OpenGL
Modelierung und Beleuchtung

Proseminar Computergrafik

René Lützner

Übersicht

- Einleitung
- Modellierung
- Beleuchtung
- Quellen

Einleitung

- Begriffserklärung
- Historische Entwicklung
- Erweiterbarkeit
- OpenGl als Zustandsautomat
- OpenGL Rendering Pipeline
- OpenGL Syntax

Begriffserklärung

- **OpenGL** (**Open G**raphics **L**ibrary) bezeichnet eine plattform- und programmiersprachenunabhängige API (Application Programming Interface).
- **GLU** (Open**GL U**tility Libary) ist eine Funtionsbibliothek, die auf Open GL aufsetzt.
- **Rendern** bezeichnet den Proßess bei dem der Rechner, dass durch geometrische Primitiven beschriebene Objekt in ein Pixel-Bild umwandelt, welches dann auf dem Bildschirm angezeigt wird.

Historische Entwicklung

- OpenGL entstand ursprünglich aus dem von Silicon Graphics (SGI) entwickelten Iris GL.
- aufgrund seiner Plattformunabhängigkeit ist OpenGL im professionellen Bereich als 3D-Standard nach wie vor führend. Im Bereich der Computerspiele wurde es jedoch in den letzten Jahren zunehmend von Microsofts DirectX 3D verdrängt.

