AUF CD 1

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

Animierte Modelle mit Milkshape 3D

Spiel-Szene

Um Spielfiguren zu entwickeln, animieren Sie Ihre Milkshape-3D-Polygonmodelle mit realistischen Knochensystemen.

CARSTEN DACHSBACHER

Den 3D-Modellen, die Sie in der letzten Ausgabe von PC Underground (10/01, S. 238) geschaffen haben, fehlt noch ein Knochengerüst. In diesem Beitrag zeigen wir Ihnen, wie Sie mit Milkshape Ihre 3D-Modelle mit Hilfe von Knochensystemen (Bone Systems) realistisch animieren und in eigene Spiele einbauen.

Beim Modellieren eines humanoiden Lebewesens bilden Sie ein Knochenskelett nach. Dieses enthält nicht so viele Knochen wie das menschliche Skelett. Im Egoshooter *Unreal Tournament* finden Sie ein Skelett, das etwa 40 Knochen enthält.

Bauen Sie das 3D-Modell (Dreiecke) auf. Die Form des Modells sollte sich den Gegebenheiten des möglichst genau gebildeten Skeletts anpassen. Gehen Sie vor, wie in der letzten Ausgabe beschrieben: Texturieren Sie das 3D-Modell.

Jedem Vertex des Modells weisen Sie einen Knochen zu, mit dem er assoziiert ist. Die Bewegung dieses Knochens wirkt sich auf den Vertex und damit assoziierte Vertizes aus. Sie müssen bei der Animation des 3D-Modells nur das Skelett animieren, das Modell passt sich den Bewegungen an. Sie können so Animationen, die Sie einmal auf Basis des Skeletts angelegt haben, für mehrere 3D-Modelle verwenden.

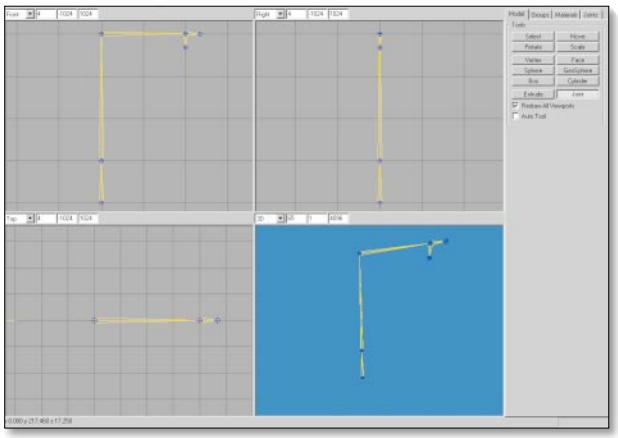
Das Skelett ist hierarchisch aufgebaut: Knochen können anderen Knochen untergeordnet sein. So bewegen sich der Unterarm und alle daran befestigten Knochen mit, wenn das Modell den Oberarm bewegt. Diese Animationstechnik heißt *Skeletal Animation*. Auf Grund der Rechenleistung heutiger Hardware hat sie in nahezu allen modernen Spielen Einzug gehalten. Ihre Vorteile: Sie können Animationen wiederverwenden, verschiedene Animation überblenden und überlagern.

So animieren Sie zwei Abläufe unterschiedlich voneinander: In der ersten Animation lassen Sie eine Spielfigur laufen. Die zweite beschreibt eine Drehung des Kopfes zur Seite. Die Spielfigur kann gehend den Kopf zur Seite drehen.

