WiSe 20/21

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Prof. Dr. Winfried Kurth
Alex Tavkhelidze

# Praktikum Computergrafik

*Folien zu #2*

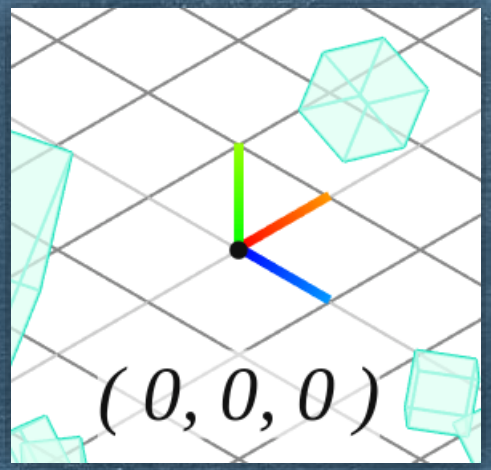
## Viewing-Pipeline
## &
## Transformationen

Stufen

Funktionsübersicht

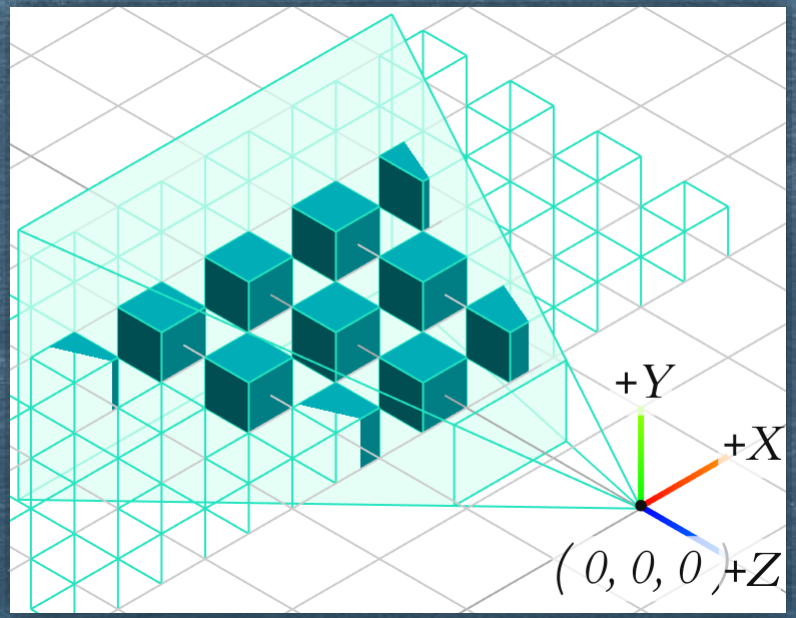
Push-in/Pop-out

# Welt



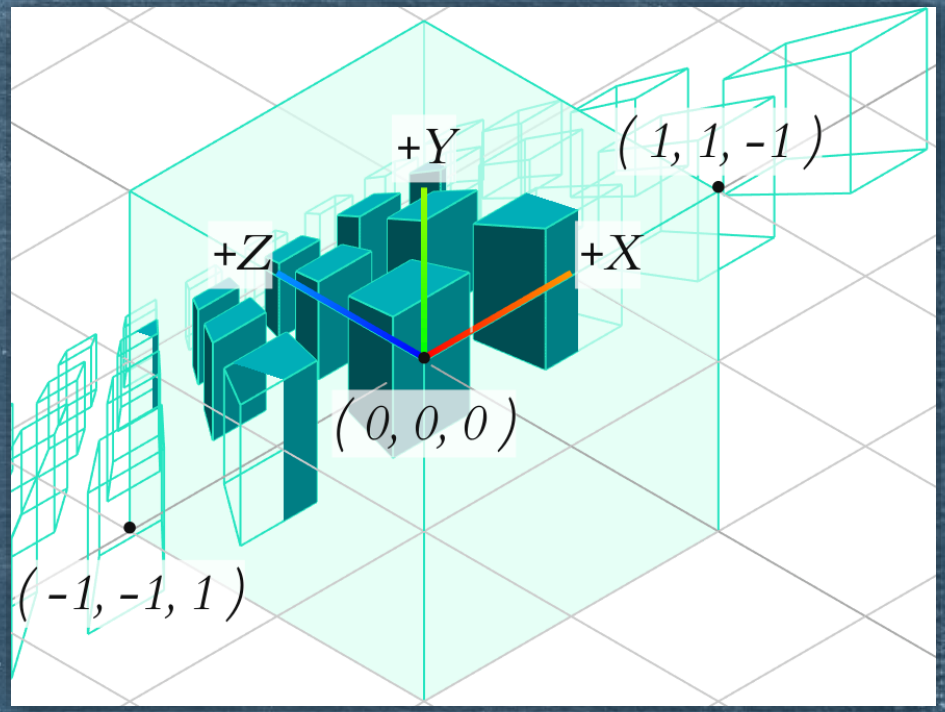

Objekt ⟶ Welt

# Betrachter




Welt ⟶ Kamera

# Auge

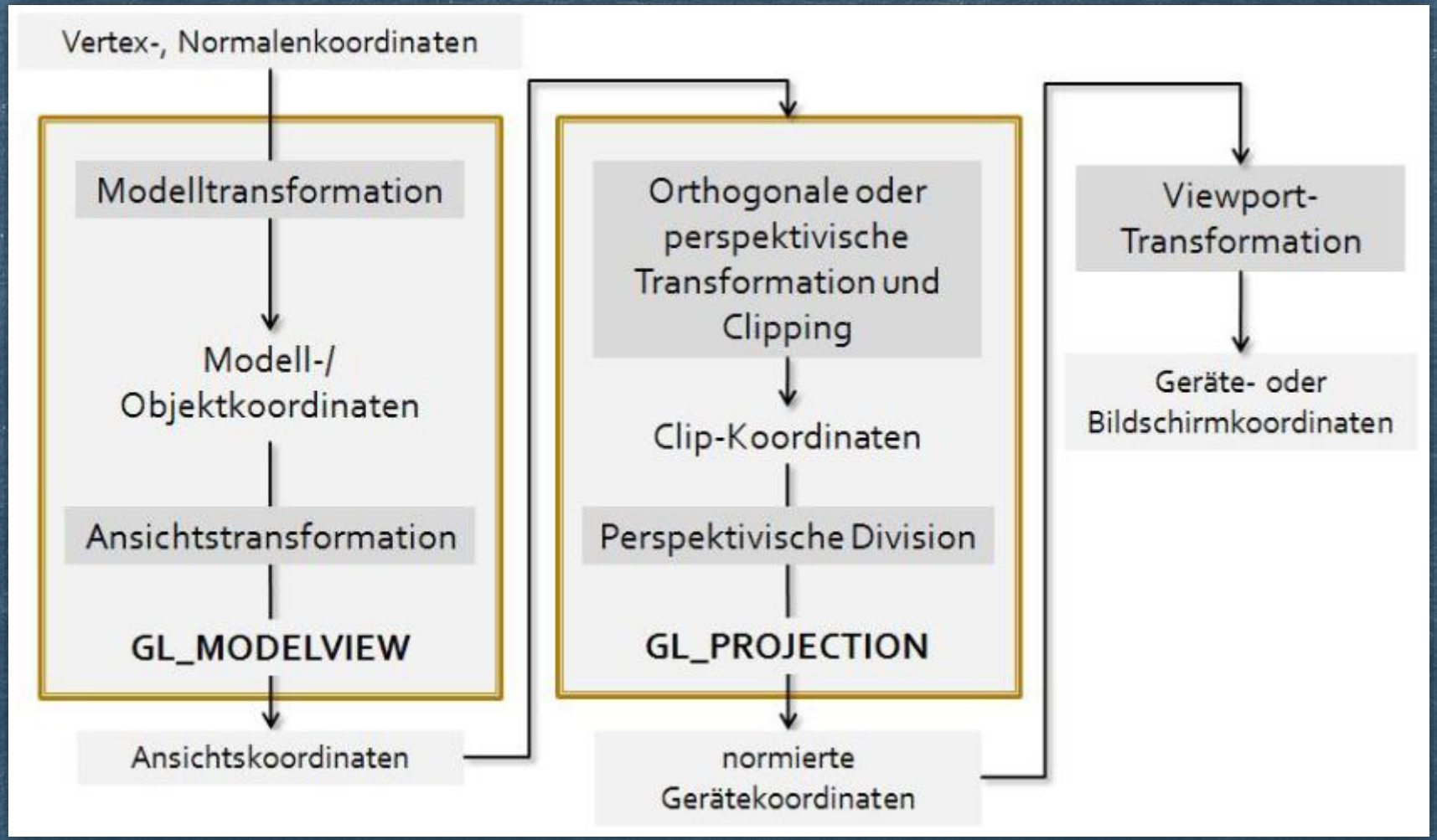



