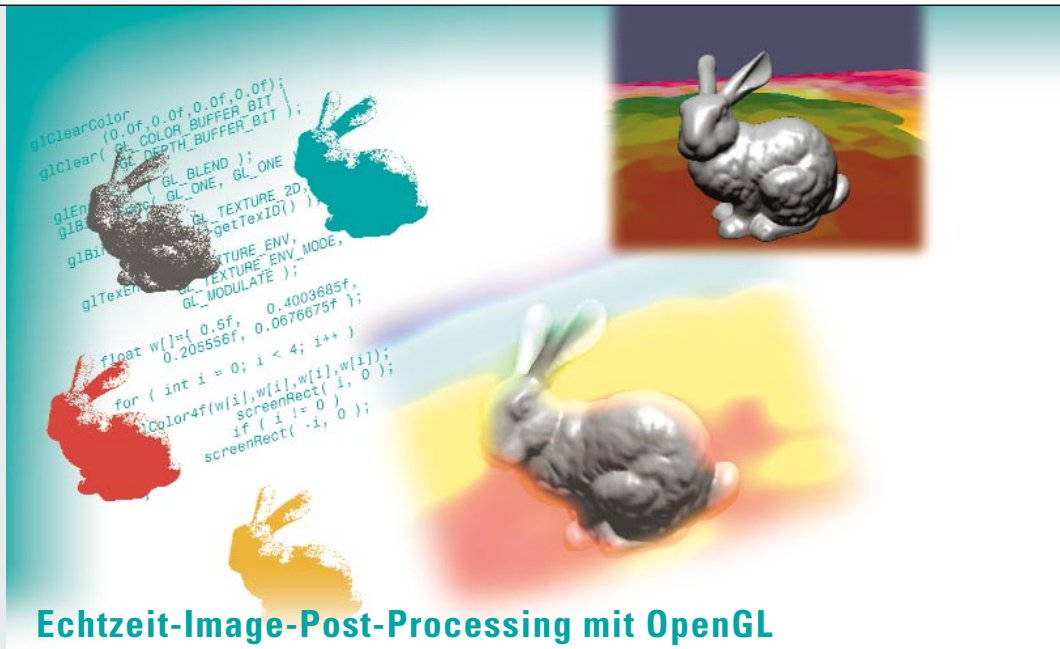
workshop

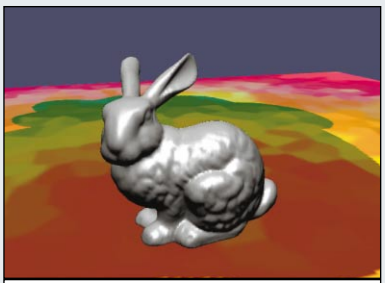


Echtzeit-Image-Post-Processing mit OpenGL

Bewegende Botschaften

Entlocken Sie Ihrer Grafikkarte ungeahnte Effekte wie Dithering, Kantenfilter und Leuchtspuren. Bereichern Sie Ihre Rendering- Szenen mit Post-Processing in Echtzeit per OpenGL. Mit der Macht von Bildern spielen Sie mit den Gefühlen des Betrachters.

Carsten Dachsbacher



Szene: Mit diesem Bild testen Sie alle Post-Processing-Effekte.

Grafikkarten schaffen es mittlerweile, zunehmend komplexe Szenen immer realistischer darzustellen. Das verbessert die Geometrie, Texturierung und Beleuchtung einer virtuellen Szene. Nicht nur das Rendering einer Szene kann den Betrachter überzeugend beeindrucken. Eine wichtige Rolle spielen auch die so genannten Post-Processing-Effekte. Solche Effekte fügen Sie dem Bild nach dem Rendering hinzu. Diese Art der Nachbearbeitung kann für viele Effekte auch mit Hilfe der Grafikkarte erfolgen. Einige Beispiele, manche mit Praxisrelevanz, andere eher aus der akademischen Ecke, lernen Sie in dieser Ausgabe kennen und programmieren. Für einige Effekte genügen Grafikkarten der Direct3D-8-Klasse wie nVidia-GeForce-3 und GeForce-4 oder ATI-Radeon-8500/9000. Aber Sie sehen auch, wie viel einfacher und effizienter es ist, solche Effekte auf den Grafikkarten der neuesten Generation, also der ATI-Radeon-9700 und nVidia-GeForce-FX, zu programmieren.

