3D-Vektorgraphik: Mesh

Mit Maschenwerk = Mesh bezeichnet man:
1) ein von DirectX unterstütztes Vektor-Fileformat = **X-File** mit der Extension `.x`.
2) eine Direct3D-Klasse aus dem Namespace `Microsoft.DirectX.Direct3DX`, die solche `.x`-Files auswertet und rendert.

Primitive Meshes

Direct3D bietet mehrere Standard-3D-Polygone in seiner `Mesh`-Klasse, aus denen man einfache Figuren zusammenbauen kann.
Ihre Vertices sind vom Typ `CustomVertex.PositionNormal`, sind deshalb gerichtet beleuchtbar aber sie besitzen weder Farben noch Texturkoordinaten und sind deshalb nicht texturierbar. Sie benutzen die `Material`-Eigenschaften von `Device`.
Beispiele:

```
static Mesh myPolygon  = Mesh.Polygon ( device, 0.3f, 8 ); //line length + no of symmetric vertices
static Mesh myBox      = Mesh.Box     ( device, 0.5f, 0.5f, 0.5f ); //xSize, ySize, zSize
static Mesh mySphere   = Mesh.Sphere  ( device, 0.5f, 20, 20 ); //radius, no slices, no stacks
static Mesh myTorus    = Mesh.Torus   ( device, 0.2f, 0.4f, 20, 20 );//in+out radii, slices+stacks
static Mesh myCylinder = Mesh.Cylinder( device, 0.5f, 0.2f, 0.8f, 20, 20 ); //front+back radii, length,
slices+stacks
static Mesh myTeapot   = Mesh.Teapot  ( device );
static Mesh myText     = Mesh.TextFromFont( device,
                                            new System.Drawing.Font( FontFamily.GenericSerif, 12 ),
                                            text, 0.01f, 0.25f ); //string, smooth, thick

device.BeginScene();
  myPolygon .DrawSubset( 0 );
  myBox     .DrawSubset( 0 );
  mySphere  .DrawSubset( 0 );
  myTorus   .DrawSubset( 0 );
  myCylinder.DrawSubset( 0 );
  myTeapot  .DrawSubset( 0 );
  myText    .DrawSubset( 0 );
device.EndScene();
```

X-Files

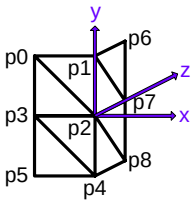
Das X-Fileformat, auch `DXF`-Format genannt, stammt von Microsoft und ist, ähnlich wie das `BMP`-Format für Rasterbilder, im Laufe der Zeit universell populär geworden. Es existiert als a) kompaktes aber unlesbares Binärformat und b) als editierbares Textformat.
Eine Spezifikation finden Sie unter: **http://astronomy.swin.edu.au/~pbourke/geomformats/directx/**.
Man kann aus allen 3D-Modell-Editoren X-Files exportieren:
Maya 5 und **Maya 6** Plug-Ins siehe: `C:\DXSDK\Utilities\Bin\plugins\Maya`
Adobe Photoshop Plug-In siehe: `C:\DXSDK\Utilities\Bin\plugins\Photoshop`
Autodesk 3ds Max Plug-In siehe: **www.andytather.co.uk/Panda/directxmax_downloads.aspx**
AC3D siehe: **www.ac3d.org**
LightWave 3D siehe: **www.newtek.com/products/lightwave/product/index.html**
MeshX siehe: **www.spinnerbaker.com/meshx.htm**
Shareware **MilkShape 3D** siehe: **www.swissquake.ch/chumbalum-soft/ms3d/index.html**

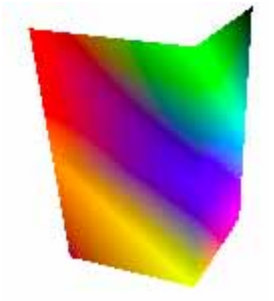
X-File Sammlung: **www.3dcafe.com/asp/anatomy.asp#DXF**
lesenswert: **pluralsight.com/wiki/default.aspx/Craig.DirectX/MeshBasicsTutorial.html**

Man kann ein X-File im Textformat schreiben, was zwar mühsam aber aufschlussreich ist.

