AUF DER HEFT-CD

download
www.pc-magazin.de

## Demo-Programmierung unter Windows 95/NT

# Frei wie ein Vogel

**Den Grand Canyon und die Schluchten der Ozeane bereisen Sie heute schon virtuell am Bildschirm. Wie Sie Ihre eigene Cyberwelt mit Hilfe des Voxelspace schaffen, zeigt Ihnen dieser Beitrag.**

CARSTEN DACHSBACHER/
NILS PIPENBRINCK

In dieser Ausgabe von PC Underground stellen wie Ihnen die sogenannte Voxelspace-Grafik vor. Voxelspace ist ein Verfahren zur Darstellung dreidimensionaler Objekte. Im Gegensatz zur 3D-Engine aus den Ausgaben 8/98 und 9/98 basiert das Verfahren aber nicht auf Polygonen, sondern auf sogenannte Voxels (Volume-Elements), aus denen sich das 3D-Objekt zusammensetzt. Sie können sich ein Voxel einfach als Miniquader vorstellen, denn es ist gewissermaßen das dreidimensionale Pendant eines Pixels (Picture-Element).

**WEISS-BLAUER HIMMEL**, grüne Landschaft, schneebedeckte Gipfel – mit Voxelspace zur PC-Idylle

Es gibt sehr unterschiedliche Anwendungen für Voxelspace. In der Medizin wird das Verfahren zum Beispiel eingesetzt, um Daten eines Kernspintomographen zu visualisieren. Eine andere Möglichkeit kennen Sie vielleicht aus PC-Spielen, zum Beispiel dem Hubschraubersimulator *Comanche*. Hier werden Voxels verwendet, um eine Landschaft darzustellen. Und genau diese Variante stellen wir Ihnen in dieser Ausgabe vor.

Wenn Sie ein dreidimensionales Objekt mit Voxels darstellen, verwenden Sie dazu ein dreidimensionales Array von Werten. Jeder Eintrag in diesem Array hält zunächst fest, ob sich an der entsprechenden Stelle überhaupt ein Voxel befindet. Ist ein Voxel vorhanden, enthält der Eintrag darüber hinaus zusätzliche Informationen, zum Beispiel die Farbe des Voxel.

Die Berechnung eines Bilds erfolgt, indem für jeden Pixel am Bildschirm ermittelt wird, welcher Voxel des Objekts dem Betrachter am nächsten ist. Leider führt dieses Verfahren auch bei relativ einfachen Objekten sehr schnell zu einer immensen Datenmenge. Wir zeigen Ihnen, wie Sie dieses Hindernis mit einem kleinen Trick umgehen.

## Landschaft erfassen

Stellen Sie sich einen Landschaftsausschnitt aus der Vogelperspektive vor. Wenn Sie ein gedachtes Gitter über den Ausschnitt legen, können Sie für jedes einzelne Feld im Gitter eine Farbe und eine Höhe der Landschaft angeben. Diese Daten reichen schon aus: Sie bilden diese Landschaft durch Voxels nach, die auf einer Ebene stehen, und zwar nur mit der Grundfläche eines Gitterfelds und der angegebenen Höhe und Farbe.

Die Daten speichern Sie als Bitmaps. Die erste Bitmap, die Heightmap, enthält für jede Position $(x,y)$ die Höhe der Landschaft bzw. die Höhe des Voxel, das sich an dieser Position befindet. Die zweite Bitmap enthält die dazugehörige Farbe. Sie generieren solche Landschaftsdaten entweder selbst mit Photoshop oder ähnlichen Bildbearbeitungsprogrammen, oder Sie verwenden unsere Beispiele auf der Heft-CD.

Wenn Sie sich auf Landschaften mit einer Größe von 256 x 256 oder 512 x 512 Pixeln beschränken, hält sich die Datenmenge in Grenzen. Diese Auflösungen reichen für eine Demo auch vollkommen aus, da in Demos und Beispielprogrammen fast immer *seamless*-Texturen, das heißt beliebig aneinanderreihbare Texturen, verwendet werden. Die Kantenlängen der Bitmaps berechnen Sie mit $2^n$, um eine möglichst schnelle Berechnung im Programm zu gewährleisten.