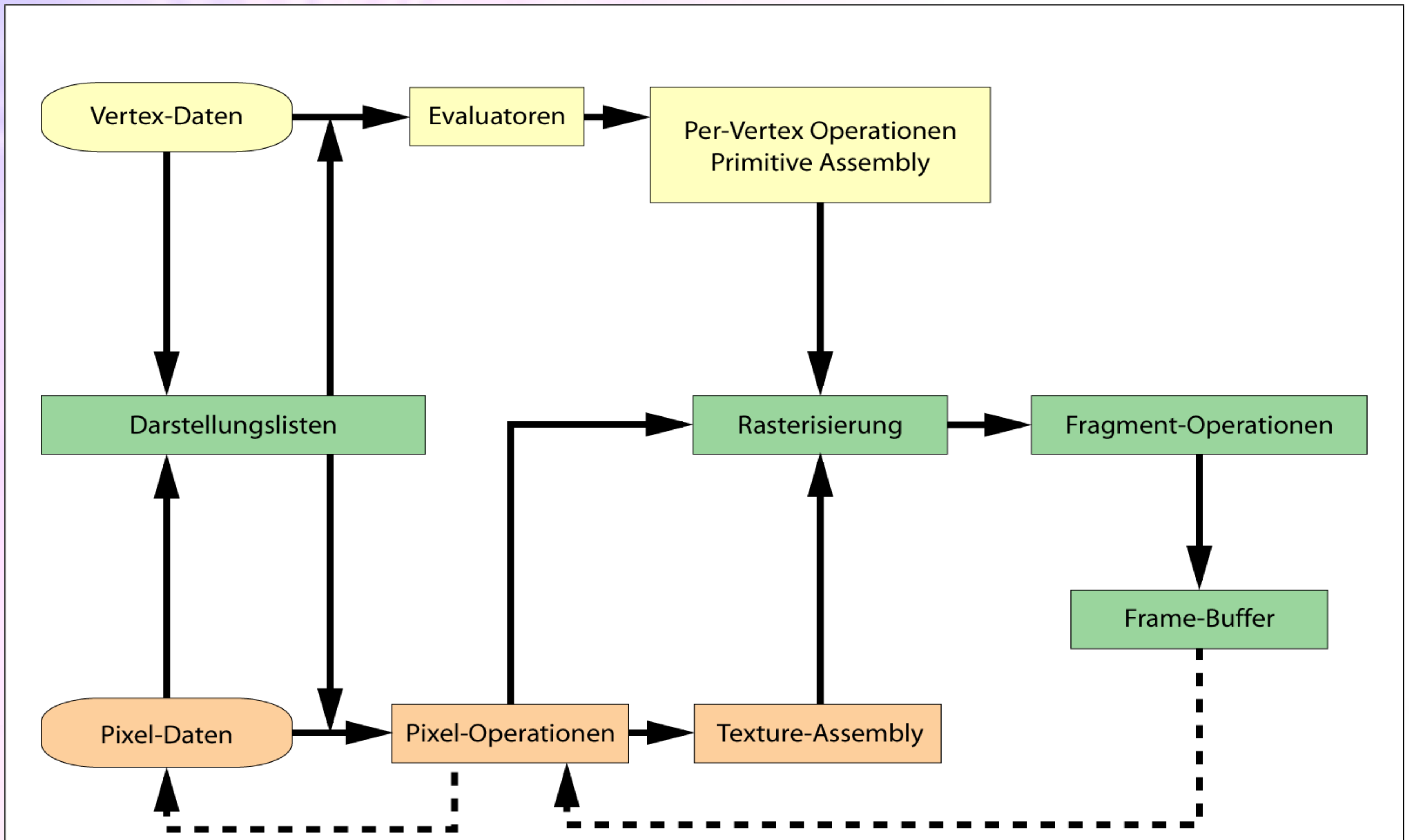
Erweiterbarkeit

- Ablauf bei der Einführung einer Erweiterung:
  - ein einzelne Anbieter (typischerweise Grafikkartenhersteller) erweitert die Zustandsmaschine von OpenGL
  - Ausliefern der C-Headerdatei, deren Funktionsnamen und Konstanten erhalten einen herstellerspezif. Postfix
  - Einigung mehrerer Hersteller über die Benutzung der Erweiterung (Postfix EXT)
  - Einigung des ARB (Architecture Review Board) zur Standardisierung der Erweiterung (Postfix ARB)
  - Aufnahme in den Core

OpenGL als Zustandsautomat

- Die Darstellung von gerenderten Objekten ist von vielen Parametern abhänging z.B. vom Licht, Texturen und Oberflächenbeschaffenheiten.
- OpenGL wurde als Zustandsautomat entworfen, d.h. dass nicht bei jedem Funktionsaufruf alle Parameter übergeben werden.
- Mit Hilfe dieser ZustandsVariablen kann man eine aufwändige Reorganisationen der Graphikpipeline solange wie möglich vermeiden.

Rendering Pipeline



OpenGL Syntax

- OpenGL Befehle nutzen einen Präfix **gl** und alle Befehle fangen nach diesem Präfix mit einem Großbuchstaben an.
- Für Konstanten gilt, dass sie mit GL_ anfangen und nur Großbuchstaben und Unterstriche benutzt werden.
- Auch am Ende von einigen Befehlen treten Suffixe auf, die z.B. die Zahl der Parameter angeben.
- OpenGL bietet auch eigene Typen an.

OpenGL Syntax

- Bsp glColor:
  - ohne Pointer:
    - `glColor3f(1.0, 0.0, 0.0);`
  - mit Pointer:
    - `GLfloat color_array[] = {1.0, 0.0, 0.0};`
    - `glColor3fv(color_array);`
- Statt der spezifischen Funktionsnamen lässt sich folgender Syntax verwenden glColor*() der für alle Variationen dieser Funktion steht.

Modellierung

- Erzwingen des vollständigen Zeichnens
- Punkte, Linie und Polygone
- OpenGL Primitiven
- Normalen Vektoren
- Beispiele (Rechteck, Ikosaeder)
- Quadrics
- Bezier-Kurven

Erzwingen des vollst. Zeichnens

- Die meisten modernen Graphiksysteme kann man sich als eine zusammengesetzte Abarbeitungs-Linie vorstellen.
- Die CPU gibt einen Zeichnen-Befehl aus und andere noch vorhandene Hardware übernimmt die folgenden Bearbeitungsschritte.
- In High-End Architekturen werden diese Schritte alle von unterschiedlichen Hardwarekomponenten erledigt, somit gibt es keinen Grund für die CPU darauf zu warten, bis der Zeichnenbefehl durchgeführt wurde.
- Um Performanzverluste zu vermeiden, werden Packete übergeben statt nur eines einzelnen Vertexes.
- Mit glFlush() erfolgt ein sofortiges Absenden des Packetes.