Beispiel für ein `.x`-File und ein dazu passendes DirectX-Render-Programm:

```
xof 0302txt 0064  //mandatory X-file header
Mesh {
 //VertexBuffer
 9;              //vertex count
 -1.0; 1.0; 0.0;, //p0
  0.0; 1.0; 0.0;, //p1
  0.0; 0.0; 0.0;, //p2
 -1.0; 0.0; 0.0;, //p3
  0.0;-1.0; 0.0;, //p4
 -1.0;-1.0; 0.0;, //p5
  0.0; 1.0; 1.0;, //p6
  0.0; 0.0; 1.0;, //p7
  0.0;-1.0; 1.0;; //p8
 //IndexBuffer
 8;             //face count
 3; 0, 1, 2;,   //triangle0
 3; 0, 2, 3;,   //triangle1
 3; 3, 2, 4;,   //triangle2
 3; 3, 4, 5;,   //triangle3
 3; 1, 6, 7;,   //triangle4
 3; 1, 7, 2;,   //triangle5
 3; 2, 7, 8;,   //triangle6
 3; 2, 8, 4;;   //triangle7
 MeshVertexColors {
  9;                     //vertex count
  0; 1.0; 0.0; 0.0; 0;;, //red
  1; 0.0; 1.0; 0.0; 0;;, //green
  2; 0.0; 0.0; 1.0; 0;;, //blue
  3; 1.0; 0.6; 0.0; 0;;, //orange
  4; 1.0; 1.0; 0.0; 0;;, //yellow
  5; 0.6; 0.2; 0.2; 0;;, //brown
  6; 0.0; 0.0; 0.0; 0;;, //black
  7; 0.0; 1.0; 1.0; 0;;, //cyan
  8; 1.0; 0.0; 1.0; 0;;; //magenta
 }
 MeshMaterialList { //void but mandatory
  1;1;0;;
  Material { 0;0;0;0;;0;0;0;0;;0;0;0;; }
 }
} //end of Mesh
```





Kopieren Sie diesen Text in eine Textdatei `C:\temp\mesh.x`.
Beachten Sie:
1) `.x`-Files müssen mit der Zeile `xof 0302txt 0064` beginnen.
2) Danach enthalten `.x`-Files normalerweise so genannte `Templates` = Syntaxbeschreibungen der Blöcke. Diese `Templates` sind nicht unbedingt nötig und sind hier weggelassen.
3) Block `Mesh` enthält den Vertex- und den IndexBuffer und zwei Unterblöcke `MeshVertexColors` und `MeshMaterialList`, wobei der letztere obligatorisch aber in diesem Beispiel ohne weitere Bedeutung ist. Blöcke werden hinter ihrem Bezeichner durch geschweifte Klammern umschlossen.
4) Mit `//` beginnen Kommentare, die wie Leerzeichen und Zeilensprünge keine technische Wirkung haben.
5) Jeder Block beginnt mit einem Integer plus Semikolon = Anzahl der folgenden Elemente.
6) Die Elemente werden durch Komma getrennt, die Zahlen innerhalb der Elemente werden durch Semikolon getrennt. Nach dem letzten Element folgt kein Komma, sondern ein Semikolon.
7) Jedes Element des IndexBuffers benötigt einen `index count` (hier: `3;` in der vorderen Spalte).
8) Jedes Element des `MeshVertexColors`-Blocks benötigt eine Vertexnummer (hier: `0; ... 8;` in der vorderen Spalte).
9) Die Farben R, G, B und die Transparenz werden codiert zwischen `0.0` = nicht vorhanden und `1.0` = voll.
10) Die Schreibweisen `0.0` und `0` sind für `floats` gleichwertig, nicht aber `1.0` und `1`.
11) Kleine syntaktische Fehler machen das File unbrauchbar und sind schwer zu finden. Sorgfältiges Layout in Spaltenform ist wichtig.

