Übung „Augmented Reality“
Einführung in GLUT

Fabian Hennecke

05.05.09

Agenda (I)

- OpenGL Utility Toolkit (GLUT)
  - Initialisierung
  - Callback-Funktionen
- OpenGL
  - Initialisierung
  - 2D-Objekte
  - 3D-Objekte
  - Grundlegende Transformationen

Agenda (II)

- Geometrische Transformationen in 3D
  - Koordinatensysteme
  - Translation, Rotation und Skalierung
  - Homogene Koordinaten und Transformationsmatrix
  - Projection- und Modelview-Matrix und Matrixoperationen in OpenGL
- ARToolkit

OpenGL Utility Toolkit (GLUT)

- Nachfolger von *libaux*
- Systemunabhängiges Toolkit
- Erlaubt mehrere OpenGL-Fenster
- Ereignisverarbeitung durch Callbacks
- Funktionen für verschiedene Objekte
- Fensterverwaltung
- Verwendung von GLUT:

```
#include <GL/glut.h>
// bindet auch GL/gl.h und GL/glu.h ein
```

GLUT-Initialisierung

- GLUT initialisieren:

```
void glutInit(int* argcp, char** argv);
```

- GLUT-Fenster initialisieren:

```
void glutInitWindowSize(int w, int h);
void glutInitWindowPosition(int x, int y);
```

- Displaymodus (später mehr):

```
void glutInitDisplayMode(unsigned int mode);
```

- Fenster erzeugen:

```
int glutCreateWindow(char* title);
```


GLUT-Initialisierung

- Fenster zerstören:

```
void glutDestroyWindow(int window);
```

- Fenster neu zeichnen:

```
void glutPostRedisplay();
```

- Bildschirmspeicher wechseln:

```
void glutSwapBuffers();
```

  - GLUT zeichnet im Displaymodus `DOUBLE` jeweils in den Hintergrund-Buffer
  - Am Ende des Zeichnens wird der aktuell dargestellte Buffer und der Hintergrund-Buffer gewechselt

GLUT - Callback-Funktionen

- GLUT verwendet einige Callback-Funktionen, die durch Ereignisse aufgerufen werden
- Idle Callback:
  - Wird aufgerufen, wenn keine anderen Events aufgetreten sind → Zeichnet meistens die Szene neu

```
void glutIdleFunc(void (*func)(void));
```


GLUT – Callback-Funktionen

- Display Callback:
  - Wird aufgerufen, wenn das **aktuelle** Fenster neu gezeichnet werden muss

```
void glutDisplayFunc(void (*func)(void));
```

- Reshape Callback:
  - Wird aufgerufen, wenn sich die Größe des **aktuellen** Fensters verändert

```
void glutReshapeFunc(void (*func)
                     (int w, int h));
```

GLUT – Callback-Funktionen

- Keyboard Callback:
  - Wird aufgerufen, wenn eine Standard-Taste (keine F-Tasten o.ä.) gedrückt wurde

```
void glutKeyboardFunc(void (*func)
      (unsigned char key, int x, int y));
```

- Special Callback:
  - Wird aufgerufen, wenn eine Sondertaste gedrückt wurde

```
void glutSpecialFunc(void (*func)
      (int key, int x, int y));
```


GLUT – Callback-Funktionen

- Mouse Callback:
  - Wird aufgerufen, wenn eine Maus-Taste gedrückt wurde

```
void glutMouseFunc(void (*func)
      (int button, int state, int x, int y));
```

- Motion Callback:
  - Wird aufgerufen, wenn die Maus bei gedrückter Maustaste bewegt wurde

```
void glutSpecialFunc(void (*func)
      (int x, int y));
```

GLUT – Callback-Funktionen

- Passive Motion Callback:
  - Wird aufgerufen, wenn die Maus ohne gedrückte Maustaste bewegt wurde

```
void glutPassiveMotionFunc(void (*func)
     (int x, int y));
```

- Starten der Ereignisverarbeitung:
  - Wird nach der Initialisierung von GLUT und OpenGL aufgerufen und startet eine Endlosschleife

```
void glutMainLoop(void);
```


