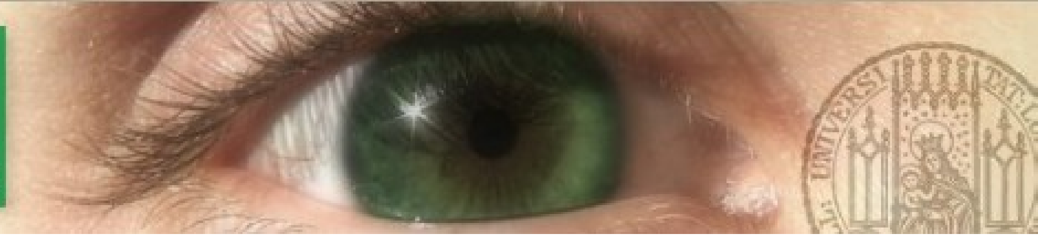
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LEHR- UND FORSCHUNGSEINHEIT MEDIENINFORMATIK



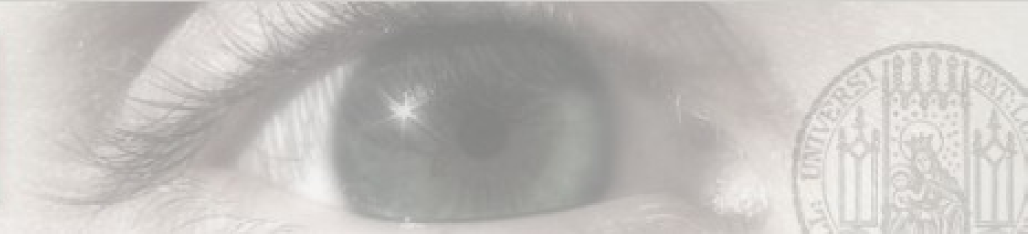
Prof. Dr. Andreas Butz

Dipl.-Medieninf. Hendrik Richter
Dipl.-Medieninf. Raphael Wimmer

Computergrafik 1 Übung 4

Was ist OpenGL?

- OpenGL = Open Graphics Library
- API für Echtzeit-3D-Grafik
- Hardwarebeschleunigt (GPU)
- Betriebssystem- / GUI-unabhängig
- wenige geometrische Primitive: Punkte, Zeilen, Polygone, Bilder
- Out of the Box kein Raytracing, Radiosity, Schattenberechnung, Spiegelungseffekte, etc.
- als \`state machine´ (Zustandsmaschine) implementiert, d.h. der Zustand der Render-Engine wird über Schaltfunktionen eingestellt und bleibt erhalten bis zu einer Zustandsänderung



Geschichte

Jahr		
1992	OpenGL 1.0	
1994		
1996		
1998		
2000		
2002	OpenGL ES	GLSL (OpenGL Shading Language)
2004	OpenGL 2.0	
2006		
2008	OpenGL 3.0	OpenCL (Open Computing Language)
2010	OpenGL 4.0	

OpenGL Internals

Transform Feedback

Vertex Data

Vertex Shading and Per-Vertex Operations

Primitive Assembly and Rasterization

Fragment Shading and Per-Fragment Operations

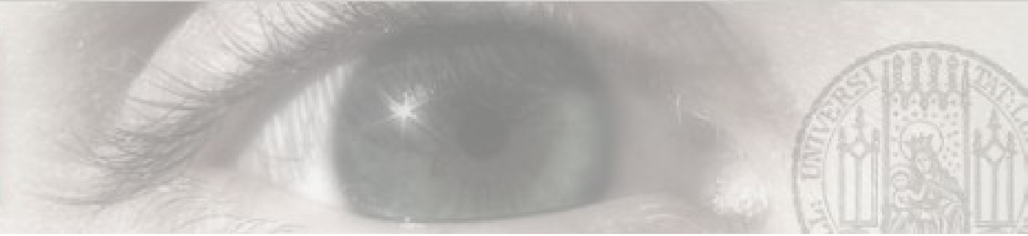
Framebuffer

Pixel Data

Texture Memory

Pixel Pack/Unpack

Source: OpenGL 4.0 Specification



Fünf Wege, um Primitive zu zeichnen (I)

Immediate Mode

```
glBegin(GL_QUADS);
    glVertex3f(1.0,1.0,0.0);
    glVertex3f(-1.0,0.5,0.0);
    glVertex3f(-1.0,-1.0,0.0);
    glVertex3f(1.0,-1.0,0.0);
glEnd();
```

Display Lists

```
GLuint index = glGenLists(1);
glNewList(index, GL_COMPILE);
    glBegin(GL_QUADS);
        glVertex3f(1.0,1.0,0.0);
        glVertex3f(-1.0,0.5,0.0);
        glVertex3f(-1.0,-1.0,0.0);
        glVertex3f(1.0,-1.0,0.0);
    glEnd();
glEndList();
glCallList(index); // draw
glDeleteLists(index, 1);
```

