AUF CD

Die Quelltexte sowie die fertig übersetzten Routinen finden Sie im Verzeichnis *Heft Add-ons/Programmierung/PC Underground.*

Constructive Solid Geometry, Texturen in OpenGL

Glanz im Strahlenkranz

Bringen Sie Ihrer Grafikkarte Boolesche Operationen bei und rendern Sie damit Constructive-Solid-Geometry-Objekte. Mit einfachen Textur-Tricks erhalten Sie beeindruckende Lichteffekte!

CARSTEN DACHSBACHER

Lernen Sie die Booleschen Operationen für 3D-Körper kennen, die eigentlich nicht für polygonal definierte, sondern mathematisch beschriebenen Körper eingeführt wurden. Die drei primären Booleschen Operationen sind:

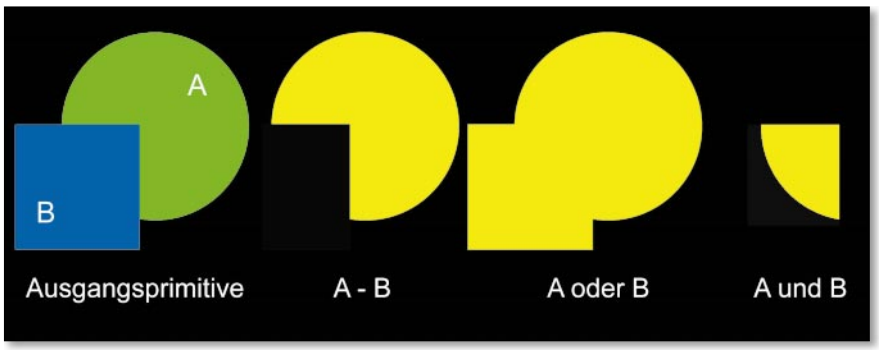
- die Subtraktion,
- die Vereinigung
- und die Schnittberechnung.

Die letzteren beiden werden auch – analog zur Mengenlehre – als *Oder-* bzw. *Und*-Verknüpfung bezeichnet. Die Subtraktion schneidet das Volumen bzw. die Form eines Objekts aus einem anderen Objekt heraus. Die *Oder*-Operation vereinigt die beiden Objekte. Die *Und*-Operation resultiert in Volumen, das von beiden Objekten belegt ist. Als Schreibweise verwenden Sie *A – B*, *A or B* oder *A and B*. CSG-Operationen (Constructive Solid Geometry) finden Sie in 3D-Modelling-Paketen, mit denen sich Objekte intuitiv gestalten lassen, oder dazu, um 3D-Grafik in Echtzeit aufzubauen. Wir zeigen Ihnen den spielerischen Einsatz, unterstützt von interessanten Textur-Effekten.

Polygonale Techniken

Als ersten wollen wir die CSG-Operationen mit polygonalen Techniken durchführen, also die Dreiecke des einen Objekts mit den Dreiecken des anderen schneiden und die richtigen Dreiecke auswählen. Es sind die Dreiecke richtig, die als Menge der Ausgangs- und durch Schnittberechnungen entstandenen Dreiecke entstehen. Diese resultieren aus der CSG-Operation. Ergebnisse liefern verschiedene Ansätze. Für 3D-Modelling-Programme gibt es zwei bekannte und gute Referenzen.

- Der erste Ansatz verwendet die so genannten BSP-Trees (Binary Space Partitioning). Er wird in *Set Operations on Polyhedra Using Binary Space Partitioning Trees* von Thibault und Naylor beschrieben.
- Die zweite Technik stellt das Arbeitspapier *Constructive Solid Geometry for Polyhedral Objects* vor. Beide Dokumente finden Sie im Internet, wobei http://citeseer.com/ eine sehr gute Ausgangslage bietet. Auch das Standardwerk der Computergrafik (*Computer Graphics Principles and Practice*) widmet sich ausführlich diesen beiden Techniken. Wenn Sie allerdings nur das Rendering der CSG-Objekte behandeln wollen, können Sie sich sehr viel Arbeit sparen, indem Sie keine Polygonale- sondern eine Render-Technik verwenden.



