Schatten in 3D-Szenen darstellen

Licht im Schatten

Die virtuelle Welt wirkt mit Schatten in 3D-Objekten realistisch. Wir zeigen, wie Sie Schatten rendern.

CARSTEN DACHSBACHER

Die Programmierer schaffen es mehr und mehr, die 3D-Grafik neuer Spiele der Realität anzupassen. Indem sie den Schatten von Objekten darstellen, verstärken sie den dreidimensionalen Eindruck.

Es gibt viele Verfahren, um in der Hardware-beschleunigten Computergrafik Schatten in Echtzeit zu gestalten. Sie unterscheiden sich in Qualität, Leistungsfähigkeit und Flexibilität.

Wir wollen Ihnen die Technik der Stencil-Buffer-Schatten vorführen (zu Stencil Buffers vgl. Heft 5/02, ab S. 190). Die Rendering-Auflösung dieser Schatten ist identisch mit der des restlichen Bildes. Ein weiterer Vorteil: 3D-Objekte können sich selbst beschatten.

Methoden der Schattendarstellung

Es gibt mehrere Methoden, wie Sie Schatten darstellen können.

• Die einfachste Variante sind die Projektionsschatten. Projizieren Sie das 3D-Objekt, das Schatten werfen soll, mit einer Projektionsabbildung auf eine Ebene. Eine Projektionsabbildung geben Sie in OpenGL mit einer Matrix an. Die projizierten Dreiecke zeichnen Sie in Schwarz oder einer anderen dunklen Farbe. Wenn Sie 3D-Objekte haben, die auf einen Boden oder wenige große Flächen (Ebenen) Schatten werfen sollen, ist dieses Verfahren zu empfehlen. Es ist aber viel zu aufwändig, wenn ein 3D-Objekt sich selbst oder andere 3D-Objekte beschatten soll, weil das Schatten werfende Objekt auf jede Ebene, die durch ein Polygon bestimmt wird, projiziert werden muss.

• Um Schattentexturen zu verwenden, platzieren Sie die OpenGL- Kamera an der Position der Lichtquelle und lassen sie in Richtung des Schatten werfenden Objekts zeigen. Den Framebuffer löschen Sie mit weißer Farbe. Rendern Sie das Objekt in Schwarz und kopieren Sie das gerenderte Bild in die Schattentextur. Wenn Sie ein anderes 3D-Objekt für das finale Bild rendern, projizieren Sie die Schattentextur auf dieses 3D-Objekt. Dazu generieren Sie etwa Textur-Koordinaten von OpenGL. Anschließend modulieren Sie die Helligkeit der gezeichneten Pixel mit den Helligkeitsintensitäten aus der Schattentextur. Der Vorteil dieser Methode: Sie müssen das Schatten werfende 3D-Objekt nur in die Textur zeichnen. Der Nachteil: Die Schattentextur entspricht nicht der Bildschirmauflösung, dadurch treten manchmal Aliasing-Effekte auf. Auch können 3D-Objekte nicht selbst Schatten werfen.