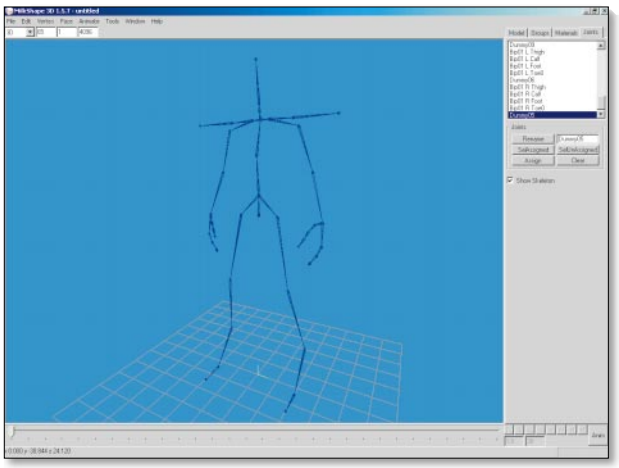
Für diese Animationstechnik gibt es Hardware, die über 30000 Mark kostet. Diese Motioncapturing-Systeme bauen grob ein Skelett und versehen eine Person, von der Sie Bewegungen aufzeichnen wollen, mit Sensoren oder Marken. Anschließend zeichnet der Computer anhand der Sensoren oder Kamerabilder die Bewegungsabläufe auf und speichert diese für Ihr Skelett. So erreichen Studios realistische Bewegungen in Perfektion.

Ein Roboterarm entsteht

Anhand eines Roboterarms erkennen Sie die hierarchische Animation, und Sie überschauen die Anzahl der Knochen (Bones) – in diesem Fall der Armteile. Der Roboterarm soll komplett drehbar sein, zwei Armsegmente und eine Greifhand besitzen.



UNSER ROBOTERARM besteht aus fünf Bones und sechs Joints.

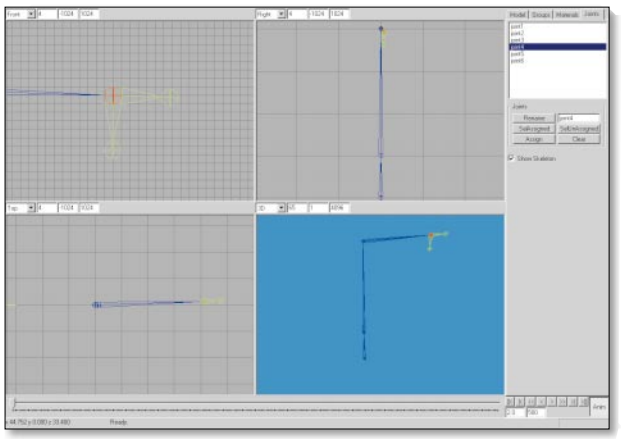


DAS KNOCHENSKELETT des Ego-Shooters *Unreal Tournament*

Sie benötigen fünf Knochen. Diese sind in Milkshape 3D durch Joints (Verbindungsstücke/Gelenke) definiert. Ein Bone beginnt und endet in einem Joint.

Sie legen Joints an, indem Sie im Model-Tab von Milkshape 3D das *Joint-Tool* wählen, und die Joints in den *Modelling*-Fenstern per Mausklick positionieren. Milkshape 3D erzeugt automatisch einen Bone zwischen einem neu eingefügten und dem letzten Joint. Wenn Sie an einem Joint einen zweiten Bone befestigen wollen, dann wählen Sie den entsprechenden Joint im *Joints-Tab*. Er wird dann rot dargestellt. Die schon daran befindlichen Bones sind grün zu erkennen.

Im nächsten Schritt modellieren Sie den Roboterarm. Das Beispiel begnügt sich mit einfachen Zylindern und einer Kugel, die grob die Form gestalten. Sie haben nun die Möglichkeit das Aussehen des Roboters nach Ihren Vorstellungen zu verfeinern.



AN EINEM JOINT können Sie mehrere Bones befestigen.

