**AUF CD**

Die Quelltexte sowie die fertig übersetzten Routinen finden Sie im Verzeichnis *Heft Add-ons/Programmierung/PC Underground.*

Shadow Depth Maps mit OpenGL

# Im Schein-werferlicht

Mit wenigen OpenGL-Befehlen und etwas Theorie entlocken Sie Ihrer modernen Grafikkarte Schatteneffekte. Dabei können sich die 3D-Objekte auch selbst beschatten.

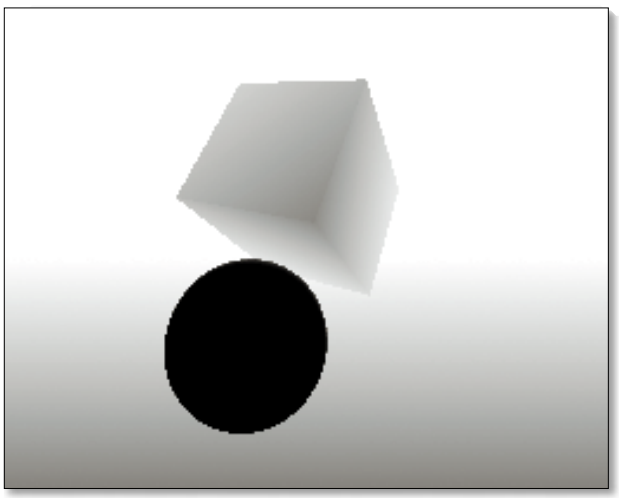
CARSTEN DACHSBACHER

Das Rendering von Schatten in 3D-Engines und Computerspielen ist heutzutage schon fast ein Muss. Die Leistungsfähigkeit moderner 3D-Beschleuniger und CPUs stellen Ihnen die technischen Werkzeuge zur Verfügung. Einzige Voraussetzung: Sie müssen die Theorie dahinter kennen – dafür sorgt dieser Artikel.

Sie hatten sich bereits in PC Underground 6/02, ab S. 196, mit einem Verfahren vertraut gemacht, um in Echtzeit Schatten zu rendern. Dabei handelte es sich um die Stencil-Buffer-Schatten. Diese Technik betrachtet Schatten eines 3D-Objekts als polygonales Volumen. Für jedes gerenderte Bild mussten Sie dabei das Volumen bestimmen – das heißt, die Silhouette des Objekts aus der Sicht der Lichtquelle bestimmen.

Dieses Verfahren ist mit Rechen- und Speicheraufwand für die Adjazenz-Information verbunden. Und es gibt weitere Nachteile: Sie können beispielsweise keinen korrekten Schatten rendern, wenn Verfahren wie die PN-Triangles der neueren ATI-Grafikkarten verwendet werden. Die PN-Triangles bieten die Option, mit einem Dreieck eine gewölbte Oberfläche darzustellen, ähnlich wie bei herkömmlichen parametrischen Flächen.

Lernen Sie die Shadow-Map-Technik kennen, die Sie mit den Stencil Shadows schon gestreift hatten.



**DER IN EINE TEXTUR KOPIERTE** Z-Buffer zeigt dunklere Teile nahe an der Lichtquelle, hellere weiter entfernt.

## Shadow Depth Maps

Bei den Shadow Maps verwenden Sie den Z-Buffer und projektives Texture Mapping, um die Schatten zu rendern. Dieses Verfahren ist für Spotlights gedacht: Lichtquellen, die einen begrenzten Lichtkegel in eine Richtung abstrahlen. Omnilights sind Lichtquellen, die in jede Richtung gleichmäßig Licht aussenden.

Bei der Shadow-Mapping-Technik transformieren Sie die 3D-Szene, die Sie mit Schatten rendern wollen. Dabei muss die OpenGL-Kamera in die gleiche Position wie das entsprechende Spotlight rücken. Mit dieser Einstellung rendern Sie die 3D-Szene aus der Sicht der Lichtquelle. Von diesem gerenderten Bild benötigen Sie nur den Z-Buffer, den Sie in die Depth Map kopieren. Mit Z-Buffer heben Sie sich also lediglich die Tiefeninformation auf.

Ein Beispiel sehen Sie im Bild links. Diese Textur verwenden Sie dann beim Rendering des fertigen Bildes wieder. Dabei projizieren Sie die Textur mit der OpenGL-Textur-Koordinaten-Generierung auf die Geometrie der 3D-Objekte.