Post-Processing allgemein

Wie erwähnt, werden Post-Processing Effekte im Nachhinein zum Rendering hinzugefügt, oder dieses wird modifiziert. Um aber jeden Pixel des Bildschirms zu modifizieren, müssen Sie auf dessen Farb- und Alpha-Wert zugreifen. Das bedeutet, dass Sie eine Textur benötigen, in die Sie die 3D-Szene zeichnen. Dies können Sie prinzipiell auf zwei Arten erreichen: entweder Sie rendern die Szene in den Backbuffer (wie bei jedem herkömmlichen Rendering-Vorgang), oder Sie rendern direkt in eine Textur. Letzteres, auch die effektivste Variante, erreichen Sie mit den so genannten P-Buffers (PC Underground 3/03, ab. S.168). Im Beispiel-Programm zu dieser Ausgabe ist die P-Buffer Klasse enthalten, die Sie wie folgt einsetzen.

Ein P-Buffer (Pixel-Buffer) ist ein Speicherbereich, den Sie wie den normalen Frame-Buffer als Rendertarget, d.h. um dort etwas zu rendern, verwenden können. Ein P-Buffer kann einen eigenen Stencil und Depth Buffer besitzen und verschiedene Farbformate unterstützen. Sein großer Vorteil: Sie können ihn für ein anderes Rendertarget (wie für Frame- oder P-Buffer) als Textur verwenden. Dies bedeutet: Sie können Ihre 3D-Szene in eine dynamische Textur (den P-Buffer) zeichnen und anschließend mit dieser Textur Post-Processing Effekte anwenden. Dabei kann die Auflösung des P-Buffers der des Frame-Buffers entsprechen, muss aber nicht.

Die Verwendung der P-Buffer Klasse *CPBuffer* ist denkbar einfach: dem Konstruktor übergeben Sie die Auflösung des P-Buffers in X- und Y-Richtung sowie den Device Context des OpenGL-Fensters, womit ein P-Buffer mit 32 Bit Farbtiefe und einem 16 Bit Depth Buffer an-

gelegt wird. Um den P-Buffer als Rendertarget zu aktivieren, rufen Sie die Methode *makeCurrent()* auf. Diese verwendet intern die Funktion *wglMakeCurrent(...)*, mit der Sie auch den Frame-Buffer wieder als Rendertarget aktivieren. Die Parameter sind dabei der Device Context und OpenGL Rendering Context des Fensters.

So verwenden Sie nun einen P-Buffer als Textur: Zunächst wählen Sie eine OpenGL Textur mit Hilfe einer ID aus. Diese ID legt der Konstruktor bereits für Sie an. Im Anschluss binden Sie den P-Buffer an diese Textur ID:

```
glBindTexture( GL_TEXTURE_2D,
          pBuffer->getTexID() );
pBuffer->bind();
```

Jetzt können Sie die Textur wie üblich in OpenGL verwenden. Bevor Sie allerdings wieder den Inhalt des P-Buffers modifizieren wollen, müssen Sie den P-Buffer wieder von der Textur ID lösen. Dies übernimmt die Methode *release()*.

Dithering

Der erste hier vorgestellte Post-Processing Effekt ist mehr Spielerei und Beispiel: Dithering. Dieses Verfahren stellt ein Bild mit reduzierter Farbanzahl dar (im Beispiel schwarz und weiß), wobei es Muster verwendet. Dies soll den Eindruck von Farben oder Graustufen erwecken. Im Folgenden erfahren Sie, wie Sie Ihre 3D-Szenen in Echtzeit per Schwarz-Weiß-Dithering darstellen.

Bei vielen Dithering-Verfahren ist die Auflösung des resultieren Bildes höher als die des Ausgangsbildes. Solche Verfahren lassen sich mit dem Einsatz geeigneter Texturen leicht mit der Grafikkarte umsetzen. Andere Verfahren, wie z.B. das Floyd-Steinberg Verfahren, bearbeiten das Bild Pixel für Pixel und brauchen einen Übertrag, also ein gemerktes Zwischenresultat. Das Verfahren lässt sich somit nicht auf der Grafikkarte implementieren.