DIE BEI CONSTRUCTIVE SOLID GEOMETRY möglichen Operationen in zwei Dimensionen

Stencil-Buffer-Technik

Wenn Sie einfache konvexe Primitive wie Kugeln, Quader, Kegel oder Zylinder einsetzen, lassen sich die CSG-Operationen durchführen, ohne deren Geometrie bearbeiten zu müssen. Es handelt sich dabei um ein Rendering-Verfahren mit Tricks. Es bedient sich der Stencil Buffers.

Stencil Buffering (vgl. PC Underground in Heft 5/02 und 6/02) setzen Sie ein, um Spiegelungen und Schatten in 3D-Szenen zu rendern. In der Tat ist das im Folgenden vorgestellte Verfahren verwandt mit den Stencil-Buffer-Schatten.

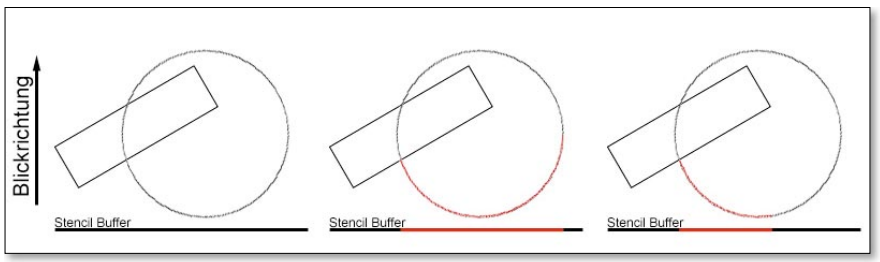
Mit einem Stencil Buffer steuern Sie das Rendering auf Pixelbasis, also für einzelne Pixel. Sie können das Setzen eines Pixels, abhängig vom Ergebnis von bestimmten Vergleichsoperationen, gestatten oder verbieten. Zusätzlich zum Frame- (bzw. Color-)Buffer und zum Z-Buffer für Farb-/Alpha- und Tiefeninformation, die Sie zumeist beim Rendering verwenden, können Grafikkarten den Stencil Buffer zur Verfügung stellen. Dieser besitzt die gleiche Auflösung, also die gleiche Anzahl von Pixeln wie die andern beiden Buffer. Die Bit-Tiefe des Stencil Buffer ist nicht fest vorgegeben.

Unterschiedliche Hardware stellt unterschiedliche Bitbreiten zur Verfügung. Je nach Anwendung benötigen Sie mehr oder weniger Bits pro Pixel. Sie können mindestens ein Bit oder auch acht Bit Stencil Buffer anfordern. Für den Stencil Buffer benötigen Sie keine speziellen OpenGL-Erweiterungen (Extensions), weil ihn fast jede 3D-Hardware zur Verfügung stellt.

In Ihrer Render-Pipeline können Sie den Stencil Buffer löschen und beschreiben. Zunächst müssen Sie OpenGL aber mitteilen, dass Sie einen Stencil Buffer ▶

verwenden möchten. Das erledigen Sie dadurch, dass Sie das gewünschte Pixelformat beschreiben, während Sie den Render-Kontext erzeugen. Ansonsten unterscheidet sich die Initialisierung nicht von der herkömmlichen. Den zugehörigen Sourcecode finden Sie auf der Heft CD. Zusätzlich experimentieren Sie mit dem vollständigen Programm.

Stencil Buffers löschen Sie, wie die anderen Buffers mit dem folgenden OpenGL-Befehl:



STENCIL-BUFFER-OPERATIONEN für die Subtraktion und die Schnittmenge zweier Objekte

```
glClear
 ( GL_STENCIL_BUFFER_BIT );
```

Stencil Buffers beschreiben Sie mit Rendering-Optionen, etwa denen für Polygone. Sie aktivieren den Aufruf mit

```
glEnable
 ( GL_STENCIL_TEST );
```

Jetzt führt OpenGL beim Setzen jedes Pixels eine Vergleichsoperation durch, die Sie mit dem Befehl

```
glStencilFunc(...)
```

festlegen.

Der erste Parameter gibt die Vergleichsfunktion an. Diese legt fest, ob der Wert im Stencil Buffer kleiner (gleich), größer (gleich) oder gleich einem Referenzwert (der zweite Parameter) sein muss, um ein positives Testresultat zu erhalten. Sie können auch festlegen, dass der Test immer ein positives oder negatives Ergebnis liefert, wenn Sie den Stencil Buffer mit bestimmten Werten füllen wollen.