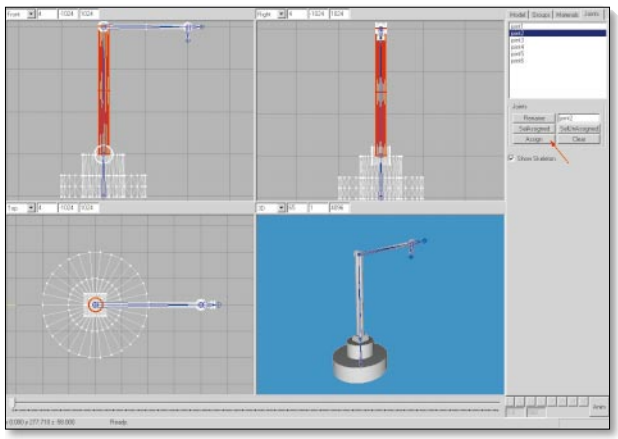
Weisen Sie den Vertizes der Zylinder jeweils einen Bone zu. Dazu markieren Sie die Vertizes mit dem *Select-Tool* aus dem *Model-Tab*. Wählen Sie im *Joints-Tab* den *Joint* und damit alle daran befestigten (untergeordneten) *Bones* aus, und klicken Sie auf den *Assign*-Button. Beachten Sie folgende Punkte:

•Beginnen Sie in der Hierarchie ganz oben, und arbeiten Sie sich nach unten durch. Das vereinfacht die Zuweisungen. Beim Roboterarm fällt die Entscheidung, welche Vertizes welchen Bones zugeordnet werden sollen, leicht, weil Sie die 3D-Teile des Arms der Reihe nach anlegen. Bei einem Low-Polygon-Modell eines Menschen sind eventuell mehrere Versuche nötig, um das beste Ergebnis zu erzielen. Das 3D-Modell soll möglichst genau entlang dem Skelett verlaufen. Weisen Sie alle Vertizes zu. Um herauszufinden, welche Sie noch nicht bearbeitet haben, deselektieren Sie zunächst alle eventuell markierten Vertizes. Wenn Sie auf den *Select unassigned* (*SelUnAssigned*) Button klicken, markiert Milkshape 3D alle diese Vertices für Sie. Analog bewirkt der *Select Assigned* (*SelAssigned*) Button, dass alle Vertices, die mit dem gerade gewählten Bone assoziiert sind, sichtbar werden.

Skelett animieren

Animieren Sie das Skelett. Aktivieren Sie im Menü die Funktion *Show Keyframer*. Im unteren Teil des Programmfensters erscheinen eine Zeitachse und diverse Buttons. Mit dem *Anim*-Button aktivieren Sie die Animationsmodellierung.

Beim Keyframing legen Sie die Stellung der Gelenke, also die Position und Drehung der Joints, nur für bestimmte Zeitpunkte fest. Die Positionen und Winkel zwischen den angegebenen Zeitpunkten bestimmt das Animationsprogramm durch Interpolation.



WEISEN SIE allen Vertizes einen Bone zu.

Mit dem Schieberegler wählen Sie einen Zeitpunkt (ein *Frame*), für den Sie einen Animationsschritt festlegen wollen. Mit den Optionen *Select*, *Move* und *Rotate* aus dem *Model-Tab* können Sie den Roboterarm in jede mögliche Position bringen. Wenn Ihnen das Resultat zusagt, speichern Sie das Frame mit dem Menüpunkt *Animate/Set Keyframe* oder mit der Tastenkombination *[Strg-K]*.

Wenn Sie alle Keyframes festgelegt haben, können Sie die fertige Animation mit dem *Play*-Button oder durch Bewegung des Schiebereglers auf der Zeitachse ansehen und überprüfen. Wenn Sie Ihre Arbeit als *MS3D*-Datei speichern, werden alle *Bone*- und *Keyframing*-Informationen automatisch mit abgelegt.

Animationen in eigenen Programmen

Das Beispielprogramm der letzten Ausgabe las die Vertex-, Polygon- und Material-Informationen von *MS3D*-Dateien aus. Diese Informationen erschienen in den Dateien immer in der gleichen Reihenfolge. Ebenso verhält es sich mit Animationsdaten, die den Materialdaten folgen. Zuerst finden Sie die *Frames per Second*, das in Milkshape 3D ausgewählte Frame und die Gesamtzahl der Frames in der *MS3D*-Datei:

```
float  fAnimationFPS;
float  fCurrentTime;
int    iTotalFrames;
```

Als nächstes finden Sie den 16-Bit-Wert *unsigned word*, der die Anzahl der Joints im Skelett angibt:

```
word nJoints;
```

