AUF CD 1
Die Quelltexte sowie die fertig übersetzten Routinen finden Sie im Verzeichnis *Praxis/Programmierung/PC Underground*.

Texturen generieren

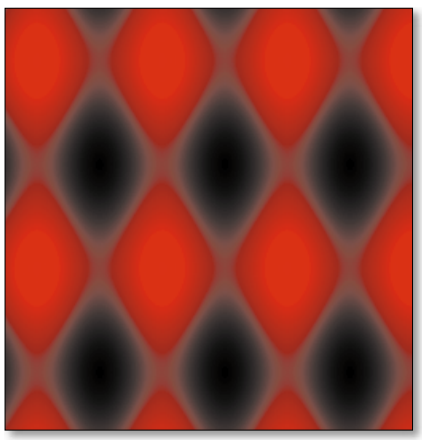
Mit Marmor oder Holz

Mit wenigen Zeilen Programmcode gestalten Sie die schönsten Oberflächen mit Texturen aller Art. Ihrer Fantasie sind keine Grenzen gesetzt, um die Aufmerksamkeit des Betrachters zu erringen.

CARSTEN DACHSBACHER

Generieren Sie mit wenig Aufwand schrill bunte oder realistisch anmutende Texturen. Diese verwenden Sie für eigene Grafiken, Ihre Webpages oder 3D-Modelle. Diese Texturen sind dann garantiert frei von Rechten Dritter. Das ist bei Texturen von Modelling-Programmen oder CD-ROMs nicht immer der Fall.

Die Texturen in Ihren Programmen speichern Sie über die Parameter, mit denen Sie sie angelegt haben. Damit sparen Sie viel Platz im Gegensatz zur Datenmenge, die eine fertige Textur als Bild belegt. Textur-Parameter speichern Sie mit etwa 100 Byte oder weniger.



DIE SINUSFUNKTIONEN färbt die Textur.

Wir zeigen Ihnen Schritt für Schritt, wie Sie Ihren eigenen Textur-Generator bauen. Dieser arbeitet mit Layers. Das ist ein Speicherbereich, der eine temporäre Textur oder andere für die Texturgenerierung notwendige Daten enthält. Ein Layer hat die Ausmaße der Textur, die Sie generieren wollen. Er besteht aus drei Kanälen, je einen für die Grundfarben Rot, Grün und Blau. Ein Textur-Generator, wie Sie ihn programmieren, arbeitet mit vier Arbeitsschritten:

- Er generiert einfache temporäre Basis texturen,
- verzerrt Texturen,
- Farboperationen
- und Filterfunktionen.

Notwendig ist nur der erste Schritt. Die drei anderen sorgen dafür, dass die Texturen interessant wirken. Sie generieren einfache Texturen auf einigen Layers. Dann verzerren Sie den ersten Layer mit den Daten des zweiten und verändern mit dem Resultat die Textur des dritten. Einfach generierte Texturen können Sie aneinander legen, ohne einen erkennbarer Rand zu lassen.

Zunächst definieren Sie die Datenstrukturen für Ihre Layer. Diese bestehen aus einer Struktur für die Farbkanäle eines Pixels und einer Liste aus Bildern der gewünschten Größe. Der Wertebereich unserer Texturen bewegt sich zwischen *0* bis *255*, wie der verwendete *unsigned char*-Wert definiert ist:

```
// Definitionen für die Layer
typedef struct
{ unsigned char r, g, b; }
 COLOR;
COLOR layer
[ MAXLAYER ][ SIZE * SIZE ];
```