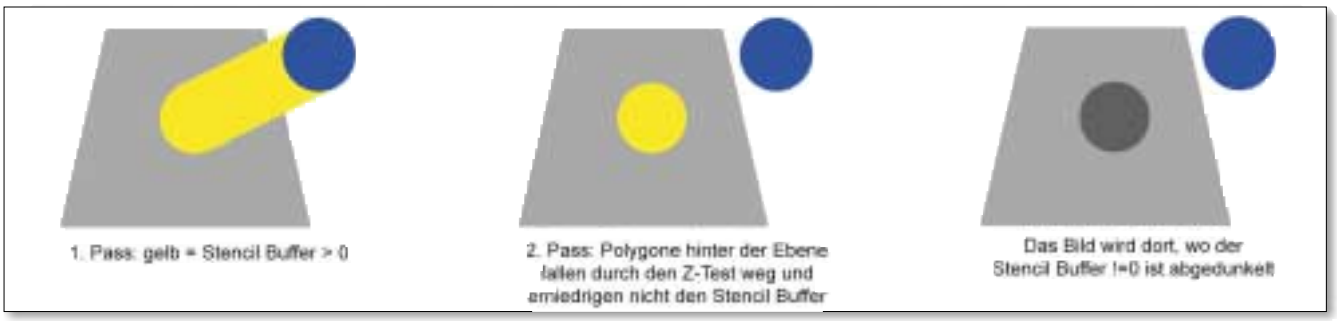
• Shadow Maps verwenden den Z-Buffer und projektives Texture Mapping (wie eben beschrieben). Auch hier entsprechen sich Kameraposition/-richtung und Position/Richtung der Lichtquelle. Damit rendern Sie die 3D-Szene aus der Sicht der Lichtquelle, wobei Sie auch den Z-Buffer zum Rendern verwenden. Den Z-Buffer kopieren Sie in eine Textur. Diese enthält Grauwerte, die die Entfernung der sichtbaren Teilen der 3D-Szene von der Lichtquelle repräsentieren. Diese Textur wird beim Rendern des fertigen Bildes mit der Textur-Transformation-Matrix von OpenGL und Textur-Koordinaten auf die 3D-Objekte gesetzt. Die Koordinatengenerierung stellen Sie so ein, dass die Koordinaten (x,y,z) im Raum eines Punktes und die Textur-Koordinaten (s,t) eines Pixels in der Shadow Map sind. Der Parameter r enthält die Entfernung zur Lichtquelle, wobei r die dritte Komponente der OpenGL Textur-Koordinatengenerierung ist.

Dann liefert OpenGL die Methode, um den Wert von r mit dem des Texels an der Stelle (s,t) in der Shadow Map zu vergleichen. Wenn r größer als der Texelwert ist, liegt der Pixel im Schatten. Dann liegt eine andere Stelle der 3D-Szene näher an der Lichtquelle und beschattet den gerade betrachteten Punkt.

Dieses Verfahren ist sehr flexibel und lässt auch Selbstbeschattung zu. Aller-

AUF DVD

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



MIT DREI SCHRITTEN bekommen Sie mit Stencil Buffers Schatten in die 3D-Szene.

dings ist, wie bei den Schattentexturen die Auflösung der Schattenränder an die Auflösung der Shadow Maps gebunden.

Stencil Buffers

Mit dem Stencil Buffer können Sie das Rendering für einzelne Pixel steuern. Im Stencil Buffer befindet sich für jeden Pixel eine Zahl, deren Bitbreite je nach Hardware und Anforderung unterschiedlich ist. Sie teilen dies OpenGL bei der Intialisierung mit, wenn Sie den Renderkontext gestalten.

Gebräuchliche Bitbreiten sind 1 oder 8 Bit pro Pixel. Diese Werte können Sie im Ganzen löschen und mit OpenGL-Primitiven wie Dreiecken, Linien, Punkten etc. beschreiben. Als Operationen sind denkbar: löschen, mit einem Wert beschreiben, erhöhen, erniedrigen und invertieren. Um das Rendering zu steuern, stellt Ihnen der Stencil Buffer die Methode zur Verfügung, um vor dem Setzen eines Pixels einen gegebenen Wert mit dem des Stencil Buffer an dieser Stelle zu vergleichen. Das Ergebnis des Vergleichs bestimmt, ob ein Pixel gezeichnet wird oder nicht. Den Stencil Buffer löschen Sie mit der Zeile:

```
glClear
( GL_STENCIL_BUFFER_BIT );
```

Um den Stencil Buffer zu beschreiben oder Vergleiche damit auszuführen, aktivieren Sie den Stencil-Test:

```
glEnable( GL_STENCIL_TEST );
```

Die Vergleichsoperation, die OpenGL beim Setzen jedes Pixels durchführt, legen Sie mit dem Befehl *glStencilFunc(...)* fest. Der erste Parameter gibt die Vergleichsfunktion an. Diese legt fest, ob der Wert im Stencil Buffer kleiner (gleich), größer (gleich) oder gleich einem Referenzwert sein muss, um den Test als gelungen zu bezeichnen. Der Referenzwert ist der zweite Parameter.