## Bild berechnen

Um das endgültige Bild zu berechnen, müßten Sie für jeden Bildpunkt eine Sichtbarkeitsprüfung durchführen, um herauszufinden, welcher Voxel sichtbar ist. Eine solche Sichtbarkeitsprüfung wird mit dem sogenanntem Raycasting durchgeführt: Sie schießen einen Strahl von der Betrachterposition durch das Pixel des Bilds und durch den Voxelspace,


**GRENZENLOSE FREIHEIT:** durch Wüsten und Täler zur nächsten Oase

bis Sie einen Voxel getroffen haben oder der Strahl eine bestimmte maximale Suchstrecke zurückgelegt hat, bei der Sie die Berechnung abbrechen.

Bei einer Auflösung von 320 x 240 Punkten wären das 76 800 Operationen, die allesamt sehr viel Rechenzeit kosten. Dies wäre ohne sehr komplizierte Verfahren und entsprechend kleinliche Assembler-Optimierung nicht in Echtzeit möglich. Wenn Sie aber die Möglichkeiten der Betrachterposition und des -blickwinkels etwas einschränken, brauchen Sie im Prinzip nur einen einzigen Ray (Strahl) pro Bildschirmspalte auszuwerten.

Zuerst berechnen Sie, wie die Strahlen verlaufen. Nehmen Sie für den Anfang an, der Betrachter, der über diese Landschaft fliegt, blickt immer waagerecht, das heißt nicht nach unten und nicht nach oben.

Im Bild rechts sehen Sie einen Landschaftsausschnitt aus der Vogelperspektive. Darauf sind die Betrachterposition und der Kegel des sichtbaren Bereichs eingezeichnet. Der Kegel hat einen vorher festgelegten Öffnungswinkel, der dem Öffnungswinkel der virtuellen Kamera entspricht.

In diesem Kegel befinden sich in konstanten Winkelabständen die Strahlen, die Sie schießen. In Wirklichkeit sind es natürlich deutlich mehr, als in dieser Skizze eingezeichnet sind (wie bereits erwähnt, ein Strahl pro Bildschirmspalte). Diese Strahlen geben Sie durch einen Startpunkt (Ortsvektor) und einen Richtungsvektor an.

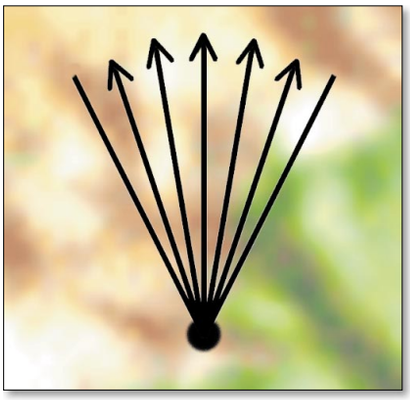
Da Sie sich im Moment noch im Zweidimensionalen befinden, hat jeder dieser Vektoren zwei Komponenten. Daß diese Komponenten in Fixpoint-Arithmetik gespeichert werden, haben Sie sicher schon geahnt. Der Startpunkt ist die Position der Betrachterkamera und somit für alle Strahlen konstant. Den Richtungsvektor erhalten Sie für einen bestimmten Blickwinkel mit Hilfe der Sinus- und der Cosinus-Funktion:

```
delta_x = cos(winkel)*FIXPOINT;
delta_y = sin(winkel)*FIXPOINT;
```

Da die Sinus- bzw. Cosinus-Funktionen der meisten Prozessoren sehr langsam sind, verwenden Sie statt dessen vorberechnete Funktionswerte in einer Tabelle. Die Schleife, die für jede Spalte des Bildschirms die Richtungsvektoren berechnet, sieht dann – mit Hinzunahme der Parameter für den Öffnungs- und Blickwinkel des Betrachters – folgendermaßen aus:

```
for ( x = 0; x < SCREEN_X; x++ )
{
   //Abweichung vom Blickwinkel
   //für diese Spalte:
   winkel = (OEFFNUNGSWINKEL *
      (SCREEN_X - x * 2)) /
➥SCREEN_X;
   // Werte aus den Tabellen
   //auslesen:
   delta_x = COS
➥( blickwinkel + winkel );
   delta_y = SIN
➥( blickwinkel + winkel );
   //an dieser Stelle wird die
   //Funktion aufgerufen,die eine
   //Bildschirmspalte zeichnet
}
```

Damit haben Sie die Richtung der Strahlen im Zweidimensionalen bestimmt



**IM RAYCASTING-VERFAHREN** überprüfen Sie, ob ein Voxel sichtbar oder verdeckt ist.

und können somit die ersten Bilder berechnen.