Kamera ⟶ Auge

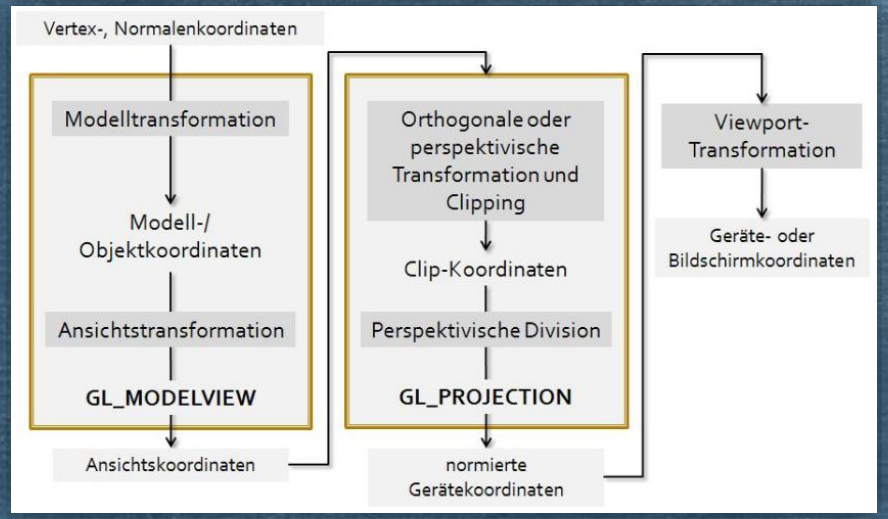
# Transformationen in OpenGL

Bildquelle: https://jsantell.com/model-view-projection/

# Transformationsstufen (1)



# Transformationsstufen (2)



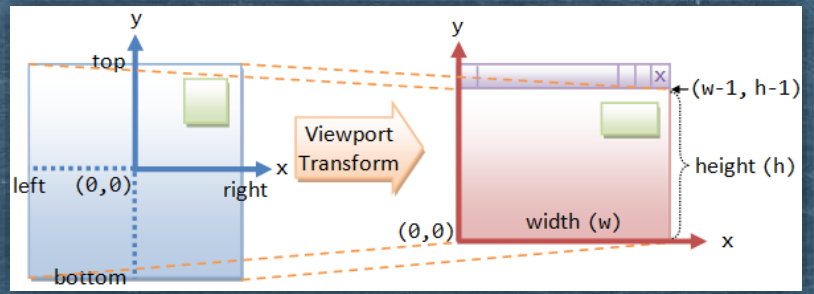

- Nah steht für den Parameter "Nah-Tiefenschnittfläche"
- Weit steht für den Parameter "Weit-Tiefenschnittfläche"
- Bei glFrustum , gluPerspective und glOrtho

## Mathe-Ecke:

$$\begin{pmatrix} x_{Kamera} \\ y_{Kamera} \\ z_{Kamera} \\ w_{Kamera} \end{pmatrix} = M_{MODELVIEW} \cdot \begin{pmatrix} x_{Objekt} \\ y_{Objekt} \\ z_{Objekt} \\ w_{Objekt} \end{pmatrix} \quad \begin{pmatrix} x_{Clip} \\ y_{Clip} \\ z_{Clip} \\ w_{Clip} \end{pmatrix} = M_{PROJECTION} \cdot \begin{pmatrix} x_{Kamera} \\ y_{Kamera} \\ z_{Kamera} \\ w_{Kamera} \end{pmatrix}$$

$$M_{MODELVIEW} = M_{MODEL} \cdot M_{VIEW}$$

Wie wird das Seitenverhältnis aka „*aspect ratio*" beibehalten?