GLUT – Callback-Funktionen

- Beispiel:

```
void display(void) {
  // some OpenGL drawing
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  …
  glutDisplayFunc(display);
  …
  glutMainLoop();
  return 0;
}
```

OpenGL

- Low Level Graphik API
- Grundlegende 3D API (in der Industrie standardisiert)
- Verwendung meistens mit GLU (OpenGL Utility Library)
- Ermöglicht direkten Hardware-Zugriff
- Erweiterungen:
  - **Open Inventor:** flexible und erweiterbare
- Szenengraph-API
  - **GLUT:** OpenGL Utility Toolkit

OpenGL - Initialisierung

- Wichtige Schritte vor dem Starten der GLUT MainLoop:
  - Spezifikation der Hintergrundfarbe der Buffer

```
glClearColor(float red, float green,
      float blue, float alpha);
```

  - Spezifikation des Shader-Modells

```
glShadeModel(GLenum mode);
// mode = GL_FLAT or GL_SMOOTH
```

  - Spezifikation wie tief im z-Buffer gelöscht wird

```
glClearDepth(float depth);
```

OpenGL - Initialisierung

- Wichtige Schritte vor dem Starten der GLUT MainLoop:
  - Einschalten der OpenGL-Funktionen

```
glEnable(GLenum capability);
// capabilities:
// GL_BLEND, GL_DEPTH_TEST,
// GL_TEXTURE_2D, GL_LIGHTING, …
glDisable(GLenum capability);
```

  - Spezifikation der Tiefenfunktion

```
glDepthFunc(GLenum function);
// functions: GL_NEVER, GL_LESS, GL_EQUAL,
// GL_LEQUAL, GL_GREATER, GL_ALWAYS, …
```


OpenGL - Initialisierung

- Wichtige Schritte vor dem Starten der GLUT MainLoop:
  - Spezifikation der Zeicheneigenschaften

```
glHint(GLenum target, GLenum mode);
// target: GL_PERSPECTIVE_CORRECTION_MODE
// mode: GL_NICEST
```

  - Spezifikation der Renderingeigenschaften für Polygone

```
glPolygonMode(GLenum face, GLenum mode);
// face: GL_FRONT_AND_BACK, GL_FRONT, …
// mode: GL_FILL, …
```

OpenGL - Initialisierung

- Beispiel:

```
bool initGL() {
    glClearColor(0.0f, 0.0f, 0.5f, 1.0f);
    glShadeModel(GL_FLAT);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDisable(GL_BLEND);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT,
                GL_NICEST);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    return true;
}
```


OpenGL - Initialisierung

- Wichtig (für die Reshape Funktion):

```
glViewport(GLint x, GLint y, GLsizei width,
    GLsizei height);
gluPerspective(GLdouble fovy, GLdouble
    aspect, GLdouble zNear, GLdouble zFar);
```

- Beispiel:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, w, h);
gluPerspective(45, 1.0f * w/h, 0.1f, 100.0f);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

OpenGL – 2D-Objekte

- Alle 2D-Objekte liegen im Raum in einer Ebene und bestehen aus einer Menge von Punkten
- Mögliche 2D-Objekte:
  - Points (1D)
  - Lines (2D)
  - Line Strips (2D) → Verkettete Linien
  - Line Loops (2D) → geschlossener Linienzug
  - Polygons (2D) → geschlossener, gefüllter Linienzug