## Raycasting einsetzen

Das nächste Problem, das Sie lösen, ist die Strahlenverfolgung. Dabei stellen Sie fest, welcher Voxel von einem dieser Strahlen getroffen wird. An dieser Stelle haben Sie bereits das *delta_x* und das *delta_y* für den Strahl bestimmt und wählen noch eine (vorläufig) feste Flughöhe *z*.

Außerdem kennen Sie die Startposition *(x/y)* des Strahls. Sie addieren jetzt *delta_x* auf *x* und *delta_y* auf *y* und prüfen jeweils, ob der Voxel an der Position *(x/y)* höher als oder gleich hoch wie die Flughöhe ist. Haben Sie einen solchen Voxel gefunden, müssen die dahinter liegenden Voxels, um sichtbar zu sein, natürlich nicht nur höher als die Flughöhe, sondern auch höher als die bisher gefundenen Voxels sein.

Damit diese Schleife nicht ewig läuft und auch um den Fall abzufangen, daß ein Strahl nie einen Voxel trifft, verwenden Sie zusätzlich einen Zähler, den Sie nach jedem Schleifendurchlauf inkrementieren. Wenn dieser Zähler einen bestimmten, vorher festgelegten Wert überschreitet, brechen Sie die Schleife ab. Dieser Wert bestimmt also auch die Entfernung, bis zu der Voxels sichtbar sind. Prinzipiell sieht das Raycasting folgendermaßen aus:

```
entf = 0;
  z = 100;   //Flughöhe
  while ( entf++ < MAX_ENTF )
    {
    x += delta_x;
      y += delta_y;
        if ( hoehe[x,y] >= z )
        {
     // Voxel getroffen!
             z = hoehe[x,y];
        }

    }
```

Mit dieser Routine berechnen Sie natürlich keine zufriedenstellenden Bilder, die einen Eindruck einer perspektivischen Darstellung der Landschaft vermitteln. An dieser Stelle kommt unser angekündigter Trick ins Spiel. Das Auf- und Abblicken, das heißt die Neigung der Betrachterkamera, drücken Sie durch ein *delta_z* aus, das Sie nach jedem Raycasting-Schritt zur Flughöhe addieren. Außerdem erhöhen Sie den *entf*-Wert nicht um 1, sondern um eine Variable *ph*. Zusätzlich deklarieren Sie eine Konstante *VSCALE*, mit der Sie die vertikale Skalierung der Landschaft festlegen.

Wenn Sie in der Schleife einen Voxel erreichen, dessen Höhe größer als der momentane *z*-Wert ist, dann erhöhen Sie *delta_z* um *VSCALE* und *z* um *ph*, solange dies der Fall ist. Bei der Berechnung einer Spalte des Bilds starten Sie bei dieser Routine in der untersten Zeile.

Die Pixel zeichnen Sie ebenfalls in dieser Schleife, indem Sie den aktuellen Pixel auf die Farbe des getroffenen Voxels setzen und die nachfolgende Pixel-Koordinate auf den Pixel darüber setzen. Wenn Sie die oberste Zeile des Bilds überschritten haben, brechen Sie die Berechnung dieser Bildschirmspalte ab. Die fertige Routine zum Zeichnen sieht dann folgendermaßen aus:

```
while (ph < MAX_ENTF*VSCALE)
{
  y += dy;
  x += dx;
  z += dz;
  ph += VSCALE;

  h = hoehe[x,y];

  // Schnittpunkt ?
  if (h > z)
  {
    c = farbe[x,y];
    do
    {
      // Steigung erhöhen
      dz += VSCALE;
      screen[pixel] = c;
      z += ph;
```

```
        pixel -= SCREEN_X;
        if (pixel < 0) return;
      } while ( h > z );

    }
  }
```


**DIE VERTEILUNG DER GRAUSTUFEN** für eine Wasseroberfläche bearbeiten Sie mit einer Graduierungskurve.

Allerdings ist diese Methode mathematisch nicht ganz korrekt. Sie liefert aber für Neigungen, die nicht zu groß sind (ca. bis +/- 45° nach oben und unten), akzeptable Ergebnisse. Und Sie wissen ja, der Zweck heiligt bekanntlich die Mittel.