$$\begin{pmatrix} x_{normiert} \\ y_{normiert} \\ z_{normiert} \end{pmatrix} = \begin{pmatrix} \frac{x_{Clip}}{w_{Clip}} \\ \frac{y_{Clip}}{w_{Clip}} \\ \frac{z_{Clip}}{w_{Clip}} \end{pmatrix}$$

$$\begin{pmatrix} x_{VIEWPORT} \\ y_{VIEWPORT} \\ z_{VIEWPORT} \end{pmatrix} = \begin{pmatrix} x_{ul} + \frac{Breite}{2} \cdot (1 + x_{normiert}) \\ y_{ul} + \frac{Höhe}{2} \cdot (1 + y_{normiert}) \\ \frac{Weit - Nah}{2} \cdot z_{normiert} + \frac{Weit + Nah}{2} \end{pmatrix}$$

glViewport($x_{ul}$, $y_{ul}$, Breite, Höhe)

# Funktionsübersicht

**Bibliothek**

com.jogamp.opengl.glu.GLU

**Befehl**

| **Projektion:** | **Kamera:** |
|---|---|
| *perspektivisch*<br>gluPerspective() | einrichten<br>gluLookAt() |

com.jogamp.opengl.GL2

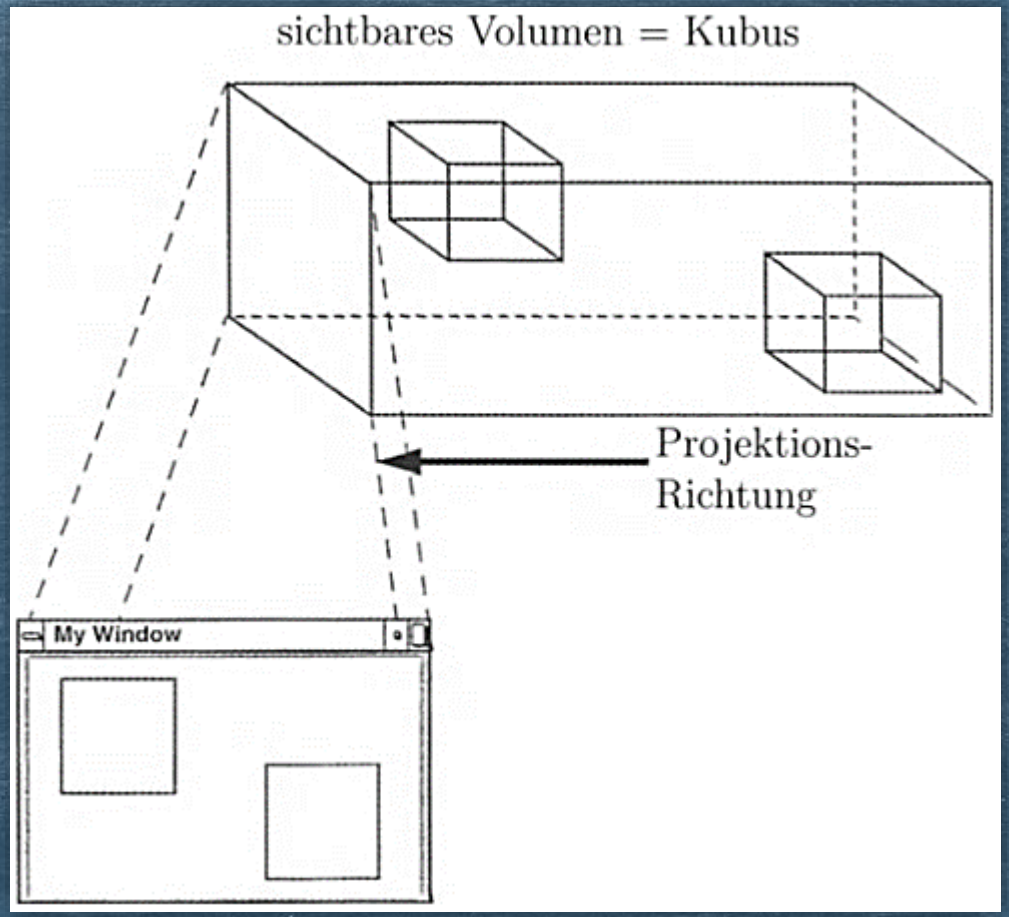
| **Projektion:** | **Transformationsbausteine:** | |
|---|---|---|
| *perspektivisch*<br>glFrustum() | glTranslate() | glMatrixMode() |
| *orthogonal*<br>glOrtho() | glRotate() | glLoadIdentity() |
| *viewport*<br>glViewport() | glScale() | glLoadMatrix()<br>glMultMatrix() |

