Realistische 3D-Bilder durch mangelnde Schärfe

Tiefenunschärfe mit Direct3D

Tiefenunschärfe ist ein Effekt, der deutlich zum realistischen Eindruck einer Animation beiträgt – aber allzu oft ignoriert wird. Wir zeigen Ihnen, wie Sie diesen Effekt mit Direct3D in Echtzeit erreichen.

Carsten Dachsbacher

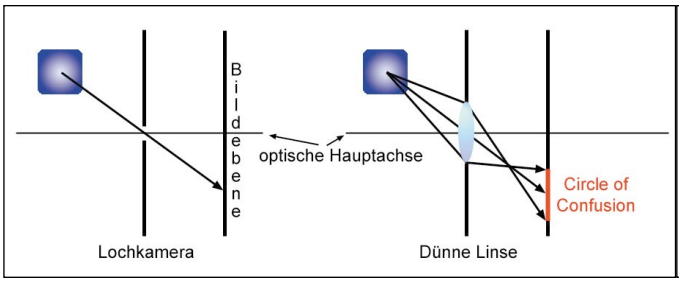
Das *Deferred Shading* konstruiert die Geometrie einer 3D-Szene zunächst ohne Beleuchtungsberechnung. Dabei zeichnen Sie nicht in den normalen sichtbaren Frame Buffer, sondern in so genannte Fat Buffers.

Tiefenunschärfe bewirkt, dass Objekte, die außerhalb des Fokus liegen, unscharf erscheinen. Typischerweise verwendet die Echtzeit-Computergrafik ein Lochkamera-Modell, um perfekt scharfe Bilder der ganzen Szene zu erzeugen. Reale Kameras brauchen Linsen mit endlichen Maßen, um die Szene auf die Bildebene abzubilden. Das verursacht die Tiefenunschärfe. Für höchste Qualität und fotorealistisches Rendering sind Tiefenunschärfe-Effekte ein wichtiger Bestandteil, sowohl für den realistischen Eindruck, als auch als Stilmittel, um die wesentlichen Komponenten der Szene zu unterstreichen. Für Depth-of-Field-Effekte (DOF) gibt es verschiedene Ansätze. Ein sehr eleganter Ansatz für DirectX9-Grafik-Hardware, den wir Ihnen hier auch vorstellen, ist der von T. Scheuermann/ATI Research.

Im Bild sehen Sie die Unterschiede zwischen einer Lochkamera und einer Abbildung durch eine Linse. Im ersten Fall passiert für jede Richtung nur ein Lichtstrahl. Bei der Abbildung eines Objekts außerhalb des Fokus auf die Bildebene mit dem Linsenmodell mit einer dünnen Linse tragen mehrere bzw. viele Strahlen zu einem Bildpunkt bei. Der Bereich auf der Ebene lässt sich gut durch einen Kreis approximieren, dem Circle of Confusion (CoC).

Implementation

Im vorgestellten Ansatz müssen Sie eventuell bestehende Implementationen nicht großartig modifizieren, um den DOF-Effekt darzustellen. Lediglich der Alpha-Kanal wird verwendet, um die Tiefe für jeden Pixel zu speichern. Wenn Sie den Alpha-Kanal für Blending Operationen



Kameramodelle: Für das mathematische Modell ziehen Sie die Loch- der Linsenkamera vor.