Punkte, Linien und Polygone

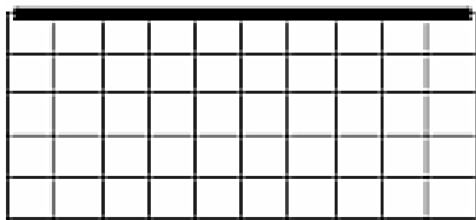
- Punkte
  - Ein Punkt wird durch eine Menge von Floating-Point-Zahlen representiert den man Vertex nennt. Alle internen Berechnungen werden so vorgenommen als wären diese Vertices dreidimensional.
- Linien
  - In OpenGL bezieht sich die Bezeichnung Linie auf ein Liniensegment, da es einen Anfang und ein Ende gibt und sie somit nicht unendlich lang ist.

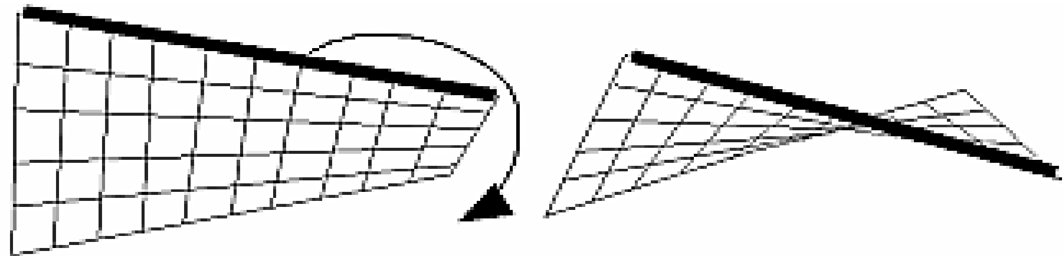
Punkte, Linien und Polygone

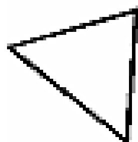
- Polygone
  - Polygone sind abgeschlossene Bereiche, die von einzelnen Liniensegment-Schleifen umschlossen sind.
  - Bezüglich der Komplexität gibt es einige Beschränkungen von OpenGL, es muss sich um primitive Polygone handeln.
  - In der realen Welt gibt es jedoch nicht konvexe Polygone, diese lassen sich aber aus einfachen Polygonen zusammensetzten.

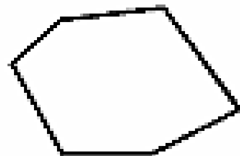
Punkte, Linien und Polygone

- Da OpenGL Vertices immer dreidimensional sind gibt es außer bei Dreiecken keinen Garant dafür, dass alle Punkte die das Polygon formen in einer Ebene liegen.