Mesh lesen

C#-Programm `mesh1`:

```
//Form1.cs ********************************************************************
//Add References: Microsoft.DirectX, Microsoft.Direct3D, Microsoft.Direct3DX, Vers. >= 1.0.2902.0
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
public class Form1 : Form
{ static void Main() { Application.Run( new Form1() ); }
  static Device device;
  static Mesh   mymesh;
  static float  fAngle;
  PresentParameters presentParams;
  Timer myTimer = new Timer();
  public Form1()
  { myTimer.Tick += new EventHandler( OnTimer );
    myTimer.Interval = 1;
    ClientSize = new Size( 400, 300 );
  }
  protected override void OnResize( System.EventArgs e )
  { myTimer.Stop();
    try
    { presentParams = new PresentParameters();
      presentParams.Windowed = true;
      presentParams.SwapEffect = SwapEffect.Discard;
      presentParams.EnableAutoDepthStencil = true;
      presentParams.AutoDepthStencilFormat = DepthFormat.D16;
      if ( device != null ) device.Dispose();
      device = new Device( 0, DeviceType.Hardware, this,
                           CreateFlags.SoftwareVertexProcessing, presentParams );
      if ( mymesh != null ) mymesh.Dispose();
      try { mymesh = Mesh.FromFile( @"C:\temp\mesh.x", MeshFlags.SystemMemory, device ); }
      catch { MessageBox.Show( @"Missing or Invalid: C:\temp\mesh.x" ); return; }
      device.Transform.View = Matrix.LookAtLH(
        new Vector3( 0f,0f,-4f ), new Vector3( 0f,0f,0f ), new Vector3( 0f,1f,0f ) );
      device.Transform.Projection = Matrix.PerspectiveFovLH( (float)Math.PI/4, 1f, 1f, 10f );
      device.RenderState.CullMode = Cull.None;
      device.RenderState.Ambient  = Color.White;
      device.RenderState.AmbientMaterialSource = ColorSource.Color1;
      myTimer.Start();
    }
    catch (DirectXException ex) { MessageBox.Show( ex.ToString() ); return; }
  }
  protected static void OnTimer( Object myObject, EventArgs myEventArgs )
  { if (device == null) return;
    device.Clear( ClearFlags.Target | ClearFlags.ZBuffer, Color.Blue, 1f, 0 );
    device.Transform.World = Matrix.RotationY( fAngle += 0.01f );
    device.BeginScene();
      mymesh.DrawSubset(0);
    device.EndScene();
    device.Present();
  }
}
```

Das Programm enthält keinerlei Vertices, Indices oder Farben, sondern entnimmt alles aus dem `.x`-File `C:\temp\mesh.x`.
Sie müssen mit den drei üblichen Referenzen `Microsoft.DirectX`, `Microsoft.DirectX.Direct3D` und `Microsoft.DirectX.Direct3DX` linken.

Extended Material

a) In der `MeshMaterialList` kann der `Material`-Block eine Textur enthalten = `ExtendedMaterial`.
Das `.x`-File muss dann zusätzlich Texturkoordinaten enthalten.

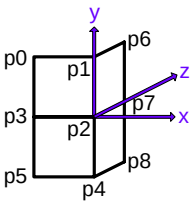
```
MeshMaterialList {
 1;1;0;;
 Material { 1.0;1.0;1.0;0;;0;0;0;0;;0;0;0;; TextureFilename { "C:\\temp\\bmp1.bmp"; } }
}
MeshTextureCoords {
 4;
 0.0, 0.0; //left  upper corner
 1.0, 0.0; //rigth upper corner
 0.0, 1.0; //left  lower corner
 1.0, 1.0;;//right lower corner
}
```

b) In der `MeshMaterialList` können zwei oder mehr `Material`-Blöcke enthalten sein.
In diesem Fall muss man die `ExtendedMaterials` einzeln laden, in `static Material[]`- und `static Texture[]`-Arrays umkopieren und rendern.
Beispiel mit 4 Vierecken und 2 `ExtendedMaterials`:
Wenn Sie dieses Beispiel nachvollziehen wollen, dann laden Sie die beiden linken Bilder nach `C:\temp`:

