die Taste festgehalten wird, wird nichts passieren.

```
        SwapBuffers(hDC); // Swap Buffers (Double Buffering)

        if (keys['L'] && !lp) // L wurde gedrückt und nicht festgehalten?
        {
```

Wenn lp auf False gesetzt war, was bedeutet, dass die Taste 'L' noch nicht gedrückt wurde oder sie losgelassen wurde, wird lp auf True gesetzt. Das zwingt den Benutzer, die 'L'-Taste loszulassen, bevor der Code erneut ausgeführt werden kann. Wenn wir nicht überprüfen, ob die Taste gehalten wird, würde die Beleuchtung ständig an- und ausgehen, da das Programm denken würde, dass Sie die 'L'-Taste immer wieder drücken würden, wenn man diesen Codeabschnitt erreicht.

Wenn lp einmal auf True gesetzt wurde, wird dem Computer somit mitgeteilt, dass 'L' gedrückt wird und wir die Beleuchtung an, bzw. aus schalten. Die Variable light kann nur auf true oder false gesetzt werden. Wenn wir also light=!light sagen, sagen wir also light ist NICHT gleich light. Was auf deutsch bedeutet, dass wenn light gleich true ist, dass wir es ungleich true machen (auf false setzen) und wenn light gleich false ist, wir light ungleich false machen (auf true setzen). Wenn light also true war, setzen wir es auf false und wenn light false war, setzen wir es auf true.

```
        lp=TRUE;        // lp wird TRUE
        light=!light;   // wechsel Light zu TRUE/FALSE
```

Nun überprüfen wir, was mit der Beleuchtung geschehen soll. Die erste Zeile heißt auf gut deutsch: wenn light gleich false ist. Wenn Sie also alles zusammenfügen, machen die Zeilen folgendes: Wenn light gleich false ist, deaktiviere Beleuchtung. Das schaltet die Beleuchtung aus. Der Befehl else

bedeutet: wenn es nicht gleich false war. Wenn light also nicht gleich false war, muss es gleich true sein, weshalb wir die Beleuchtung anschalten.

```
		if (!light) // wenn keine Beleuchtung
		{
			glDisable(GL_LIGHTING); // deaktiviere Beleuchtung
		}
		else // ansonsten
		{
			glEnable(GL_LIGHTING); // aktiviere Beleuchtung
		}
	}
```

Die folgende Zeile überprüft, ob wir aufgehört haben, die 'L'-Taste zu drücken. Falls ja, setzen wir die Variable lp auf False, was bedeutet, dass die 'L'-Taste nicht gedrückt wird. Wenn wir nicht überprüfen, ob die Taste losgelassen wurde, sind wir nur einmal in der Lage die Beleuchtung anzuschalten, da der Computer denkt, dass 'L' gedrückt ist und uns sie nicht mehr ausschalten lässt.

```
		if (!keys['L']) // wurde die L Taste losgelassen?
		{
			lp=FALSE; // wenn ja, wird lp gleich FALSE
		}
```

Nun machen wir etwas ähnliches mit der 'F'-Taste. Wenn die Taste gedrückt ist und sie nicht festgehalten wird oder sie niemals zuvor gedrückt wurde, wird die Variable fp auf True gesetzt, was bedeutet, dass die Taste nun gedrückt wird. Dann wird die Variable namens filter inkrementiert. Wenn filter größer als 2 ist (was texture[3] sein würde und diese Textur existiert nicht), setzen wir die Variable filter zurück auf null.

```
		if (keys['F'] && !fp) // wurde die F Taste gedrückt?
		{
			fp=TRUE;		// fp wird TRUE
			filter+=1;		// filter Wert um eins erhöhen

			if (filter>2) // Ist Wert größer als 2?
			{
				filter=0; // Wenn ja, setze filter auf 0
			}
		}

		if (!keys['F']) // wurde die F Taste losgelassen?
		{
			fp=FALSE;		// wenn ja, wird fp gleich FALSE
		}
```

Die nächsten vier Zeilen überprüfen, ob wir die 'Seite hoch' Taste drücken. Wenn ja, wird die Variable z dekrementiert. Wenn diese Variable dekrementiert wird, wird sich der Würfel weiter weg, in die Ferne bewegen, wegen des glTranslatef(0.0f,0.0f,z) Befehls, der in der DrawGLScene Prozedure verwendet wird.

```
if (keys[VK_PRIOR]) // Wurde die Seite nach oben Taste gedrückt?
{
        z-=0.02f; // wenn ja, bewege in den Screen hinein
}
```

Diese vier Zeilen überprüfen, ob die 'Seite runter' Taste gedrückt wurde. Wenn ja, wird die Variable z inkrementiert und der Würfel bewegt sich in Richtung des Betrachters, wegen des glTranslatef(0.0f,0.0f,z) Befehls, der in der DrawGLScene Prozedur verwendet wird.

```
if (keys[VK_NEXT]) // Wurde die Seite nach unten Taste gedrückt?
{
        z+=0.02f; // wenn ja, bewege in Richtung Betrachter
}
```