# glOrtho(left, right, bottom, top, z_near, z_far)

```
public void display(GLAutoDrawable drawable)
{
  GL2 gl = drawable.getGL().getGL2();
  //
  gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  gl.glClear(GL.GL_COLOR_BUFFER_BIT);
  gl.glColor3f(1.0f, 1.0f, 1.0f);
  gl.glMatrixMode(GL2.GL_PROJECTION);
  gl.glLoadIdentity();
  gl.glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f);
```

```
public void reshape(GLAutoDrawable drawable, int x, int y, int w, int h) {
    GL2 gl = drawable.getGL().getGL2();
    //
    gl.glViewport(0, 0, w, h);
    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glOrtho(0.0, 8.0, 0.0, 8.0, -0.5, 2.5);
```

- Standardzustand:
  - glOrtho(-1, 1, -1, 1, 1, -1)

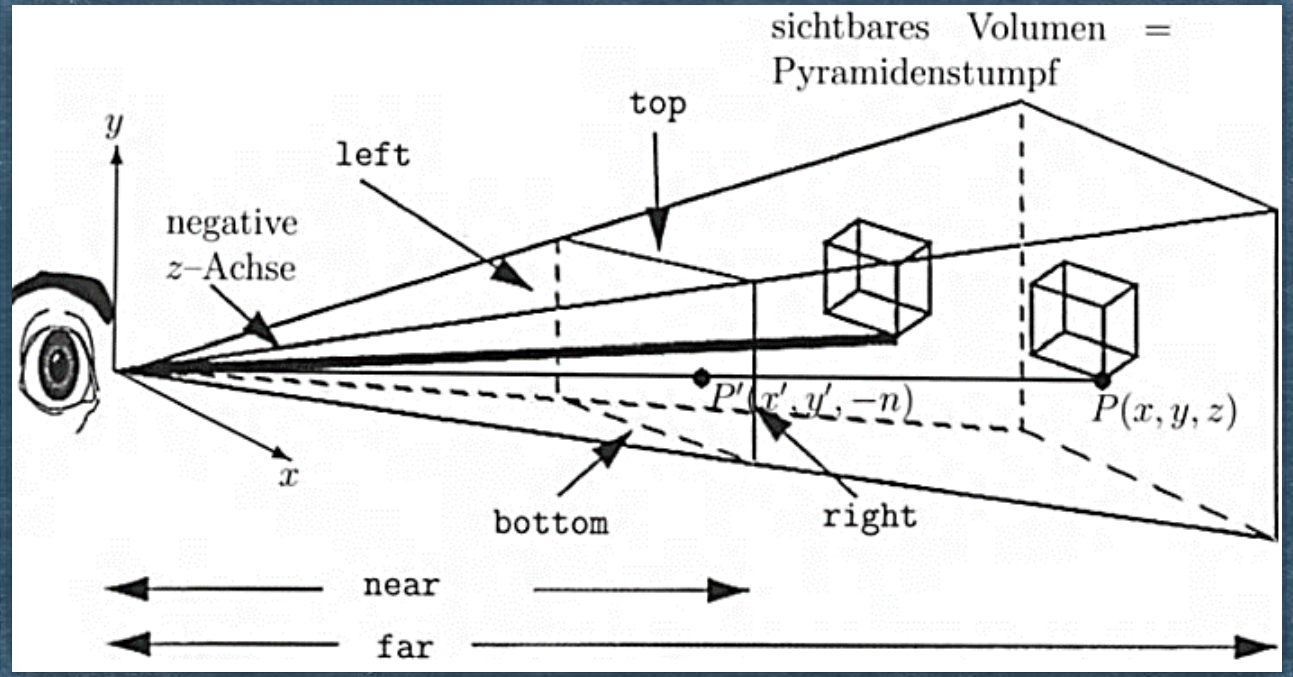



Bildquelle: https://doi.org/10.1007/978-3-322-92916-7

# glFrustum(left, right, bottom, top, z_near, z_far)

```
  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
System.out.println("Listener.reshape()");
GL2 gl = drawable.getGL().getGL2();

float h = (float)height / (float)width;

gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glLoadIdentity();
gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
```

```
gl.glMatrixMode(GL2.GL_PROJECTION);

gl.glLoadIdentity();

if (h<1)
        gl.glFrustum(-1.0f, 1.0f, -h, h, 1.0f, 60.0f);
else
{
        h = 1.0f/h;
        gl.glFrustum(-h, h, -1.0f, 1.0f, 1.0f, 60.0f);
}
```