benötigen macht das nichts – die Tiefeninformation schreiben Sie erst im jeweils letzten Renderpass in den Alpha-Kanal, also nachdem Sie diesen nicht mehr benötigen.
Der Tiefenwert, den Sie in den Alpha-Kanal schreiben, ist die relative Tiefe. Das bedeutet, die Entfernung der *Near-Plane* entspricht der Tiefe *-1*, der *Far-Plane* *+1* und der Focus-Ebene entspricht ein Tiefenwert von *0*. Diese drei Werte übergeben Sie an den Pixel-Shader, mit dem Sie auch den Alpha-Wert berechnen. Ein Maß für die Größe des CoC erhalten Sie, wenn Sie den Absolutwert dieser Funktion nehmen. Vor der Ausgabe des Resultats müssen Sie das Intervall *[-1;+1]* auf *[0;1]* abbilden, da der Framebuffer nur diesen Wertebereich kennt. Folgender HLSL (High Level Shader Language) Code zeigt die Abbildung. Im Vertex-Shader berechnen Sie die Entfernung entlang der Blickrichtung:

```
// matWV: model->world transformation
float4 toVertex =
mul(matWV,vertex.pos)-
                  cameraPosition;
dist = dot(cameraDirection,toVertex);
```

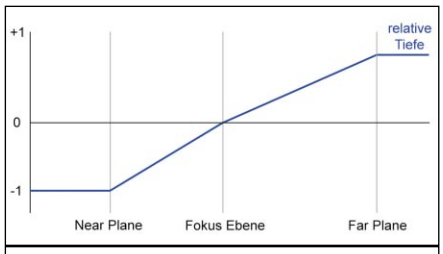
Im Pixel-Shader nehmen Sie die Abbildung vor:

```
float a;
if ( dist < zFocal )
  a=(dist-zFocal)/(zFocal-zNear); else
  a=(distance-zFocal)/(zFar-zFocal);
a = a * 0.5f + 0.5f;
```

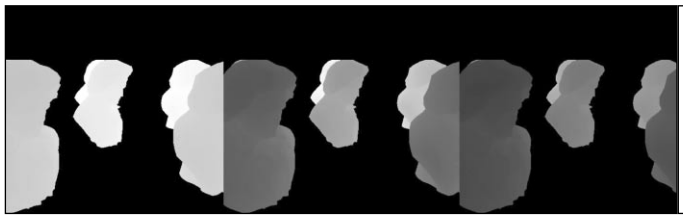
Das ist schon alles, was Sie während des normalen Renderings vornehmen müssen. Wie sich die Wahl der Focal-Plane auswirkt, sehen Sie im Bild. Als stilistische Erweiterung können Sie in der obigen Berechnung eine obere oder untere Grenze für die relative Tiefe angeben. Der Effekt: Sie erzwingen damit, dass bestimmte Gegenstände Ihrer 3D-Szene immer schärfer oder verschwommener dargestellt werden, als es eigentlich – aufgrund des Betrachterabstandes – der Fall wäre. Man kann dabei auch von semantischer Tiefenunschärfe sprechen.
Alle weiteren Schritte sind Aufgabe des Post-Processing, entstehen also durch Nachbearbeitung aus dem fertig gerenderten Bild, welches Sie am besten in eine Textur haben rendern lassen. Zunächst beginnen Sie damit – wie immer, wenn Sie einen Blur-Effekt darstellen wollen – eine niedriger aufgelöste Version Ihres Bildes anzulegen. Diese sollte etwa ein Viertel der Kantenlänge des Originals groß sein. Zusätzlich wenden Sie zum Glätten dieser Version noch einen Gauss-Filter an.
Der PC Underground Beitrag *Echtzeit Processing* (06/03, S.188, Heft-CD) beschreibt eine effiziente Art, einen Gauss-Filter auf ein Bild anzuwenden. Aufgrund der Separierbarkeit in horizontale und vertikale Anteile können Sie auch sehr große Filter-Kernel wie *7x7* in nur zwei Renderpasses anwenden. In diesem Fall genügt aber bereits ein Filter-Kernel der Größe *3x3*, den Sie in einem Renderpass implementieren, indem Sie im Vertex-Shader acht Textur-Koordinaten berechnen und den neunten Texel mit einer Textur-Koordinate auslesen, die Sie im Pixel-Shader berechnen – also mit einem Dependent-Textur-Lookup. Beachten Sie bei Textur-Loopups, dass Sie in Direct3D (und im Gegensatz zu OpenGL) zur Textur-Koordinate die Hälfte der Größe eines Texels addieren müssen, um einen Texel exakt in seiner Mitte auszulesen. Sie erhalten also nicht durch bilineare Interpolation einen interpolierten Farbwert.
Für den *3x3*-Gauss-Filter verwenden Sie folgenden Code im HLSL-Vertex-Shader (bei einer Textur-Größe von *512x512*), wenn Sie in *texCoord* die Textur-Koordinaten annehmen. So bilden Sie die Textur vollständig auf den Viewport ab:

```
float  ofs = 1.0/512.0;
float2 texelOffset =
  float4( 0.5/512.0, 0.5/512.0 );
texcoord0 = texCoord +
  float2( -ofs, -ofs ) + texelOffset;
texcoord1 = texCoord +
  float2( 0.0f, -ofs ) + texelOffset;
```