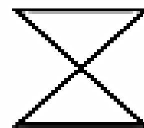


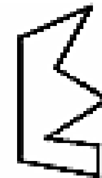






Valid

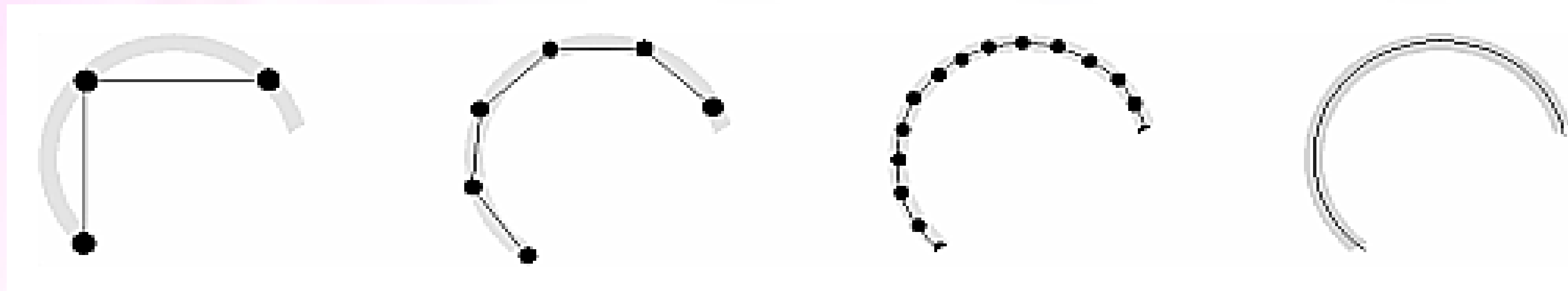




Invalid

Punkte, Linien und Polygone

- Rechtecke
  - OpenGL unterstützt die Erstellung von gefüllten Rechtecken durch glRect*().
- Kurven
  - Durch eine Folge von Liniensegmenten ist eine geschwungenen Kurvenform beliebiger Genauigkeit aproximierbar.



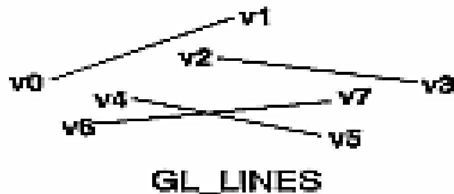
OpenGL Zeichnen von Primitiven

- Um ein geometrisches Objekt aus mehreren Vertices aufzubauen müssen diese Vertices innerhalb von glBegin() und glEnd() angeben werden .

