**AUF CD**

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

## Polygon-Modelle mit Milkshape 3D

# Spiel mit Puppen

Legen Sie Ihre eigenen Low-Polygon-3D-Modelle mit Milkshape 3D an, die Sie texturieren und in eigenen Programmen verwenden. Mit Plug-ins haben Sie Dateiformate im Griff.

CARSTEN DACHSBACHER

Für Programmierer und Grafiker, die 3D-Spielfiguren erschaffen, mit Texturen versehen und animieren möchten, ist das 3D-Modelling-Programm Milkshape geeignet. Modellieren Sie zunächst eine Figur, um deren Daten anschließend in Ihre eigenen OpenGL-Programme zu importieren.

Beim Programmieren mit Milkshape 3D verwendet der 3D-Grafiker wenige Polygone: mehrere Hundert, bis zu etwa Tausend Dreiecke. Das spart Rechenzeit.

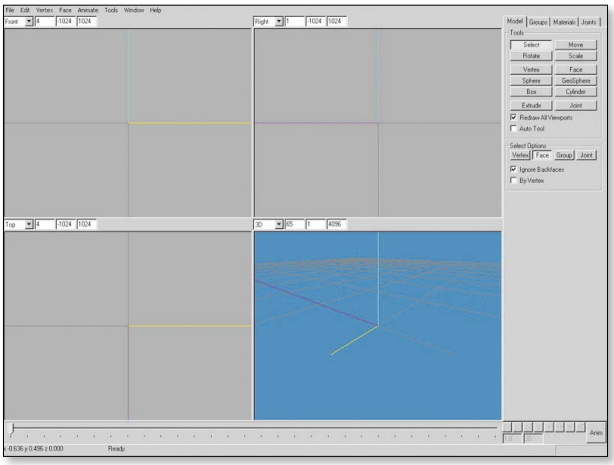
Sie können für unsere Einführung eine Testversion von Milkshape 3D unter dieser URL downloaden: www.swissquake.ch/chumbalum-soft/ms3d1x/ Die aktuelle Version ist 1.5.7.

Wenn Sie Milkshape 3D starten, sehen Sie den Hauptbildschirm. Den meisten Platz nehmen die vier Ansichten des noch nicht vorhandenen 3D-Modells ein. Voreingestellt sind die achsenparallelen Ansichten von vorne, rechts und oben. Das vierte Fenster zeigt die Szene aus der Perspektive einer frei positionierbaren Kamera. Auf der rechten Seite sehen Sie die Toolbar, in der Sie fast alle Befehle finden, mit denen Sie später modellieren, texturieren und animieren. Am unteren Bildschirmrand sehen Sie die Kontrollbuttons für die Animation.

## Das erste 3D-Modell

Wenn Sie mit Ihrem 3D-Objekt beginnen, sollten Sie sich ungefähr vorstellen können, was Sie modellieren wollen. Am besten, Sie fertigen einige kleine Skizzen von verschiedenen Blickwinkeln an. Damit werden Sie besser zurechtkommen und im Endeffekt meist auch schneller am Ziel sein.

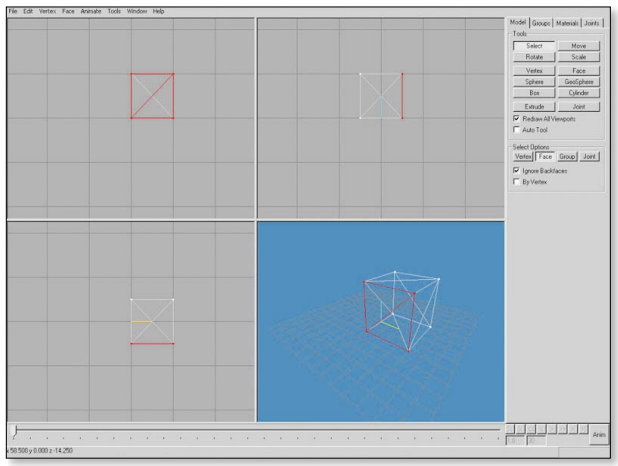
Für die Skizzen legen Sie die Proportionen fest. Bei der Charaktermodellierung, werden die Modelle meist aus kleinen, einfachen geometrischen Objekten zusammengesetzt. Diese Modelle finden Sie in Ego-Shootern. Damit geben Sie die grobe Form vor, die Sie dann in Feinarbeit verbessern.