Sie können auch festlegen, dass der Test immer ein positives oder negatives Ergebnis liefert, wenn es nur darum geht, den Stencil Buffer mit Werten zu füllen. Mit dem dritten Parameter können Sie eine Bitmaske übergeben. Auf beide Werte, dem aus dem Stencil Buffer und dem Referenzwert, führen Sie vor dem Vergleich ein bitweises *AND*-Verfahren durch.

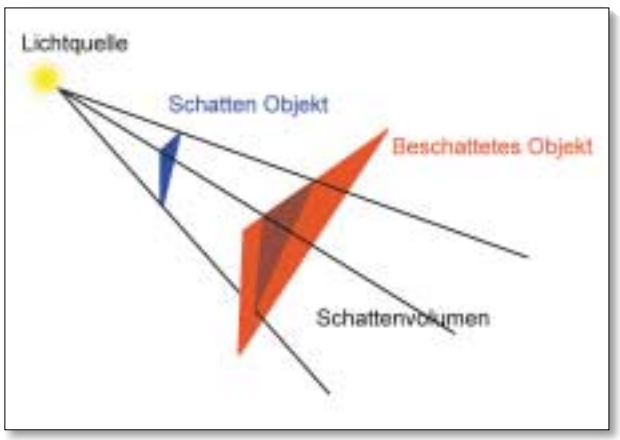
Mit dem OpenGL-Befehl *glStencil Op(...)* bestimmen Sie, was nach dem Stencil Test abhängig vom Ergebniss passiert, ob ein Pixel gesetzt wird oder nicht, und ob der Stencil-Buffer-Wert verändert wird oder nicht. Dabei unterscheiden Sie drei Fälle, bei denen auch das Ergebnis des Z-Buffer-Tests relevant ist:

- Der Stencil-Buffer-Test liefert ein negatives Ergebnis,
- der Stencil-Buffer-Test liefern ein positives Ergebnis, aber der Z-Buffer Test schlägt fehl,
- beide Tests liefern ein positives Resultat. Beim Z-Buffer Test kann das auch auftreten, wenn dieser deaktiviert ist.

Für jeden dieser drei Fälle geben Sie bei der Funktion *glStencilOp(...)* an, welche der oben genannten Operationen durchgeführt werden soll.

Schatten mit dem Stencil Buffer

Diese Technik betrachtet die Schatten eines Objektes als polygonale Volumen. Stellen Sie sich zunächst den einfachsten Fall vor, dass ein einzelnes Dreieck einen Schatten werfen soll. Halbgeraden, die von der Lichtquelle aus durch die Eckpunkte des Dreiecks gehen, begrenzen das Schattenvolumen. Dieses wird auf der einen Seite auch durch das Dreieck selbst begrenzt, in der anderen Richtung ist es theoretisch unendlich lang. Daraus erhalten Sie einen Pyramidenstumpf. Alle Teile der 3D-Objekte, die innerhalb dieses Pyramidenstumpfes liegen, befinden sich im Schatten des Dreiecks und werden nicht direkt von der Lichtquelle beleuchtet. Die Normale des Schatten werfenden Dreiecks zeigt zur Lichtquelle.



DAS SCHATTENVOLUMEN eines Dreiecks

Sie verwenden den Stencil Buffer nun, um festzustellen, welche Teile der 3D-Szene im Schattenvolumen sind und welche nicht. Für jeden Pixel des Bildes wird die Strecke zwischen Betrachter und Punkt in der Szene untersucht, was Sie wieder als Halbgerade auffassen können. Die Richtung ist durch den Ort des Pixels im Bild und den Tiefen-Wert des Z-Buffers an der entsprechenden Stelle gegeben. Der Wert des Stencil Buffer wird erhöht, wenn die Halbgerade in ein Schattenvolumen eindringt, und erniedrigt, wenn ein Schattenvolumen verlassen wird.

