Einführung in die API OpenGL

Überblick

- Was ist OpenGL?
- OpenGL und Windows
- 3D Grafik Grundlagen
- OpenGL benutzen
  - Transformationen, Licht, Texture Mapping
- Ein kleines Beispielprogramm
- Fortgeschrittene Themen
- Anregungen zur Diskussion

Was ist OpenGL?

- 3D Grafik API
  - Schnittstelle zur 3D Programmierung
  - Unterstützung für Hardwarebeschleunigung
  - Unabhängig vom Betriebssystem
  - Low-level API
- 1992 von SGI aus IrisGL hervorgegangen
- Offener Standard, Architecture Review Board
- Erweiterbar mittels Extensions
- Nur Grafik, keine Unterstützung für Input, Sound etc.
- Ursprünglich für C geschrieben, aber auch andere Sprachen werden unterstützt (Delphi, Java, etc.)

Was ist OpenGL?

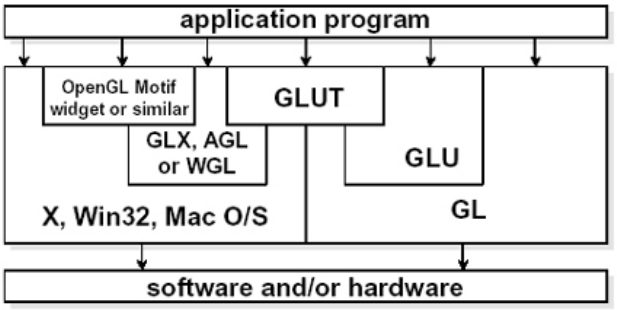
- Vorteile
  - Sehr weit verbreitet, vor allem in der Industrie
  - Unterstützt von allen führenden Herstellern
  - Portabel
  - Relativ leicht zu erlernen
  - Sehr performant
- Nachteile
  - Nur Grafik, im Gegensatz zu DirectX
  - Standard erst bei Version 1.5, trotz rasender Hardwareentwicklung
  - Extensions sehr unübersichtlich

OpenGL und Windows

- Das erste mal unter Windows NT 3.5 implentiert.
- OpenGL nutzt Fensterfunktionen des jeweiligen Betriebssystems
- Es gibt verschiedene Anbindungen
  - Windows – WGL
  - Linux – GLX
  - Apple Macintosh - AGL
- Alternative: GL Utility Toolkit (GLUT)
  - Portabel und sehr kurz
  - Unflexibel

OpenGL und Windows

OpenGL and Related APIs



Vorbereitungen

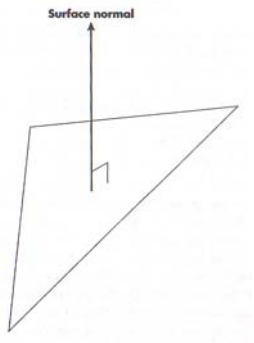
- Header Dateien einbinden
  - #include <GL/gl.h>
  - #include <GL/glu.h>
- Libraries im Compiler linken
  - Für Windows: openGL32.lib GLu32.lib
- OpenGL führt neue Datentypen ein
  - GLfloat, GLint, GLenum, etc.
  - Sinn: Einfacher zu portieren

3D Grundlagen

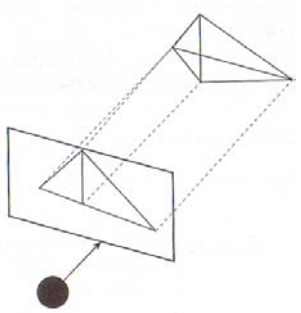
- 3D Grafik kann sehr mathematisch sein
  ⇨ Lineare Algebra ist nützlich(!)
- Affiner Vektorraum R^3
- Transformationen mittels Matrizen
- Normalisierung von Vektoren
- Skalarprodukt zur Winkelberechnung
- Berechnung des Normalenvektors mit dem Kreuzprodukt C=AxB
- Hintereinanderausführung von Transformationen mittels Matrixmultiplikation