**MILKSHAPE 3D** nach dem Start

Unser Beispiel soll den Oberkörper einer Spielfigur darstellen. Als einfaches geometrisches Objekt beginnen Sie mit einem Würfel. Wählen Sie dazu in der Toolbar unter dem *Model-Tab* eine *Box Tool* aus. Klicken Sie in die Front-Ansicht, und halten Sie den Knopf gedrückt, um ein Viereck aufzuziehen. In der Kameraansicht sehen Sie einen grauen Würfel. Die Darstellungsparameter einer Ansicht ändern Sie im Kontextmenü, das Sie über die rechte Maustaste erreichen. Um eine Ansicht zu verschieben oder zu skalieren, halten Sie die *[Strg]*- oder *[Shift]*-Taste gedrückt. Bewegen Sie die Maus, während Sie die Rechte Taste drükken, im entsprechenden Fenster.

Aus diesem Würfel werden Sie den Körper des 3D-Charakters modellieren. Wählen Sie in der Toolbar *Model/ Select*. Bei den *Select*-Optionen benötigen Sie die Einträge *Face* und *Ignore Backfaces*. Damit selektieren Sie per Mausklick einzelne Dreiecke, wobei Sie von hinten sichtbare Polygone ignorieren. Wählen Sie in der Frontansicht die beiden Dreiecke.

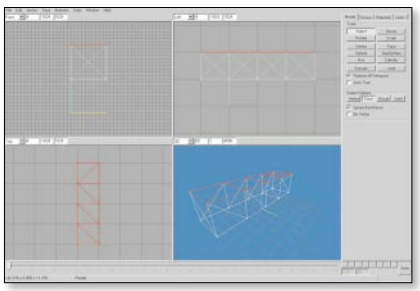


**DER EXTRUDE-BEFEHL** erzeugt weitere Würfel.

Mit *Toolbar/Extrude* können Sie neue Teile aus einem bestehenden 3D-Modell herausziehen. Ziehen Sie die markierten Dreiecke nach außen. Damit erhalten Sie einen zweiten Würfel, der mit dem ersten verbunden ist. Diesen Vorgang wie-

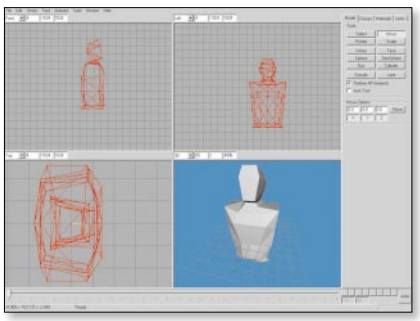
derholen Sie, bis Sie vier bis sechs Würfel haben, je nachdem, wie viele Polygone Sie für die Figur verwenden wollen.

Markieren Sie im Bild die obere Seite der Würfelreihe, und vervielfältigen Sie diese mehrmals, um mehrere Würfelreihen zu bekommen. Mit dem Befehl *Toolbar/Scale* können Sie die Breite unseres noch ziemlich ungestalteten Oberkörpers an markierten Vertices ändern. Zunächst erhalten Sie eine grobe Form, die Sie verfeinern.