## Hintergrund berechnen

Nun sind Sie so weit, daß Sie eine Voxel-Landschaft darstellen können. Aber was wäre eine idyllische Landschaft ohne entsprechenden Himmel?

Für den Himmel benutzen Sie eine Textur, die Sie entsprechend der Kamerabewegung am Bildschirm verschieben. Die Textur sollte natürlich *seamless*, das heißt beliebig aneinanderreihbar sein, um sie mehrmals nebeneinander zeichnen zu können.

Die Verschiebung entlang der x-Achse berechnen Sie aus dem Blickwinkel der Kamera. Die Verschiebung nach oben bzw. unten erhalten Sie aus dem Neigungswert der Kamera.

Die Himmel-Textur, die Sie in den Beispielprogrammen zu diesem Artikel auf der Heft-CD finden, wurde übrigens mit Hilfe von prozeduralen Texturen mit dem Shareware-Raytracer POVRay berechnet. Sie erzeugen solche Texturen auch mit der Routine für fraktale Plasmawolken und den Verzerrungen aus PC Underground der Ausgabe 10/98.

## Nebel einbauen

Ein weiteres hübsches Feature, das Sie Ihrem Voxel-Programm hinzufügen können, ist das sogenannte Fogging (Nebel). Hierbei werden die Farben der Voxels je nach Entfernung vom Betrachter mit weißer Farbe gemischt.

So geben Sie der Landschaftsdarstellung ein weitaus realeres Aussehen. Außerdem vermeiden Sie einen störenden Effekt in der Darstellung: Bei der Berechnung des Raycasting müssen Sie die Suchschleife nach einer gewissen Strecke abbrechen, entweder weil der Strahl keinen Voxel trifft oder weil Sie einfach nicht genug Rechenzeit zur Verfügung haben, den Strahl sehr lange Zeit zu verfolgen.

Durch das Abbrechen des Raycasting kann es passieren, daß hohe Berge plötzlich in einiger Entfernung des Betrachters aus dem Boden herauszuwachsen scheinen. Sie programmieren das Fogging dann so, daß die Voxels, die die maximale Entfernung vom Betrachter haben, beinahe weiß sind. Dann tauchen die Berge aus dem Nebel heraus auf, statt im Hintergrund in die Höhe zu sprießen.

Sie programmieren das Fogging, indem Sie der Schleife für das Raycasting eine Variable hinzufügen, die die Anzahl der Durchläufe enthält. Zudem legen Sie eine Shading-Tabelle an, die für jede Entfernung und jede Farbe der Landschafts-Bitmap die Mischfarbe mit Weiß enthält. Diese Tabelle berechnen Sie zum Beispiel so wie im Listing-Teil dieses Artikels.

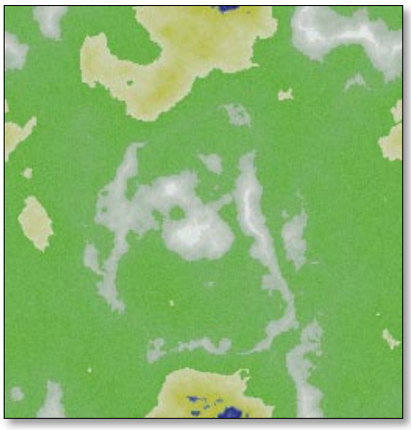
Übrigens: Ohne großen Aufwand implementieren Sie auch eine Drehung der Kamera um die eigene Achse. Hierzu sollten Sie die Ausmaße des berechneten Voxel-Bilds auf 256 x 256 ändern und dieses Bild als Textur für den Rotozoomer aus Ausgabe 11/98 verwenden.

## Landschaften generieren

Neue Landschaften zu generieren erfordert ein bißchen Übung. Wir zeigen Ihnen deshalb, wie eine unserer Beispiellandschaften erzeugt wurde.

Als Software haben wir das bekannte Bildbearbeitungsprogramm Photoshop von Adobe benutzt. Photoshop ist durch die vielen Filterfunktionen perfekt für Ihre Aufgabe gerüstet. Da fast alle neueren Bildbearbeitungsprogramme die entsprechenden Fähigkeiten besitzen, können Sie dieses Beispiel auch mit anderen Programmen nachvollziehen.