Mit dem dritten Parameter übergeben Sie eine Bitmaske. Für den Wert aus dem Stencil Buffer und dem Referenzwert führt OpenGL vor dem Vergleich eine bitweise *AND*-Operation durch.

Mit der OpenGL Funktion *glStencilOp(...)* bestimmen Sie, was nach dem Stencil-Test passieren soll: Setzt das Ergebnis ein Pixel oder nicht, verändert es den Stencil-Buffer-Wert oder nicht? Sie können unterschiedliche Operationen für drei denkbare Fälle angeben, die dadurch entstehen, dass auch das Ergebnis des Z-Buffer-Tests relevant ist:

- Der Stencil-Buffer-Test liefert ein negatives Ergebnis.
- Der Stencil-Buffer-Test liefert ein positives Ergebnis, aber der Z-Buffer-Test ein negatives.
- Beide Tests liefern ein positives Ergebnis.

Beachten Sie dabei, dass der Z-Buffer-Test immer als positiv angenommen wird, wenn das Z-Buffering deaktiviert ist.

Für jeden dieser drei Fälle gibt der *glStencilOp(...)*-Befehl an, welche Operation durchgeführt werden soll. Diese Operationen sind: Stencil-Buffer-Wert unverändert lassen, Wert auf Null setzen, auf den Referenz Wert (der *glStencilFunc(...)*-Funktion) setzen, erhöhen, erniedrigen oder bitweise invertieren. Die Steuerung der Stencil-Buffer-Operationen gestattet viele verschiedene Spezialeffekte.

CSG-Operationen mit Stencil Buffers

Bei CSG-Operationen handelt es sich nur um einen Rendering-Trick. Das Dreiecksnetz, welches das Ergebnis einer geometrisch durchgeführten CSG-Operation wäre, wird nicht erzeugt. Sie verwenden den Z-Buffer und den Stencil Buffer, um Teile der Ausgangsprimitive zu rendern oder wegzuschneiden.

Die geometrischen Primitive im Beispielprogramm erzeugen Sie mit den OpenGL-Befehlen für Quadriken. Das Beispiel zeigt, wie Sie eine Display-Liste für eine texturierte Kugel generieren:

```
GLUquadric *sphere;
GLuint     sphereList;

GLfloat    mat[] =
  { 0.0f, 0.5f, 0.0f, 1.0f };

sphereList = glGenLists( 1 );
glNewList
 ( sphereList, GL_COMPILE );

sphere = gluNewQuadric();
gluQuadricTexture
 ( sphere, GL_TRUE );
glMaterialfv(
  GL_FRONT_AND_BACK,
  GL_AMBIENT_AND_DIFFUSE, mat );
gluSphere
 ( sphere, 20.0f, 64, 64 );

glEndList();
```

Für die spätere Verwendung kapseln Sie den Aufruf zum Rendern einer Displayliste in eine Funktion:

```
void drawSphere()
{
  glPushMatrix();
  // event. Transformationen
  ...
  glCallList( sphereList );
  glPopMatrix();
}
```

Die *Oder*-Verknüpfung bzw. die Vereinigung können Sie einfach rendern. Dazu benötigen Sie lediglich den Z-Buffer. Die zwei Parameter der Funktion sind Zeiger auf weitere Funktionen, die jeweils das Rendern eines der geometrischen Primitive gekapselt haben:

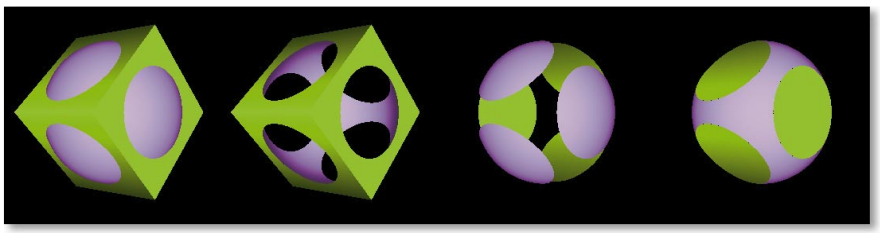
```
void  renderUnion(
   void (*A)(), void (*B)() )
{
  glEnable( GL_DEPTH_TEST );
  A();
  B();
}
```