Um die Animationsdaten einzulesen, definieren Sie folgende Strukturen:

```
typedef struct
{
  float time;
  float rotation[ 3 ];
} MS3D_KEYFRAME_ROT;

typedef struct
{
  float time;
  float position[ 3 ];
} MS3D_KEYFRAME_POS;

typedef struct
{
  byte  flags;
  char  name[ 32 ];
  char  parentName[ 32 ];
  float rotation[ 3 ];
  float position[ 3 ];

  word  nKeyFramesRot;
  word  nKeyFramesPos;

  MS3D_KEYFRAME_ROT *keyFramesRot;
  MS3D_KEYFRAME_POS *keyFramesPos;
} MS3D_JOINT;
```

SPIEL-PROJEKTE UND LINKS

Spiel	Link	Genre
Die Germanen	www.bigbytesoftware.de	3D-Strategie
Operation Red Falcon	www.operationredfalcon.com	Halflife Mod
SCS	www.nirwana-games-development.4d2.de	Weltraum Shooter
Takatis	www.poke53280.de.vu	2D Shoot'Em'Up
Tank Hunter	www.stefanzerbst.de	Ego Shooter
VVL Extreme	www.cwr-spiele.de	Ego Shooter
Weststorm	www.banshee-interactive.de	3D-Shooter
DUSMANIA2001	www.xenoage.de/dusmania2001/	Berichte zur DUS
Hobby-Spieleentwickler	www.untergrund-spiele.4players.de	Portal

Die Strukturen *MS3D_KEYFRAME_ROT* und *MS3D_KEYFRAME_POS* enthalten die Rotation oder Position eines Joints für ein Keyframe. Diese Werte werden in einem Array der *MS3D_JOINT*-Struktur gespeichert. Diese Struktur enthält außer Flags deren Namen und den Namen des in der Hierarchie höheren Joints (*parentName*). Außerdem sind die initiale Position und Rotation gespeichert. Die Daten lesen Sie in der Reihenfolge, die Sie in der *MS3D_JOINT*-Struktur finden, aus der *MS3D*-Datei aus.

Erste Positionierung der Bones

Nachdem Sie alle Daten aus der *MS3D*-Datei gelesen haben, müssen Sie die Vertizes entsprechend der initialen Position und Rotation der zugehörigen Bones transformieren. Auf Grund der zusammengesetzten Rotation und Translation (Verschiebung) und des hierarchischen Aufbaus der Animation setzen Sie Matrizen ein. Speichern Sie für jeden Bone mehrere Matrizen. Diese umfassen die Zwischenergebnisse, die relative Transformation eines Bones sowie die absolute:

```
typedef float MATRIX[4][4];

typedef struct
{
  MATRIX mRelative;
  MATRIX mAbsolute;
  MATRIX mRelativeFinal;
  MATRIX mFinal;
} BONE;

pBone = new BONE[ nJoints ];
```

Die initiale Positionierung berechnen Sie für jeden Bone. Die folgende Routine geht davon aus, dass in der Bone-Liste die Hierarchie absteigend ist. Das heißt, entweder ist ein Joint Beginn eines Knochens, oder das in der Hierarchie nächsthöhere Joint, mit dem es einen Bone bildet, ist vorher in der Liste enthalten.

Für jeden Joint berechnen Sie die Rotationsmatrix (*createRotationMatrix(...)*) und fügen die Translation hinzu, die sich bei einer 4x4-Matrix in der rechten Spalte befindet. Die Matrixroutinen finden Sie in der Datei *matrix.h* und in fast jeder mathematischen Formelsammlung:

```
for ( i = 0; i < nJoints; i++ )
{
MS3D_JOINT *bone = &pJoint[ i ];
 createRotationMatrix(
  pBone[ i ].mRelative,
  bone->rotation[ 0 ] );

// Translation
pBone[i].mRelative[0][3]=
 bone->pos[ 0 ];
pBone[i].mRelative[1][3]=
 bone->pos[ 1 ];
pBone[i].mRelative[2][3]=
 bone->pos[ 2 ];
```