Analog bestimmen Sie die verbleibenden sieben Nachbar-Texel. Im Pixel-Shader lesen Sie die Textur aus, gewichten die Farb-Samples entsprechend der Gauss-Glockenfunktion und summieren die Farbwerte auf. So erhalten Sie das gewünschte Resultat.
An dieser Stelle kommt der Post-Processing-Effekt: Um das Bild unscharf zu machen, benötigen Sie einen Filter, für den Sie zunächst auf einer Kreisscheibe stochastisch verteilte Abtastpunkte gemäß einer Poisson-Verteilung wählen. Mit diesem Filter-Kernel tasten Sie die



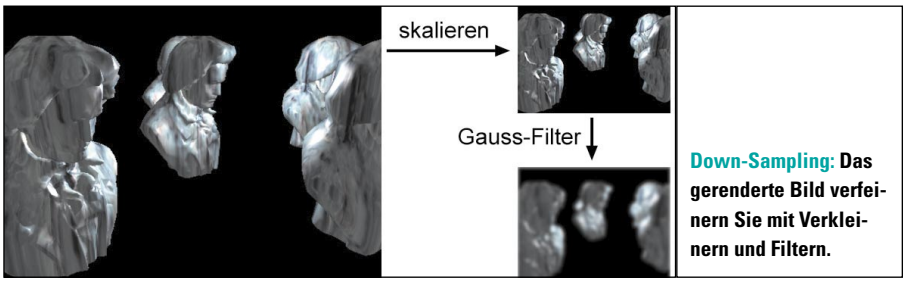
Entfernung: So bilden Sie die Tiefe auf eine relative Entfernung ab.



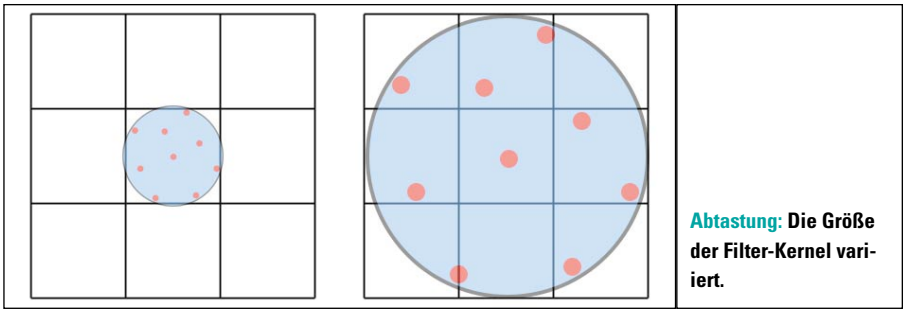
Im Alpha-Kanal: Die relative Tiefe hängt von der Wahl der Fokus-Ebene nah, mittel oder fern ab.