OpenGL – 2D-Objekte

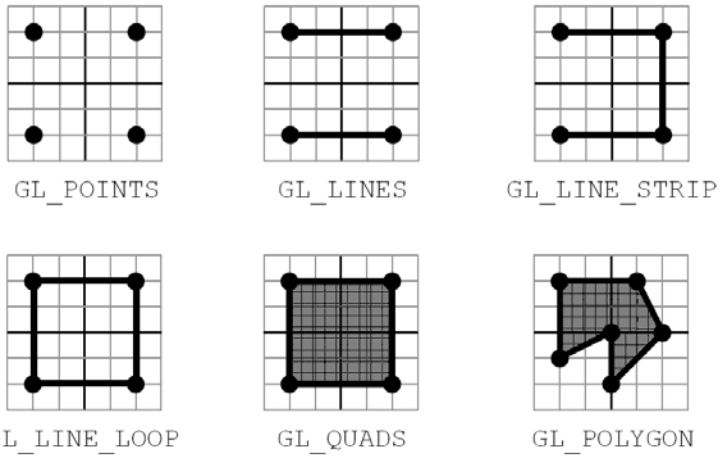
- Beispiel:

```
glBegin(GL_POLYGON);
glColor3f(1.0f, 0.0f, 0.0f); // red
glVertex3f(-1.0f, -1.0f, 0.0f);
glVertex3f(1.0f, -1.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f); // blue
glVertex3f(1.0f, 1.0f, 0.0f);
glVertex3f(-1.0f, 1.0f, 0.0f);
glEnd();
```

- Weitere Formen:
  `GL_POINTS, GL_LINES (je 2 Punkte verbunden)`
  `GL_LINE_STRIP, GL_LINE_LOOP`
  `GL_POLYGON, GL_QUADS, GL_TRIANGLES`

OpenGL – 2D-Objekte

- Beispiele:



OpenGL – 3D-Objekte

- Werden aus 2D-Objekten erstellt
- Vertices einer Fläche unterscheiden sich in allen drei Koordinaten
- Beispiel:
  - Würfel aus 6 Quadraten
  - Würfel aus 12 Dreiecken
  - Kugel aus *n* Dreiecken
  - Tetraeder aus 4 Dreiecken
  - …

OpenGL - Zeichnungsablauf

- Zunächst Löschen des Buffers:
```
glClear(GL_COLOR_BUFFER_BIT |GL_DEPTH_BUFFER_BIT);
```
- Vor dem Zeichnen: Transformation des zu zeichnenden Objektes:
```
glLoadIdentity(); // Laden der Einheitsmatrix

glTranslatef(GLfloat x, GLfloat y, GLfloat z);

glRotatef(GLfloat angle, GLfloat x, GLfloat y,
    GLfloat z);
```
- Nach dem Zeichnen der Objekte: Wechsel des Buffers
```
glutSwapBuffers();
```


OpenGL – Grundlegende Transform.

- Die Einheitsmatrix:
```
glLoadIdentity();
```
  – Lädt die Position an der sich die Kamera befindet

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Die Translation:
```
glTranslatef(x, y, z);
glTranslated(x, y, z);
```
  – Verschiebt den Ursprung an die neue Position

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL – Grundlegende Transform.

- Die Rotation:

```
glRotatef(a, x, y, z);
glRotated(a, x, y, z);
```

  – Rotiert den Ursprung um den Vektor (x,y,z) entgegen dem Uhrzeigersinn

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$glRotate*(\alpha,1,0,0)$

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

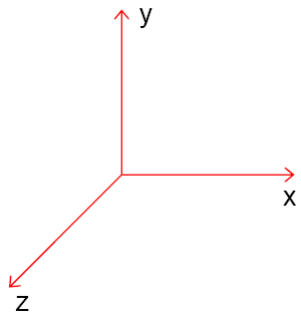
$glRotate*(\alpha,0,0,1);$

$$\begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$glRotate*(\alpha,0,1,0);$

Geometrische Transformationen in 3D

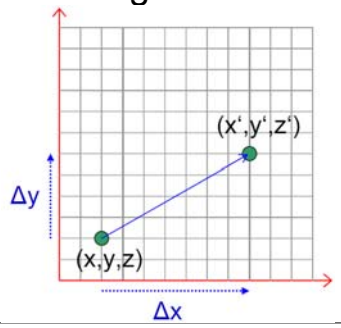
- Koordinatensysteme
  – Rechtshändiges karthesisches Koordinatensystem:



Translation, Rotation und Skalierung

- Translation:
  - Verschiebung eines Punktes auf einen anderen
  - Beschreibung der Translation:

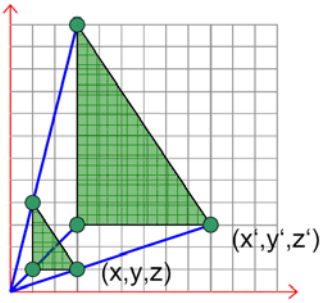
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$$