Das Beispiel verwendet ein einfaches Verfahren, das aus einem 32-Bit Farbbild mit Hilfe der nVidia Register *Combiner* (gewissermaßen das OpenGL Pendant zu den Direct3D acht Pixel Shaders) ein Schwarz-Weiß-Bild erzeugt. Die Register Combiner bestehen aus mehreren (abhängig vom GeForce Modell) *General Combiners*, die mit komponentenweiser Multiplikation, Skalarprodukten oder Summen operieren können und einem *Final Combiner*, der aus den Zwischenergebnissen den endgültigen Farbwert bestimmt. Auf die Ein- und Ausgabewerte eines Combiners können Sie so genannte *Mappings* anwenden. Mappings strecken, stauchen oder invertieren den Wertebereich und ändern Vorzeichen (PC Magazin Spezial 27).

So bearbeiten Sie jeden Pixel: Zunächst berechnen Sie aus dem Farbwert eine Helligkeit. Dabei können Sie die unterschiedliche perzeptive Wahrnehmung von rot, grün und blau durch unsere Augen berücksichtigen, indem Sie die Werte mit *0.3, 0.59* und *0.11* (in der Summe 100 Prozent) gewichten und aufsummieren. Dazu bilden Sie aus dem Farbwert der Textur und einem konstanten Vektor (*0.3, 0.59, 0.11, 0.0*) ein Skalarprodukt. Das Resultat speichern Sie in allen Komponenten des Zielregisters. An dieser Stelle setzen Sie eine zweite Graustufen-Textur ein. In ihr speichern Sie in jedem Texel einen Zufallswert zwischen *0* und *1*. Den Zufallswert, dessen Intervall Sie von *[0;1]* im Register Combiner auf *[-0.5; 0.5]* abbilden, addieren Sie zum Wert des Skalarproduktes und verwenden das Resultat als Alpha-Wert, der letztendlich zum Zeichnen verwendet wird.

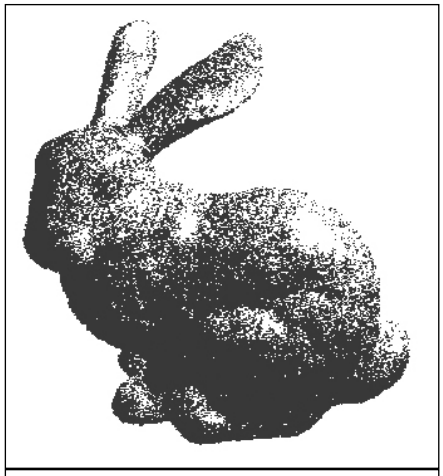
Zusammengefasst: Der erste General Combiner berechnet den Grauwert mit Hilfe des Skalarproduktes und damit den Alpha-Wert und setzt die RGB-Werte auf *1*. Der Final Combiner lässt den bereits berechneten RGBA Vektor durch.

Was haben Sie damit erreicht? Vom berechneten Grauwert addieren Sie eine Zufallszahl zwischen *-0.5* und *0.5* und erweitern das Werteintervall des Alpha-Werts auf *[-0.5, 1.5]*. Um diesen Post-Processing Effekt zu verwenden, löschen Sie den Frame-Buffer mit schwarzem Hintergrund und schalten vor dem Rendering den Alpha-Test ein, so dass nur Pixel gezeichnet werden, deren Alpha-Wert größer als *0.5* ist. Dann schalten Sie die Register Combiner mit dem oben beschriebenen Setup ein und zeichnen die gerenderte Szene (enthalten in der P-Buffer Textur) mit zwei Dreiecken über den ganzen sichtbaren Bereich. Damit sehen Sie den Dithering Effekt.

Kantenfilter

Der zweite Effekt ist ein aus Bildbearbeitungsprogrammen bekannter Bild-Filter, der Kanten in 2D-Bildern hervorhebt. Im Gegensatz zum ersten Beispiel verwenden Sie diesen Effekt in der Praxis wie beim Non-Photorealistic Rendering oder Toon Shading.

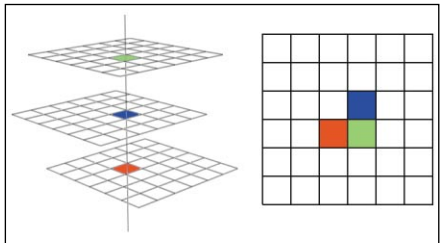
Kantenfilter untersuchen die Nachbar-Pixel jedes Pixels im Bild, bzw. die Differenz ihrer Helligkeitswerte. Überschreitet die Differenz betragsmäßig einen vorher festgelegten Schwellwert, so nimmt man an, dass es sich beim betrachteten Pixel um den Teil einer Kante handelt. Zunächst stellen Sie sich einen einfachen Kantenfilter vor, der nur vier Nachbar Pixel betrachtet, und den Sie mit den Register Combiners berechnen. Es wird dazu die Differenz der Helligkeiten des oben und unten, bzw. links und rechts benachbarten Pixels benötigt. Das Bild stellt einen Bildfilter dar, wie er in der Praxis üblich ist: in den Kästchen sind



Dithering: Frei nach Dürer stellen Sie die 3D-Szene schwarz-weiß dar.