Bestimmung von Abtastpunkten

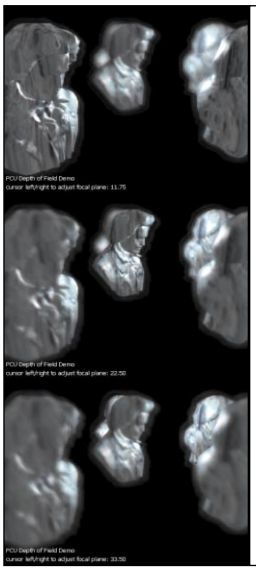
Aufgrund der Hardware-Beschränkungen können Sie pro Filter-Kernel nur sehr wenige Abtastpunkte wählen. Daher soll die Wahl nicht unbedacht erfolgen. Ein regelmäßiges Abtastmuster ist zwar leicht anzulegen und bietet eine gleichförmige Verteilung, allerdings ist die menschliche Wahrnehmung für die daraus resultierenden Artefakte sehr empfänglich. Deshalb ist es besser, eine Reihe zufällig verteilter Abtastpunkte zu wählen. Es genügt dabei aber nicht, einfach per Zufallszahlengenerator acht oder 12 Koordinatenpaare zu erzeugen. Denn zu leicht können zwei oder mehr Punkte dicht beieinander liegen.
Deshalb bietet sich folgende Methode an: Gehen Sie davon aus, dass Sie eine Liste mit 2D-Punkten haben, die sich alle in $[-1;1]^2$ befinden. Um nun die Liste um einen weiteren Punkt zu erweitern, generieren Sie zufällig einige Kandidatenpunkte aus je zwei Zufallszahlen *[-1;1]*. Achten Sie nur darauf, dass der Abstand des Kandidaten vom Ursprung kleiner oder gleich *1* ist – schließlich wollen Sie eine Kreisscheibe abtasten.
Für jeden dieser Kandidatenpunkte berechnen Sie nun den kleinsten Abstand zu den Punkten, die sich bereits in der Liste befinden. Schließlich fügen Sie den Kandidaten mit dem größten Abstand zur Liste hinzu. Jetzt fragt sich noch, mit welcher Liste Sie beginnen.
Angesichts der Tatsache, dass Sie einen Filter-Kernel bestimmen wollen, beginnen Sie mit der Liste, die nur den Punkt *(0,0)* enthält. Mit dieser Methode können Sie sukzessive beliebig viele Abtastpunkte erzeugen.



Down-Sampling: Das gerenderte Bild verfeinern Sie mit Verkleinern und Filtern.



Abtastung: Die Größe der Filter-Kernel variiert.



Das Resultat: Eine einfache Test-Szene gestaltet mit verschiedenen Fokus-Ebenen.

beiden Texturen (das hoch und niedrig aufgelöste Bild) in der Umgebung jedes Pixels ab. Die Größe der Kreisscheibe wählen Sie anhand des Unschärfewertes, den Sie im Alpha-Kanal abgelegt haben.

Um zu verhindern, dass Farbwerte von Objekten, die sich näher am Betrachter befinden, in weiter hinten liegende unscharfe Bereiche verwaschen, führen Sie für jeden Abtastpunkt noch einen Tiefentest durch. Wie Sie solche Abtastpunkte für den Filter-Kernel bestimmen können, zeigt der Kasten *Bestimmung von Abtastpunkten*.

Das Folgende erklärt die notwendigen Schritte anhand von HLSL-Programmcode, den Sie in unserem Beispielprogramm finden. Sie rendern in den Post-Processing-Schritten jeweils ein Rechteck, das den ganzen Bildschirm bedeckt. Die Textur-Koordinaten sind so gewählt, dass die Original-Bilder auf den Bildschirm gemapped werden. Unser Beispiel geht davon aus, dass die hochaufgelöste Version 512^2 Pixel enthält, die niedrige 128^2. Im Pixel Shader lesen Sie als erstes für jeden Pixel den entsprechenden Wert aus der *Hi-Res*-Textur. Die Tiefe entnehmen Sie dem Alpha-Kanal. Dort haben Sie sie ja gespeichert:

```
temp = tex2D( hiresImage, texcoord );
depth = temp.a;
```