Basistexturen

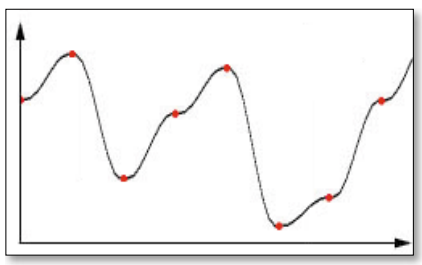
Um einfache Texturen zu erzeugen, setzen Sie Pixel auf den Layer. Am besten bestimmen Sie die Farbwerte durch zwei überlagerte Sinusfunktionen über der Textur:

```
// Sinusfunktion
for ( y = 0; y < SIZE; y++ )
 for ( x = 0; x < SIZE; x++ )
 {
  wert=127+63.5*sin(x*faktor)
      + 63.5 * sin(y*faktor );
  layer[0][y*SIZE+x].r=wert;
 }
```

Das Resultat überzeugt nicht. Sie können Layers dieser Art aber verwenden, um andere Layers zu verzerren. Verwenden Sie die zweite einfache Basistextur, um die Helligkeit anderer Layer zu ändern. Betrachten Sie jeden Pixel, und weisen Sie ihm eine Farbe entsprechend seinem Abstand zum Mittelpunkt zu. Es gilt dann der Satz: Je kleiner der Abstand, desto heller die Farbe.

Eine etwas komplexere Methode stellen Sub-Plasmas dar. Zuerst setzen Sie einige Pixel mit zufälliger Helligkeit an bestimmte Positionen im Layer. Beispielsweise wählen Sie einen Abstand von acht Pixeln zwischen zwei Zufallswerten. Mit diesen Zufallswerten interpolieren Sie die Farbwerte aller anderen Pixel.

Dabei setzen Sie im zweidimensionalen Raum entsprechend alle Pixel an jeder durch acht teilbaren *x*- und *y*-Koordinate. Bei der Interpolation ist entscheidend, welches Verfahren Sie verwenden. Eine lineare Interpolation führt zu unschönen Ergebnissen. Das Bild markiert die gesetzten Zufallswerte durch rote Punkte.



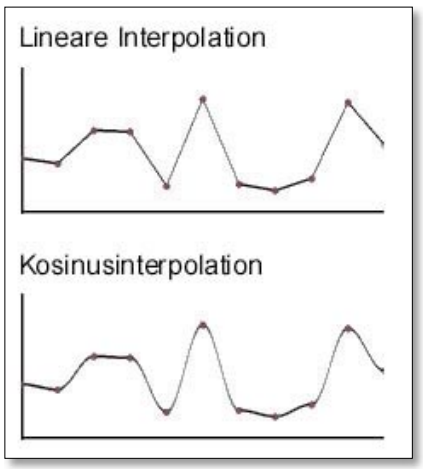
EINE NOISE-FUNKTION, die durch wenige Stützpunkte festgelegt ist

Der Rest der Kurve ist gleichmäßig glatt und führt zu einem hervorragenden Sub-Plasma wie im vorigen Bild. Wer sich in die mathematische Welt der Splines wagt, verwendet für die Interpolation die Catmull-Rom-Splines. Mit deutlich weniger Rechenaufwand kommen Sie aus, wenn Sie die Kosinus-Interpolation verwenden.

Verwenden Sie das Verfahren von Ken Perlin, um Texturen zu synthetisieren. Die Homepage von Ken Perlin, auf der Sie auch seinen Artikel über prozedurale Texturen lesen, finden Sie unter http://mrl.nyu.edu/perlin.

Eine Perlin-Noise-Funktion liefert zu einem Parameter (in diesem Fall eine ganze Zahl) eine Zufallszahl zurück. Wenn Sie zweimal denselben Parameter übergeben, muss die Funktion auch zweimal dasselbe Resultat erzeugen. Anderenfalls erhalten Sie trotz gleicher Startparameter nie zweimal dieselbe Textur.

Perlin-Noise-Funktionen sind Erweiterungen der Sub-Plasmas: Mehrere Sub-Plasmas mit unterschiedlicher Amplitude und Frequenz werden addiert. Die Amplitude bezeichnet den höchsten vorkommenden Zufallswert eines Sub-Plasmas, und mit der Frequenz bestimmen Sie den Abstand der Zufallswerte.