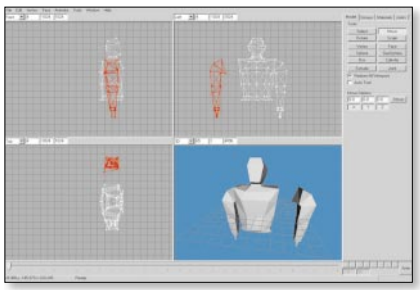
**NUN VERVIELFÄLTIGEN SIE** die obere Seite der Würfelreihe.

Den Arm der Figur erzeugen Sie mit dem *Extrude*-Befehl, um der groben Form Feinschliff zu geben. Damit Sie sich nicht zweimal die Arbeit machen, einen Arm zu modellieren, markieren Sie die Polygone des ersten Arms. Mit *Edit/Duplicate Selection* erhalten Sie eine Kopie des Arms. Diese liegt an derselben Position wie das Original und ist deshalb nicht zu sehen. Spiegeln Sie den Arm (im Menü unter *Vertex/Mirror*), und schieben Sie ihn möglichst genau an die Schulter.



**NACH GETANER FEINARBEIT** kann sich das Ergebnis sehen lassen.

Um den Arm und den Oberkörper zu verbinden, markieren Sie jeweils die zwei Vertices, die Sie verschmelzen wollen, und wählen *Vertex/Snap together*. Damit bekommen die beiden Vertices dieselbe Position, sind aber noch getrennt. Nachdem Sie alle Vertices angepasst haben, führen Sie den Befehl *Vertex/Weld together* aus, um die doppelten Vertices zu eliminieren. Die Hände unseres 3D-Modells sind aus separaten Würfeln aufgebaut. Die Beine modellieren Sie ebenso wie die Arme.



**JETZT KLEBEN SIE** den kopierten Arm an.

## Oberflächenmaterial und Texturen

Die meisten Ego-Shooter kommen mit wenigen Texturen für die 3D-Charaktere aus, meist reicht eine mit 256 x 256 Pixeln. Dadurch werden der Aufwand beim Rendering und der Textur-Speicherverbrauch gesenkt.

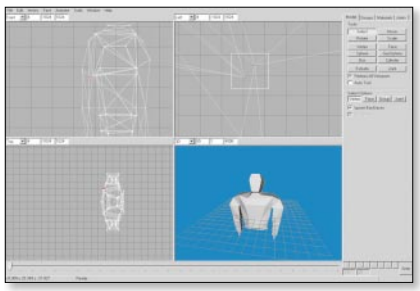
Zunächst gruppieren Sie Ihr 3D-Modell neu. In der Toolbar unter *Groups* finden Sie eine Liste von Polygongruppen, die Sie getrennt selektieren, löschen, benennen und verstecken können. Wählen Sie die Gruppen *Oberkörper*, *Kopf* und *Arme* aus, und wählen Sie *Regroup*, um sie zusammenzufassen. Die *Smoothing Groups* unten in der Toolbar fassen die Polygone zusammen, die zu einer glatten Oberfläche gehören. Falsch zusammengefasste Polygone machen sich beim Rendering durch Schattierungsfehler bemerkbar.

Wählen Sie in der Toolbar den *Material-Tab*, und erzeugen Sie mit *New* neues Material. Wenn Sie alle Polygone ausgewählt haben, drücken Sie *Assign*, um das neue Material zuzuweisen. Für das Material können Sie beliebige *Ambient-*, *Diffuse-* und *Specular-* Farben einstellen.

Texturen heißen bei 3D-Modellen auch *Skins*. Um dem Material eine Textur hinzuzufügen, drücken Sie auf den oberen der beiden Knöpfe, die mit *<none>* beschriftet sind, und wählen eine Bilddatei aus. Viel Zeit kostet die Anpassung der Textur an die Polygone, also die Festlegung der Texturkoordinaten. Wenn Sie ein oder mehrere Dreiecke auswählen und im Menü unter *Window* den *Texture Coordinate Editor* wählen (oder *[Strg-t]* drücken), öffnet sich ein Fenster. Dort sehen Sie die Textur des Materials und die Vertices der markierten Polygone. Durch die Verschiebung der Vertices in der Textur legen Sie die Texturbereiche fest, die jeweils auf ein Polygon projiziert (gemappt) werden. Angesichts der Anzahl der Vertices ist das sehr aufwendig. Aber gerade die Texturierung und die exakte Zuweisung der Koordinaten lässt ein Low-Polygon-Modell wirklich gut aussehen.