Für die Subtraktion und die *Und*-Verknüpfung definieren Sie zwei Hilfsfunktionen, um den Überblick über die einzelnen Render-Schritte zu behalten. Die erste Funktion dient dazu, ein Objekt zu rendern und dabei die Z-Buffer-Werte zu schreiben. Sie deaktivieren dabei den Z-Buffer-Test, den Stencil-Test und das Schreiben der Farbwerte in den Framebuffer:

```
void fixZBuffer
 ( void (*A)() )
{
```

GLSTENCILFUNC – STENCIL-BUFFER-VERGLEICHE

Funktion	Resultat
GL_NEVER	immer negativ
GL_LESS	positiv, wenn *(ref & mask) ‹ (stencil & mask).*
GL_LEQUAL	positiv, wenn *(ref & mask) ? (stencil & mask).*
GL_GREATER	positiv, wenn *(ref & mask) › (stencil & mask).*
GL_GEQUAL	positiv, wenn *(ref & mask) ? (stencil & mask).*
GL_EQUAL	positiv, wenn *(ref & mask) = (stencil & mask).*
GL_NOTEQUAL	positiv, wenn *(ref & mask) != (stencil & mask).*
GL_ALWAYS	immer positiva



UNSER BEISPIELPROGRAMM zeigt die drei CSG-Operationen.

```
    glColorMask
     ( GL_FALSE, GL_FALSE,
     GL_FALSE, GL_FALSE );
    glEnable( GL_DEPTH_TEST );
    glDisable( GL_STENCIL_TEST );
    glDepthFunc( GL_ALWAYS );
    A();
    glDepthFunc( GL_LESS );
}
```

Die zweite Funktion ist das Herzstück der CSG-Operationen. Hiermit rendern Sie den Teil des Objekts *A*, der sich innerhalb des Objekts *B* befindet. Um die Funktion flexibel einsetzen zu können, geben weitere Parameter an, ob die Innen- oder Außenseiten von *A* gerendert werden, und wie der abschließende Stencil-Test durchgeführt werden soll.

Als erstes rendern Sie die gewünschte Seite von *A*, ohne Stencil-Test und ohne den Framebuffer zu beschreiben:

```
void AinsideB
   (void(*A)(),void(*B)(),
  GLenum cullFace, GLenum test )
{
  glEnable( GL_DEPTH_TEST );
  glColorMask
   ( GL_FALSE, GL_FALSE,
     GL_FALSE, GL_FALSE );
  glCullFace( cullFace );
  A();
```

Anschließend markieren Sie die Teile des Bildes (bzw. des Stencil Buffers), an denen sich ein Teil des Objekts *A* innerhalb des Objekts *B* befindet. Dazu rendern Sie die Vorderseite von *B*, mit dem Z-Buffer-Test, ohne den Frame- oder Z-Buffer zu beschreiben. Dabei inkrementieren Sie den Stencil-Wert jedes Pixels:

```
 glDepthMask( GL_FALSE
);
  glEnable( GL_STEN-
CIL_TEST );
  glStencilFunc
   ( GL_ALWAYS, 0, 0 );
  glStencilOp(
    GL_KEEP, GL_KEEP,
GL_INCR );
  glCullFace( GL_BACK
);
  B();
```

Umgekehrt dekrementieren Sie die Stencil-Buffer-Werte dort, wo auch die Rückseite von *B* den Z-Buffer-Test besteht:

```
    glStencilOp(
      GL_KEEP, GL_KEEP, GL_DECR );
    glCullFace( GL_FRONT );
    B();
```

Diese Operationen sehen Sie im Bild an einem zweidimensionalen Beispiel: die zwei Objekte (im Z-Buffer) vor der Stencil-Operation nach dem In- und nach dem Dekrementieren.