Normalen-Vektor und Projektion

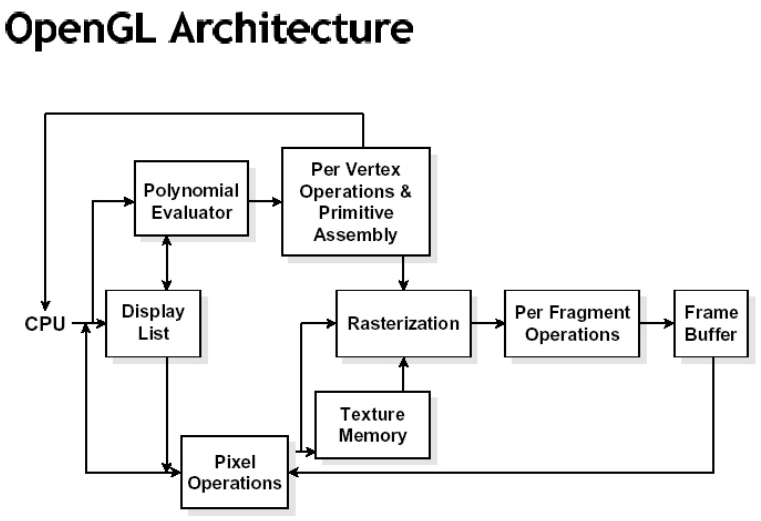
- Die Normale ist der orthogonale Vektor einer Fläche



- Projektion 3D -> 2D

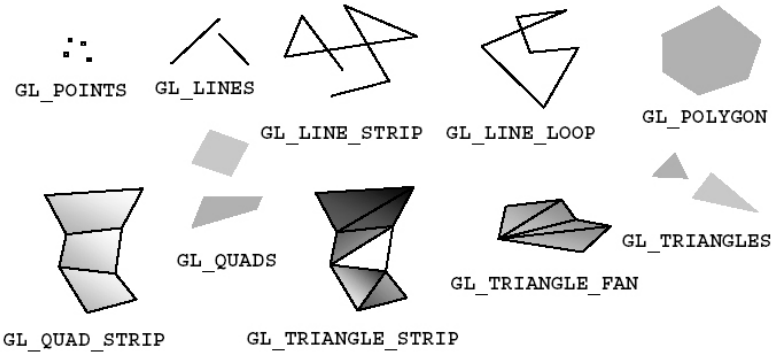


OpenGL Architektur



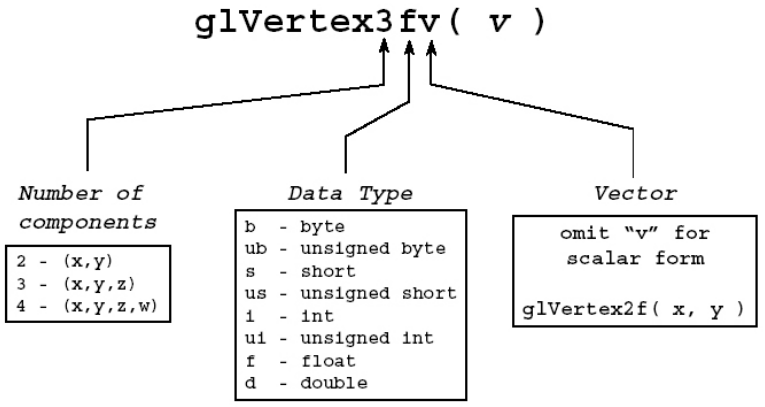
OpenGL Primitive

- Alle geometrischen Primitive sind durch Vertices definiert



OpenGL Kommando-Formate

- OpenGL Anweisungen folgen einem bestimmten Aufbau



Primitive spezifizieren

- Primitive werden durch folgende Anweisung erstellt

```
glBegin( primType);
  glVertex3f(1.0f,1.0f,1.0f);
  glVertex3f(…);
  …
glEnd();
```

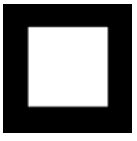
Typen:	
GL_POINTS	GL_LINESTRIP
GL_LINES	GL_LINE_LOOP
GL_POLYGON	GL_TRIANGLE_STRIP
GL_TRIANGLES	GL_TRIANGLE_FAN
GL_QUADS	GL_QUAD_STRIP

OpenGL benutzen

- Schema eines OpenGL Programms

```
void initGL() {
  glShadeModel(GL_SMOOTH);
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  …
}
void CreateGLWindow() {
  …
}
void render() {
 …
}
int WINAPI WinMain(…) {
  //main message loop
  while (!done) {
    dispatchMessages();
    render();
}}
```

```
void render() {
  glClear( GL_COLOR_BUFFER_BIT |
           GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glTranslatef(0.0f,0.0f,-6.0f);
  glBegin(GL_QUADS);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(1.0f,1.0f,-1.0f);
    glVertex3f(1.0f,-1.0f,-1.0f);
  glEnd();
  glpopMatrix();
}
```