Translation, Rotation und Skalierung

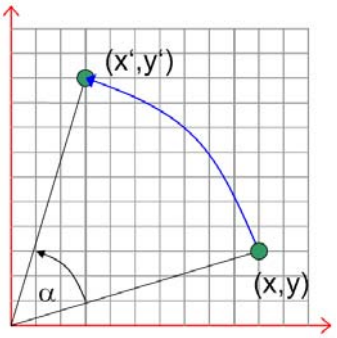
- Skalierung (uniform):
  - Streckung in alle Richtungen mit dem Ursprung als Zentrum um α
  - Beschreibung der Skalierung:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \alpha \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} \alpha \cdot x \\ \alpha \cdot y \\ \alpha \cdot z \end{pmatrix}$$



Translation, Rotation und Skalierung

- Rotation:
  - Drehung eines Punktes um den Ursprung mit dem Winkel α (α > 0 □ CCW)
  - Beschreibung der Rotation (2D):



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Translation, Rotation und Skalierung

- Rotation (in 3D):
  - Drei verschiedene Matrizen für die Drehung eines Punktes um eine Achse

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um die x-Achse

$$\begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um die y-Achse

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um die z-Achse

Translation, Rotation und Skalierung

- Bisher:
  - Translation: Addition eines Vektors
  - Skalierung: Multiplikation des Faktors
  - Rotation: Matrixmultiplikation
- Problem:
  - Keine einheitliche Behandlung
  - Zusammengesetzte Transformationen nur schwer zu realisieren
  - Umkehrung von verketteten Transformationen nicht möglich (da nicht affin)

Homogene Koordinaten

- Einheitliche Repräsentation von allen Transformationen
- Überführung eines Koordinatensystems in ein homogenes Koordinatensystem durch Hinzufügen einer weiteren Dimension
- Repräsentation eines 3D-Punktes in homogenen Koordinaten:

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda \cdot x \\ \lambda \cdot y \\ \lambda \cdot z \\ \lambda \end{pmatrix}, \lambda \neq 0$$

Homogene Koordinaten

- Vorteile:
  - Repräsentation aller Punkte in homogenen Koordinaten ermöglicht nun die einheitliche Behandlung der Transformationen
- Beispiel:

$$\begin{pmatrix} \lambda \cdot x \\ \lambda \cdot y \\ \lambda \cdot z \\ \lambda \end{pmatrix} = ? \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- ? → Transformationen als Matrizen
- Verknüpfung durch Matrixmultiplikation

Homogene Koordinaten

- Die allgemeine Transformationsmatrix:

$$T = \begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{pmatrix}$$

  - Skalierung
  - Rotation
  - Translation

Projection- und Modelview-Matrix

- OpenGL verwendet zwei Matrizen:
  - Modelview-Matrix:
    - Transformiert die Objektkoordinaten
    - Erzeugt Kamerakoordinaten
    - Beschreibt die Beziehung zwischen Objekt und Kamera (als Transformationsmatrix)
  - Projection-Matrix:
    - Transformiert die Kamerakoordinaten
    - Erzeugt Schnittkoordinaten
  - Schnittkoordinaten werden mit einem Faktor letztlich auf den Bildschirm übertragen

Projection- und Modelview-Matrix

- Matrixoperationen in OpenGL:
  - Die Wahl der Matrix:

```
void glMatrixMode(GLenum mode);
// mode = GL_MODELVIEW, GL_PROJECTION
oder GL_TEXTURE
```

  - Laden der Einheitsmatrix:
    - Setzt den Ursprung an die Position des Betrachters (also der Kamera)

```
void glLoadIdentity();
```

Projection- und Modelview-Matrix

- Matrixoperationen in OpenGL:
  - Laden einer Matrix:
    - Matrizen sind 1D repräsentiert, d.h. bei einer 4x4-Matrix handelt es sich um ein Feld der Länge 16
    - Matrizen werden “row-major” behandelt, d.h. die ersten vier Elemente bilden die erste Spalte, usw.

```
void glLoadMatrixf(GLfloat* mp);
```