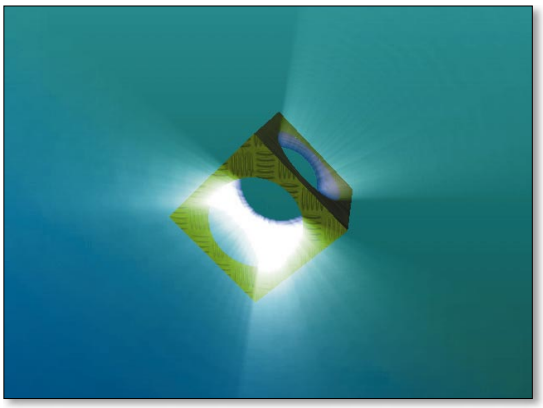
Im letzten Schritt unserer Funktion zeichnen Sie den sichtbaren Teil des Objekts *A* (abhängig vom Stencil-Test) in den Framebuffer:

```
    glDepthMask( GL_TRUE );
   glColorMask( GL_TRUE, GL_TRUE,
  GL_TRUE, GL_TRUE );

    glDisable( GL_DEPTH_TEST );
    glStencilFunc( test, 0, 1 );
    glCullFace( cullFace );
    A();
  }
```

Mit diesen beiden Routinen gestalten sich die CSG-Operationen übersichtlicher, trotzdem benötigen Sie Vorstellungskraft, um die Schritte nachzuvollziehen. Am Besten Sie fertigen Hand-Skizzen an, in denen Sie die Einzelschritte einzeichnen. Für die *Und*-Verknüpfung, also die CSG-Schnittoperation, rufen Sie folgende Funktionen auf:

```
void  renderIntersection(
     void (*A)(), void (*B)() )
{
  AinsideB
   (A,B,GL_BACK,GL_NOTEQUAL);
  fixZBuffer( B );
  AinsideB
   (B,A,GL_BACK,GL_NOTEQUAL);
  glDisable( GL_STENCIL_TEST );
}
```



DER RADIAL BLUR EFFEKT mit Luminanz

Im ersten Schritt zeichnen Sie den Teil der Vorderseite des Objekts *A*, der sich innerhalb des Objekts *B* befindet. Den Teilbereich bestimmen Sie durch die Paramter *GL_BACK* und *GL_NOTEQUAL*. Als nächstes fixieren Sie die Tiefeninformation auf das *B*-Objekt. Dadurch können Sie jetzt den Teil der *B*-Vorderseite rendern, der sich in *A* befindet.

Bei der CSG-Substraktion verwenden Sie leicht abgeänderte Parameter:

```
void renderSubstraction(
      void (*A)(), void (*B)() )
{
  AinsideB
   (A,B,GL_FRONT,GL_NOTEQUAL);
  fixZBuffer(B);
 AinsideB(B,A,GL_BACK,GL_EQUAL);
  glDisable( GL_STENCIL_TEST );
}
```

Der Unterschied zur *Und*-Verknüpfung liegt beim zweiten Teil des Renderings. Hier wird nicht der Teil von *B* gerendert, der sich innerhalb vom Objekt *A* befindet, sondern genau das Gegenteil: durch den *GL_EQUAL*-Test werden die nicht im Inneren liegenden Teile gerendert.

Im Bild links oben sehen Sie Screenshots von unserem Beispielprogramm, die die Vereinigung, Schnittmenge und Subtraktion eines Würfels und einer Kugel zeigen.

DER RADIAL-BLUR-EFFEKT mit RGB-Farbwerten

Radial-Blur-Effekte

Im zweiten Teil dieses Artikels wollen wir Ihnen einen Textur-Effekt vorstellen, der aus technischer Sicht nichts mit den CSG-Operationen oder dem Stencil Buffering zu tun hat. Allerdings lassen sich in Verbindung mit den CSG-Körpern interessante Effekte erzeugen, wie Sie im Bild oben können.

Der Effekt basiert darauf, Teile des Bildes von der Mitte nach außen zu ziehen und zu verwaschen. Effekte dieser Art nennt man auch Radial-Blur-Effekte. *Radial*, weil die Vergrößerung des Bildteile (nach außen ziehen) kreisförmig um die Mitte stattfindet, und *Blur*, weil die weiter außen liegenderen Teile verwaschen erscheinen.