Nun müssen wir alle Pfeil-Tasten überprüfen. Indem links oder rechts gedrückt wird, wird xspeed inkrementiert oder dekrementiert. Indem hoch oder runter gedrückt wird, wird yspeed inkrementiert oder dekrementiert. Erinnern Sie sich daran, dass ich vorher in diesem Tutorial gesagt habe, dass, je höher der Wert in xspeed oder yspeed ist, desto schneller rotiert der Würfel. Je länger Sie eine Pfeil-Taste drücken, desto schneller wird der Würfel in diese Richtung rotieren.

```
if (keys[VK_UP]) // Wurde die Pfeil nach oben Taste gedrückt?
{
        xspeed-=0.01f; // wenn ja, dekrementierexspeed
}

if (keys[VK_DOWN]) // Wurde die Pfeil nach unten Taste gedrückt?
{
        xspeed+=0.01f; // wenn ja, inkrementierexspeed
}

if (keys[VK_RIGHT]) // wurde die rechte Pfeiltaste gedrückt?
{
        yspeed+=0.01f; // wenn ja, inkrementiere yspeed
}

if (keys[VK_LEFT]) // wurde die linke Pfeiltaste gedrückt?
{
        yspeed-=0.01f; // wenn ja, inkrementiereyspeed
}
```

Wie in allen vorangegangen Tutorials, stellen wir sicher, dass der Titel des Fensters korrekt ist.

```
if (keys[VK_F1]) // Wurde F1 gedrückt?
{
        keys[VK_F1]=FALSE;    // Wenn ja, setze die Taste auf FALSE
        KillGLWindow();       // Kill unser aktuelles Fenster
        fullscreen=!fullscreen; // Wechsel zwischen Fullscreen und Fester-Modus
```

```
                        // Erzeuge unser OpenGL-Fenster neu
                        if (!CreateGLWindow("NeHe's Textures, Lighting & Keyboard
                                            Tutorial",640,480,16,fullscreen))
                        {
                                return 0; // Beenden, wenn das Fenster nicht erzeugt wurde
                        }
                }
            }
        }
    }

        // Shutdown
        KillGLWindow();         // Kill das Fenster
        return (msg.wParam);  // beende das Programm
}
```

Nach diesem Tutorial sollten Sie in der Lage sein, hochqualitative, realistisch aussehende texturierte Objekte aus Quadraten zu erzeugen und mit ihnen zu interagieren. Sie sollten die Vorteile der drei Filter, die in diesem Tutorial verwendet wurden, verstanden haben. Indem Sie die bestimmten Tasten auf der Tastatur drücken, sollten Sie mit dem/n Objekt(en) auf dem Screen interagieren können und Sie sollten wissen, wie man einfache Beleuchtung in einer Szene einsetzt, um diese realistischer wirken zu lassen.

Jeff Molofee (**NeHe**)

* DOWNLOAD Visual C++ Code für diese Lektion.

* DOWNLOAD Borland C++ Builder 6 Code für diese Lektion. (Conversion by Christian Kindahl)
* DOWNLOAD C# Code für diese Lektion. (Conversion by Brian Holley)
* DOWNLOAD Code Warrior 5.3 Code für diese Lektion. (Conversion by Scott Lupton)
* DOWNLOAD Cygwin Code für diese Lektion. (Conversion by Stephan Ferraro)
* DOWNLOAD D Language Code für diese Lektion. (Conversion by Familia Pineda Garcia)
* DOWNLOAD Delphi Code für diese Lektion. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code für diese Lektion. (Conversion by Dan)
* DOWNLOAD Euphoria Code für diese Lektion. (Conversion by Evan Marshall)
* DOWNLOAD Game GLUT Code für diese Lektion. (Conversion by Milikas Anastasios)
* DOWNLOAD GLUT Code für diese Lektion. (Conversion by Andy Restad)
* DOWNLOAD Irix Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Java Code für diese Lektion. (Conversion by Jeff Kirby)
* DOWNLOAD Jedi-SDL Code für diese Lektion. (Conversion by Dominique Louis)
* DOWNLOAD JoGL Code für diese Lektion. (Conversion by Kevin J. Duling)
* DOWNLOAD LCC Win32 Code für diese Lektion. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux Code für diese Lektion. (Conversion by Richard Campbell)
* DOWNLOAD Linux/GLX Code für diese Lektion. (Conversion by Mihael Vrbanec)
* DOWNLOAD Linux/SDL Code für diese Lektion. (Conversion by Ti Leggett)
* DOWNLOAD LWJGL Code für diese Lektion. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS Code für diese Lektion. (Conversion by Anthony Parker)
* DOWNLOAD Mac OS X/Cocoa Code für diese Lektion. (Conversion by Bryan Blackburn)
* DOWNLOAD MASM Code für diese Lektion. (Conversion by Nico (Scalp))
* DOWNLOAD Visual C++ / OpenIL Code für diese Lektion. (Conversion by Denton Woods)
* DOWNLOAD Power Basic Code für diese Lektion. (Conversion by Angus Law)
* DOWNLOAD Pelles C Code für diese Lektion. (Conversion by Pelle Orinius)
* DOWNLOAD Scheme Code für diese Lektion. (Conversion by Brendan Burns)
* DOWNLOAD Solaris Code für diese Lektion. (Conversion by Lakmal Gunasekara)
* DOWNLOAD Visual Basic Code für diese Lektion. (Conversion by Peter De Tagyos)
* DOWNLOAD Visual Fortran Code für diese Lektion. (Conversion by Jean-Philippe Perois)
* DOWNLOAD Visual Studio .NET Code für diese Lektion. (Conversion by Grant James)

Der original Text ist hier zu finden.

Die original OpenGL Tutorials stammen von NeHe's Seite.