Zunächst erzeugen Sie eine Graustufen-Bitmap mit den Maßen 256 x 256 Pixel. Diese Bitmap wird Ihre Höhen-Bitmap. Um eine gute Landschaftsstruktur zu bekommen, wählen Sie als Zeichenfarbe Weiß und als Hintergrundfarbe Schwarz.



**OHNE LICHTEFFEKTE** sieht die Landschafts-Textur noch nicht sehr aufregend aus.

Über das Menü *Filter/Rendering Filter/Wolken* lassen Sie Photoshop die Grundstruktur Ihrer Landschaft berechnen. Rufen Sie zudem mehrfach den Filter *Differenz Wolken* aus dem gleichen Menü auf. Damit bekommen Sie zusätzliche feine Strukturen in Ihrer Landschaft. Speichern Sie dieses Bild, denn Sie brauchen es später noch, um der Landschaft den letzen Schliff zu geben.

Diese Bitmap sieht noch nicht sehr realistisch aus. Es fehlen Wasserflächen, Niederungen und Hochplateaus, die Sie im nächsten Schritt hinzufügen.

Als erstes ist es nötig, den ganzen Wertebereich der Graustufen-Bitmap auszunutzen, um einen geeigneten Wertebereich für die Voxel-Berechnung zu erhalten. Das erledigen Sie mit der Funktion *Auto-Tonwertkorrektur*, die Sie im Menü *Bild/Einstellungen* finden.

Jetzt folgt die Generierung der Hochplateaus und Wasserflächen: Über die Graduierungskurve im gleichen Menü bearbeiten Sie die Verteilung der Graustufen. Wählen Sie als Werkzeug den Stift, und zeichnen Sie Bereiche gleicher Höhe durch horizontale Linien ein.

Wenn Sie mit der Höhen-Bitmap fertig sind, ändern Sie über die Helligkeit-Kontrast-Regelung noch das Höhenniveau, falls Ihre Berge zu hoch sind. Sind Ihre Berge zu spitz und zackig, dann spielen Sie am besten so lange mit den Weichzeichnungsfiltern, bis Sie zufrieden sind.

Laden Sie nun die Bitmap, die Sie vorher gespeichert haben. Diese Grafik verwandeln Sie durch einfaches Einfärben und Nachbearbeiten in eine sehr realistische Landschafts-Textur.

Mit Hilfe des Paletteneditors im Menü *Bild/Modus/Farbtabelle* färben Sie Ihre

Textur ein. Von Hochgebirgen bis hin zu Fantasielandschaften ist alles möglich. Wir haben uns in den Beispielen eher konservativ an der Natur orientiert: Die Wasserflächen sind blau, die Berge grau mit weißen, schneebedeckten Gipfeln und dazwischen ein wenig Braun- und Grüntöne für Sand und Grasflächen.

Jetzt zum Feinschliff: Wandeln Sie Ihr Bild mit *Bild/Modus/RGB-Farbe* in das RGB-Format um. Fügen Sie dem Bild etwas Rauschen hinzu, um der Landschaft mehr Realismus zu verleihen. Was jetzt noch fehlt, sind Lichteffekte, die Sonneneinstrahlung und Schattenwurf der Berge simulieren. Dank der Rendering-Filter von Photoshop ist dies denkbar einfach.

Laden Sie Ihre Höhen-Bitmap, und stellen Sie sicher, daß die Bitmap als Graustufen-Bitmap vorliegt. Photoshop hat nämlich die unangenehme Eigenschaft, *bmp*-Dateien immer als palettierte Bilder zu laden, auch wenn gar keine Farben benutzt wurden.

Die Ebene Ihrer Höhen-Bitmap wird nun dupliziert und in die Textur-Bitmap eingefügt. Öffnen Sie hierzu die Ebenen-Toolbox, und wählen Sie die Ebene aus. Die Funktion zum Duplizieren rufen Sie über die rechte Maustaste ab. Sollten Sie die Ebenen-Toolbox nicht finden, ist diese vermutlich ausgeblendet. Sie finden sie im Menü *Fenster/Ebenen einblenden*. Die duplizierte Ebene wird Ihnen als Relief-Kanal für die Beleuchtung dienen. Rufen Sie hierzu den Filter *Filter/Rendering/Beleuchtungseffekte* auf. Als Relief-Kanal wählen Sie den neuen Kanal der Höhen-Bitmap. Zum Start empfehlen wir eine diffuse Lichtquelle.