Um das Verfahren mit der Hardware-Beschleunigung zu nutzen, rendern Sie die 3D-Szene. Anschließend zeichnen Sie die Schattenvolumina in den Stencil Buffer. Zeichnen Sie nur die Polygone der Schattenvolumina, die zum Betrachter hinzeigen, stellen Sie die Stencil-Buffer-Operation auf *Inkrementieren* und aktivieren Sie den Z-Buffer-Test. Damit erhöhen Sie den Stencil-Buffer-Wert für das Eindringen in alle Schattenvolumina, die (für einen Pixel) zwischen Betrachter und der nächsten sichtbaren Stelle der 3D-Szene liegen. Mit den vom Betrachter wegzeigenden Polygonen verfahren Sie analog, nur dass Sie den Stencil-Buffer-Wert dekrementieren.

Als Resultat haben alle Pixel, die sich in einem Schattenvolumen befinden, einen Stencil-Buffer-Wert ungleich Null. Am einfachsten ist es jetzt, ein dunkles halbtransparentes Rechteck über das gesamte Bild zu zeichnen, wobei der Stencil-Buffer-Test auch auf ungleich Null gesetzt wird. Damit werden die beschatteten Teile abgedunkelt.

Diese Variante ist nicht ganz korrekt, weil beim Zeichnen der Szene die Beleuchtung normalerweise angeschaltet ist, und die beschatteten Teile jetzt nur abgedunkelt, aber beleuchtet, wurden. Im Beispielprogramm finden Sie die gerade beschriebene vereinfachte Variante. Wenn Sie ganz korrekt vorgehen wollen, verfahren Sie wie folgt:

- Zeichnen Sie die 3D-Szene mit Beleuchtungsattributen für Schatten (nur ambientes Licht).
- Deaktivieren Sie *Color- und Z-Buffer schreiben*, und aktivieren Sie den Z-Buffer Vergleich.
- Zeichnen Sie wie zuvor beschrieben die Schattenvolumina.
- Zeichen Sie die volle Beleuchtungsberechnung und Szene, wobei Sie den Stencil-Buffer-Vergleich auf Null setzen.

Beim Löschen des Stencil Buffer gilt: Wenn sich der Betrachter innerhalb eines Schattenvolumens befindet, müssen Sie den Stencil Buffer auf den Wert *1* statt *0* initialisieren. Der Grund: Betrachten Sie den Fall eines einzigen Schattenvolumens, dann würden nur die Dreiecke gezeichnet, die den Stencil-Buffer-Wert dekrementieren. Andere befinden sich hinter dem Betrachter. Der Stencil Buffer kann aber nicht kleinere Werte als Null annehmen, und somit würde die ganze Szene als beleuchtet dargestellt.

Optimierung und Implementation

Für komplexere Objekte ist es nicht sinnvoll, für jedes Dreieck ein eigenes Schattenvolumen zu konstruieren. Stattdessen verwenden Sie die Kanten, die die Silhouette des Objektes (betrachtet von der Lichtquelle aus) bilden. Daraus konstruieren Sie sich die Seitenflächen des Schattenvolumens.

Die Kanten der Silhouette sind solche, die Randkanten des 3D-Objektes sind, und solche, die Teil zweier Dreiecke sind, bei denen eines zur Lichtquelle hinzeigt und eines nicht.