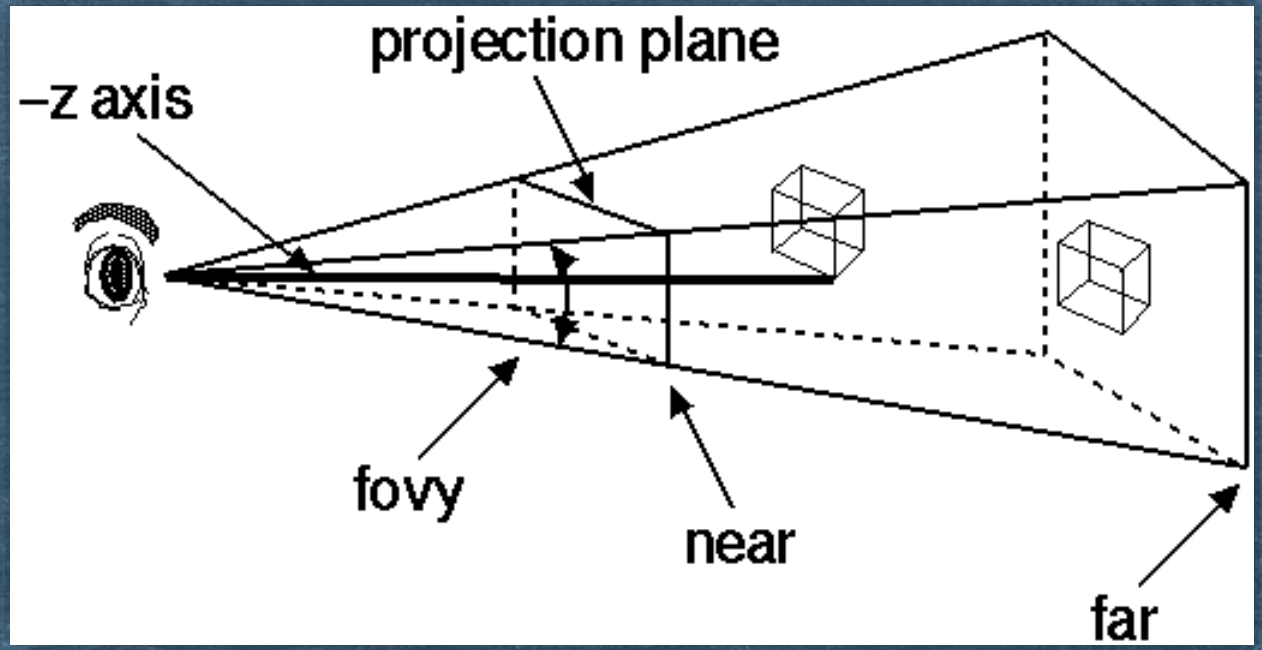

Bildquelle: https://doi.org/10.1007/978-3-322-92916-7

# gluPerspective(fovy, aspect, z_near, z_far)

```
public void reshape(GLAutoDrawable drawable, int x, int y, int w, int h) {
    GL2 gl = drawable.getGL().getGL2();
    //
    gl.glViewport(0, 0, w, h);
    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();
    glu.gluPerspective(45.0, (float) w / (float) h, 3.0, 5.0);
```

```
public void display(GLAutoDrawable drawable)
{

  GL2 gl = drawable.getGL().getGL2();

  // The usual OpenGL stuff to clear the screen and set up viewing.
  gl.glClearColor(.25f, .25f, .25f, 1.0f);
  gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

  gl.glMatrixMode(GL2ES1.GL_PROJECTION);
  gl.glLoadIdentity();
  glu.gluPerspective(30.0f, 1.0f, .1f, 100);
```