`C:\temp\bmp1.bmp`

`C:\temp\bmp2.bmp`

von `mesh.x` geliefertes Ergebnis




```
xof 0302txt 0064  //mandatory X-file header
Mesh {
 //VertexBuffer
 9;                //vertex count
 -1.0; 1.0; 0.0;, //p0
  0.0; 1.0; 0.0;, //p1
  0.0; 0.0; 0.0;, //p2
 -1.0; 0.0; 0.0;, //p3
  0.0;-1.0; 0.0;, //p4
 -1.0;-1.0; 0.0;, //p5
  0.0; 1.0; 1.0;, //p6
  0.0; 0.0; 1.0;, //p7
  0.0;-1.0; 1.0;; //p8
 //IndexBuffer
 4;                //face count
 4; 0, 1, 2, 3;,  //quad0
 4; 3, 2, 4, 5;,  //quad1
 4; 1, 6, 7, 2;,  //quad2
 4; 2, 7, 8, 4;;  //quad3
 MeshTextureCoords { //each pi needs a ti
  9;
  0.0, 0.0; //t0      → upper left
  1.0, 0.0; //t1      → upper rigth
  1.0, 1.0; //t2      → lower right
  0.0, 1.0; //t3      → lower left
  1.0, 0.0; //t4 = t1 → mirror down
  0.0, 0.0; //t5 = t0 → mirror down
  0.0, 0.0; //t6 = t0 → mirror right
  0.0, 1.0; //t7 = t3 → mirror right
  0.0, 0.0;;//t8 = t0 → mirror down and right
 }
 MeshMaterialList {
  2;4;0,1,1,0;; //2 mats for 4 faces: mat 0 for faces 0,3, mat 1 for faces 1,2
  Material { 1.0;1.0;1.0;0;;0;0;0;0;;0;0;0;; TextureFilename { "C:\\temp\\bmp1.bmp"; } }
  Material { 1.0;1.0;1.0;0;;0;0;0;0;;0;0;0;; TextureFilename { "C:\\temp\\bmp2.bmp"; } }
 }
} //end of Mesh file C:\temp\mesh.x ************************************************************
```

```
//C#-Program: Form1.cs ********************************************************************
//Add References: Microsoft.DirectX, Microsoft.Direct3D, Microsoft.Direct3DX, Vers. >= 1.0.2902.0
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
public class Form1 : Form
{ static void Main() { Application.Run( new Form1() ); }
  static Device device;
  static Mesh   mymesh;
  static float  fAngle;
  ExtendedMaterial[] exmat;        //new !
  static Material [] mat;          //new !
  static Texture  [] tex;          //new !
  PresentParameters presentParams;
  Timer myTimer = new Timer();
  public Form1()
  { Text = "mesh1";
    myTimer.Tick += new EventHandler( OnTimer );
    myTimer.Interval = 1;
    ClientSize = new Size( 400, 300 );
  }
  protected override void OnResize( System.EventArgs e )
  { myTimer.Stop();
    try
    { presentParams = new PresentParameters();
      presentParams.Windowed = true;
      presentParams.SwapEffect = SwapEffect.Discard;
      presentParams.EnableAutoDepthStencil = true;
      presentParams.AutoDepthStencilFormat = DepthFormat.D16;
      if ( device != null ) device.Dispose();
      device = new Device( 0, DeviceType.Hardware, this,
                           CreateFlags.SoftwareVertexProcessing, presentParams );
      if ( mymesh != null ) mymesh.Dispose();
      try { mymesh = Mesh.FromFile( @"C:\temp\mesh.x", MeshFlags.SystemMemory, device, out exmat ); }
      catch { MessageBox.Show( @"Missing or Invalid: C:\temp\mesh.x" ); return; }
      if ( exmat != null && exmat.Length > 0 )
      { mat = new Material[exmat.Length];
        tex = new Texture [exmat.Length];
        for ( int i=0; i < exmat.Length; i++ )
        { mat[i].Ambient = exmat[i].Material3D.Diffuse;
          if ( exmat[i].TextureFilename != null )
            try { tex[i] = TextureLoader.FromFile( device, exmat[i].TextureFilename ); }
            catch { MessageBox.Show( "Missing: " + exmat[i].TextureFilename ); return; }
        }
      }
      device.Transform.View = Matrix.LookAtLH(
        new Vector3( 0f,2f,-3f ), new Vector3( 0f,0f,0f ), new Vector3( 0f,1f,0f ) );
      device.Transform.Projection = Matrix.PerspectiveFovLH( (float)Math.PI/4, 1f, 1f, 10f );
      device.RenderState.CullMode = Cull.None;
      device.RenderState.Ambient  = Color.White;
      device.RenderState.Lighting = true;
      myTimer.Start();
    }
    catch( DirectXException ex ) { MessageBox.Show( ex.ToString() ); return; }
  }
  protected static void OnTimer( Object myObject, EventArgs myEventArgs )
  { if (device == null) return;
    device.Clear( ClearFlags.Target | ClearFlags.ZBuffer, Color.White, 1f, 0 );
    device.Transform.World = Matrix.RotationY( fAngle += 0.01f );
    device.BeginScene();
      for ( int i=0; i < mat.Length; i++ )
      { device.Material = mat[i];
        device.SetTexture( 0, tex[i] );
        mymesh.DrawSubset( i );
      }
    device.EndScene();
    device.Present();
  }
}
```