0	+1	0
+1		-1
0	-1	0

Kantendetektion: Dies zeigt einen einfachen Filter-Kernel.



Nachbarschafts-Sampling: So arbeitet das Verfahren mit D3D8-Grafikkarten.

Kantenfilter: Die Ränder heben Sie mit Register Combiners hervor.

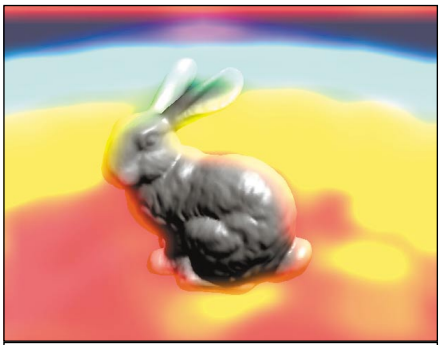
die Faktoren (Gewichte) der Pixel – in der Mitte der gerade Betrachtete – eingetragen, mit denen die Grauwerte multipliziert werden. Die Summe dieser Produkte wird (formal) am Ende durch die Summe aller Gewichte geteilt, um einen normalisierten Helligkeitswert zu erhalten. Diese Filtervorschrift wird auch als Filter-Kernel bezeichnet.

Die vier Helligkeitswerte entsprechen vier *Texture Lookups*, die Sie in einem Rendering Pass ab einer GeForce-3-Karte erledigen können. Dazu verwenden Sie die P-Buffer-Textur, die die gerenderte Szene enthält, auf vier Texture Stages. Die Texturkoordinaten der vier Stages sind dabei entsprechen um folgende Pixel Offsets verschoben: *(-1/0), (1/0), (0/-1)* und *(0/1)*. Beachten Sie, dass Sie die Verschiebung gemessen in Pixel durch die Auflösung des P-Buffers teilen müssen, um die tatsächliche Translation in Texturkoordinaten zu erhalten. Die Offsets können Sie auch mit Hilfe eines Vertex Programs berechnen, oder mit den Befehlen *glMultiTexCoord2fARB(...)* übergeben – es sind lediglich 4 * 4 Texturkoordinaten. Das Bild zeigt das Prinzip dieses Nachbarschafts-Samplings.

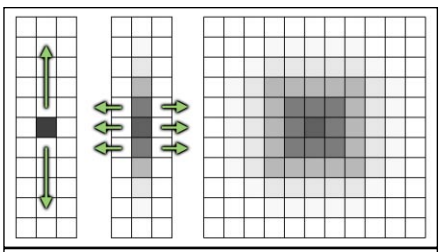
Der Sobel-Filter: Diese Technik bewältigt eine D3D9-Grafikkarte leicht.

-1	0	+1
-2		+2
-1	0	+1

Der Sobel-Filter: Zwei Filter-Kernel unterscheiden sich durch eine 90-Grad-Rotation.



Strahlende Welt: Im Glow-Effekt leuchtet die 3D-Szene.



Einfach schnell: Zweimaliges Filtern reduziert den Aufwand.

Die Berechnung des Kantenfilters sehen Sie hier in Pseudo-Notation, wobei *tex0..3* die Farbwerte der Nachbarpixel sind. *t1* und *t2* sind temporäre RGBA-Vektoren:

```
t1 = 2*(tex0-tex1);
t2 = 2*(tex2-tex3);
result = 4*[(t1 dot t1)
          + (t2 dot t2)];
```

Offensichtlich ist darin kein echter Schwellenwert-Vergleich enthalten. Vielmehr skalieren Sie stark die nicht Bool'schen Resultate der Differenzen, um einen ansprechenden Eindruck zu erhalten.

Das Beispielprogramm vervollständigt die Register-Combiner-Einstellungen jeweils mit der Syntax, wie sie die nVidia-Bibliothek *nvParse* verwendet. Um diese Bibliothek aber nicht linken zu müssen und um von ihr unabhängig zu bleiben, nimmt das Beispiel einen etwas holprigen Weg: es stellt die Combiner über die Befehle der *NV_register_combiner OpenGL Extension* ein.