DER VERGLEICH ZEIGT die Unterschiede zwischen Linear- und Kosinus-Interpolation.

Sie können mehrere Noise-Funktionen unterschiedlicher Amplitude und Frequenz zu einer Perlin-Noise- Funktion summieren.

Analog können Sie auch im Zweidimensionalen vorgehen. Verschiedene Sub-Plasmas ergeben zusammen eine Perlin-Noise-Textur.

Das Beispiel interpoliert zwischen den Zufallswerten der Sub-Plasmas nur linear, was wegen der Überlagerung der Sub-Plasmas im Endbild nicht mehr auffällt.

Die Amplitude und die Frequenz, die Sie für die einzelnen Noise Funktionen verwenden, können Sie durch die *Persistence* festlegen. Sie legen nur noch eine Amplitude und eine Frequenz für die erste Funktion fest. Für die jeweils nächste Funktion, verdoppeln Sie die Frequenz und multiplizieren die Amplitude mit der Persistence. Der Wert der Persistence sollte zwischen *0* und *1* liegen. Größere Werte bedeuten höhere Frequen-

INTERPOLATIONSVERFAHREN

Bei der Bildbearbeitung und der Generierung von Texturen entscheidet die Methode der Interpolation über die Bildqualität. Meist genügt die lineare Interpolation. Diese können Sie bedenkenlos anwenden, wenn Sie nur über wenige Pixel interpolieren oder der interpolierte Bereich noch weiter überdeckt wird (bei Perlin Noise).

Angenommen, Sie wollen Werte wie bei einer Noise-Funktion an der Stelle *x* auslesen. Dann muss *x* nicht ganzzahlig sein, die Werte sollen aber für ganzzahlige *x* bekannt sein. Sie interpolieren zwischen den Werten, die an der Stelle des auf- und des abgerundeten *x* liegen:

```
wert1 = abgerundet( x );
wert2 = aufgerundet( x );
```

Es bleibt zu entscheiden, wie Sie die beiden Werte gewichten. Bei der linearen Interpolation verwenden Sie den Nachkommaanteil von *x*. Je kleiner dieser ist, desto näher liegt der unbekannte Wert an der Stelle *x* am Wert *1*. Die Gewichtungsfaktoren berechnen Sie wie folgt, wobei die Summe 1 ergibt:

```
faktor1 = 1.0 - nachkomma(x);
faktor2 = nachkomma( x );
```

Den interpolierten Wert erhalten Sie mit

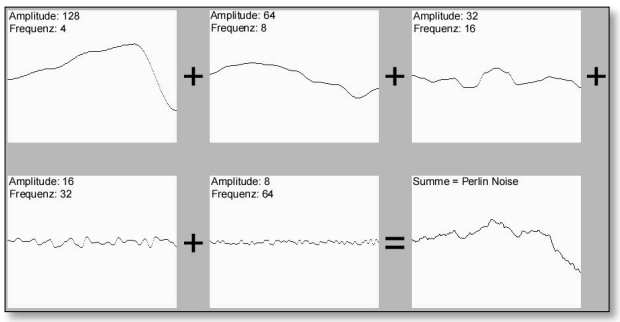
```
wert=wert1*faktor1+wert2*faktor2
```

Wenn Sie mehr Rechenaufwand investieren, erzielen Sie mit der Kosinusinterpolation abgerundete Ergebnisse:

```
// Kosinusgewichtung
ft = nachkomma( x ) * PI;
f = (1 - cos( ft ) ) * 0.5

faktor1 = 1.0 - f;
faktor2 = f;
wert=wert1*faktor1+wert2*faktor2;
```

Bei der Kosinusinterpolation steigt der Gewichtungsfaktor *f* an den Rändern langsamer. Dadurch erhalten Sie in der Nähe der Zufallswerte abgerundete Verläufe.