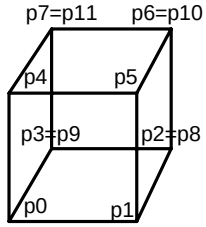
Textured Cube

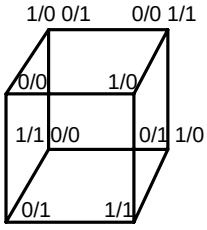
Das Zeichnen eines Würfels ist eigentlich einfach: `Mesh wuerfel = Mesh.Box( device, 1f, 1f, 1f );`
Allerdings liefert die Analyse des Würfels überraschende Ergebnisse:
`int nv = wuerfel.NumberVertices;` liefert die Eckenzahl `nv = 24` anstatt der erwarteten `8`.
`int nf = wuerfel.NumberFaces;` liefert die Flächenzahl `nf = 12` anstatt der erwarteten `6`.
`VertexFormats vf = wuerfel.VertexFormat;` liefert das VertexFormat `vf = PositionNormal`, statt wie erwartet `PositionNormalTextured`.
Offensichtlich codiert `Mesh.Box` jede Ecke 3-fach und jedes Quadrat durch 2 Dreiecke.
`Mesh.Box` besitzt außerdem keine Texturkoordinaten und `Mesh.Box` kann folglich nicht mit Texturen bedeckt werden.
Es gibt zwei Lösungen für dieses Problem:

1) `Mesh.Box` klonen in eine um Texturkoordinaten erweiterte Kopie "`MyTexturedBox`".
   Diese Lösung ist beschrieben unter **http://msdn.microsoft.com/coding4fun/zman/zmantextures**
2) Ein `cube.x`-File schreiben, das alle Vertices, Faces und Texturen selbst definiert.

`cube.x`-File schreiben:
Diese Aufgabe ist nicht ganz trivial, weil an jedem der 8 Vertices des Würfels drei Texturen hängen, aber jeder Vertex nur eine Texturkoordinate `Tu,Tv` in sein Vertexformat `PositionTextured` bzw. `PositionNormalTextured` verpacken kann. Zur Bedeckung der 6 Würfelflächen braucht man 6 Texturen und jede Textur braucht 4 Texturkoordinaten = insgesamt 6*4 = 24 Texturkoordinaten und man beginnt zu verstehen, warum `Mesh.Box` 24 Ecken erzeugt und nicht nur 8.
Folge: Wegen der Texturkoordinaten muss man jeden Vertex 3 mal programmieren, je einmal für jede der 3 Flächen, die er begrenzt. Für die freie Orientierung je eines Bildes auf je einer Würfelfläche braucht man also 24 texturierte Vertices mit entsprechend komplizierter Numerierung und viele Zeilen Programmcode.