Nun können Sie mit den Reglern experimentieren, bis Sie ein ansprechendes Resultat erzielen. Durch die zahlreichen Parameter dieses Filters ist es nicht leicht, gleich am Anfang die optimalen Ergebnisse zu erzielen. Aber mit ein bißchen Übung bekommen Sie schnell den richtigen Dreh.

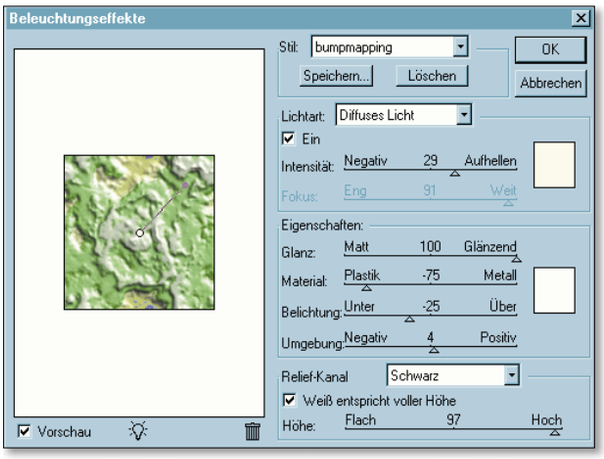
Übrigens: Vertrauen Sie nicht blind dem Vorschaufenster. In der Regel fallen die Beleuchtungseffekte immer etwas extremer aus. Wenn Sie fertig sind, vergessen Sie nicht, Ihre Textur wieder auf 256 Farben zu reduzieren, um Sie im Voxel-Programm verwenden zu können.

## Keyframing optimieren

Im Beitrag zur 3D-Engine in Ausgabe 9/98 haben wir Ihnen gezeigt, wie Sie die Kamerapositionen und alle Animationsdaten für eine angenommene Bildfrequenz speichern und wieder abspielen. Dieses Verfahren glänzt durch seine Einfachheit, hat aber einige entscheidende Nachteile: Die Animation ist auf eine bestimmte Bildfrequenz abgestimmt und wird deswegen auf unterschiedlichen PCs auch unterschiedlich schnell abgespielt. Es ist außerdem sehr schwierig, die Kamerafahrt nachträglich zu ändern. Aus diesem Grund zeigen wir Ihnen hier eine verbesserte Variante des Keyframings.

Sie legen dabei nicht mehr extra für jedes Bild eine Kameraposition ab, sondern definieren diese Position nur noch für einige Zeitpunkte. Mit ein wenig Mathematik berechnen Sie die Zwischenstufen.

Das Problem, eine weiche Kurve zwischen einigen wenigen Stützwerten zu berechnen, ist nicht neu und wurde von vielen Mathematikern behandelt. Sehr




**AUF DIE BELEUCHTUNG** kommt es an. Lichteffekte verstärken den realen Eindruck.

nützlich für die Computergrafik hat sich dabei die Familie der kubischen Splines erwiesen. Diese Splines berechnen den Kurvenverlauf zwischen zwei Stützwerten, indem eine Funktion dritten Grades aus den gegebenen Werten berechnet wird. Diese Funktion benutzen Sie, um Werte zwischen den Keyframes zu berechnen.

Es gibt eine ganze Menge verschiedener kubischer Splines. Vermutlich haben Sie sogar schon mit kubischen Splines gearbeitet, ohne es zu wissen: Die Rundungen und Bögen von TrueType-Schriften und auch die Beziér-Kurven, die von vielen Grafikprogrammen benutzt werden, sind zum Beispiel solche Kurven.

## Hermite-Kurven berechnen

Hermite-Kurven bestehen aus zwei Punkten (Im Bild *P1* und *P2* genannt) sowie aus zwei Vektoren (die Tangenten), die Einfluß auf die Richtung der Kurve haben, sobald die Kurve den Startpunkt verläßt (*T1*) bzw. den Endpunkt erreicht (*T2*). Diese vier Werte geben Ihnen eine sehr gute Kontrolle über den Verlauf der Kurve zwischen den zwei Punkten.