EINDIMENSIONALE NOISE-FUNKTIONEN addieren Sie zu einer Perlin-Noise-Funktion.

zen, also mehr Rauschen. Verdoppelte (überlagerte) Frequenzen nennt man Oktaven, da bei Klängen eine Verdoppelung der Frequenzen einem Sprung von einer Oktave entspricht. Wie viele Oktaven Sie wählen, ist Ihre Entscheidung. Berücksichtigen Sie nur, dass die Amplitude irgendwann so klein wird, dass die Funktion nicht mehr ins Gewicht fällt.

Wie erzeugen Sie Noise-Funktionen? Die herkömmlichen Zufallszahlengeneratoren, die Ihnen C anbietet, liefern bei jedem Aufruf eine neue Zahl. Da das Ergebnis aber reproduzierbar sein muss (weil Sie eine Noise-Funktion eventuell mehrmals an derselben Stelle berechnen müssen), können Sie diese nicht verwenden.

Sie können eine Funktion wählen, die relativ zufällig Werte liefert – meistens sehr große Primzahlen. Folgende Funktion berechnet eine Zufallszahl zu *x* zwischen *-1* und *1*:

```
x = (x<<13) ^ x;
return
(1.0-((x*(x*x*15731+789221)
+1376312589)&7fffffff)/
 1073741824.0);
```

Eine andere Methode legt beim Start des Programms eine Tabelle mit Zufallszahlen mit dem Generator an. Es genügen zum Beispiel 4096 verschiedene Zahlen. Als Funktion dient dann

```
return randomTable[ x & 4095 ];
```

Verzerrte Texturen

Aus Basistexturen können Sie interessante Texturen machen. Sie verzerren das Bild, indem Sie jeden Pixel des Bildes betrachten und ihm den Wert eines anderen Pixels zuweisen. Dieser Pixel liegt zum Beispiel drei Pixel tiefer und vier Pixel rechts.

Schwieriger ist es, einen Wert mit einer Verschiebung von *2.7* Pixel tiefer und *4.2* Pixel rechts auszulesen. Wenn Sie die Werte auf *3* und *4* aufrunden, erhalten Sie eine sehr körnige Textur mit ungewünschten Aliasing-Effekten.

Sie lösen die Aufgabe mit der bilineare Interpolation. Dazu nehmen Sie den Pixel, den Sie auslesen wollen. Sie ordnen ihm *x*- und *y*-Koordinaten zu, denen Sie die gerundeten Werten von *2* und *4* zuweisen. Dann betrachten Sie die vier umliegenden Punkte:

```
(x, y),
(x+1, y),
(x, y+1)
(x+1, y+1)
```

Die Farbwerte dieser Pixel gewichten Sie abhängig vom Nachkomma-Anteil der Verschiebung. Für den ersten bis zum vierten Fall ergeben sich folgende Punktwerte:

```
0.7 * 0.2 = 0.14
0.3 * 0.2 = 0.06
0.7 * 0.8 = 0.56
0.3 * 0.8 = 0.24
```

Die vier Gewichte ergeben in der Summe den Wert *1*. Mit diesen Gewichten multiplizieren Sie die Rot-, Grün- und Blau-Farbanteile der umliegenden Pixel und addieren diese. Damit erhalten Sie einen gefilterten Farbwert ohne hässliche Verzerrungen.

Texturen lassen sich nach verschiedenen Methoden verzerren:

- Bei der ersten verwenden Sie eine Funktion, die Ihnen abhängig von der Position Ihres Pixels einen Verschiebungsvektor liefert.
- Die zweite Variante verwendet den Inhalt eines oder zweier Layers, um die Verschiebung eines Pixels zu bestimmen (Map-Distortion, siehe nächsten Abschnitt).
- Sie können eine Textur auch verzerren, indem Sie die Verschiebung an einer Pixelposition durch zwei Sinusfunktionen berechnen:

```
x_move =
 sin( x * 0.03 )
* 4.0;
y_move =
 sin( x * 0.04 )
* 4.0;
```

Damit setzen Sie an der Pixelposition *(x, y)* den Wert, den Sie bei *(x+x_move, y+y_move)* auslesen.