Die State Machine

Tobias Lang

- Alle Rendering Attribute sind im OpenGL State gekapselt
  - Rendering Style
  - Shading
  - Lighting
  - Texture Mapping
- Jedesmal wenn ein Vertex verarbeitet wird, bestimmen interne Statustabellen, seine Eigenschaften

State Machine manipulieren

Tobias Lang

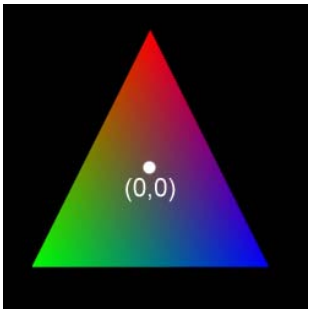
- Der aktuelle Status bestimmt die Erscheinung

```
for each ( primitive to render ) {
  update OpenGL state;
  render primitive;
}
```

- Häufigste Art den Status zu manipulieren
  - glColor3f()
  - glIndex3f()
  - glNormal*
  - glTexCoord*()
- Prinzip: Gewünschten Status einstellen, Geometriedaten übergeben, u.s.w.

Beispiel

```
GLvoid render() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glShadeModel(GL_ SMOOTH);
  glEnable(GL_ LIGHTING);
  glBegin(GL_TRIANGLES);

    glColor3f(1.0f,0.0f,0.0f);     //red
    glVertex3f(0.0f,1.0f,0.0f);

    glColor3f(0.0f,1.0f,0.0f);     //green
    glVertex3f(-1.0f, -1.0f,0.0f);

    glColor3f(0.0f,0.0f,1.0f);     //blue
    glVertex3f(1.0f,-1.0f,0.0f);

  glEnd();
}
```

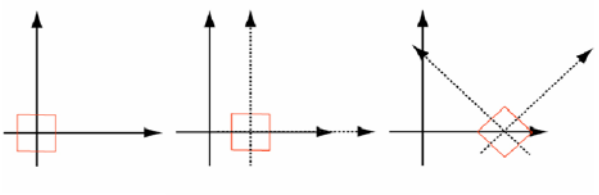


Transformationen

Transformationen in OpenGL

- Transformationen erlauben es uns Gegenstände zu bewegen, zu drehen und zu manipulieren
- Weitere Anwendung: Projektion vom 3D Raum in den 2D Raum des Bildschirms
- Achtung:

  Nicht die Objekte selbst werden bewegt, sondern ihre Koordinatensysteme.

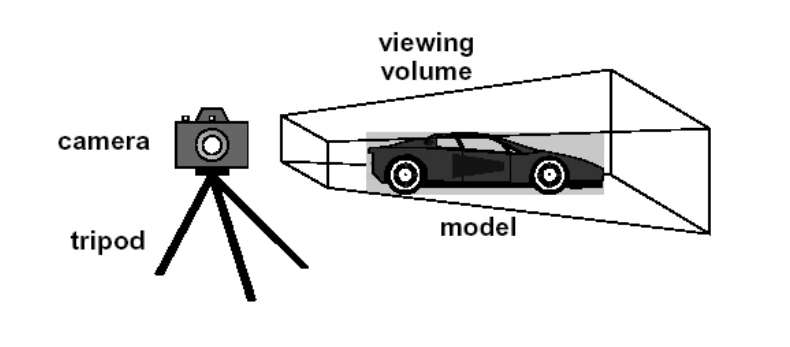


Transformationen in OpenGL

- Mehrere Arten von OpenGL Transformationen
  - *Viewing* *Orientierung der virtuellen Kamera*
  - *Modeling* *Objekte bewegen*
  - *Projection* *Sichtvolumen und clipping planes*
  - *Viewport* *2D Bild in OpenGL Fenster*
- Diese Transformationen werden in dieser bestimmten Reihenfolge ausgeführt
- Transformationen werden mittels Matrizen angegeben

Die Virtuelle Kamera

- 3D Grafik ist wie fotografieren



Die Kamera und alle Objekte sind 3D, das Resultat ist 2D!