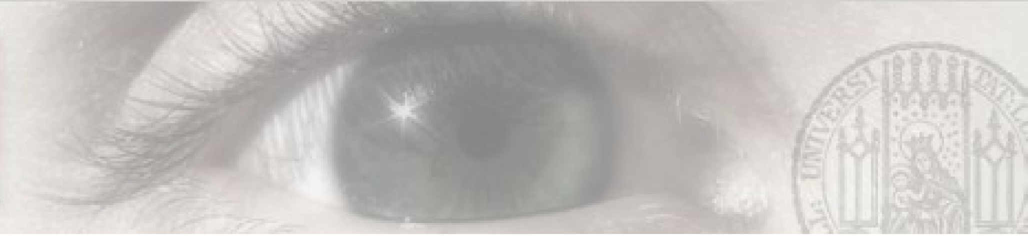
Fünf Wege, um Primitive zu zeichnen (II)

Vertex Arrays

```
GLfloat vertices[] = {1.0,1.0,0.0,
                      -1.0,0.5,0.0,
                      -1.0,-1.0,0.0,
                      1.0,-1.0,0.0};
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glDrawArrays(GL_QUADS, 0, 4);
glDisableClientState(GL_VERTEX_ARRAY);
```

Vertex Arrays mit Indices

```
GLfloat vertices[] = {1.0,1.0,0.0,
                      -1.0,0.5,0.0,
                      -1.0,-1.0,0.0,
                      1.0,-1.0,0.0};
GLfloat indices[] = {0,1,2,3};
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glDrawElements(GL_QUADS, 4,
    GL_UNSIGNED_BYTE, indices);
glDisableClientState(GL_VERTEX_ARRAY);
```



Fünf Wege, um Primitive zu zeichnen (III)

Vertex Buffer Objects

```
#define GL_GLEXT_PROTOTYPES
#include <GL/glext.h>
GLuint vboID;
glGenBuffers(1, &vboID);
glBindBuffer(GL_ARRAY_BUFFER_ARB,
    vboID);
glBufferData(GL_ARRAY_BUFFER_ARB,
    sizeof(vertices), vertices,
    GL_STATIC_DRAW_ARB);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, 0);
glDrawArrays(GL_QUADS, 0, 4);
glDisableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER_ARB, 0);
glDeleteBuffers(1, &vboID);
```

Was soll ich verwenden?

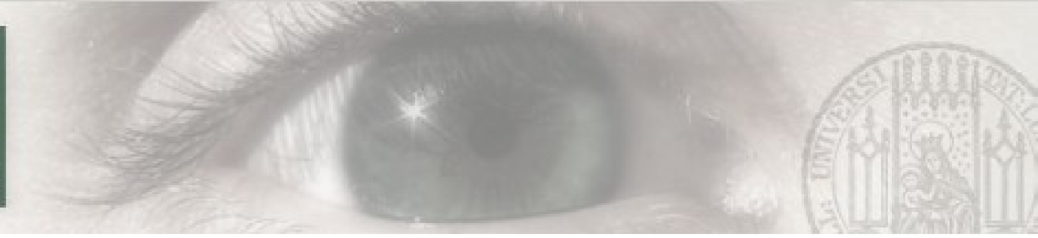
OpenGL 3.0 und 4.0 'deprecaten'
  Immediate Mode und Display Lists.

Aber diese werden auch weiterhin
  unterstützt.

Einfache Primitive:
  immediate mode

Einfache Performance-Verbesserung:
  Display Lists

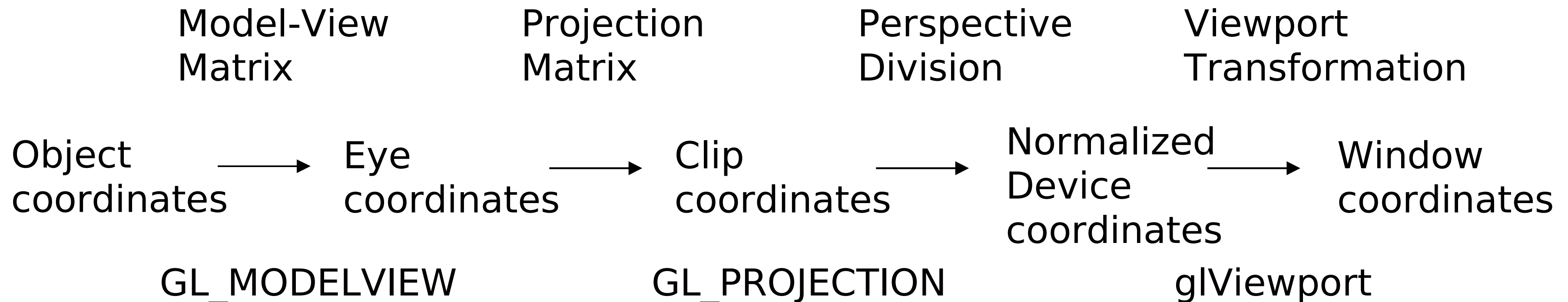
Größere Projekte:
  Vertex Buffer Objects