Die Koordinaten-Generierung stellen Sie so ein, dass die Texturkoordinaten eines Punktes den Vertexkoordinaten des Punktes bezüglich der Lichtquelle entsprechen. Diese Texturkoordinaten sind mit *(s, t, r)* bezeichnet. Mit der entsprechenden OpenGL-Erweiterung können Sie die *r*-Komponente (also den Abstand des Punktes von der Lichtquelle) mit der Tiefeninformation in der Depth Map vergleichen. Kurz: Sie vergleichen bei jedem Fragment (Pixel), das gezeichnet wird, seinen tatsächlichen Abstand zur Lichtquelle mit dem in der Depth Map gespeicherten Abstand.

In der Depth Map sind durch das Z-Buffering die jeweils kleinsten Abstände zwischen der Lichtquelle und einer Oberfläche gespeichert. Ist der tatsächliche Abstand *r* größer als der gespeicherte Wert im Texel der Depth Map, liegt der betrachtete Pixel hinter einer Oberfläche, die von der Lichtquelle aus sichtbar ist. Das bedeutet, er liegt im Schatten.

Dieses Verfahren ist universell einsetzbar und sehr flexibel. Es lässt auch Selbstbeschattung von 3D-Objekten zu. Ein weiterer Vorteil ist, dass der Aufwand des Schatten-Rendering nicht direkt von der Komplexität der Geometrie abhängt. Außerdem können Sie, um die Shadow Depth Maps zu generieren, 3D-Objekte mit reduzierten Details verwenden.

Bei diesem Screen-Space-Schattenverfahren hängt die Auflösung der Schatten von der Auflösung der Depth Map ab. Dabei sind unter Umständen Treppcheneffekte an den Rändern der Schatten zu erkennen.

## Umsetzung in OpenGL

Nachdem Sie jetzt das Prinzip kennen, implementieren Sie das Schatten- ➧ Ren-

dering. Zunächst müssen Sie sich um die benötigten OpenGL-Extensions kümmern, da die Funktionalität der Depth-Map-Vergleiche noch nicht Bestandteil einer OpenGL-Spezifikation ist. Außerdem verwenden Sie Multitexturing, um nicht nur Schatten zu rendern, sondern dem Lichtkegel auch einen attraktiven Helligkeitsverlauf und den 3D-Objekten eine normale Textur zu verpassen.

Für die Depth Maps stehen Ihnen entweder die *GL_SGIX_depth_texture* und *GL_SGIX_shadow*-Extensions zur Verfügung, oder Sie verwenden die *GL_ARB_depth_texture* und *GL_ARB_shadow*-Extentions. Ihre Wahl hängt davon ab, was die Treiber anbieten.

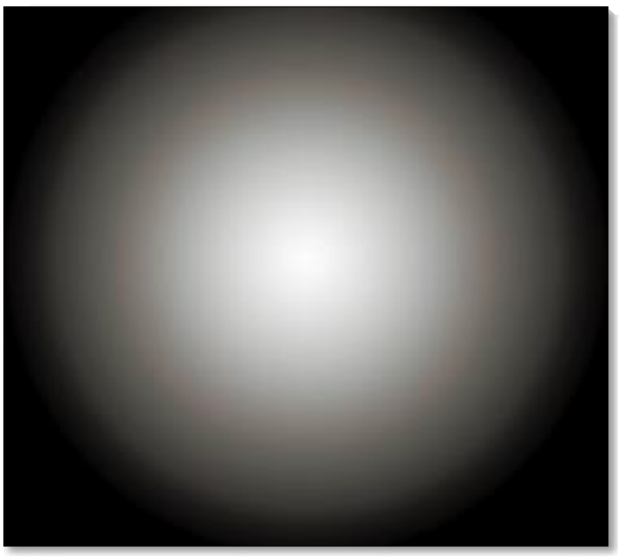
Bislang werden diese beiden Erweiterungen hauptsächlich von nVidia-Grafikkarten unterstützt. Es ist zu erwarten, dass alle neuen Karten nachziehen werden. Die beiden Varianten unterscheiden sich prinzipiell nicht, lediglich im Setup des Depth-Map-Vergleichs müssen Sie die später erwähnten Einzelheiten beachten. Die beiden Extensions definieren keine neuen OpenGL-Funktionen, sondern nur neue Tokens für die *glTexParameter*-Befehle. Für das Multitexturing verwenden Sie in dieser Ausgabe den *glActiveTextureARB*-Befehl, um die jeweils aktive Texturing-Einheit zu wählen. Mit dem *glMultiTexCoord2fARB*-Befehl geben Sie die Texturkoordinaten an.