Analogien

- Zum besseren Verständnis der OpenGL Transformation kann man die Realitiät betrachten

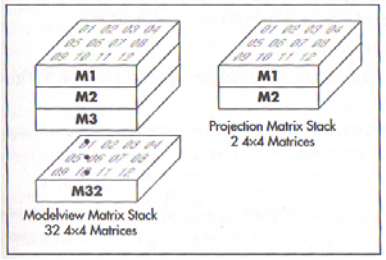
- Viewing transformations
  - Stativ – Position und Orientierung des Sichtvolumens in der Welt
- Modeling transformations
  - Modell bewegen
- Projection transformations
  - Linse der Kamera einstellen
- Viewport transformations
  - Das Bild vergrössern/verkleinern

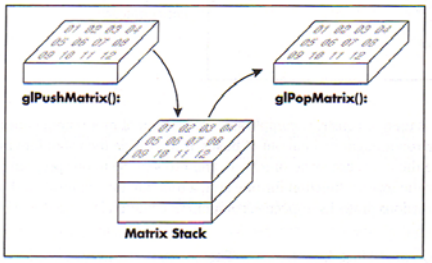
Hinweis: Im Gegensatz zum menschlichen Sehen und richtigen Kameras, ist das Sichtvolumen kein Kegel sondern pyramidenartig. Folge: Perspektivische Unterschiede, vor allem an dem Rändern

Matrix Stacks

Tobias Lang

- Matrizen werden in den sog. Matrix Stacks verwaltet
  - Modelview matrix stack
  - Projection matrix stack
  - Texture mapping stack





Transformationen – OpenGL Kommandos

Tobias Lang

- Transformationsart wählen
  - glMatrixMode(GL_MODELVIEW);
  - glMatrixMode(GL_PROJECTION);
- Einheitsmatrix laden
  - glLoadIdentity();
- Viewport festlegen
  - glViewport(0,0,width, height);
- Projection
  - gluPerspective(45.0f, width/height, 0.1f, 100.0f);
- „Klassisch"
  - glTranslatef(x,y,z);
  - glRotatef(x,y,z);
  - glScale(x,y,z);

Transformation - Codebeispiel

```
void render() {
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glPushMatrix();
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(-5.0f,1.0f,0.0f,0.0f);
    DrawCube();
  glPopMatrix();
  glPushMatrix();
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(15.0f,1.0f,0.0f,0.0f);.
    DrawSphere();
  glPopMatrix();
}
```


Beleuchtung

Beleuchtung

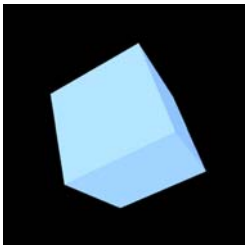
- Beleuchtung ist ein zentrales Thema in der Computergrafik
- Beleuchtung simuliert wie Objekte das Licht reflektieren
- Ergebnis ist abhängig von
  - Material-Attribute des Objekts
  - Farbe und Position der Lichtquelle
  - Globale Lichteinstellungen
    - Ambientes Licht
    - 2-seitige Beleuchtung

Beleuchtung in OpenGL

- Man kann Beleuchtung immer nur der Realität annähern
- OpenGL benutzt das Phong Modell
  - An jedem Vertex wird das Licht berechnet
  - Interpolation dazwischen
  - Schnell, aber unrealistisch bei wenig Flächen
- Das reflektierte Licht setzt sich zusammen aus
  - Ambient light - allgegenwärtiges Licht
  - Diffuse light - schattiertes Licht
  - Specular light - Glanz
  - Emissive light - Eigenleuchten (beeinflusst NICHT die Umgebung)
- In OpenGL kann man bis zu 8 Lichter erstellen

Code für Beleuchtung

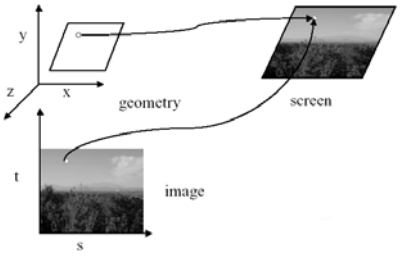
```
float ambientLight[] = {0.3f,0.5f,0.8f,1.0f};
float diffuseLighz[] = {0.25f,0.25f,0.25f,1.0f};
float Lightposition[] = {0.0f,0.0f,1.0f,0.0f};

float matAmbient[] = {1.0f,1.0f,1.0f,1.0f};
float matDiffuse[] = {1.0f,1.0f,1.0f,1.0f};

void Lighting() {
  glEnable(GL_LIGHTING);
     //set Materials
  glMaterialfv(GL_FRONT,GL_AMBIENT,matAmbient);
  glMaterialfv(GL_FRONT,GL_DIFFUSE,matDiffuse);
     // set up Light0
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
  glEnable(LIGHT0);
  DrawCube();
}
```



