

Erste Schritte in OpenGL

Stefanie Kettner

8. Mai 2007

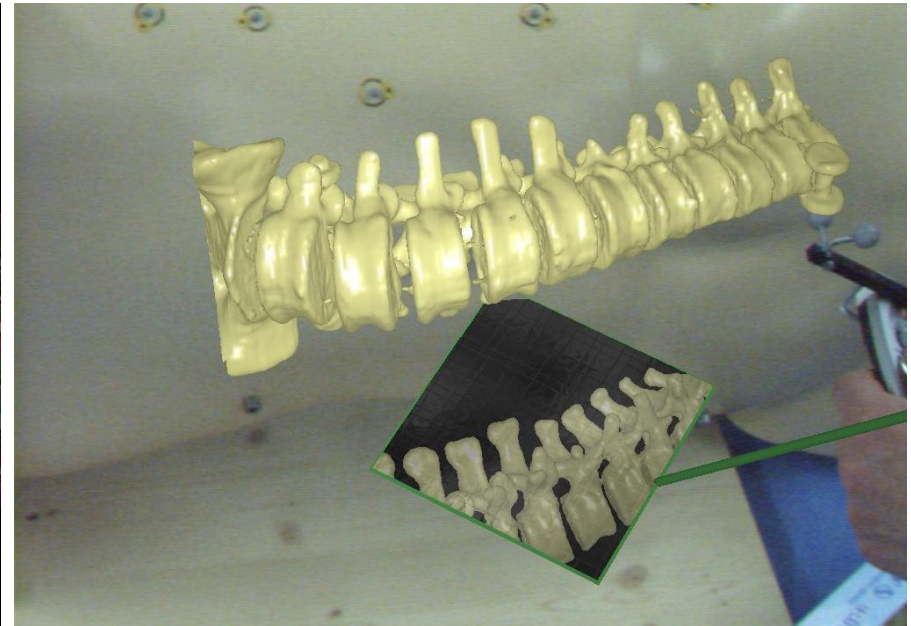
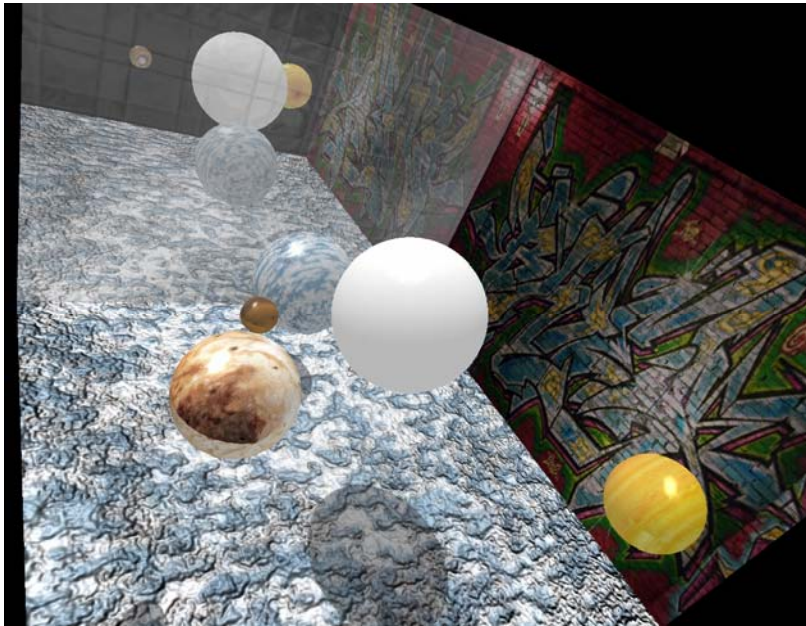
chair for computer aided medical procedures

department of computer science | technische universität münchen

Open GL (Graphics Library) seit 1992

- API Spezifikation für Plattform-, Programmiersprache-, Hardware-unabhängigen Zugriff auf (beschleunigte) Graphikhardware
- Dessen Implementierung wird i.A. mit dem Graphikkartentreiber mitgeliefert
- Zustandsbasiertes Modell
- 3D Programmierung für Visualisierung, Spielentwicklung, ...

Mit OpenGL kann man nicht nur einfach, sondern auch schnelle Graphik erzeugen



Was ist zu tun?