Projektionen in OpenGL

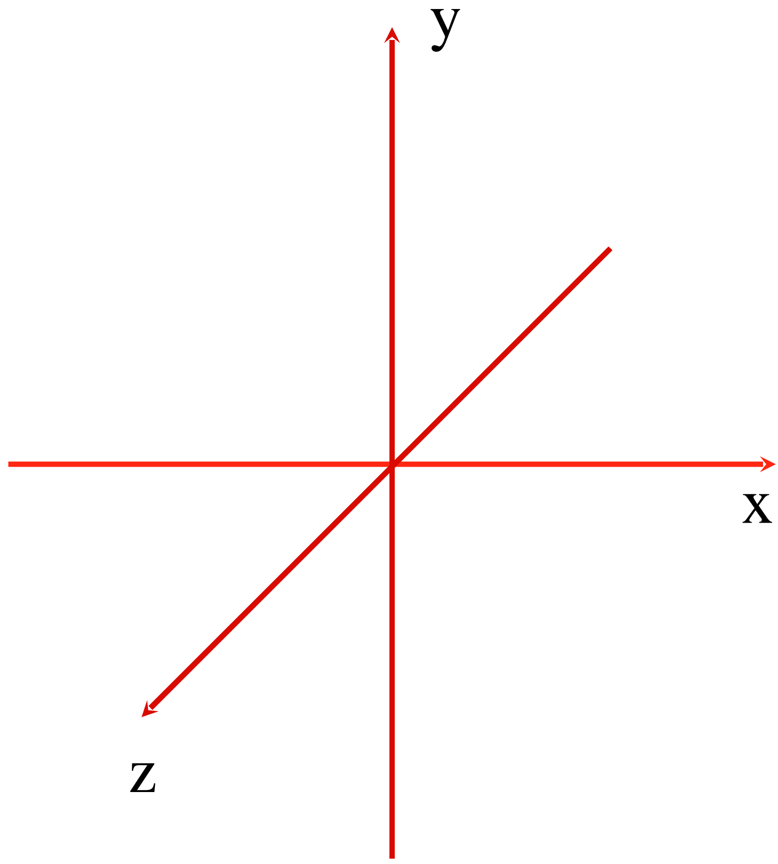
OpenGL Transformationen

- Feste Kette von Matrixmultiplikationen zur Transformation/Projektion
- Objekte in Object/World coordinates ->
  Farbige Pixel in Window coordinates



(Quelle: http://www.opengl.org/resources/faq/technical/transformations.htm)

OpenGL Object coordinates




```
glBegin(GL_POLYGON);
    glColor3f(1.0f, 0.0f, 0.0f); // red
    glVertex3f(-1.0f, -1.0f, 1.0f); // p1
    glVertex3f(1.0f, -1.0f, 1.0f);  // p2
    glColor3f(0.0f, 1.0f, 0.0f); // green
    glVertex3f(1.0f, 1.0f, -1.0f);  // p3
    glVertex3f(-1.0f, 1.0f, -1.0f); // p4
glEnd();
```

- `glVertex` erzeugt Punkte in Object coordinates
- OpenGL gibt keine Einheiten für Object coordinates vor

(Quelle i.F.:http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf)

OpenGL Eye coordinates

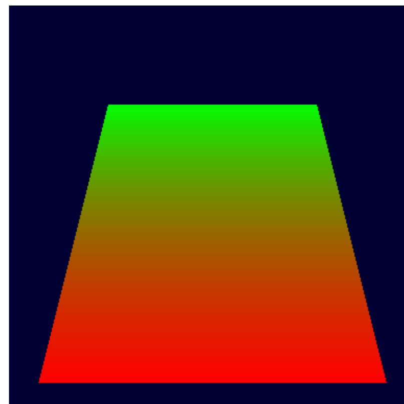
$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} = M \begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix}$$