Beginnen Sie zunächst damit, die benötigten Texturen anzulegen, allen voran die *Shadow Depth Map*:

```
GLuint shadowDepthMap;
glGenTextures
     ( 1, &shadowDepthMap );
glBindTexture( GL_TEXTURE_2D,
               shadowDepthMap );
```

Als Format wählen Sie *GL_DEPTH_COMPONENT*. Damit stellen Sie sicher, dass für die Tiefeninformation in der Textur dieselbe Bit-Tiefe wie für den tatsächlichen Z-Buffer verwendet wird (also nicht konvertiert werden muss). Sie könnten die Bit-Tiefe aber auch explizit angeben, wie mit *GL_DEPTH_COMPONENT16_SGIX* oder *GL_DEPTH_COMPONENT16_ARB* für 16 Bit. Die Größe der *Depth Map* berechnet sich nach der Formel *SHADOWSIZE x SHADOWSIZE*:

```
glCopyTexImage2D
   ( GL_TEXTURE_2D, 0,
  GL_DEPTH_COMPONENT, 0, 0,
  SHADOWSIZE, SHADOWSIZE, 0 );
```



**MIT DIESEM HELLIGKEITSVERLAUF** schinden Ihre Spotlights Eindruck.

In der Regel genügt eine Größe der Depth Map von 256 x 256 Pixeln. Weitere Textur-Parameter sind das *Clamping* der Textur, was bedeutet, dass die Textur sich nicht wiederholt. Das ist wichtig, weil die Depth Map nur für den sichtbaren Bereich der Lichtquelle gilt, aber durch die Textur-Koordinatengenerierung auch auf andere Stellen gemappt wird. Außerdem stellen Sie die Vergößerung/Verkleinerung der Textur auf bilineare Interpolation ohne Mipmaps:

```
glTexParameteri
 ( GL_TEXTURE_2D,
 GL_TEXTURE_MAG_FILTER,
 GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D,
 GL_TEXTURE_MIN_FILTER,
 GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D,
 GL_TEXTURE_WRAP_S,
 GL_CLAMP_TO_EDGE);
glTexParameteri( GL_TEXTURE_2D,
 GL_TEXTURE_WRAP_T,
 GL_CLAMP_TO_EDGE);
```

Jetzt kommt der wichtigste Teil des Textur-Setups: die Vergleichsoperation. Sie vergleicht die *r*-Komponente der Texturkoordinaten mit dem Tiefenwert in der Depth Map. Als Resultat kann eine *0* oder eine *1* vorkommen, die Sie als Luminanz (Helligkeitswert) der Textur-Stage verwenden. Um die Helligkeit mit anderen Textur-Stages zu kombinieren, verwenden Sie *GL_MODULATE* für das Textur-*Combining*:

```
glTexEnvi( GL_TEXTURE_ENV,
  GL_TEXTURE_ENV_MODE,
  GL_MODULATE );
   glTexParameteri(
   GL_TEXTURE_2D,
   GL_DEPTH_TEXTURE_MODE_ARB,
   GL_LUMINANCE );
```

Den Depth-Map-Vergleich aktivieren Sie je nach verwendeter Erweiterung so:

```
// ARB Ext
glTexParameteri( GL_TEXTURE_2D,
  GL_TEXTURE_COMPARE_MODE_ARB,
  GL_COMPARE_R_TO_TEXTURE_ARB );
glTexParameteri( GL_TEXTURE_2D,
  GL_TEXTURE_COMPARE_FUNC_ARB,
  GL_LEQUAL );

// SGIX Ext
glTexParameteri( GL_TEXTURE_2D,
   GL_TEXTURE_COMPARE_SGIX,
   GL_TRUE );
glTexParameteri( GL_TEXTURE_2D,

GL_TEXTURE_COMPARE_OPERATOR_SGIX,
  GL_TEXTURE_LEQUAL_R_SGIX );
```

Den Helligkeitsverlauf des Spotlights laden Sie als einfache Graustufen-Textur. Dafür aktivieren Sie das Textur-Clamping und *GL_MODULATE*.