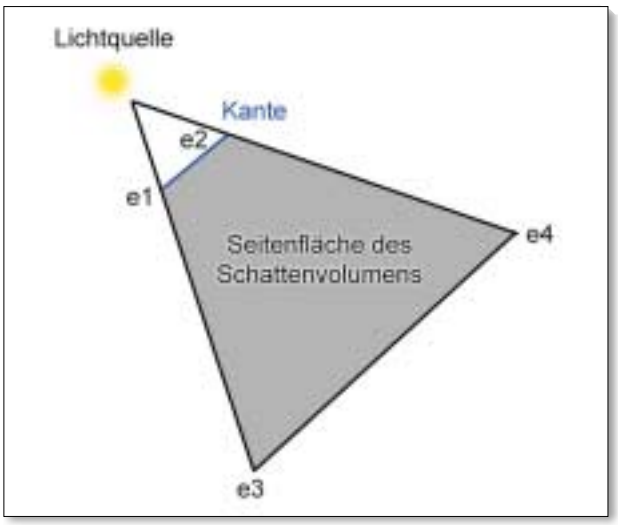
An dieser Stelle wollen wir Ihnen Schritt für Schritt vorführen, wie Sie die Stencil Buffer Schatten eines 3D-Objektes rendern. Dabei bauen Sie auf der Basis des OpenGL Frameworks auf. Dieses initialisiert OpenGL und lädt einfache 3D-Objekte für Sie, die jeweils eine Vertexliste (*pVertexList*) und eine Indexliste (*pFaceList*) inklusiver der Normalen für jedes Dreieck besitzen. Davon ausgehend erläutern wir die notwendigen Programmierschritte.

Zunächst einmal benötigen Sie die so genannte *Adjazenz*-Information, die ausweist, welche Polygone benachbart sind und sich welche Kante teilen. Dazu bilden Sie eine Liste mit allen Kanten des 3D-Modells (in *pEdgeList*), die jeweils die Referenz auf die benachbarten Dreiecke speichern. Anschliessend können Sie elegant für jedes Dreieck und jede seiner Kanten das benachbarte Dreieck feststellen:

```
// Für jedes Dreieck die Nachbarn
// finden und Pointer darauf
// speichern
for ( i = 0; i < nFaces; i++ )
{
  for ( int j = 0; j < 3; j++ )
  {
int edge =
 pFaceList[i].adjacent[j];
int *adj =
 &pEdgeList[edge].poly[0];

    if
( pEdgeList[ edge ].boundary==1)
pFaceList[i].adjacentFace[j]=
 NULL;
 else {
      if ( adj[ 0 ] == i )
pFaceList[ i ].adjacentFace[j]=
  &pFaceList[ adj[ 1 ] ]; else
pFaceList[ i ].adjacentFace[j]=
  &pFaceList[ adj[ 0 ] ];
} } }
```

Mit diesen Daten können Sie nun den Schatten eines 3D-Objekts zeichnen. Dabei beginnen Sie festzustellen, welche Polygone zur Lichtquelle hinzeigen und welche nicht. Das ist der gleiche Vorgang wie beim Backface Culling, nur dass Sie die Position der Lichtquelle statt der der Kamera heranziehen.



EINE SEITENFLÄCHE des Silhouettenvolumens wird durch eine Kante und die Lichtquelle bestimmt.

Um mit den Normalen der Dreiecke arbeiten zu können, muss die Position der Lichtquelle in dasselbe Koordinatensystem transformiert wird, in dem die Koordinaten der Vertices definiert sind, dem *Object Space*. Dazu invertieren Sie die Transformationen der Modelview Matrix durch die Umkehrung der Einzeltransformationen und deren Reihenfolge. Die Transformation der Lichtquelle in den Object Space nehmen Sie wie folgt vor:

```
VERTEX3D lightObject;
MATRIX44 modelView,
 invModelView;

// Modelview Matrix auslesen
glGetFloatv
( GL_MODELVIEW_MATRIX,
  modelView );

// Inverse Berechnen
inverseMatrix
( modelView, invModelView );

// und Position transformieren
// Vektor = Matrix * Vektor !
lightObject =
 invModelView * lightWorld;
```

Jetzt können Sie feststellen, welche Dreiecke zur Lichtquelle hinzeigen:

```
for
( int i = 0; i < nFaces; i++ )
{
  if
  (pFaceList[i].normal*
   lightObject+
      pFaceList[i].w > 0 )
  pFaceList[i].facesLight = 1;
  else
  pFaceList[i].facesLight = 0;
}
```