Jetzt suchen Sie den in der Hierarchie nächsthöheren Joint. Beginnt mit dem aktuellen Joint ein Knochenstrang, werden Sie keinen *Parent Bone* finden, womit Sie die oben berechnete Transformationsmatrix speichern. Wenn Sie einen *Parent Bone* finden, müssen Sie dessen Transformationsmatrix mit der Matrix des aktuellen Bones multiplizieren und speichern:

```
// nächsthöheren Joint suchen
  nParentBone = -1;
  for ( int j = 0; j < nJoints;
j++ )
if ( strcmp( pJoint[ j ].name,
bone->parentName ) == 0 )
    {
      nParentBone = j;
      break;
    }

  if ( nParentBone != -1 )
  {
    // Parent Bone gefunden
    pBone[ i ].mAbsolute =
      pBone[ i ].mRelative *
pBone[ nParentBone ].mAbsolute);
pBone[ i ].mFinal =
 pBone[ i ].mAbsolute;
  } else
  {
    // kein Parent Bone
pBone[ i ].mAbsolute =
 pBone[ i ].mRelative;
pBone[ i ].mFinal =
 pBone[ i ].mRelative;
  }
}
```

Transformieren Sie alle Vertizes mit der inversen Transformationsmatrix des assoziierten Bones. Es ist nicht notwendig, aufwändige mathematische Verfahren zur Matrixinversion anzuwenden, da

BERICHT VON DER DUSMANIA 2001

Am 27. und 28.07.2001 traf sich die Szene der deutschen Hobby-Spieleentwickler zum dritten Mal zur so genannten DUSmania in Freudental bei Stuttgart. *DUS* steht für *Deutsche Untergrund Spiele*. Unter den etwa 100 Gästen waren hauptsächlich Hobbyprogrammierer, -grafiker und -musiker sowie einige wenige kommerzielle Spieleentwickler.

Diskussionsrunden beschäftigten sich mit Themen wie *Neue Spiele, neue Konzepte, Distributionsmöglichkeit Internet* und *lizenzierte gegen selbst entwickelte Engines*. Am Abend fanden Projektvorstellungen verschiedener Teams statt, die ihre Spiele auf einer Großleinwand dem Publikum vorstellten. Interessant dürfte für die Hobbyentwickler die Präsenz von professionellen Teams wie Davilex oder Vulpine gewesen sein.

Vier Projekte machten deutlich, mit welchem Engagement die Hobbyentwickler ans Werk gehen. Dennoch gibt es qualitative Unterschiede zu den weltweit vermarkteten, kommerziellen Projekten:

• Das Echtzeit-Strategiespiel *Die Verbotene Welt* von Sechsta Sinn ist eine Art *Command&Conquer*-Klon im typischen Isografik-Stil. Es war das qualitativ beste Spiel auf der DUS. Screenshots und Demos können Sie von der Webseite www.sechstasinn.de laden.

• *Eisenfaust* lautet der Titel eines 2D-Shoot'em-up von Not'A'Tric. Der Projektleiter plant, die erste Episode des Spiels als Freeware im Internet unter www.notatric.de anzubieten.

• Vom Strategiespiel *Acrophy* von Lama Cru waren ein kurzer Trailer und eine angespielte Demoversion zu sehen. Das Test-Projekt finden Sie auf der Website www.lama-ware.de *DarkBasic* ist ein Basic-Compiler mit eingebauter DirectX-Unterstützung. Damit sollen Hobbyprogrammierer leichter mit 3D-Hardware umgehen und einfacher Spiele und Demos entwickeln können. Informationen zu DarkBasic und damit programmierten Spielen wie *Dark land* im Stil von Marble Madness finden Sie unter www.colorarts.de/.