Für diesen Effekt benötigen Sie zunächst eine OpenGL-Textur, die Sie während der Initialisierungsphase des Programms anlegen. Damit legen Sie die Filter für die Textur-Skalierung fest. Mit einer Konstante bestimmen Sie die Größe der Textur für den späteren Gebrauch.

```
#define BLURSIZE 256
GLuint blurTexture;

glGenTextures( 1, &blurTexture);
glBindTexture( GL_TEXTURE_2D,
        blurTexture );
glTexParameteri( GL_TEXTURE_2D,
  GL_TEXTURE_MIN_FILTER,
  GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D,
  GL_TEXTURE_MAG_FILTER,
  GL_LINEAR );
```

Vor dem Rendering des eigentlichen Bildes bzw. des CSG-Objekts rendern Sie die Szene in dieser Blur-Textur. Diesen Vorgang fassen Sie am Besten in einer Funktion zusammen:

```
void render2Texture
   ( GLenum format )
{
  // Viewport: Texgröße +löschen
  glViewport
  (0,0,BLURSIZE,BLURSIZE);

  glClearColor
  (0.0f,0.0f,0.0f,0.5f);
  glClear( GL_COLOR_BUFFER_BIT |
           GL_DEPTH_BUFFER_BIT |
         GL_STENCIL_BUFFER_BIT);

  renderCSG();

  // Texture kopieren
  glBindTexture( GL_TEXTURE_2D,
    blurTexture );
 glCopyTexImage2D(GL_TEXTURE_2D,
    0, format, 0, 0,
    BLURSIZE, BLURSIZE, 0 );

 // Viewport wieder restaurieren
  glClearColor
  (0.0f,0.0f,0.5f,0.5f);
  glClear( GL_COLOR_BUFFER_BIT |
           GL_DEPTH_BUFFER_BIT |
         GL_STENCIL_BUFFER_BIT);
  extern int windowX, windowY;
 glViewport(0,0,windowX,windowY);
 }
```

Für den *format*-Parameter wählen Sie entweder *GL_LUMINANCE* oder *GL_RGB*, je nachdem, welche Effekte Sie bei den vorigen beiden Bildern erhalten wollen.

In Ihrer Render-Pipeline erzeugen Sie die Blur-Textur, zeichnen die 3D-Szene und rendern den Radial-Blur-Effekt mit der folgenden Funktion, die wir Ihnen als Ausschnitt zeigen. Der weggelassene Teil deaktiviert den Z-Buffer und die Beleuchtungsberechnung und initialisiert die OpenGL-Matrizen, um eine orthogonale Abbildung zu bekommen.

Die Blur-Textur wird n-mal über das Bild gezeichnet, wobei die Farb- oder die Luminanz-Werte jedesmal leicht auf das Bild addiert werden. Die Intensität bestimmen Sie über den Alphawert. Die Stärke des Zooms (wie die Textur in jedem Schritt vergrößert wird) geben Sie mit dem zweiten Parameter an:

```
void renderBlur
  ( int n, float delta )
{
  float texZoom = 0.0f;
  float alpha   = 0.15f;
  glDisable( GL_DEPTH_TEST );
  glEnable( GL_BLEND );
  glBlendFunc
  ( GL_SRC_ALPHA, GL_ONE );
  glBindTexture( GL_TEXTURE_2D,
                 blurTexture );
  // ... Matrizen & Co ...
  glBegin( GL_QUADS );
  for ( int i = 0; i < n; i++ )
  {
   glColor4f( 1, 1, 1, alpha );
   glTexCoord2f
   (0+texZoom,1-texZoom);
   glVertex2f( -1, 1 );
   glTexCoord2f
   (0+texZoom,0+texZoom);
   glVertex2f( -1, -1 );
   glTexCoord2f
   (1-texZoom,0+texZoom);
   glVertex2f( 1, -1 );
   glTexCoord2f
   (1-texZoom,1-texZoom);
   glVertex2f( 1, 1 );
   texZoom += delta;
   alpha -= 0.15f / (float)n;
  }
  glEnd();
}
```

Sie erhalten schöne Resultate, wenn Sie die Textur 50-mal vergrößern und zeichnen. Als Delta-Wert hat sich *0.01* bewährt. Beachten Sie, dass die Grafikkarte den Framebuffer 50-mal beschreiben muss. Das belastet die Hardware. ET

STENCIL-BUFFER-OPERATIONEN

Operation	Stencil-Buffer-Werte
GL_KEEP	unverändert
GL_ZERO	auf Null setzen
GL_REPLACE	auf Referenzwert setzen
GL_INCR	erhöhen, mit Sättigung
GL_DECR	niedriger, nicht kleiner Null
GL_INVERT	bitweise invertieren

Literatur:

James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: *Computer Graphics Principles and Practice*, Addison Wesley Professional 1996, 1200 Seiten, 75 US-Dollar, ISBN 0-201-84840-6