- Eine andere Verzerrung (in Adobe Photoshop gebräuchlich) nutzt den Twirl-Effekt. Dieser dreht das Bild. Den Drehwinkel eines Pixels bestimmen Sie über seinen Abstand zum Mittelpunkt der Textur.

Map-Distortion

Bei der Map-Distortion berechnen Sie den Verschiebungsvektor eines Pixels durch die Helligkeitswerte der entsprechenden Pixel in den anderen Layern. Die Helligkeitswerte multiplizieren Sie am besten mit einem Wert zwischen *0* und *1*. Sie können diese Werte auch als Parameter einer Sinusfunktion auffassen und so eine Marmortextur erreichen, die zum Beispiel auf das 3D-Modell einer Vase passt.

Der Pseudocode für eine Map-Distortion sieht wie folgt aus. Hierbei verzerren Sie den roten Farbkanal von Layer *3* mit den Layern *1* und *2* und schreiben das Ergebnis in Layer *4*:

```
for ( y = 0; y < SIZE; y++ )
  for ( x = 0; x < SIZE; x++ )
  {
   x_move = layer[ 0 ].r * 0.1;
   y_move =
   sin(layer[ 1 ].r * 0.1);
   layer[ 4 ][ x + y * SIZE ] =
      interpolatePixel
  ( 3, x+x_move, y+y_move );
  }
```

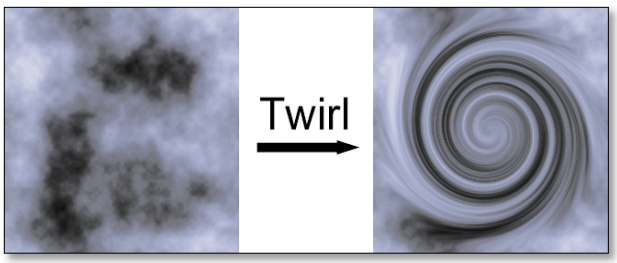
Farboperationen

Mit den Farboperationen können Sie Ihren Texturen den letzen Schliff verpassen. Dabei verändern Sie die Helligkeit, den Kontrast oder den Farbton.

Am einfachsten ändern Sie die Farben in Ihrer Textur, indem Sie die Farbkanäle invertieren. Dabei erhalten Sie interessante Farbkombinationen. Sie müssen lediglich für jeden Pixel die Rot-, Grün- und Blau-Werte wie folgt ändern:

```
layer[0][x+y*SIZE].r =
  255 - layer[0][x+y*SIZE].r;
```

Wenn Sie Finetuning an Ihren Texturen vornehmen wollen, sollten Sie das Farbmodell wechseln. Für Farbkorrekturen eignet sich das HSV-Farbmodell. Das RGB-Modell können Sie sich als einen



DIE TWIRL-FUNKTION, wie Sie sie aus Bildbearbeitungsprogrammen kennen.

Würfel vorstellen, dessen Kanten die Achsen der drei Farbwerte darstellen.

Beim HSV-Modell wird der Farbraum durch einen Kegel aufgespannt. Dabei geben Sie die Farben mit drei Werten an:

- Mit dem Hue-Wert bestimmen Sie den Farbton. Dieser Wert ist der Drehwinkel um die Achse des Kegels.
- Der zweite Wert, *S*, steht für die Sättigung (Saturation). Im Kegel interpretieren Sie ihn als Abstand zur Achse. Kleine Werte ergeben blassere Farben bis hin zu Graustufen. Mit großen Werten erzielen Sie leuchtende Farben.
- Der verbleibende dritte Parameter, *V*, steht für Value (Helligkeit).