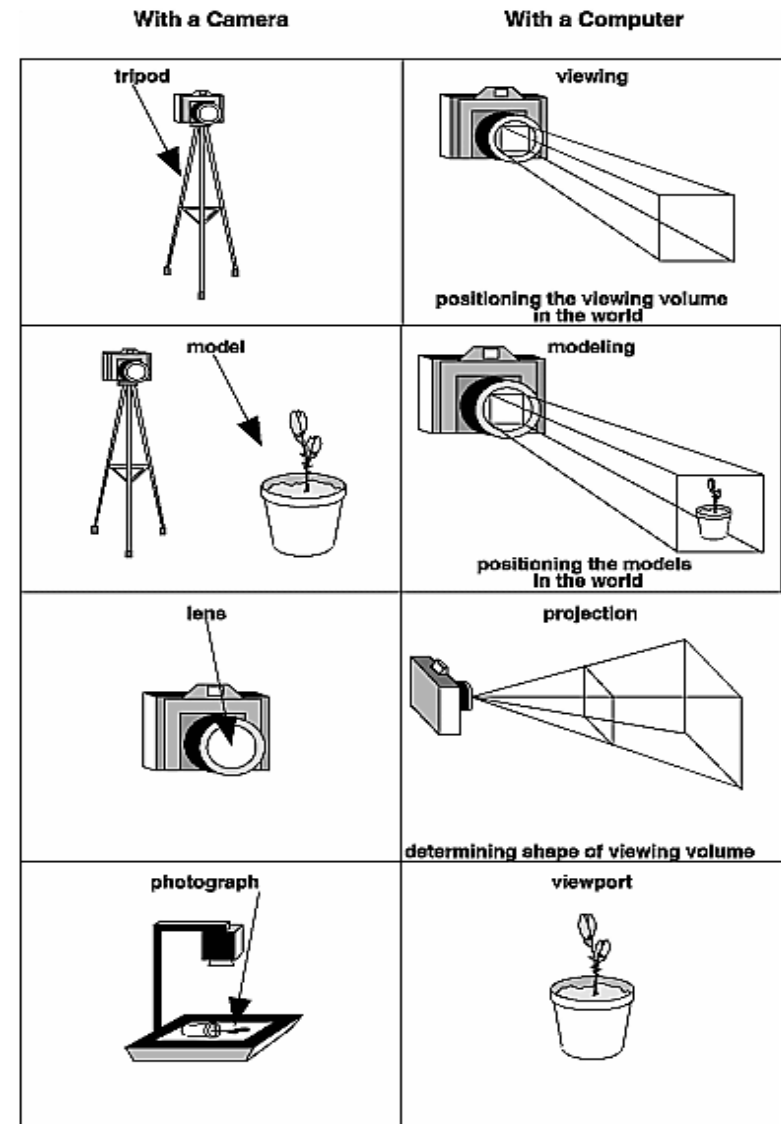
- GL Bibliothek einbinden (glu32.lib, opengl32.lib) – das Verzeichnis ist vom Betriebssystem her bekannt
 - Qt-QGLWidget erzeugen (siehe Qt-Folien)
- Nun steht alle Funktionalität zur Verfügung!

Download und Infos: <http://www.opengl.org/resources/libraries/glut/>

OpenGL Basics

Die Kamera-Analogie

1. Positionieren der Kamera
→ Viewing Transformation
2. Positionieren der Objekte
→ Modeling Transformation
3. Bestimmen der Projektion
→ Projection Matrix
4. Viewport-Größe setzen
→ Viewport setting



Die Kamera-Analogie

- Viewing Transformation: mehr dazu nächste Woche

```
gluLookAt(0.0, 0.0, -5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

- Modelling Transformation

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
glTranslatef(-1.5f, 0.0f, 0.0f); //Verschieben um 1.5 nach links
```

- Projection Matrix: mehr dazu nächste Woche

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

```
glOrtho(0.0, 1.0, 0.0, 1.0, -3.0, 3.0);
```

- Viewportgröße

```
glMatrixMode(GL_MODELVIEW);
```

```
glViewport(0, 0, width, height);
```

Beispielprogramm - Ein Dreieck

MyWidget.cpp

```
#include „MyWidget.h“

MyWidget::MyWidget(QWidget* parent)
: QGLWidget(parent)
{}

MyWidget::~MyWidget(){}

void MyWidget::initializeGL()
{ ... }

void MyWidget::paintGL()
{ ... }
```

Beispielprogramm - Ein Dreieck

MyWidget.cpp

```
...
void MyWidget::initializeGL(){
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -3.0, 3.0);
    glMatrixMode(GL_MODELVIEW);
    glViewport(0, 0, width, height);
}
void MyWidget::paintGL(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_QUADS)
        glVertex3f(-0.5f, 0.5f, 0.0f);
        glVertex3f(0.5f, 0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
    glEnd();
}
...
```