## *INTELLISENSE* FÜR OPENGL-BEFEHLE

Mit IntelliSense bezeichnet Microsoft das Feature der Visual-C++- Entwicklungsumgebung, mit dem sich automatisch Befehlsnamen vervollständigen (über *[Strg-Leertaste]*) und Typ- und Parameter-Informationen per *[Strg-t]* oder *[Strg-Shift-Leertaste]* abfragen lassen. Diese praktische Option steht zunächst nur für die Standardbefehle wie *strdup(...)* oder selbst definierte Funktionen zur Verfügung. Bei OpenGL-Befehlen kann dieses Feature nicht eingesetzt werden, es sei denn, Sie gaukeln der Entwicklungsumgebung vor, Sie hätten jeden Befehl selbst definiert.

Adam Medveczky hat dies getan und seine Header-Datei der OpenGL-Gemeinde zur Verfügung gestellt. Diese Datei müssen Sie in ein Visual-C++-Projekt über den Menüpunkt *Project/Add To Project/Files* einfügen. In der Header-Datei befinden sich alle Funktionen als leer definiert – mit einer nie erfüllten *#ifdef*-Anweisung. Dadurch werden die Funktionen für die Entwicklungsumgebung definiert, der Compiler ignoriert sie aber – was er auch tun muss – auf Grund der *#ifdef*-Anweisung. Sie finden die Header-Datei auf der Heft-CD.

## Depth Map rendern

Nach den Initialisierungen rendern Sie die einzelnen Frames. Im Folgenden gehen wir – der Übersichtlichkeit halber – davon aus, dass es eine Funktion *drawScene()* gibt, die die vollständige 3D-Szene an OpenGL übergibt. Für jeden Frame rendern Sie zunächst die Depth Map. Dabei müssen Sie die OpenGL-Kamera passend positionieren und ausrichten und den OpenGL-Viewport einstellen. Den Öffnungswinkel des Spotlights stellen Sie in der Projektions-Matrix, die Position und Richtung mit *gluLookAt(...)* in der Modelview-Matrix von OpenGL ein:

```
glViewport( 0, 0,
  SHADOWSIZE, SHADOWSIZE );

glClear( GL_COLOR_BUFFER_BIT |
         GL_DEPTH_BUFFER_BIT );

glMatrixMode( GL_PROJECTION );
glLoadIdentity();

gluPerspective( SPOT_ANGLE,
        1.0f, 1.0f, 500.0f );

glMatrixMode( GL_MODELVIEW );
glPushMatrix();

glLoadIdentity();

gluLookAt(
 lightPosition[ 0 ],
 lightPosition[ 1 ],
 lightPosition[ 2 ],
 0, 0, 0,
 0, 1, 0 );
```

Verwenden Sie *glPolygonOffset(...)*, um die gerenderten Dreiecke in der Depth Map zu verschieben. Somit vermeiden Sie Artefakte (Bildfehler) durch Ungenauigkeiten bei den Schatten. Außerdem können Sie die OpenGL-Beleuchtungsberechnung und das Beschreiben des Colorbuffers abschalten. Somit ersparen Sie Ihrer Grafikkarte unnötigen Aufwand, denn nur der Z-Buffer ist interessant:

```
glDisable( GL_LIGHTING );
glColorMask( GL_FALSE, GL_FALSE,
             GL_FALSE, GL_FALSE);

glPolygonOffset( 2, 2 );
glEnable
 ( GL_POLYGON_OFFSET_FILL );
```

Jetzt zeichnen Sie die 3D-Szene und kopieren den Z-Buffer in die Depth Map:

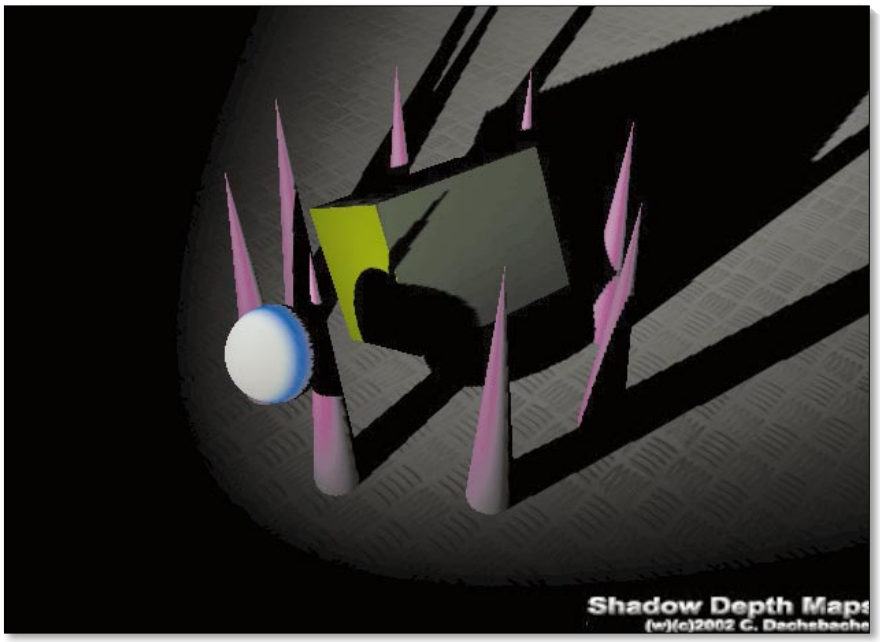
```
drawScene();

glBindTexture( GL_TEXTURE_2D,
               shadowDepthMap );
glCopyTexSubImage2D(
  GL_TEXTURE_2D, 0, 0, 0, 0, 0,
  SHADOWSIZE, SHADOWSIZE );
```