In vielen Fällen ist eine solche Freiheit der Bildorientierung nicht unbedingt notwendig. Wenn es irrelevant ist, ob ein Bild gespiegelt ist oder auf dem Kopf steht, kann man auch mit 12 Vertices auskommen.
Anleitung: Man programmiert zunächst die 4 Vertices des Fußbodens p0 bis p3 und dann die 4 der Decke p4 bis p7, dann noch einmal die beiden der Hinterkante des Fußbodens p8 = p2 und p9 = p3 und die beiden der Hinterkante der Decke p10 = p6 und p11 = p7. Im Indexbuffer kodiert man zunächst die Vorderwand, dann die rechts anschließende Wand, dann die Hinterwand und dann die linke Wand. Danach Boden und Decke unter Verwendung der doppelten Vertices p8, p9, p10 und p11.
Die Texturkoordinaten der Frontfläche sind einfach, aber die Textur der rechten Seitenfläche muss man horizontal spiegeln: An die letzte Spalte der Fronttextur hängt man die letzte Spalte der rechten Seitentextur, welche dann mit der ersten Spalte endet. An diese Spalte hängt man die erste Spalte der Hinterwandtextur und an deren letzte Spalte gespiegelt die Textur der rechten Seitenwand. Danach hängt man an die unterste Zeile der Vorderwandtextur die unterste Zeile der Bodentextur und an die oberste Zeile der Vorderwandtextur die oberste der Deckentextur.
Durch diese je zweifache Verwendung von 6 der insgesamt 12 Würfelkanten genügen 12 Vertices für eine vollständige Texturierung.





```
xof 0302txt 0064  //mandatory X-file header
Mesh {
 //VertexBuffer
 12;              //vertex count
  0.0; 0.0; 0.0;, //p0      floor
  1.0; 0.0; 0.0;, //p1
  1.0; 0.0; 1.0;, //p2
  0.0; 0.0; 1.0;, //p3
  0.0; 1.0; 0.0;, //p4      ceiling
  1.0; 1.0; 0.0;, //p5
  1.0; 1.0; 1.0;, //p6
  0.0; 1.0; 1.0;, //p7
  1.0; 0.0; 1.0;, //p8 = p2 floor   for texture
  0.0; 0.0; 1.0;, //p9 = p3
  1.0; 1.0; 1.0;, //p10= p6 ceiling for texture
  0.0; 1.0; 1.0;, //p11= p7
 //IndexBuffer
 6;               //face count
 4; 0, 1, 5, 4;,  //front   face
 4; 1, 2, 6, 5;,  //right   face
 4; 2, 3, 7, 6;,  //back    face
 4; 3, 0, 4, 7;,  //left    face
 4; 0, 1, 8, 9;,  //floor   face
 4; 4, 5,10,11;;  //ceiling face
 MeshTextureCoords {
  12;
  0.0, 1.0; //p0  floor
  1.0, 1.0; //p1
  0.0, 1.0; //p2
  1.0, 1.0; //p3
  0.0, 0.0; //p4  ceiling
  1.0, 0.0; //p5
  0.0, 0.0; //p6
  1.0, 0.0; //p7
  1.0, 0.0; //p8  floor
  0.0, 0.0; //p9
  1.0, 1.0; //p10 ceiling
  0.0, 1.0; //p11
 }
 MeshMaterialList {
  2;6;0,1,0,1,0,1;; //2 mats for 6 faces: mat 0 for faces 0,2,4, mat 1 for faces 1,3,5
  Material { 1.0;1.0;1.0;0;;0;0;0;0;;0;0;0;; TextureFilename { "C:\\temp\\bmp1.bmp"; } }
  Material { 1.0;1.0;1.0;0;;0;0;0;0;;0;0;0;; TextureFilename { "C:\\temp\\bmp2.bmp"; } }
 }
} //end of Mesh file C:\temp\cube.x
************************************************************
```

Speichern Sie dieses Mesh nach `C:\temp\cube.x`, ändern Sie im obigen C#-Programm den Filenamen im `Mesh.FromFile`-Befehl und in `MessageBox.Show` und erproben Sie `cube.x`.

Hierarchien und Animation-Keys

Man kann in einem Mesh Mutter-Kind-Enkel-Objekte kodieren.
Eine gute Einführung über Frame Hierarchy finden Sie unter: **www.jkarlsson.com/Articles/loadframes.asp**.
Zusätzlich kann man Frames zeitgesteuert animieren: **www.jkarlsson.com/Articles/animation.asp**