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(4.0, 3.0);
    glVertex2f(6.0, 1.5);
    glVertex2f(4.0, 0.0);
glEnd();
```

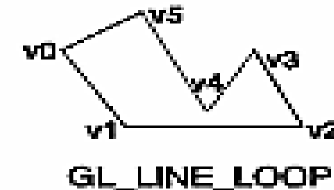
OpenGL Zeichnen von Primitiven

v4
v0 v3
v1 v2
GL_POINTS

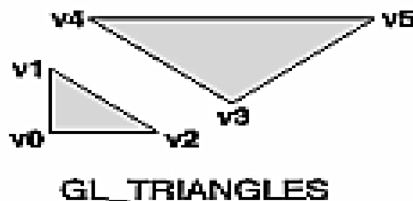


GL_LINES

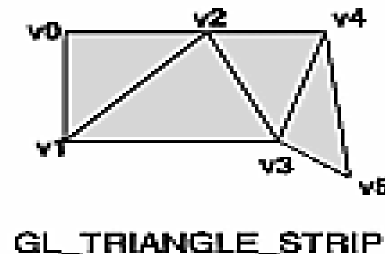
GL_LINE_STRIP



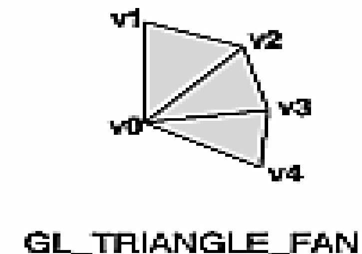
GL_LINE_LOOP



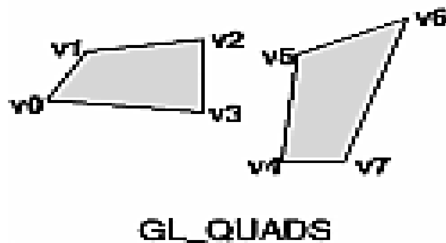
GL_TRIANGLES



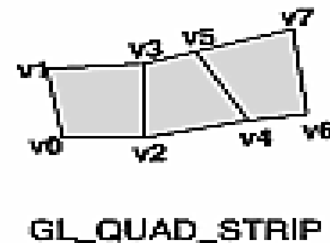
GL_TRIANGLE_STRIP



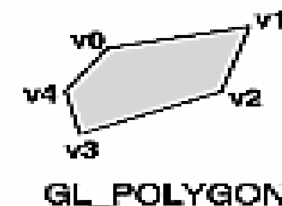
GL_TRIANGLE_FAN



GL_QUADS

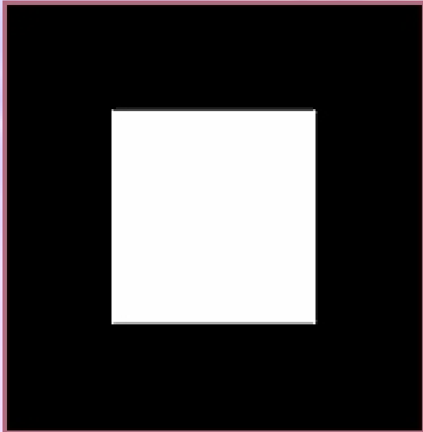


GL_QUAD_STRIP



GL_POLYGON

Rechteck

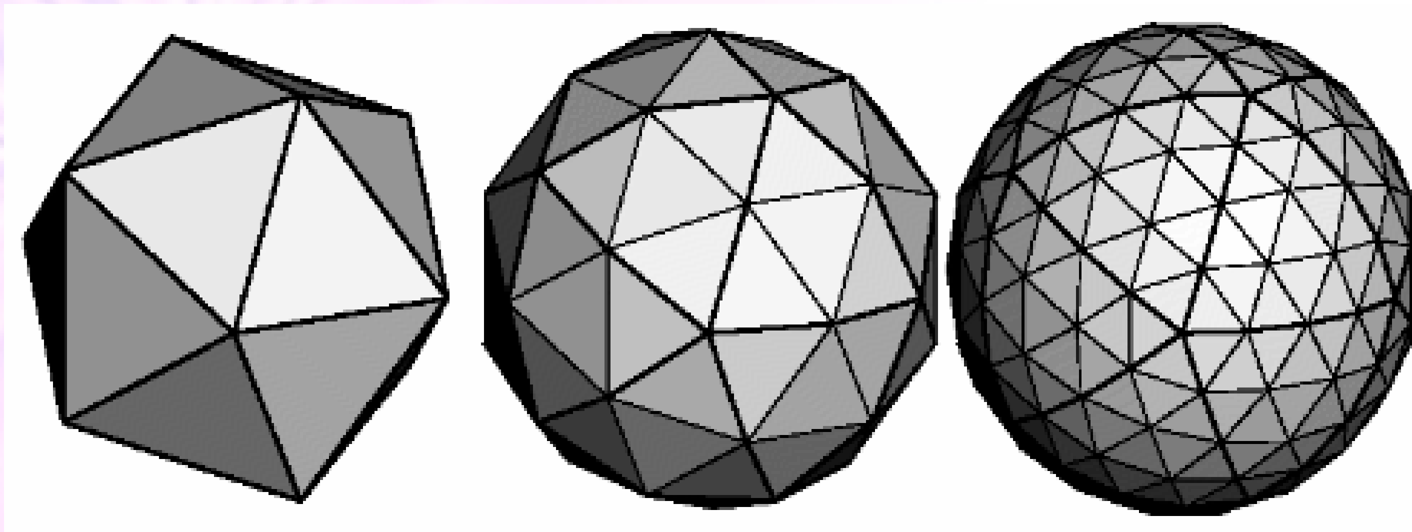


```
Main()

{

   InitializeAWindowPlease()

   GlClearColor (0.0, 0.0, 0.0, 0.0);
   glClear (GL_COLOR_BUFFER_BIT);

   glColor3f (1.0, 1.0, 1.0);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);

   glBegin(GL_POLYGON);
       glVertex3f (0.25, 0.25, 0.0);
       glVertex3f (0.75, 0.25, 0.0);
       glVertex3f (0.75, 0.75, 0.0);
       glVertex3f (0.25, 0.75, 0.0);
   GlEnd();
   glFlush();