Um eine Graustufen-Bitmap des Z-Buffers auszuwerten, können Sie die Daten so erhalten:

```
unsigned char depthMap
           [ 256 * 256 ];
glReadPixels( 0, 0, 256, 256,
  GL_DEPTH_COMPONENT,
  GL_UNSIGNED_BYTE, depthMap );
```

Anschließend stellen Sie die vorherigen Renderstates und OpenGL-Matrizen wieder her und sind fertig mit der Depth Map.




**UNSER BEISPIELPROGRAMM** in Aktion

## Mit der Depth Map rendern

Jetzt können Sie die 3D-Szene mit Schatten rendern. Sie müssen lediglich die Textur Stages und die Texturkoordinaten-Generierung einstellen. Dieser Vorgang ist für die ersten beiden Textur Stages gleich. Eine verwendet die Spotlight Textur, die andere die Depth ▶

Map. Welche Textur auf welcher Stage liegt, ist egal: Durch das *GL_MODULATE* ist die Reihenfolge hinfällig. Die jeweils aktive Textur Stage wählen Sie mit *glActiveTextureARB(GL_TEXTUREi_ARB)*. Dann stellen Sie die Textur-Matrix ein. Diese setzt sich aus drei Schritten zusammen. Die einzelnen Matrixtransformationen werden in umgekehrter Reihenfolge zu der im Programm ausgeführt.

Als erstes geben Sie an, dass Sie zur Texturkoordinaten-Generierung alle vier Vertex-Komponenten (also homogene Koordinaten) *direkt* verwenden wollen. Direkt heißt, sie werden nicht weiter transformiert, und deshalb ist in

**SOLCHE TREPPCHENEFFEKTE** treten bei zu geringer Auflösung der Depth Map auf.

*genS*, *genT*, *genR* und *genQ* eine *4x4*-Einheitsmatrix gespeichert. Sie setzen die *GL_EYE_LINEAR*-Option folgendermaßen ein:

```
glActiveTextureARB
   ( GL_TEXTURE0_ARB );

float genS[]={1.0,0.0,0.0,0.0};
float genT[]={0.0,1.0,0.0,0.0};
float genR[]={0.0,0.0,1.0,0.0};
float genQ[]={0.0,0.0,0.0,1.0};

glEnable( GL_TEXTURE_GEN_S );
glEnable( GL_TEXTURE_GEN_T );
glEnable( GL_TEXTURE_GEN_R );
glEnable( GL_TEXTURE_GEN_Q );

glTexGenfv
     (GL_S,GL_EYE_PLANE,genS );
glTexGenfv
     (GL_S,GL_EYE_PLANE,genR );
glTexGenfv
     (GL_S,GL_EYE_PLANE,genT );
glTexGenfv
     (GL_S,GL_EYE_PLANE,genQ );

glTexGeni( GL_S,
           GL_TEXTURE_GEN_MODE,
           GL_EYE_LINEAR );
glTexGeni( GL_T,
           GL_TEXTURE_GEN_MODE,
           GL_EYE_LINEAR );
glTexGeni( GL_R,
           GL_TEXTURE_GEN_MODE,
           GL_EYE_LINEAR );
glTexGeni( GL_Q,
           GL_TEXTURE_GEN_MODE,
           GL_EYE_LINEAR );
```

Die nur durchgereichten (durch die Texturkoordinaten-Generierung) Vertexkoordinaten werden mit der Textur-Matrix transformiert. Die erste Transformation in der Ausführungsreihenfolge positioniert die Lichtquelle und richtet sie aus. Alle Tranformationen werden mit denselben Parametern wie beim Rendering der Depth Map ausgeführt. Als nächstes wird die Projektionsabbildung durchgeführt, mit der Sie den Öffnungswinkel des Spotlights bestimmt haben.