Konstruieren und zeichnen Sie alle Seitenflächen des Schattenvolumens wie im Bild. Zuvor setzen Sie die Parameter für den Stencil und Z-Buffer:

```
// Z-Buffer Test, Schreiben aus
glDepthFunc( GL_LEQUAL );
glDepthMask( GL_FALSE );

// Stencil Buffer, schreiben
glEnable( GL_STENCIL_TEST );
glColorMask( 0, 0, 0, 0 );
glStencilFunc
 ( GL_ALWAYS, 1, 0xff );

// 1.: Zum Betrachter zeigende
// Schattenvolumina Flächen

// OpenGL macht Backface Culling
// der Seitenflächen !

glFrontFace(GL_CCW);
glStencilOp
 (GL_KEEP,GL_KEEP,GL_INCR);

// Seitenflächen suchen/zeichnen
for ( i = 0; i < nFaces; i ++ )
if ( pFaceList[ i ].facesLight )
{
 FACE *k;
 //Jede Kante, angrenzend eines
 //zur Lichtquelle hin- & weg-
 //zeigenden Dreiecks:
 for ( int j = 0; j < 3; j++ )

 if(( k=
pFaceList[i].adjacentFace[j])!=0
    && !k->facesLight )
 {
// e1, e2: Punkte der Kante
  VERTEX3D *e1 =
&pVertexList[pFaceList[i].p[j]];
  VERTEX3D *e2 =
&pVertexList
 [pFaceList[i].p[(j+1)%3]];
  VERTEX3D e3, e4;

// weitere Punkte der Flächen
// durch die Halbgerade & dem
// Ort der Lichtquelle bestimmt:
e3 = *e1+(*e1-lightObject )
```

```
 *SLENGTH;
e4 = *e2+(*e2-lightObject )
 *SLENGTH;

  // Seitenfläche zeichnen
  glBegin( GL_TRIANGLE_STRIP );
    glVertex3fv( (GLfloat*)e1 );
   glVertex3fv( (GLfloat*)&e3 );
    glVertex3fv( (GLfloat*)e2 );
   glVertex3fv( (GLfloat*)&e4 );
  glEnd();
 }
}
```

UNSER BEISPIELPROGRAMM in Aktion

```
  // 2. Durchgang
  glFrontFace(GL_CW);
  glStencilOp
   (GL_KEEP,GL_KEEP,GL_DECR);

  // Zeichnen wie oben
  ...
```

Wie Sie im obigen Code-Ausschnitt sehen, sind die Seitenflächen des Schattenvolumens nicht unendlich lang, Sie sind ausreichend gross, so dass sie den sichtbaren Bereich verlassen. Das erreichen Sie durch die *SLENGTH*-Konstante.

Jetzt müssen Sie nur noch die Teile der 3D-Szene, die sich im Schatten befinden, abdunkeln. Dazu zeichnen Sie ein dunkles, teilweise transparentes Rechteck, wobei Sie den Stencil-Buffer-Test auf ungleich Null stellen:

```
glStencilFunc
 ( GL_NOTEQUAL, 0, 0xff );
glStencilOp
 (GL_KEEP,GL_KEEP,GL_KEEP);
// Farbe und Blending
glColor4ub( 0, 0, 0, 128 );
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA,
 GL_ONE_MINUS_SRC_ALPHA );

// Rechteck zeichnen:
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glMatrixMode( GL_PROJECTION );
glLoadIdentity();

glDisable( GL_DEPTH_TEST );
glBegin( GL_TRIANGLE_STRIP );
  glVertex2i( -1,  1 );
  glVertex2i( -1, -1 );
  glVertex2i(  1,  1 );
  glVertex2i(  1, -1 );
glEnd();
glEnable( GL_DEPTH_TEST );
```

ET

Das Beispielprogramm, das dies alles implementiert, finden Sie auf der Heft-CD.

Info:

„Advanced Rendering Techniques Using OpenGL“, SIGGRAPH 99 Course Notes, www.opengl.org/developers/code/sig99/index.html

www.dachsbacher.de/pcu