IN DEN ROUNDTABLES wurden aktuelle Themen der Spieleentwickler diskutiert.

es sich beim Rotationsteil um eine *orthogonale* 3x3-Matrix handelt. Bei dieser ist die transponierte Matrix gleich der inversen. Die Verschiebung können Sie invertieren, indem Sie vor der Rotation die Verschiebung negiert anwenden:

```
for ( int j = 0;
j <nVertices; j++)
{
MS3D_VERTEX *pV = &pVertex[ j ];
  if ( pV->boneId != -1 )
  {
    MATRIX *a = &pBone
 [ pV->boneId ].mAbsolute;
    pV->vertex[0] -= a[0][3];
    pV->vertex[1] -= a[1][3];
    pV->vertex[2] -= a[2][3];

invrotate( temp, pBone
[ pV->boneId ].mAbsolute,
 pV->vertex );
    pV->vertex = temp;
  }
}
```

Animationframe berechnen

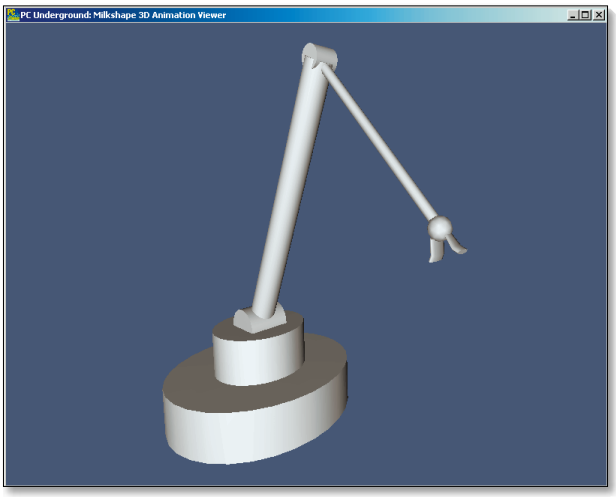
Berechnen Sie die Transformationsmatrizen für ein bestimmtes Animations-Frame. Dabei gehen Sie in etwa so vor wie bei der Initialisierung. Zunächst haben Sie ein Frame, also eine Zeit, angegeben. Behandeln Sie wieder einen Joint nach dem anderen. Für jeden Joint haben Sie die Arrays

```
  MS3D_KEYFRAME_ROT *keyFrames
Rot;
  MS3D_KEYFRAME_POS *keyFrames
Pos;
```

gespeichert, in denen die Zeitpunkte, die Rotations- oder Positionsinformationen abgelegt sind. Suchen Sie für jeden Joint jeweils die nächsten *Keyframe*-Informationen, die vor und nach dem gewünschten Frame liegt. Dadurch haben Sie ein Zeitintervall gegeben und können die Werte interpolieren, wie Sie sie an Hand der Positionsberechnung sehen:

```
MS3D_KEYFRAME_POS
  *pLastPositionKey = NULL,
  *pThisPositionKey = NULL;

for ( j = 0; j <
 nPositionKeyCount; j++ )
{
  pPositionKey =
    &bone->keyFramesPos[ j ];

if ( pPositionKey->time >=
   frame )
  {
pThisPositionKey = pPositionKey;
    break;
  }
pLastPositionKey = pPositionKey;
}

// Position interpolieren
d = pThisPositionKey->
 time - pLastPositionKey->time;
s = ( frame -
 pLastPositionKey->time ) / d;

vPos = pLastPositionKey->
 position +
( pThisPositionKey->position -
pLastPositionKey->position ) *s;
```

Mit der berechneten Rotation und Translation erzeugen Sie wieder die relative Transformationsmatrix. Wenn der gerade betrachtete Joint an einem hierarchisch höher angesiedelten Joint befestigt ist, berücksichtigen Sie wieder dessen Transformation, genau wie bei der Initialisierung. Zuletzt wenden Sie die berechneten Transformationen auf die Vertizes an: Sie erhalten genau die Animation, die Sie vorher in Milkshape 3D angelegt haben!



UNSER EIGENER Milkshape-3D-Animationsplayer

Um bei der Animation die korrekte Beleuchtung der 3D-Objekte zu gewährleisten, müssen Sie die Normalen transformieren. Wenden Sie nur die Rotation auf die Normalenvektoren an, nicht die Translation: Verwenden Sie in *matrix.h* die Funktion *rotate(...)* und nicht *transform(...)*. ET