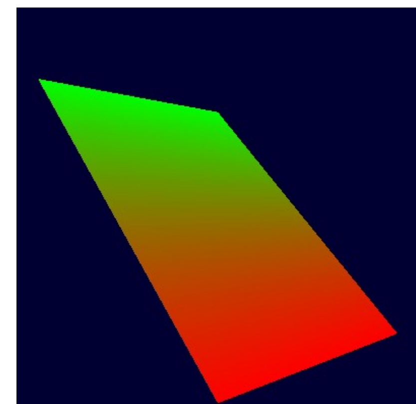
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0, 0, -4);
glRotatef(45, 0, 1, 0);
```

```
1.0   0.0   0.0   0.0
0.0   1.0   0.0   0.0
0.0   0.0   1.0   0.0
0.0   0.0   0.0   1.0
```

```
1.0   0.0   0.0   0.0
0.0   1.0   0.0   0.0
0.0   0.0   1.0  -4.0
0.0   0.0   0.0   1.0
```

```
 0.7071   0.0000   0.7071   0.0000
 0.0000   1.0000   0.0000   0.0000
-0.7071   0.0000   0.7071  -4.0000
 0.0000   0.0000   0.0000   1.0000
```





OpenGL Clip coordinates

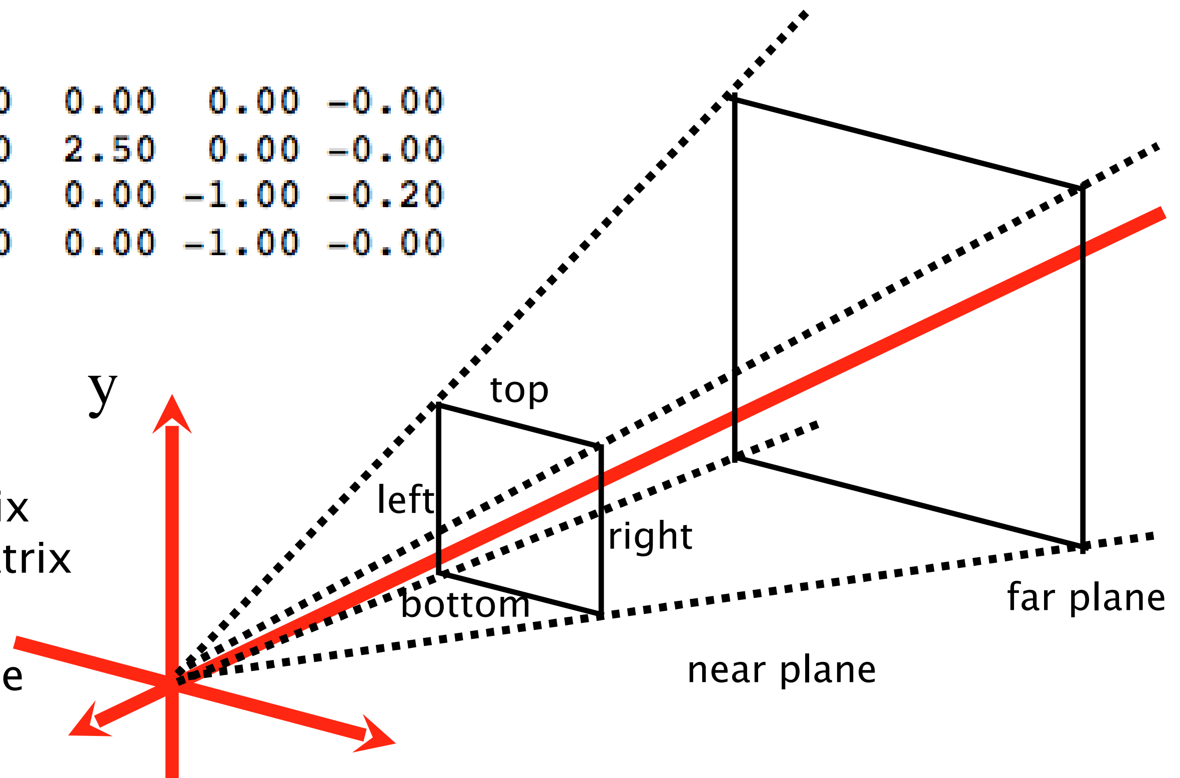
$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = P \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-0.04f, 0.04f, -0.04f, 0.04f, 0.1f, 100);
```

```
1.0  0.0  0.0  0.0
0.0  1.0  0.0  0.0
0.0  0.0  1.0  0.0
0.0  0.0  0.0  1.0
```

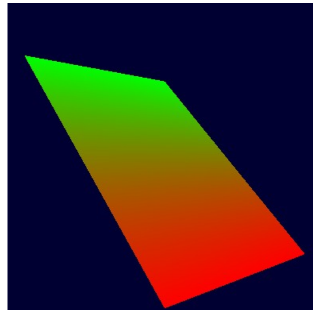
```
2.50  0.00   0.00  -0.00
0.00  2.50   0.00  -0.00
0.00  0.00  -1.00  -0.20
0.00  0.00  -1.00  -0.00
```