Beispielprogramm - Ein Dreieck

MyWidget.cpp

```
...  
void MyWidget::initializeGL(){  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    glOrtho(0.0, 1.0, 0.0, 1.0, -3.0, 3.0);  
    glMatrixMode(GL_MODELVIEW);  
    glViewport(0, 0, width, height);  
}  
void MyWidget::paintGL(){  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
    glLoadIdentity();  
    glTranslatef(-0.5f, 0.0f, 0.0f);  
    glBegin(GL_QUADS)  
        glVertex3f(-0.5f, 0.5f, 0.0f);  
        glVertex3f(0.5f, 0.5f, 0.0f);
```

← Modeling Transformation

Beispielprogramm – Ein Dreieck

- Aufruf von main-Methode

```
#include „MyWidget.h“  
#include <QApplication>  
  
int main(int arg, char* argv[]) {  
  
    QApplication qapp(arg, argv); //Qt application  
    MyWidget* widget = new MyWidget();  
    widget->show(); //evokes paintGL()  
  
    qapp.exec(); //executes the Qt application  
}
```

main.cpp

Transformationen: Etwas Mathematik

- Darstellung von euklidischen Transformationen (also nur Rotation und Translation)
 - in euklidischen Koordinaten $p_{image} = \mathbf{R} p + \mathbf{t}$
 - in homogenen Koordinaten $p_{image} = \mathbf{H} p$
- Euklidische Koordinaten ist die übliche Darstellung mit $p = (x, y, z)^T$
- Homogene Koordinaten ist die Darstellung mit $p = (x, y, z, 1)^T$
oder $p = (\omega x, \omega y, \omega z, \omega)^T$
- Eine Transformation in Matrixschreibweise in homogenen Koordinaten lautet

$$\mathbf{H} = \begin{pmatrix} r_{11} & r_{21} & r_{31} & t_x \\ r_{12} & r_{22} & r_{32} & t_y \\ r_{13} & r_{23} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Modelview matrix vs. projection matrix

- Man könnte alle Transformationen (Rotation & Translation, Projektion 3D-2D) zu einer homogenen Matrix zusammenfassen, aber aus praktischen Gründen bleiben sie getrennt

$$p_{image} = \mathbf{H}_{\text{projection}} \cdot \mathbf{H}_{\text{modelview}} \mathbf{P}_{3D}$$

- Die *Projektionsmatrix* modelliert die virtuelle Kamera von OpenGL
- Die *Modelviewmatrix* modelliert die Position der Objekte im Weltkoordinatensystem
- Die Projektionsmatrix ist standardmäßig so gesetzt, dass die virtuelle Kamera eine orthogonale Projektion erfüllt

Befehle zum ändern der Matrizen

- Wählen, welche Matrix geändert wird `glMatrixMode(GL_MODELVIEW)` oder `glMatrixMode(GL_PROJECTION)`
- Ändern der Matrix $M_{\text{neu}} = M_{\text{alt}} \cdot M_{\text{Transformation}}$
 - **Verschiebung:** `glTranslated(GLdouble x, GLdouble y, GLdouble z)`
 - **Drehung:** `glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z)`
 - **Multiplizieren einer Matrix:** `glMultMatrixd(const GLdouble *m)`
Größe der Nachfolgenden Objekte ändern: `glScaled(GLdouble x, GLdouble y, GLdouble z)`
- Laden einer Matrix
 - **Identitätsmatrix:** `glLoadIdentity()`
 - **Beliebige Matrix:** `glLoadMatrixd(const GLdouble *m)`

OpenGL ist ein Zustandsautomat

- D.h. wenn man Sachen ändert, bleiben sie solange bestehen, bis man sie wieder ändert

- Man kann einen Zustand in den Keller [Stack] „retten“, um ihn später wiederherzustellen
 - `glPushAttrib(GL_EIGENSCHAFT)` rettet den Wert von `GL_EIGENSCHAFT` in den Keller [Stack]
 - `glPopAttrib(GL_EIGENSCHAFT)` stellt den Wert wieder her
 - `glPushAttrib()` rettet alle Attribute / Eigenschaften

- Das gleiche gilt auch für die Matrizen
 - `glPushMatrix()` rettet die aktuelle Matrix
 - `glPopMatrix()` stellt die gerettete Matrix wieder her

Hilfe?!

- Delphi Wiki (<http://www.delphigl.com/>), ist zwar nicht C++, sondern Delphi, aber OpenGL ist sprachunabhängig!
- NeHe Online Tutorials samt Code (<http://nehe.gamedev.net/>) oder dessen deutsche Übersetzung (http://www.joachimrohde.com/cms/xoops/modules/articles/index.php?cat_id=1)
- Originalreferenz von SGI (<http://www.opengl.org/>)
- Das RedBook Version 1.0 online (http://www.opengl.org/documentation/red_book/)
- GLUT Spezifikation (<http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>)