Um einen Punkt dieser Kurve zu berechnen, verwenden Sie die Formeln des Mathematikers Hermite. Diese Funktionen werden Hermite Basis Funktionen genannt und lauten:

```
h1(t)  =  2 * t^3 - 3 * t^2 + 1
h2(t)  = -2 * t^3 + 3 * t^2
h3(t)  =  1 * t^3 - 2 * t^2 + t
h4(t)  =  1 * t^3 - 1 * t^2
```

Der Wert *t* bestimmt den Punkt der Kurve, der berechnet werden soll. Der Wert ist *0* für den Fall, daß Sie am Anfang der Kurve sind, und *1*, wenn Sie den Endpunkt berechnen. Alle Werte zwischen *0* und *1* berechnen die erwünschten Zwischenwerte.

Um einen Wert auf dieser Kurve zu berechnen, multiplizieren Sie nur die Punkte und Tangenten mit den vier Funktionen.

Für die Berechnung einer zweidimensionale Kurve gehen Sie so vor: Berechnen Sie zuerst für Ihren *t*-Wert die Funktionen *h1, h2, h3* und *h4*. Wenden Sie anschließend folgende Formeln an, um die Koordinaten zu erhalten:

```
x = (h1 * P1.x) + (h2 * P2.x) +
➥(h3 * T1.x) + (h4 * T2.x);
y = (h1 * P1.y) + (h2 * P2.y) +
➥(h3 * T1.y) + (h4 * T2.y);
```

## Mehrere Kurvenabschnitte

Die Tangenten, die von den Hermite-Kurven benötigt werden, sind nicht gerade anschaulich. Sie sollen sich jedoch das Erstellen der Keyframes so einfach wie möglich machen. Es ist auch gar nicht in Ihrem Interesse, daß die Kurven extrem flexibel sind. Ein Knick an den ▶

Stützwerten zum Beispiel würde bei einem Kameraflug nur störend wirken.

Wenn Sie mehr als einen Kurvenabschnitt haben, zum Beispiel bei einer Abfolge von Kamerapositionen, berechnen Sie Tangenten, die diese Bedingungen erfüllen. Die Tangentenformel für die Cardinal Splines lautet:

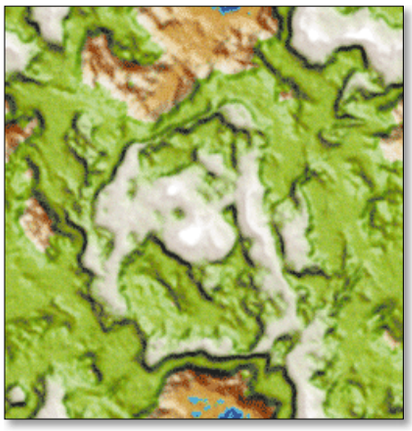
```
T[i].x = a *
➥( P[i+1].x - P[i-1].x );
T[i].y = a *
➥( P[i+1].y - P[i-1].y );
```

Über die Konstante *a* stellen Sie ein, wie weich der Übergang zwischen den Kurven sein soll. Sinnvolle Werte für *a* liegen zwischen *0* und *1*. Cardinal Splines mit dem *a*-Wert *0.5* haben übrigens einen eigenen Namen: Sie werden Catmull-Rom Splines genannt. Und genau diese Splines sind optimal für den Zweck, den Sie hier verfolgen.

Die Tangenten-Formel hat einen kleinen Nachteil: Sie benötigen zum Berechnen jeweils den vorherigen und den nächsten Spline-Punkt. Für den ersten und letzten Punkt funktioniert sie nicht. In unserer Implementation wird die Tangente für den ersten und letzten Stützpunkt jeweils auf *0* gesetzt.

## C++-Klasse einsetzen

Um Ihnen die Arbeit mit den Splines zu erleichtern, finden Sie auf der Heft-CD in dieser Ausgabe eine leicht zu benutzende C++-Klasse. Sie ist in der Lage, die ganze Familie von Cardinal Splines zu berechnen – und zwar für so viele Stützwerte gleichzeitig, wie Sie wollen. Die genaue Implementation finden Sie in den beiden Quelltext-Dateien *Spline.cpp* und *Spline.h*. Einen kleinen Beispiel-Code, der die Verwendung der Klasse demonstriert, zeigt Listing 1. Im kompletten Quelltext zum Artikel befindet sich außerdem noch ein zusätzliches Beispielprogramm.