   UpdateTheWindowAndCheckForEvents(); }
```

NormalenVektor

- Die Normalen eines Objektes legen die Orientierung der Oberfläche im Raum und damit ihre relative Orientierung zur Lichtquelle fest.
- glNormal*() wird genutzt um die Normale zu setzen, danach legt man den Vertex fest, dem dann die Normale zugewiesen wird.

Ikosaeder



- Zum Zeichen des Ikosaeders selbst kann man sich seine besonderen geometrischen Eigenschaften zu Nutze machen.

Ikosaeder

```
#define X .525731112119133606

#define Z .850650808352039932

 static GLfloat vdata[12][3] = {
     {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
     {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
     {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0} };

 static GLuint tindices[20][3] = {
    {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
    {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
    {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
    {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11} };
```

Ikosaeder

```
void subdivide(float *v1, float *v2, float *v3, long depth)
{   GLfloat v12[3], v23[3], v31[3];
    GLint i;
    if (depth == 0)
    {   drawtriangle(v1, v2, v3);
        return; }
    for (i = 0; i < 3; i++)
    {   v12[i] = v1[i]+v2[i];
        v23[i] = v2[i]+v3[i];
        v31[i] = v3[i]+v1[i]; }
     normalize(v12);
     normalize(v23);
     normalize(v31);
     subdivide(v1, v12, v31, depth-1);
     subdivide(v2, v23, v12, depth-1);
     subdivide(v3, v31, v23, depth-1);
     subdivide(v12, v23, v31, depth-1); }
```

Ikosaeder

```
for (i = 0; i < 20; i++)

{   subdivide(&vdata[tindices[i][0]][0],
              &vdata[tindices[i][1]][0],
              &vdata[tindices[i][2]][0]);}

void drawtriangle(float *v1, float *v2, float *v3)

{   glBegin(GL_TRIANGLES);

      glNormal3fv(v1); vlVertex3fv(v1);
      glNormal3fv(v2); vlVertex3fv(v2);
      glNormal3fv(v3); vlVertex3fv(v3);

  glEnd(); }
```

Quadrics

- Erstellen von Quadric Objekten
  - GLUquadricObj* gluNewQuadric (void)
  - void gluDeleteQuadric (GLUquadricObj *qobj)
  - void gluQuadricCallback
- Quadric Attribute
  - gluQuadricOrientation()
  - gluQuadricDrawStyle()
  - gluQuadricNormals()
  - gluQuadricTexture()
- Quadric Primitiven
  - gluSphere(), gluCylinder(), gluDisk(), or gluPartialDisk()

Bezier Kurven

Kontrollpolynome n-ten Grades

$$C(t)=\sum_{i=0}^{n} B_{i,n} P_n$$

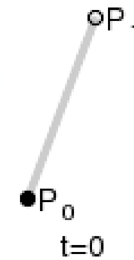
$$B_{i,n}(t)=\binom{n}{i} t^i (1-t)^{n-i}$$

$$C(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} P_{j,i} B_{i,n}(u) B_{j,m}(v)$$

Bezier Kurven

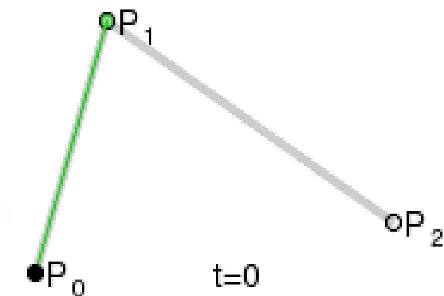
$$C_{i,n}(t)=\sum_{i=0}^{1} t^{i}(1-t)^{n-i}P_{i}$$

$$C_{i,n}(t)=(1-t)P_{0}+tP_{1}$$

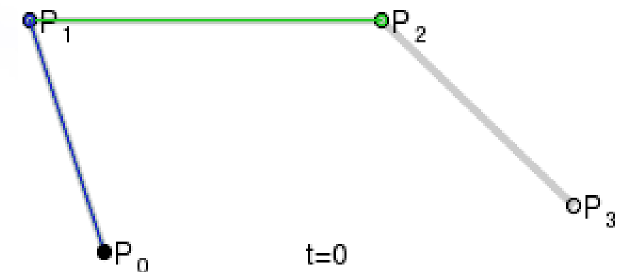


$$C_{i,n}(t)=\sum_{i=0}^{2}\binom{2}{i}t^{i}(2-t)^{n-i}P_{i}$$

$$C_{i,n}(t)=(1-t)^{2}P_{0}+2\mathrm{t}(1-t)P_{1}+t^{2}P_{2}$$



$$C_{i,n}(t)=\sum_{i=0}^{3}\binom{3}{i}t^{i}(3-t)^{n-i}P_{i}$$



Bezier Kurven

- Zum erstellen von Bezier Kurven ist es nötig vorher bei der Initialisierung mit Hilfe von glMap1() und GL_MAP1_VERTEX_3 als ersten Parameter einen Evaluator für die Vertices anzulegen.
- glEvalCoord1*() wird zum Zeichen der Bezierkurve genutzt
- Um z.B. Farben festzulegen müssen mit glMap1*() und GL_MAP1_COLOR_4 weitere Evaluators angelegt werden.

Bezier Kurve