- `glFrustum (left, right, bottom, top, near, far)` multipliziert die aktuelle Matrix mit einer perspektivischen Matrix
- `gluPerspective` ist die benutzerfreundlichere Variante



GL_PROJECTION Perspektivisch

```
2.50  0.00  0.00 -0.00
0.00  2.50  0.00 -0.00
0.00  0.00 -1.00 -0.20
0.00  0.00 -1.00 -0.00
```



$$\begin{bmatrix} \frac{2nearVal}{right-left} & 0 & A & 0 \\ 0 & \frac{2nearVal}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$A = \frac{right+left}{right-left}$$

$$B = \frac{top+bottom}{top-bottom}$$

$$C = -\frac{farVal+nearVal}{farVal-nearVal}$$

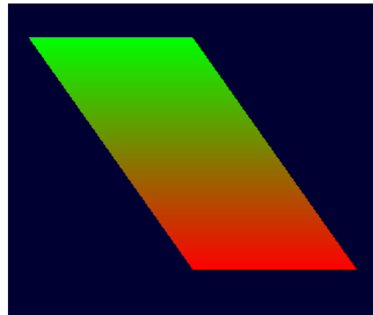
$$D = -\frac{2\,farVal\,nearVal}{farVal-nearVal}$$

- `glFrustum` (s. rechts) und `gluPerspective` produzieren Matrizen die nicht mehr dem Schema $a_4 = b_4 = c_4 = 0$, $d_4 = 1$ folgen!
- Nach der Multiplikation mit GL_PROJECTION entspricht die homogene Komponente w_c der Punkte nicht mehr unbedingt 1

(Quelle:http://www.opengl.org/sdk/docs/man/xhtml/glFrustum.xml)

GL_PROJECTION Orthographisch

```
0.50   0.00 -0.00   0.00
0.00   0.50 -0.00   0.00
0.00   0.00 -0.02  -1.00
0.00   0.00 -0.00   1.00
```



- `glOrtho` (s. rechts) erzeugt Matrizen, die dem Schema $a_4 = b_4 = c_4 = 0$, $d_4 = 1$ folgen!
- Nach der Multiplikation mit GL_PROJECTION ist die homogene Komponente w_c der Punkte weiterhin 1

```
glOrtho(-2, 2, -2, 2, 0.1f, 100);
```

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{farVal-nearVal} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

$$t_x = -\frac{right+left}{right-left}$$

$$t_y = -\frac{top+bottom}{top-bottom}$$

$$t_z = -\frac{farVal+nearVal}{farVal-nearVal}$$

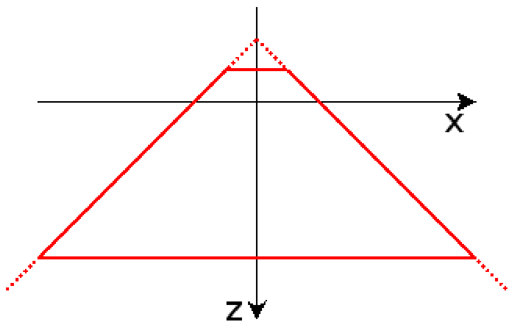
(Quelle:http://www.opengl.org/sdk/docs/man/xhtml/glOrtho.xml)

Normalized Device Coordinates

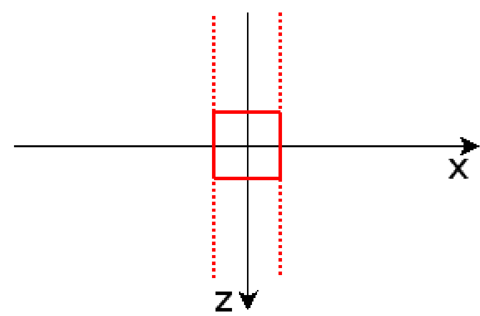
$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{pmatrix}$$

- Clip coordinates liegen im Wertebereich ($-w_c$; $+w_c$)
- Um die Darstellung einheitlich zu machen wird durch w_c geteilt, um die Koordinaten in den Wertebereich (–1; 1) zu bringen