**VIRTUAL REALITY:** Erst durch das Zusammenspiel von Licht und Schatten treten die Landschaftskonturen deutlich hervor.

## Splines einbauen

Das Einbauen der Splines in den Voxelspace ist einfach. Zunächst einmal werden für den Voxelspace Key-Informationen erzeugt, indem Sie eine Handsteuerung – ähnlich der eines Flugsimulators – in das Programm einbauen. Dann speichern Sie in regelmäßigen Abständen (zum Beispiel jede Sekunde) die aktuelle Kameraposition in eine Datei. Wenn Sie sich für diese Variante entscheiden, führen Sie einige kleine Änderungen im Demosystem durch.

Ein Tastendruck darf nicht mehr dazu führen, daß das Programm beendet wird. Dafür kommentieren Sie die Zeile

```
„case: WM_KEYDOWN“
```

in der Datei *demosys.cpp* aus (Funktion: *WindowProc*). Jetzt fragen Sie die Tasten ab. Sie verwenden dafür die Windows-Funktion *GetAsyncKeyState*. Informationen zu dieser Funktion finden Sie in der Hilfe zur Win32-API.

Die Auswertung der Keyframing-Informationen, die Sie in den Dateien *keys1.txt* und *keys2.txt* finden, geschieht mit der *CardinalSpline*-Klasse. Bei der Initialisierung des Effekts wird ein Spline erzeugt.

Die Datei wird zeilenweise ausgelesen und die Informationen als Keyframes der Spline-Klasse übergeben. Am Anfang dieser Dateien geben Sie noch die Landschafts- und Höhen-Bitmap sowie die Anzahl der Spline-Stützwerte und die Abspielgeschwindigkeit der Spline-Interpolation an.

In der Funktion *Lenkung* werden für den aktuellen Zeitpunkt die Kameradaten interpoliert (Klassenmethode *CardinalSpline::Get*) und an die Voxelspace-Routine übergeben. JR

Die komplette Demo, alle Quelltexte und die C++-Klasse zur Berechnung der Cardinal Splines finden Sie auf unserer Heft-CD und im Internet-Angebot des PC Magazin unter

```
www.pc-magazin.de/magazin
➥extras.htm
```

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

**1 Fog**

```
// Entfernung 0 bis 31
    // cmap ist das Landschaftsfarbbitmap
    for ( int j = 0; j < 32;  j++ )
      for ( int i = 0; i < 256; i++ )
      {
            entf = j*j/32;
            fogtable[i][j] =
              ColorCode( (cmap.cColors[i*4+0]*(32-entf)+
entf*256)/32,
                         (cmap.cColors[i*4+1]*(32-entf)+
entf*256)/32,
                         (cmap.cColors[i*4+2]*(32-entf)+
entf*256)/32);
      }
```

**Mit diesem Listing erzeugen Sie die *Fog*-Tabelle**

**2 Spline**

```
// Tabelle mit X und Y Koordinaten eines 2D Splines

float splinedata[] = { 20,   70,
                       50,   20,
                       230,  20,
                       300, 160};

// Spline Erstellen:
//   2    Dimension (zweidimensional, also X und Y)
//   0.5  ist die A-Konstante
//   4    ist die Anzahl der Stützwerte

CardinalSpline spline (2, 0.5, 4);

// 4 Stützwerte setzen
//    erster Parameter ist der Zeitpunkt bzw Framenummer des
Stützwertes
//    zweiter Parameter ist ein Pointer auf die Stützwert-
Koordinaten

spline.set (0,  &splinedata[0]);
spline.set (10, &splinedata[2]);
spline.set (20, &splinedata[4]);
spline.set (30, &splinedata[6]);

// Spline initialisiern, Tangenten berechnen.
spline.prepare();

// Kurve stückweise Berechnen:
for ( int i=0; i<30; i++)
{
   float data[2];
   spline.get (i, data);
   float x= data[0];
   float y= data[1];
   // Hier kann man die Werte benutzen und
   // zum Beispiel Punkte setzen.
   putpixel (x,y);
}
```

**Mit diesem Listing berechnen Sie Splines**