Mit Direct3D-9-fähigen Grafikkarten (ATI Radeon 9700 und GeForce FX) können Sie eine Textur in einem Fragment-Programm mehrfach samplen (PC Underground 4/03, ab S. 212). Die Texturkoordinaten Offsets berechnen Sie am besten in einem Vertex-Programm. So können Sie den so genannten *Sobel*-Filter berechnen, den auch die Mustererkennung verwendet, um Kanten in 2D-Bildern zu erkennen. Dieser besteht aus einem horizontalen und einem vertikalen Filter-Kernel, deren Ergebnisse durch Maximumbildung verknüpft werden. Der Sobel-Filter ist dem obigen einfachen Filter deutlich überlegen. Neue Grafikkarten berechnen in einem Post-Processing Schritt den Sobel-Filter vollständig.

Der letzte Effekt dieser Ausgabe zeigt eine *Glow*- oder Leuchttechnik. Diese vermittelt den Eindruck, dass einzelne Teile der Szene hell schimmern und leuchten. Das Prinzip ist einfach: Zunächst zeichnen Sie die 3D-Szene normal in den Backbuffer des Rendertargets. Die Teile der 3D-Szene, die den Leuchteffekt besitzen sollen, zeichnen Sie in eine P-Buffer Textur. Der Trick bei diesem Effekt ist nun, auf die P-Buffer Textur einen starken Unschärfefilter anzuwenden und das dadurch entstehende Bild auf den Backbuffer Inhalt zu addieren. Somit hellen die Teile der 3D-Szene mit Leucht-effekt ihre Umgebung farblich auf.

Der Glow-Effekt

Einen solchen Unschärfefilter für die CPU zu programmieren, ist zwar leicht, jedoch unerwünscht. Erstrebenswert ist eine effiziente Lösung mit Hilfe der Grafikhardware. Den P-Buffer-Inhalt zu kopieren und die zusätzlich entstehenden Textur-Locks zu rendern, würde die Geschwindigkeit massiv einschränken. Um eine starke Unschärfe zu erzeugen, benötigen Sie, um einen neuen Farbwertes für einen Pixel zu berechnen auch die Farbwerte von einer n^2 *(n=8)* Pixels großen Nachbarschaft. Die Gewichte der Nachbarpixel können Sie z.B. mit einer Gauss'schen Glockenfunktion bestimmen, um eine Abnahme der Gewichte mit dem Abstand zum betrachteten Pixel zu erreichen.

Zunächst könnte man annehmen, Sie würden dazu *8*8=64* Texturzugriffe benötigen. Glücklicherweise ist dem nicht so, denn viele solche Filter-Operationen lassen sich aufspalten: in zweimaliges Filtern, mit einer Filtergröße von *n*, also in diesem Beispiel nur acht Nachbarpixeln.

Anschaulich bedeutet dies: als Erstes wenden Sie einen horizontalen Filter an. Anschließend einen vertikalen Unschärfefilter auf das bereits gefilterte Bild. Dazu benötigen Sie außer dem P-Buffer, der die leuchtenden Teile der 3D-Szene enthält einen weiteren P-Buffer, in den Sie das Ergebnis nach dem ersten Filtervorgang schreiben und den Sie als Textur, also Quelle, des zweiten Filterns verwenden.

Um die *n=8* Nachbarpixel zu gewichten und aufzusummieren, benötigen Sie mit einer Direct3D-8-Grafikkarte zwei Renderpasses pro Filtervorgang, weil Sie nur auf vier Texturen bzw. Texel pro Pass zugreifen können. Der Zugriff erfolgt dabei nach demselben Prinzip wie im Bild *Nachbarschafts Sampling*: Sie verwenden dieselbe Textur vierfach mit unterschiedlichen Texturkoordinaten. Die zwei Werte (entstanden aus jeweils 4 Farbwerten) können Sie durch additives Blending mit der Funktion *glBlendFunc(GL_ONE, GL_ONE)* aufsummieren.