```
init(void)

{

    GlClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpts[0][0]);
    glEnable(GL_MAP1_VERTEX_3); }

void display(void)

{

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_STRIP);

     for (int i = 0; i <= 30; i++)
        glEvalCoord1f((GLfloat) i/30.0);

    glEnd();

    glFlush(); }
```

Beleuchtung

- Hidden Surface Removal
- Reale and OpenGL Beleuchtung
- Ambientes, Diffuses, and Spekulares Licht
- Festlegen der Materialeigenschaften
- RGB Werte für Licht und Material
- Erstellen und Aktivieren einer Lichtquelle
- Position und Dämpfung

Hidden Surface Removal

- Jene Polygone die durch andere näher am Auge befindliche Objekte verdeckt sind, werden beseitig.
- Diese Beseitigung von verdeckten Objekten nennt man „Hidden-Surface-Removal“ und der einfachste Weg dies zu errreichen ist die Nutzung des Depth Buffers.
- Der Depth Buffer speichert für jedes Pixel die TiefenWerte und falls ein Pixel „zweimal“ vorkommt, so wird jenes welches weiter vom Betrachter entfernt ist überschrieben.

Reale und OpenGL Beleuchtung

- Bei einem realen Objekt hängt die wahrgenommen Farbe, von der vom Objekt zurückgeworfenen Lichtstrahlen bzw. Wellenlängen des Lichtes, die dann auf die entsprechenden Sehzellen des Auges treffen, ab.
- OpenGL approximiert Licht und Beleuchtung als etwas das sich in rote, grüne und blaue Komponenten aufsplitten lässt.
- Im Lichtmodell von OpenGL kommt Licht von mehreren Lichtquellen die individuell aus und ein geschaltet werden können.

Reale und OpenGL Beleuchtung

- In OpenGL hat eine Lichtquelle nur dann einen sichtbaren Effekt, wenn es eine Oberfläche gibt, die das Licht absorbiert und reflektiert. Jede Oberfläche verfügt über best. Materialeigenschaften.
- Das OpenGL Lichtmodel teilt die Beleuchtung in vier unabhängige Gruppen ein: emissives, ambientes, diffuses, and spekulares Licht.

Ambientes, Diffuses und Spekulares Licht

- Ambientes Licht ist das Licht, welches im ganzen Raum verstreut auftritt und dessen Richtung nicht feststellbar ist. Wenn Ambientes Licht auf eine Oberfläche trifft so wird es genauso in alle Richtungen gestreut.
- Diffuses Licht kommt von einer bestimmten Richtung, somit ist es heller wenn es genau auf eine Oberfläche trifft, als wenn es ein Objekt nur schimmernd streift. Auch dieses Licht wird beim Auftreffen auf eine Oberfläche in alle Richtungen gestreut.

Ambientes, Diffuses und Spekulares Licht

- Spekulares Licht kommt ebenfalls von einer best. Richtung und wird in eine best. Richtung zurückgeworfen. Ein Laser der auf einen hochwertigen Spiegel trifft produziert fast 100% speculare Reflexion. Glänzendes Metall oder Plastik haben einen hohen Anteil an spekularen Licht und Kreide hat nahezu keinen.

Erstellen und aktivieren einer Lichtquelle

- Lichtquellen besitzen eine Reihe von Eigenschaften, wie Farbe, Position, und Ausrichtung. Der Befehl der zum Erstellen einer Lichtquelle genutzt wird ist glLight*()
- In OpenGL müssen die Lichtquellen explizit aktiviert werden. Wenn die Beleuchtung nicht aktiviert ist werden die Farben einfach den jeweiligen Vertex zugewiesen und es finden keinerlei Lichtberechnugen statt.

Festlegen der Marterialeigenschaften

- Viele der Materialeigenschaften sind prinzipiell die selben, wie sie auch zum Erstellen der Lichtquellen genutzt werden. Der Mechanismus zum Setzen ist der gleiche mit der Außnahme das der Befehl glMaterial*() lautet.
- Diffuse Reflexion
- Ambiente Reflexion