Wenn Sie den Hue-Wert ändern, modifizieren Sie bei einer Farbe im HSV-Modell nur den Farbton. Helligkeit und Sättigung bleiben unverändert. So können Sie aus einer blauen Textur zum Beispiel eine rote oder eine blasse blaue Textur erzeugen, ohne den umständlichen Weg über die RGB-Farbwerte zu gehen. Dazu wandeln Sie die RGB- in HSV-Werte um, verändern diese und konvertieren sie zurück. Die Routine zu dieser Konvertierung finden Sie im Sourcecode zu dieser Ausgabe.

Filterfunktionen

Texturen bearbeiten Sie vielfältig: So können Sie aus einer Fraktalplasma-Textur eine mit Holzmaserung erzeugen. Schieben Sie die Bits der Farbwerte um drei bis fünf Bits nach links. Wenn Sie diese Bits wieder unten einmaskieren, haben Sie eine Holzmaserung:

```
f = layer[ 0 ][ x+y * SIZE ].r;
f = (f << 3) | ( (f>5) & 7 );
layer[ 1 ][ x+y * SIZE ].r =f;
```

Besonders schöne Holztexturen erhalten Sie durch Perlin-Noise-Texturen. Dabei sollte die Persistenz, also der hochfrequente Anteil (Rauschen), in Ihrer Textur nicht zu hoch ist. Wenn Sie zusätzlich eine andere Startfrequenz für die *x*- oder *y*-Achse wählen, erhalten Sie noch bessere Ergebnisse.

Eine klassische Filterfunktion besteht aus einer Matrix, die für einen Pixel angibt, wie Sie seinen Farbwert und die seiner Nachbarn gewichten müssen, um einen neuen Farbwert zu erhalten. Dies ist ein Filter für Bildglättung:

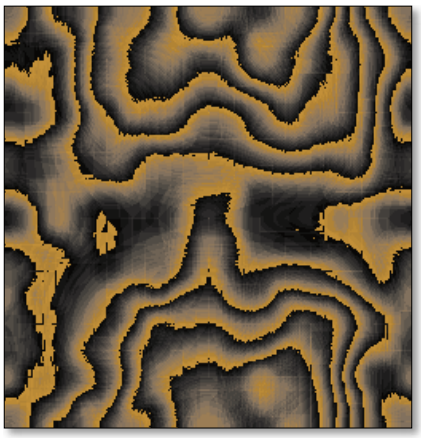
```
1 2 1
2 4 2
1 2 1
```

Den Farbwert an der aktuellen Position multiplizieren Sie mit 4, die der direkten Nachbarn mit 2 und die der diagonalen Nachbarn mit 1.

Nachdem Sie die Farbwerte aufaddiert haben, multiplizieren Sie sie mit *1/16*, um die Gesamthelligkeit des Bildes zu erhalten. Dies ist der Emboss-Filter:

```
-1 0 1
-1 0 1
-1 0 1
```

Mit dem Emboss-Filter entsteht ein Beleuchtungseffekt. Mit diesem Beleuchtungsinformation können Sie die Ausgangstextur multiplizieren. Wenden Sie den Emboss-Filter auf jeden Pixel an, und addieren Sie *128*, um das graue Bild zu erhalten. Mit diesem Wert skalieren Sie den Originalfarbwert. Achten Sie bei den Filtern darauf, dass Sie die Pixel nicht gleich mit Ihren neuen Farbwerten überschreiben. Diese Werte benötigen Sie noch zum Filtern des Nachbarpixels.



EINE HOLZTEXTUR ERZEUGEN Sie durch eine einfache Bit-Operation.

Sie müssen das Resultat immer in einen temporären Layer schreiben.

Sie können die Bildglättung auch so modifizieren, dass Sie das Bild in eine bestimmte Richtung verwischen. Wenn Sie diese Richtung von der Position des Pixels abhängig machen, erhalten Sie wieder neue interessante Effekte. ET

Die Quelltexte sowie die fertig übersetzten Routinen finden Sie auch auf unserer Website unter www.pc-magazin.de/magazin/extras.htm

Klicken Sie unter *Online Extras* im Menü *Praxis* auf das entsprechende *Download*-Feld.