Mit einer Direct3D-9-Grafikkarte können Sie einen dieser Filtervorgänge in einem Renderpass

durchführen, da Sie eine Textur achtmal an beliebigen Texels auslesen und die Farbwerte gewichten und aufsummieren können.
Als Notlösung bei älteren Grafikkarten rendern Sie jeden Filtervorgang achtmalig. Die Gewichtung der einzelnen Nachbarpixel durch die Verschiebung der Textur erreichen Sie über den OpenGL-Farbwert, wenn Sie das Textur Environment auf *GL_MODULATE* stellen. Um die Werte zu summieren, setzen Sie wieder additives Blending ein.
Der Quelltext für die horizontale Filterung *(screenRect(du,dv)* zeichnet ein Quadrat mit der P-Buffer Textur über den gesamten sichtbaren Bereich. Die Parameter bezeichnen dabei die Verschiebung der Texturkoordinaten in Texeln:

```
glClearColor (0.0f,0.0f,0.0f,0.0f);
glClear( GL_COLOR_BUFFER_BIT |
         GL_DEPTH_BUFFER_BIT );

glEnable    ( GL_BLEND );
glBlendFunc( GL_ONE, GL_ONE );

glBindTexture( GL_TEXTURE_2D,
        pBuffer->getTexID() );
glTexEnvi( GL_TEXTURE_ENV,
           GL_TEXTURE_ENV_MODE,
           GL_MODULATE );
float w[]={ 0.5f,   0.4003685f,
       0.205556f, 0.0676675f };

for ( int i = 0; i < 4; i++ )
{
 glColor4f(w[i],w[i],w[i],w[i]);
             screenRect( i, 0 );
   if ( i != 0 )screenRect( -i, 0 );
}
```

Einen horizontalen bzw. vertikalen Filterdurchgang erreichen Sie mit der DirectX9-Generation von Grafikkarten in einem Rendering Pass. Das dazugehörige Fragment Program sieht folgendermaßen aus:

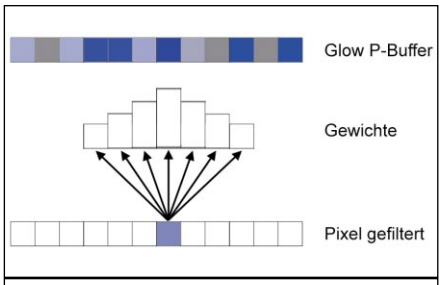
```
OUTPUT color = result.color;
TEMP color0, color1, color2, color3,
color4, color5, color6;
ALIAS temp = color0;
PARAM weights = { 0.0676675, 0.205556,
0.4003685, 0.5 };

TEX  color0, fragment.texcoord[ 0 ],
texture[ 0 ], 2D;
....
TEX  color6, fragment.texcoord[ 6 ],
texture[ 0 ], 2D;

MUL  temp, color0, weights.w;
MAD  temp, color1, weights.z, temp;
...
MAD  result.color, color6, weights.x,
temp;
END
```

Blurring: Das Verfahren benötigt mit D3D8-Hardware zwei Renderpasses pro Filtervorgang, weil nur vier Texturen pro Pass zugreifen können.

Allgemein gilt so, dass Sie mit weniger Renderpasses eine höhere Genauigkeit erhalten, die nicht nur theoretisch, sondern auch sichtbar ist. Der Grund: Die Register-Combiner und die Fragment-Programme arbeiten intern mit höherer Präzision, letztere sogar mit Floating Point Genauigkeit.
Beim Aufsummieren durch additives Blending hingegen wird mit 8 Bit pro Farbkomponente gerechnet, was einen deutlichen Datenverlust bedeuten kann.
Um einen guten Glow Effekt zu erhalten, müssen Sie keine hohe Auflösung des P-Buffers wählen. Im Gegenteil: eine niedrigere Auflösung verstärkt die Leuchtbereiche und berechnet den Effekt schneller. Bei zu geringer Auflösung stören Aliasing Effekte, die sogar dazu führen können, dass kleinere leuchtende Objekte übersprungen werden. Deshalb experimentieren Sie am besten, um eine geeignete Auflösung zu finden.
Eine bessere Kontrolle über die Leuchteffekte erreichen Sie, indem Sie nicht einen Glow Effekt für den ganzen Bildschirm, sondern selektiv für einzelne Bereiche oder 3D-Objekte berechnen. Dieser zusätzliche Aufwand lohnt sich bei komplexeren Leuchteffekten. : et



Blurring: Und so arbeitet das Verfahren mit D3D9-Grafikkarten.

Verweise
www.dachsbacher.de/pcu
www.ati.com
www.nvidia.com