```
GLfloat mat_amb_diff[] = { 0.1, 0.5, 0.8, 1.0 };
glMaterialfv(GL_FRONT_AND_BACK,
             GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
```

Festlegen der Marterialeigenschaften

- Spekulare Reflexion

```
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
```

- Emission

```
GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
```

RGB Werte für Licht und Material

- Die Spezifikation der Farbkomponenten für das Licht unterscheidet sich etwas von der für die Materalien.
- Bei Materialien stehen die Zahlen für die Reflexionsverhältnisse der Farben.
- (LR*MR, LG*MG, LB*MB)
- Genauso gilt für zwei Lichtquellen:
  (R1+R2, G1+G2, B1+B2).
- Falls eine der Summen größer als eins ist, so wird diese Komponente auf eins beschränkt.

Position und Dämpfung

- direktionale Lichtquelle
- positionale Lichtquelle
- `GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };`
- `glLightfv(GL_LIGHT0, GL_POSITION, light_position);`
- Wie oben zu sehen ist gibt man für die Position des Lichtes einen Vektor aus vier Variablen (x,y,z,w), sowie GL_POSITION als zu verändernde Eigenschaft an. Die standard Position für Licht0 ist (0,0,1,0), das Licht verläuft dann also entlang der negativen z Achse.

Position und Dämpfung

$$\text{attenuation factor} = \frac{1}{k_c + k_l d + k_q d^2}$$

- d = Distanz zwischen Lichtquelle und Vertex
- kc = GL_CONSTANT_ATTENUATION
- kl = GL_LINEAR_ATTENUATION
- kq = GL_QUADRATIC_ATTENUATION

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Position und Dämpfung

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Quellen

- http://www.glprogramming.com/red/
- http://nehe.gamedev.net/
- http://de.wikipedia.org/wiki/OpenGL
- http://de.wikipedia.org/wiki/B%C3%A9zierkurve
- http://www-users.rwth-aachen.de/Tobias.Weyand/OpenGL//OpenGL-Ausarbeitung.pdf
- http://graphics.cs.uni-sb.de/Courses/ws9900/cg-seminar/Ausarbeitung/Philipp.Walter/openglstandard.htm
- http://www.icare3d.org/myprojects/3d_gallery/opengl_3d_logo.html