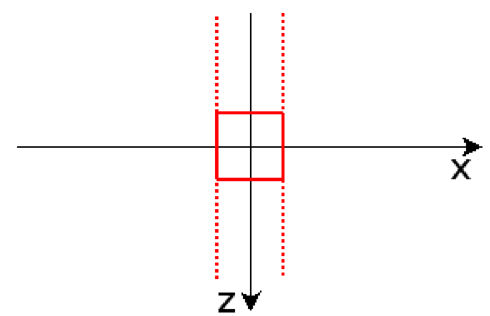
eye coordinates



clip coordinates



normalized device coordinates



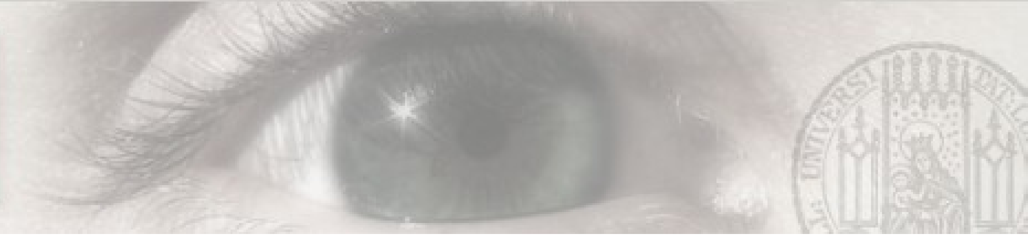
(Quelle:http://homepages.uni-paderborn.de/prefect/glmath/spaces.html)

OpenGL Window coordinates

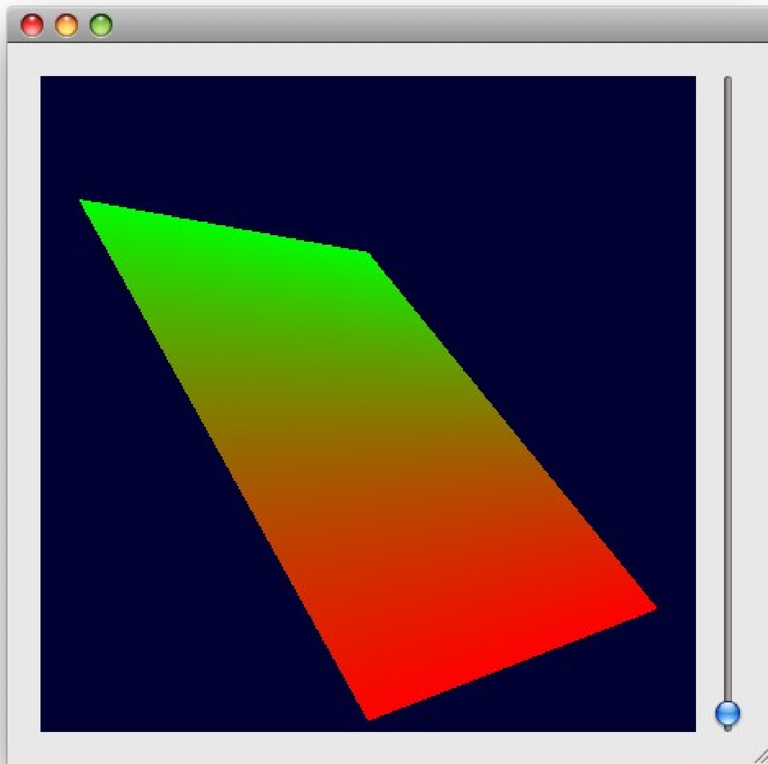
$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} (p_x/2)x_d + o_x \\ (p_y/2)y_d + o_y \\ [(f-n)/2]z_d + (n+f)/2 \end{pmatrix}$$

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

- p_d und p_d sind Breite und Höhe, o_x und o_y der Mittelpunkt des Ausgabefensters
- `glViewport(x0, y0, width, height)` berechnet aus den Ecken des Fensters automatisch p_d, p_d, o_x und o_y
- `n` und `f` sind initial auf 0 und 1 gesetzt, können mit `glDepthRange` verändert werden und haben Auswirkungen auf den Depth Buffer (s.u.)



OpenGL Transformationen - Beispiel



```
void GlTest::resizeGL(int width, int height){
    printf("%d, %d", width, height);
    glViewport(0, 0, (GLint)width, (GLint)height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-0.04f, 0.04f, -0.04f, 0.04f, 0.1f, 100);
    glMatrixMode(GL_MODELVIEW);
}

void GlTest::paintGL(){
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -4);
    glRotatef(45, 0, 1, 0);
    glBegin(GL_POLYGON);
        glColor3f(1.0f, 0.0f, 0.0f); // red
        glVertex3f(-1.0f, -1.0f, 1.0f); // p1
        glVertex3f(1.0f, -1.0f, 1.0f);  // p2
        glColor3f(0.0f, 1.0f, 0.0f); // green
        glVertex3f(1.0f, 1.0f, -1.0f);  // p3
        glVertex3f(-1.0f, 1.0f, -1.0f); // p4
    glEnd();

}
```

..