  - Multiplikation der aktuellen Matrix:

```
void glMultMatrixf(GLfloat* mp);
```


Projection- und Modelview-Matrix

- Matrixoperationen in OpenGL:
  - Der Matrix-Stack:
    - OpenGL besitzt für jede Matrix einen Stack
    - Matrizen können “gesichert” werden, um sie nach weiteren Transformationen wieder laden zu können
    - Ohne Stack müssten alle Transformationen rückgängig gemacht werden (→ hoher Aufwand)
    - Der Modelview-Stack kann mindestens 32, die beiden anderen können mindestens 2 Matrizen aufnehmen

Projection und Modelview-Matrix

- Matrixoperationen in OpenGL:
  - Der Matrix-Stack:
    - Speichern einer Matrix auf dem Stack:

```
void glPushMatrix();
```

    - Laden einer Matrix vom Stack:

```
void glPopMatrix();
```

    - Push- und Pop-Operationen müssen sich **nicht** abwechseln, da die Stacks eine gewisse Tiefe zur Verfügung stellen

Projection- und Modelview-Matrix

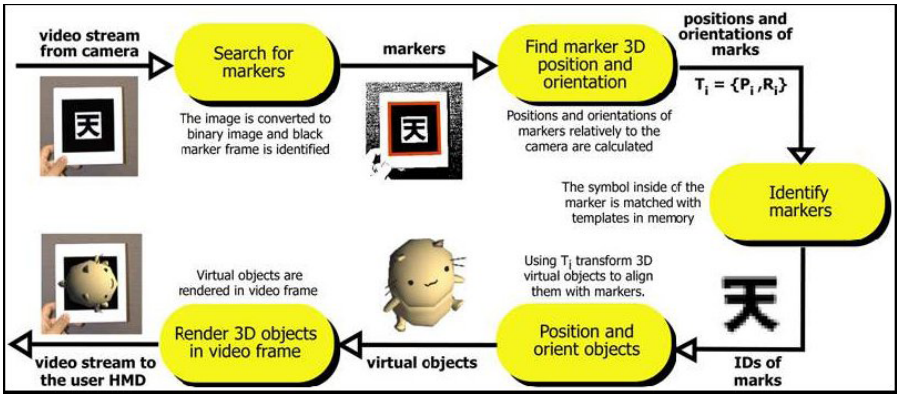
- • Matrixoperationen in OpenGL:
  - Der Matrix-Stack:
    - Die Aufrufe zu `glTranslatef(…)`, `glRotatef(…)` und `glScalef(…)` führen lediglich eine Multiplikation der aktuellen Transformationsmatrix durch
    - Die Reihenfolge der Multiplikationen wird durch den Programmierer festgelegt
    - **Achtung:** Es ist entscheidend, ob erst eine Translation gefolgt von einer Rotation durchgeführt wird, oder umgekehrt!

ARToolKit

- Marker-basiertes Tracking

ARToolKit

- Funktionsweise



ARToolKit

- Entwicklungsprinzip:
  - Initialisierung: Aufnahme, Marker
  - Hauptschleife:
    - Nächstes Videoframe holen
    - Marker erkennen
    - Kameratransformation bestimmen
    - Virtuelle Objekte zeichnen
  - Deinitialisierung: Aufnahme beenden

ARToolKit

- Grundlegende Funktionen:
  - Werden meist in der Callback-Funktion `mainLoop` aufgerufen
  - Initialisierung:

    `init`
  - Nächstes Frame einlesen:

    `arVideoGetImage`
  - Marker erkennen:

    `arDetectMarker`

ARToolKit

- Grundlegende Funktionen:
  - Kameratransformation berechnen:

    `arGetTransMat`
  - Virtuelle Objekte zeichnen:

    `draw`
  - Videoaufnahme beenden:

    `cleanup`

ARToolKit

- Aufbau eines Programms:

```
int main(int argc, char** argv)
{
    init();
    arVideoCapStart();
    argMainLoop(NULL, keyEvent, mainLoop);
    return 0;
}
```