## Milkshape-3D-Daten

Wenn Sie keine eigenen 3D-Modelle erzeugen wollen, können Sie Milkshape 3D nutzen, um zahlreiche Formate wie *MD2*, *MD3*, *MDL*, *3D*-Studio, Lightwave, Autocad DXF ins eigene *MS3D*-Format zu konvertieren. Das *MS3D*-Format von Milkshape 3D ist sehr einfach aufgebaut, lässt sich komfortabel laden und in OpenGL darstellen. Als Framework für die OpenGL- Darstellung verwenden Sie den OpenGL-Startup, den Sie aus den letzten PC-Underground-Ausgaben kennen. Dessen Aufbau müssen Sie aber nicht im Kopf haben, um ihn anzuwenden.



**DIESE BEIDEN VERTICES** müssen Sie verschmelzen.

Im *MS3D*-Dateiformat sind für Sie zunächst fünf Strukturen interessant:

„1 Am Beginn einer *MS3D*-Datei befindet sich der Header. Die ID muss *MS3D000000* sein, die Versionsnummer *3*:

```
typedef struct
{
    char    id[ 10 ];
    int     version;
} MS3D_HEADER;
```

Anschließend folgt ein 16-Bit-Interrupt-Aufruf *(unsigned int)*, der die Anzahl der Vertices des 3D-Objekts angibt. Entsprechend oft finden Sie folgende Struktur in der Datei, die die Vertexdaten enthält:

```
typedef struct
{
 byte  flags;
 float vertex[ 3 ];
 char  boneId;
 byte  refCount;
} MS3D_VERTEX;
```

In der *MS3D_VERTEX*-Struktur sind unter anderem die drei Floatwerte interessant, die die Koordinaten des Vertexes enthalten.

Es folgt wieder ein *unsigned int* (16 Bit), in dem die Anzahl der Dreiecke gespeichert ist. Entsprechend lesen Sie folgende Struktur aus:

**EIN *SKIN*** ist eine Textur für einen 3D-Charakter.

```
typedef struct
{
 word  flags;
 word  vertexIndices[ 3 ];
 float vertexNormals[ 3 ][ 3 ];
 float s[ 3 ];
 float t[ 3 ];
 byte  smoothingGroup;
 byte  groupIndex;
} MS3D_TRIANGLE;
```

Wichtig sind hier die *vertexIndices*, die Indizes der Dreieckspunkte in der Vertexliste, die dazugehörigen Normalen und die Texturkoordinaten (*s* und *t*).

Für die *Smoothing Groups* lesen Sie zuerst wieder die Anzahl aus und dann entsprechend oft die dazugehörige Struktur. Achten Sie darauf, dass die *Smoothing Groups* eine variierende Anzahl von Dreiecken enthalten (*nTriangles* und **triangleIndices*):

```
typedef struct
{
 byte flags;
 char name[ 32 ];
 word nTriangles;
 word *triangleIndices;
 char materialIndex;
} MS3D_GROUP;
```

Zuletzt folgen die Materialdefinitionen. In der Struktur sind alle Daten enthalten, die Sie in Milkshape in der *Toolbar/Material* einstellen können. Zusätzlich finden Sie in der Struktur einen Zeiger auf ein *PCUTexture*-Objekt, um in der *MS3D*-Laderoutine die Texture-Bitmaps zu laden.