Texture Mapping

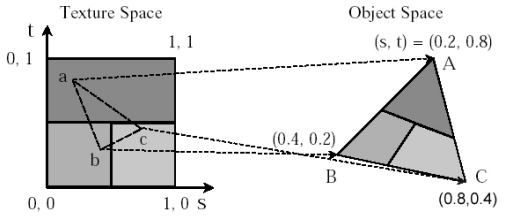
Texture Mapping

- Mit Texture Mapping bezeichnet man das „überziehen" von Polygonen mit Bildern
- Motivation
  - Materialien simulieren (z.b. Teppich)
  - Komplexität der Geometrie verringern
  - Effekte (Texturen sind drehbar etc.)
  - Echtzeit Reflektionen



Texture Mapping

- Diese Schritte sind für Texturen notwendig
  - Ein Bild laden oder generieren (2^n)
  - Das Bild einer Textur zuweisen
    - Damit wird das Bild in den Texturspeicher der Grafikkarte geladen
  - Den Vertices Texturkoordinaten zuweisen
  - Texturparameter einstellen
    - z.b Filtering

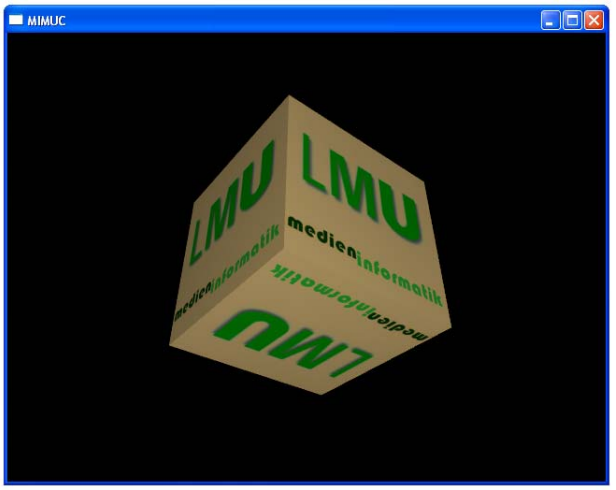


Texture Mapping in OpenGL


```
unsigned int texture;    //Texur ID

void DrawObject() {
  glBegin(GL_QUADS);
    glTexCoord2f(0.0f,1.0f);          //Texturkoordinaten für jeden Vertex
    glVertex3f(-0.5f,0.5f,0.5f);
    …
  glEnd();

void SetupTexture() {
  bitmapData = myLoadBitmapFile(„holz.jpg“);
     // TexturId reservieren und in texture speichern
  glGenTextures(1,&texture);
     // Textur mit ID 1 definieren/selektieren
  glBindTexture(GL_TEXTURE_2D,texture);
     //Filtering für Magnification/Minification
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
     //Textur laden
  glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,breite,hoehe,0,GL_RGB, GL_UNSIGNED_BYTE,
     bitmapData);
  DrawObject();
}
```


Beispiel

- Code Beispiel



Fortgeschrittene Themen

- Buffers
- Multitexturing
- DisplayLists
- Partikeleffekte
- Vertex/Pixel Shader

OpenGL und Augmented Reality

- OpenGL ist die wohl wichtigste Grafik-API für Augmented Reality
- Einsatz in Universitäten
- Geeignet für Hochleistungsrechner (Ursprung)
- DirectX und OpenGL sind kombinierbar(!)

Quellen & Links

- OpenGL Game Programming
- http://www.opengl.org/
- http://nehe.gamedev.net/(!)
- http://www.opengl.org/resources/tutorials/s2001/notes/opengl.pdf

Diskussion

- Viele Spiele nutzen DirectX – was ist nun besser?
- Ist eine Low-Level API nicht zu aufwendig?
- Wird es noch andere Standards neben DirectX und OpenGL in der nahen Zukunft geben?
- Ist OpenGL für die neuen Grafikkarten gerüstet?
- Wird das Hardware Rendering das normale Rendering z.b. mit 3ds Max ablösen?

Danke