ARToolKit

- Aufbau der `init`-Methode:

```
[…]
/* open the video path */
if( arVideoOpen( vconf ) < 0 ) exit(0);

/* find the size of the window */
if( arVideoInqSize(&xsize, &ysize) < 0 )
    exit(0);
printf("Image size (x,y) =
    (%d, %d)\n", xsize, ysize);
```


ARToolKit

- Aufbau der `init`-Methode:

```
/* set the initial camera parameters */
if(arParamLoad(cparaname, 1, &wparam) < 0)
{
    printf("Camera parameter load error!!\n");
    exit(0);
}
arParamChangeSize(&wparam, xsize, ysize,
    &cparam);
arInitCparam(&cparam);
printf("*** Camera Parameter ***\n");
arParamDisp(&cparam);
```

ARToolKit

- Aufbau der `init`-Methode:

```
if((patt_id=arLoadPatt(patt_name)) < 0)
{
    printf("pattern load error !!\n");
    exit(0);
}
/* open the graphics window */
argInit(&cparam, 1.0, 0, 0, 0, 0);
```


ARToolKit

- Aufbau der `mainLoop`-Methode:

```
[…]
/* grab a video frame */
if((dataPtr = (ARUint8*)arVideoGetImage())
    == NULL) {
    arUtilSleep(2); return;
}
argDrawMode2D();
argDispImage(dataPtr, 0, 0);
if(arDetectMarker(dataPtr, thresh,
    &marker_info, &marker_num) < 0 ) {
    cleanup(); exit(0);
}
```

ARToolKit

- Aufbau der `mainLoop`-Methode:

```
arVideoCapNext();
/* check for object visibility */
k = -1;
for(j = 0; j < marker_num; j++ ) {
if(patt_id == marker_info[j].id) {
          if(k == -1) k = j;
          else if(marker_info[k].cf <
                marker_info[j].cf ) k = j;
    }
}
```


ARToolKit

- Aufbau der `mainLoop`-Methode:

```
if(k == -1)
{
    argSwapBuffers();
return;
}
/* get the transformation between the marker
and the real camera */
arGetTransMat(&marker_info[k], patt_center,
    patt_width, patt_trans);
draw(patt_trans);
argSwapBuffers();
```

ARToolKit

- Aufbau der `draw`-Methode:
- Parameter: **`double`** `trans[3][4]`

```
argDrawMode3D(); argDraw3dCamera( 0, 0 );
glClearDepth( 1.0 );
glClear(GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);

/* load the camera transformation matrix */
argConvGlpara(patt_trans, gl_para);
glMatrixMode(GL_MODELVIEW);
glLoadMatrixd(gl_para);
```


ARToolKit

- Aufbau der `draw`-Methode:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
glLightfv(GL_LIGHT0, GL_DIFFUSE,
    lightZeroColor);
glMaterialfv(GL_FRONT, GL_SPECULAR,
    mat_flash);
glMaterialfv(GL_FRONT, GL_SHININESS,
    mat_flash_shiny);
glMaterialfv(GL_FRONT, GL_AMBIENT,
    mat_ambient);
glMatrixMode(GL_MODELVIEW);
```

ARToolKit

- Aufbau der `draw`-Methode:

```
glTranslatef(0.0, 0.0, 25.0);
glutSolidCube(50.0);
glDisable(GL_LIGHTING);
glDisable(GL_DEPTH_TEST);
```

- Aufbau der `cleanup`-Methode:

```
arVideoCapStop();
arVideoClose();
argCleanup();
```


ARToolKit auf dem Übungsblatt

- Benötigt: USB-Webcam, WDM-Treiber
- Wer hat *keine* eigene Webcam?
- Drucken der Marker: *kein* Anpassen der Seitengröße (Druckmenü im Acrobat Reader)

Literatur

- OpenGL Reference Page:
  http://www.mevis.de/opengl/opengl.html
- DGL OpenGL-Wiki:
  http://wiki.delphigl.com/index.php
- 3Dsource:
  http://www.3dsource.de/faq/index.htm
- GLUT Man Pages:
  http://www.cs.uccs.edu/~semwal/man.html
- ARToolkit Tutorial:
  http://www.hitl.washington.edu/artoolkit/documentation/