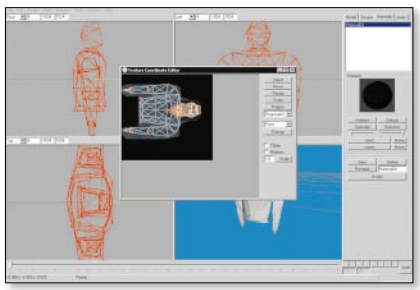
```
typedef struct
{
 char  name[ 32 ];
 float ambient[ 4 ];
 float diffuse[ 4 ];
 float specular[ 4 ];
 float emissive[ 4 ];
 float shininess;
 float transparency;
 char  mode;
 char  texture[ 128 ];
 char  alphamap[ 128 ];

PCUTexture *textureMap;
} MS3D_MATERIAL;
```

Die gelesenen Daten speichern Sie am besten zusammen in einer Klasse, die Sie im Sourcecode zu dieser Ausgabe finden:

```
class MS3DObject
{
private:
 word nVertices;
 word nTriangles;
 word nGroups;
 word nMaterials;

 MS3D_HEADER     header;
 MS3D_VERTEX     *pVertex;
 MS3D_TRIANGLE   *pTriangle;
 MS3D_GROUP      *pGroup;
 MS3D_MATERIAL   *pMaterial;

public:
 MS3DObject();
 ~MS3DObject();

 int  loadObject
 ( char *path, char *name );
 void renderObject();
};
```

## *MS3D* mit OpenGL rendern

Die Daten aus den *MS3D*-Dateien sind prädestiniert, um sie mit OpenGL zu rendern. Die Materialparameter können Sie direkt an OpenGL weitergeben. Rendern Sie die *Groups* nacheinander.



**MIT DEM *TEXTURECOORDINATE-EDITOR*** passen Sie die Textur an.

Setzen Sie jeweils die entsprechenden Materialparameter und gegebenenfalls die Textur. Zeichnen Sie alle Dreiecke der Group, indem Sie die Normalen, Texturkoordinaten und Vertexkoordinaten übermitteln:

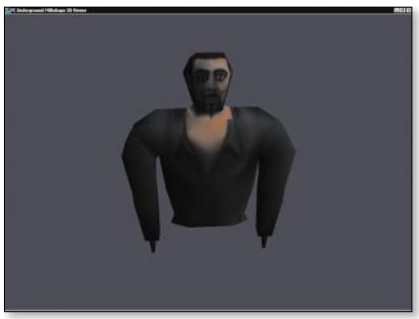
```
void MS3DObject::renderObject()
{
 for ( int i = 0; i <nGroups;
i++> )
 {
// Materialparameter
int m = pGroup[ i ].materialIn-
dex;

glMaterialfv( GL_FRONT_AND_BACK,
 GL_AMBIENT,pMaterial[m].am-
bient);
glMaterialfv( GL_FRONT_AND_BACK,
 GL_DIFFUSE,pMaterial[m].diffuse
);
glMaterialfv( GL_FRONT_AND_BACK,
 GL_SPECULAR,pMaterial[m].specu-
lar);
glMaterialfv( GL_FRONT_AND_BACK,
 GL_SHININESS,
 &pMaterial[m].shininess );
glMaterialfv( GL_FRONT_AND_BACK,
 GL_EMISSION,
 pMaterial[m].emissive );

if ( pMaterial[m].textureMap )
  pMaterial[m].textureMap->sel-
ect();

glBegin( GL_TRIANGLES );

for
(j=0;j<pGroup[i].nTriangles;j++)
{
 idx = pGroup[i].triangleIndi-
ces[j];
 MS3D_TRIANGLE
*tri=&pTriangle[idx];

 for (k=0;k<3;k++)
 {
  glNormal3fv(
    tri->vertexNormals[k]);
  glTexCoord2f(
    tri->s[k],tri->t[k]);
  glVertex3fv(
    pVertex[tri-
>vertexIndices[k]].vertex);
 }
}
 glEnd();
 }
}
```