OpenGL Matrixoperationen

- `glMatrixMode` Wechselt die aktuelle Matrix (`GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE`)

- `glLoadIdentity` Lädt die Identitätsmatrix
- `glMultMatrixf (GLdouble* m)` Multipliziert die aktuelle Matrix mit m

OpenGL Matrixoperationen

- OpenGL hat 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 mindestens zwei aufnehmen

OpenGL Matrixoperationen

- Speichern einer Matrix auf dem Stack:

  ```
  void glPushMatrix();
  ```

- Laden einer Matrix vom Stack:

  ```
  void glPopMatrix();
  ```

- Dadurch lassen sich Transformationen ohne großen Aufwand rückgängig machen

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -10.0f); // pos(1)
glPushMatrix(); // speichern von pos(1)
glTranslatef(2.0f, 0.0f, 0.0f);
glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
// zeichne etwas
glPopMatrix(); // zurück zu pos(1)
glTranslatef(-2.0f, 0.0f, 0.0f);
glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
// zeichne etwas
```

Transformationen (GL_MODELVIEW)

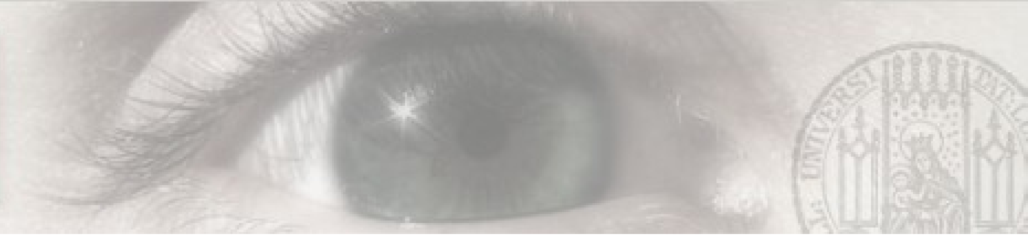
- Verschiedene Möglichkeiten Objekte zu transformieren:
  - `glTranslatef(float x, float y, float z)`
    - Verschiebt alle nachfolgenden Objekte entlang der drei Koordinatenachsen (Translation)
  - `glRotatef(float angle, float x, float y, float z)`
    - Rotiert alle nachfolgenden Objekte um Winkel `angle` (in Grad) um eine beliebige Achse (Rotation)
  - `glScalef(float x, float y, float z)`
    - Skaliert alle nachfolgenden Objekte entlang der drei Koordinatenachsen (Skalierung)

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} xx(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & yy(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & zz(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(Quelle: http://www.opengl.org)

OpenGL Matrixausgabe

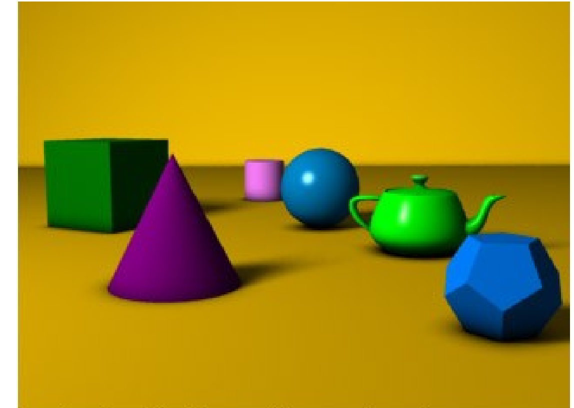
```
GLint viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);
printf("viewport: (%d, %d, %d, %d)\n",
        viewport[0], viewport[1], viewport[2], viewport[3]);

printf("modelview:\n");
GLdouble modelview[16];
glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
for(int i = 0; i < 4; i++){
    printf("%3.4f %3.4f %3.4f %3.4f\n",
            modelview[i], modelview[i + 4], modelview[i + 8], modelview[i + 12]);
}
printf("\n");