Aus dem Tiefenwert berechnen Sie die Größe des Filter-Kernels für das *Hi-Res* Bild und skalieren daraus das Format für das *Low-Res*-Bild.

```
radius = abs( depth * 10.0f - 5.0f );
radiusLow = discRadius * radiusScale;
result = 0.0f;
```

Die nächsten Instruktionen führen Sie in einer Schleife für jeden Abtastpunkt des Kernels durch. Zuerst berechnen Sie die Textur-Koordinaten, an denen die Bilder abgetastet werden. Das *taps*-Array enthält die relativen 2D-Koordinaten im Bereich $[-1;+1]^2$ für den Filter.

```
float2 coordHigh, coordLow;
  float4 tapHigh, tapLow, tap;
  coordHigh = texcoord +
    1.0/512.0 * taps[ t ] * radius;
  coordLow  = texcoord +
    1.0/128.0 * taps[ t ] * radiusLow;
```

Mit den Faktoren *1/512* bzw. *1/128* passen Sie die Filter-Kernel der Textur-Auflösung an. Um Instruktionen zu sparen, können Sie auch Filter-Tap-Tabellen vorberechnen. Als nächstes lesen Sie die beiden Texturen aus:

```
tapHigh=tex2D(hiresImage,coordHigh);
  tapLow =tex2D(loresImage,coordLow);
```

Für jeden der Abtastpunkte bestimmen Sie, wie stark der Unschärfe-Effekt sein soll. Dazu dient der Unschärfewert aus dem Tiefenwert. Anhand dieses Wertes interpolieren Sie linear zwischen dem Farbwert des *Hi-Res*- und des *Lo-Res*-Bildes:

```
tapBlur = abs( tapHigh.a*2.0f-1.0f );
  tap = lerp(tapHigh,tapLow,tapBlur);
```

Um das Verwaschen der Farben von scharfen Objekten im Vordergrund in unscharfe, weiter entfernte Bereiche zu vermeiden, führen Sie diesen Tiefentest durch:

```
tap.a = ( tap.a < depth ) ? 1.0f :
   abs( tap.a * 2.0f - 1.0f );
```

Abschließend modulieren Sie die RGB-Werte des Filters und akkumulieren die Farbwerte:

```
tap.rgb *= tap.a;
  result += tap;
```

Wenn Sie diese Arbeitsschritte für jeden Abtastwert durchgeführt haben, erhalten Sie den resultierenden Farbwert durch einen letzten Normalisierungsschritt

```
return result / result.a;
```

Hardware-Betrachtungen

Dieser Pixel-Shader ist – selbst für moderne Grafik-Hardware – aufwändig. Auf einer ATI Radeon 9700 könnten Sie nicht mehr als vier Abtastpunkte verwenden, was nicht wirklich zufrieden stellende Resultate zeigt. Zwei Wege verbessern das Bild: Der erste ist ein optimierter Pixel-Shader, so dass mehr Abtastpunkte greifen. Dies können Sie z.B. mit Filter-Tap-Tabellen erreichen, die mit der jeweiligen Textur-Auflösung von *Hi-* und *Lo-Res*-Bild vormultipliziert sind. Immer müssen Sie aber die Farbwerte und deren Akkumulation gewichten. Das optimierte Beispielprogramm liest in einem Renderpass immerhin fünf Abtastwerte aus und liefert akzeptable Resultate. Für eine höhere Qualität (acht bis 12 Abtastwerte sind gut) benötigen Sie mehr Renderpasses – das ist der zweite Weg. Dabei rendern Sie beispielsweise drei Renderpasses mit unterschiedlichen Abtastpunkten. Das Resultat dieser Passes, also die Farbwerte, müssen Sie addieren. Entweder Sie verwenden ein additives Blending im Frame-Buffer, was aufgrund der 8-Bit-Genauigkeit pro Komponente schlechte Resultate liefert, oder Sie investieren mehr Aufwand. : et

Info
www.dachsbacher.de/pcu
www.ati.com/developer/techpapers.html