Damit die Materialparameter, die Sie an OpenGL übergeben, korrekt dargestellt werden, schalten Sie mit *glEnable( GL_LIGHTING)* die OpenGL-Beleuchtungsberechnung an und definieren noch eine Lichtquelle:

```
glEnable( GL_LIGHTING );
glEnable( GL_LIGHT0 );

GLfloat ldiffuse[]={1,1,1,1};
GLfloat lambient[]={0.1f,0.1f,
                    0.1f,1.0f};

glLightfv( GL_LIGHT0, GL_AMBIENT,
lambient );
glLightfv( GL_LIGHT0, GL_DIFFUSE,
ldiffuse );
glLightfv( GL_LIGHT0, GL_SPECU-
LAR,ldiffuse );

GLfloat lpositi-
on[4]={30,30,30,0};
glLightfv( GL_LIGHT0, GL_POSITI-
ON, lposition );
```

Im Bild sehen Sie unser Beispielprogramm zusammen mit dem 3D-Modell, das Sie im ersten Teil des Artikels gestalten konnten.

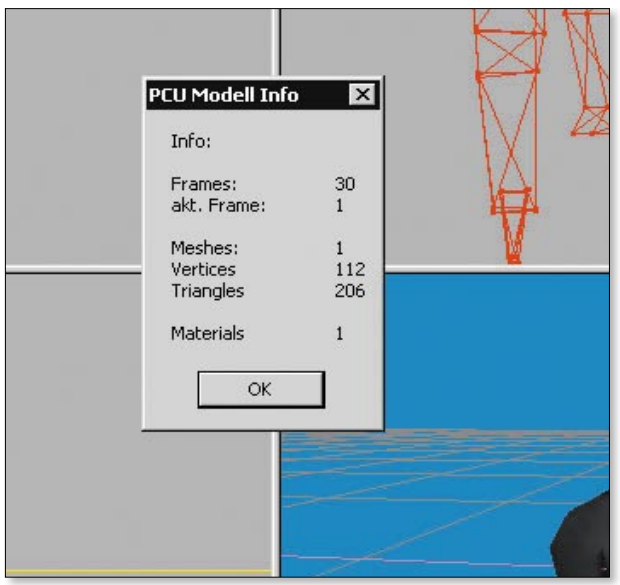
Wenn Sie weitere 3D-Objekte anlegen wollen, empfehlen wir Ihnen Tutorials zu Milkshape 3D. Eine gute Sammlung von Links zu diesem Thema finden Sie unter www.swissquake.ch/ chumbalum-soft/ ms3d1x/tutorials.html Weitere freie Testmodelle, die Sie unter Umständen noch mit Milkshape 3D konvertieren müssen, finden Sie unter www.3dcafe.com oder wenn Sie mit Suchmaschinen zum Beispiel nach *md2 model* suchen.



**DAS BEISPIELPROGRAMM** stellt unser 3D-Modell in OpenGL dar.

## Plug-ins für Milkshape 3D

Milkshape 3D unterstützt viele Dateiformate. Sie können Daten an Milkshape 3D auch übergeben, indem Sie Ihr eigenes Plug-in in Form einer DLL-Datei schreiben. Wenn Sie es ins Milkshape-3D-Programmverzeichnis kopieren, können Sie es vom Hauptmenü aus aufrufen und laden. Eigene Plug-ins programmieren Sie mit den Libraries, die Sie im SDK (Software Developer Kit) finden. Das SDK können Sie zusammen mit Milkshape 3D downloaden. Legen Sie dazu in Ihrem C++-Compiler ein DLL-Projekt an. Linken Sie die *msModelLibd.lib* aus dem SDK dazu, und binden Sie die Header *msPlugInImpl.h* und *msLib.h* ein. Schreiben Sie ein Plug-in, das nur Informationen über ein 3D-Modell in einer Windows-MessageBox ausgibt. Dies demonstriert, wie Sie auf die Modelldaten für ein Export-/Import-Plug-in zugreifen.