printf("projection:\n");
GLdouble projection[16];
glGetDoublev(GL_PROJECTION_MATRIX, projection);
for(int i = 0; i < 4; i++){
    printf("%3.4f %3.4f %3.4f %3.4f\n",
            projection[i], projection[i + 4], projection[i + 8], projection[i + 12]);
}
```



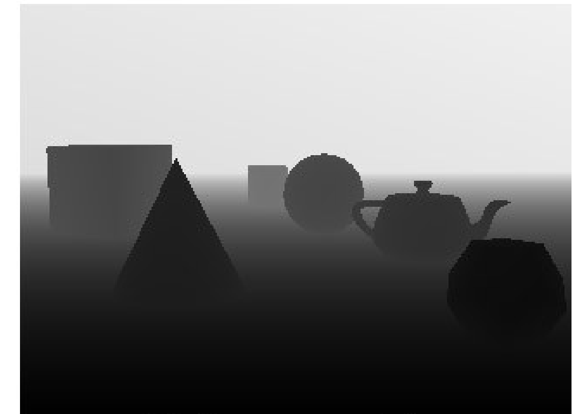
Depth Buffering

Depth Buffering

- Durch die Projektion von 3D nach 2D werden Objekte verdeckt
- Depth Buffering (Z-Buffering) führt dazu, dass es die richtigen trifft (z-Culling)
- Implementierung: Matrix in Bildgröße, die z-Werte enthält

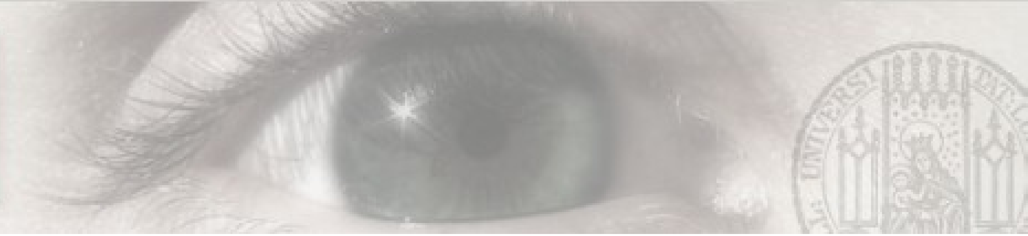


A simple three dimensional scene



Z-buffer representation

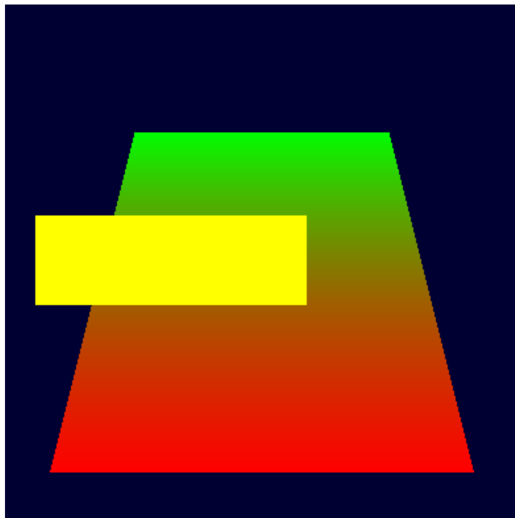
(Quelle: http://en.wikipedia.org/wiki/File:Z-buffer.jpg)

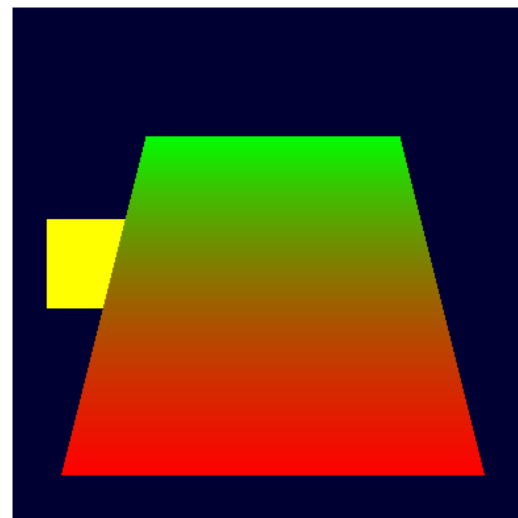


OpenGL Depth Buffer

- Wird der Depth Buffer in OpenGL nicht aktiviert können Objekte im Hintergrund solche im Vordergrund verdecken (Zeichenreihenfolge!)
- `glEnable(GL_DEPTH_TEST)` aktiviert den Depth Buffer
- `glClear(GL_DEPTH_BUFFER_BIT)` löscht den Depth Buffer (am Besten bei jedem Neuzeichnen:

  ```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ```
)





OpenGL Depth Buffer

- Weitere OpenGL Depth Buffer Funktionen:
- `glDepthRange (nearClip, farClip)` setzt n und f auf Werte zwischen 0 und 1 damit nicht der gesamte Depth Buffer ausgenutzt wird
- `glDepthFunc` legt fest wann ein Pixel gezeichnet wird (Gültige Werte: `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`) (Default: `GL_LESS`)
- `glClearDepth` bestimmt bis zu welcher Tiefe der Depth Buffer beim Aufruf von `glClear` gelöscht wird (Default: 1)

Literatur

Weiterführende Literatur

- http://phong.informatik.uni-leipzig.de/~kuska/oglscript/openglsc.pdf
- www.codesampler.org (Codebeispiele)
- http://www.opengl.org/documentation/red_book/
- …

http://farm1.static.flickr.com/194/472097903_b781a0f4f8.jpg

Vielen Dank!

Nächsten Freitag:

Porgrammierberatung