Zuletzt müssen Sie die nach der Projektion erhaltenen Koordinaten im Wertebereich *[-1;1] x [-1;1]* auf einen brauchbaren Bereich für die Texturen mit Clamping transformieren, also auf *[0;1] x [0;1]*. Zusammengefasst sieht das so aus:

```
glMatrixMode( GL_TEXTURE );
glLoadIdentity();
// [-1;1]x[-1;1]->[0;1]x[0;1]
glTranslatef( 0.5f, 0.5f, 0.5f);
glScalef( 0.5f, 0.5f, 0.5f );

// Projektion
gluPerspective(
  SPOT_ANGLE, 1.0f,1.0f,500.0f);

// Position/Richtung
gluLookAt(
          lightPosition[ 0 ],
          lightPosition[ 1 ],
          lightPosition[ 2 ],
          0, 0, 0,
          0, 1, 0 );
```

Mit diesen eingestellten Parametern können Sie die Szene mit Schatten rendern, sobald Sie das Texture Mapping angeschaltet und die Spotlight- oder Depth-Map-Textur mit *glBind(...)* aktiviert haben. Schalten Sie aber vorher noch die OpenGL-Beleuchtungsberechnung ein und platzieren Sie die Lichtquelle an der richtigen Stelle: an der Position, die Sie auch bei *gluLookAt(...)* angegeben haben.

## Texture Mapping und Shadow Depth Maps

Durch dieses Verfahren zum Schatten-Rendering sind zwei Texture Stages belegt. Bei modernen Grafikkarten stehen Ihnen mindestens noch zwei weitere Texture Stages für andere Texture Maps zur Verfügung. Wenn Ihnen das nicht reicht, können Sie auf die Spotlight-Textur verzichten. Diese ist nur eine optische Verschönerung, die Sie für die Schattenberechnung nicht unbedingt brauchen.

Wenn Sie die Spotlight Textur beibehalten wollen, gehen Sie wie folgt vor: Zu Beginn eines Frames erzeugen Sie die Depth Map. Beim Rendering der eigentlichen Kameraansicht rendern Sie ohne die Schattenberechnung, aber mit OpenGL-Beleuchtungsberechnung und Texturen. Damit haben Sie alle Textur Stages Ihrer Grafikkarte zur Verfügung.

In einem zweiten Renderpass der Szene deaktivieren Sie alle Features und verwenden das Schatten-Rendering. Zusätzlich aktivieren Sie das Blending, um die Farben im Colorbuffer mit den Helligkeitswerten aus der Schattenberechnung zu modulieren. Das erledigen Sie zum Beispiel mit:

```
glEnable( GL_BLEND );
glBlendFunc
       ( GL_ZERO, GL_SRC_COLOR );
```

Wenn Sie mit den verbleibenden Textur Stages auskommen, müssen Sie darauf achten, dass als erste Stage *GL_TEXTURE2_ARB* frei ist. Die Ausgabe der beiden darunter liegenden Stages ist, wenn Sie sie wie im Beispielprogramm konfigurieren, die mit der Helligkeit modulierte Grundfarbe des OpenGL-Materials. Das bedeutet, Sie selektieren für die Textur auf der Stage 2 (Zählung beginnt bei Null) wieder *Env Mode GL_MODULATE*:

```
glActiveTextureARB
          ( GL_TEXTURE2_ARB );
glEnable
             ( GL_TEXTURE_2D );
glBindTexture
        ( GL_TEXTURE_2D, tex );
glTexEnvi
              ( GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
                GL_MODULATE );
```

Die Texturkoordinaten müssen Sie dann mit den *glMultiTexCoord2fARB(...)*-Befehlen angeben, deren Funktions-Pointer Sie über die *wglGetProcAddress(...)*-Methode erhalten.

Achten Sie beim Einsatz von Multitexturing immer darauf, dass die Grafik-Hardware genügend Textur Stages zur Verfügung stellt. Die unterstützte Anzahl erfragen Sie von OpenGL:

```
GLint maxTexelUnits;
glGetIntegerv
    ( GL_MAX_TEXTURE_UNITS_ARB,
              &maxTexelUnits );
```

ET