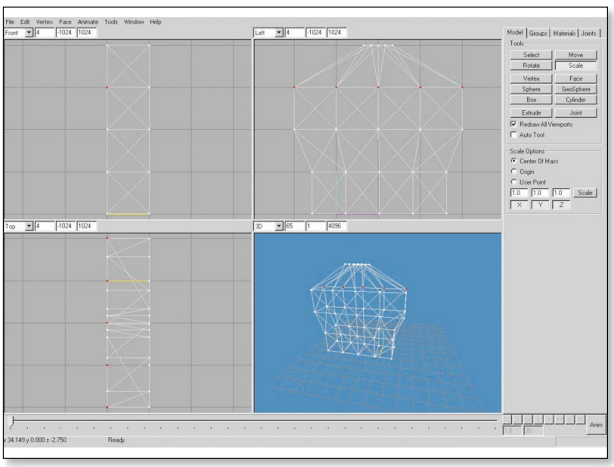

**UNSER KLEINES BEISPIEL-PLUG-IN** informiert Sie über das 3D-Modell.

Der Aufbau der Plug-in-DLLs ist immer gleich: Sie benötigen einen *DLLMain Entry Point*, den jede DLL unter Windows besitzen muss. Um die DLL zu einem Milkshape-Plug-in zu machen, müssen Sie folgende Klasse implementieren:

```
class cPlugIn : public cMsPlugIn
{
  char szTitle[64];

public:
  cPlugIn ();
  virtual ~cPlugIn ();
public:
 int          GetType ();
 const char * GetTitle ();
 int
Execute (msModel* pModel);
};
```

Die Methode *GetType()* liefert zwei Funktionen: Export oder Import. Auf diese Funktionen greifen Sie in der DLL unter *cMsPlugIn::eTypeExport* oder *cMsPlugIn::eTypeImport* zurück.

Die Methode *GetTitle()* liefert den Titel des Plug-ins zurück, der im Hauptmenü angezeigt wird. Die *Execute()*-Methode enthält einen Zeiger auf ein *msModel* (Milkshape-3D-Modell), in dem alle Daten gespeichert sind. Auf diese Daten können Sie mit Funktionen aus den Library-Dateien zugreifen. Das kleine Beispiel soll Daten nur in eine *MessageBox* exportieren. Dazu implementieren Sie die *Execute()Methode*:



**DEM WÜRFEL** verpassen Sie ungefähr die Form eines Oberkörpers.

```
int cPlugIn::Execute
(msModel *pModel)
{
  // sicherheitsabfragen:
  if ( !pModel )
    return -1;

  if (msModel_GetMeshCount( pMo-
del )==0)
  {
    // kein modell vorhanden
    return 0;
  }

  anzahl_frames =
   msModel_GetTotalFrames( pModel
);
  akt_frame =
   msModel_GetFrame( pModel );
  anzahl_meshes =
   msModel_GetMeshCount( pModel
);

  for (i=0;i<anzahl_meshes;i++)
  {
    msMesh *pMesh =
      msModel_GetMeshAt( pModel,
i );

    anzahl_vertices[i] =

msMesh_GetVertexCount(pMesh);
    anzahl_triangles[i] =

msMesh_GetTriangleCount(pMesh);
  );

  // ausgabe zusammenbauen als
string
  ...

  MessageBox( NULL, text, "PCU
Modell Info", MB_OK );

  // wichtig: wieder freigeben:
  msModel_Destroy (pModel);

  return 0;
}
```

Analog bietet die Library auch Funktionen, um Daten zu schreiben oder um beliebige Datei-Formate zu importieren. Sie können damit sogar eigene Tools, die 3D-Daten generieren oder automatisch modifizieren, direkt einbinden. ET

Nähere Informationen zu diesem Beitrag finden Sie auf der Website www.dachsbacher.de/pcu