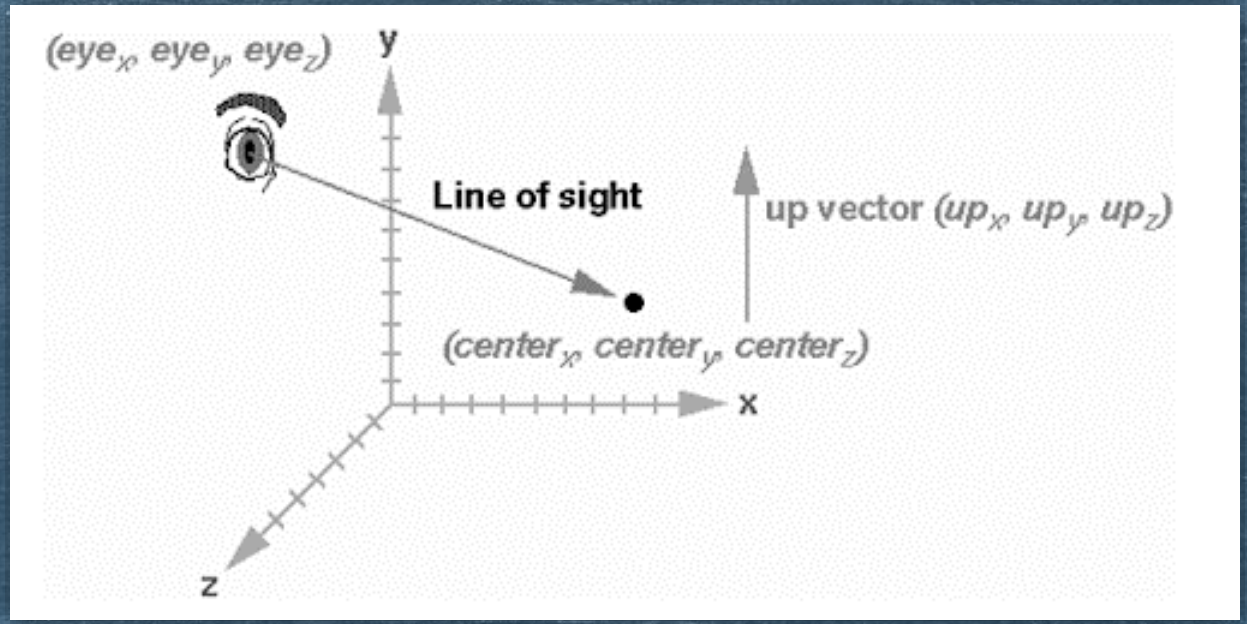
```
public void reshape(GLAutoDrawable drawable, int x, int y, int w, int h) {
    GL2 gl = drawable.getGL().getGL2();
    //
    gl.glViewport(0, 0, w, h);
    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();
    glu.gluPerspective(45.0, (float) w / (float) h, 3.0, 5.0);
```



# gluLookAt(eye_x, eye_y, eye_z, center_x, center_y, center_z, up_x, up_y, up_z)

```
public void reshape(GLAutoDrawable drawable, int x, int y, int width,
        int height) {
    GL2 gl = drawable.getGL().getGL2();
    GLU glu = new GLU();
    //
    gl.glViewport(0, 0, width, height);
    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();
    glu.gluPerspective(30, (float) width / (float) height, 1.0, 100.0);
    gl.glMatrixMode(GL2.GL_MODELVIEW);
    gl.glLoadIdentity();
    glu.gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
}
```

```
public void display(GLAutoDrawable arg0) {
        GL2 gl = arg0.getGL().getGL2();

        if(reloadShaders) {
                reloadShaders = false;
                ReloadShaders(gl);
        }
        GLU glu = GLU.createGLU(gl);

        g_numGeoPasses = 0;

        gl.glMatrixMode(GL2.GL_MODELVIEW);
        gl.glLoadIdentity();
        glu.gluLookAt(g_pos[0], g_pos[1], g_pos[2], g_pos[0], g_pos[1], 0, 0, 1, 0);
        gl.glRotatef(g_rot[0], 1, 0, 0);
        gl.glRotatef(g_rot[1], 0, 1, 0);
        gl.glTranslatef(g_bbTrans[0], g_bbTrans[1], g_bbTrans[2]);
        gl.glScalef(g_bbScale, g_bbScale, g_bbScale);
```

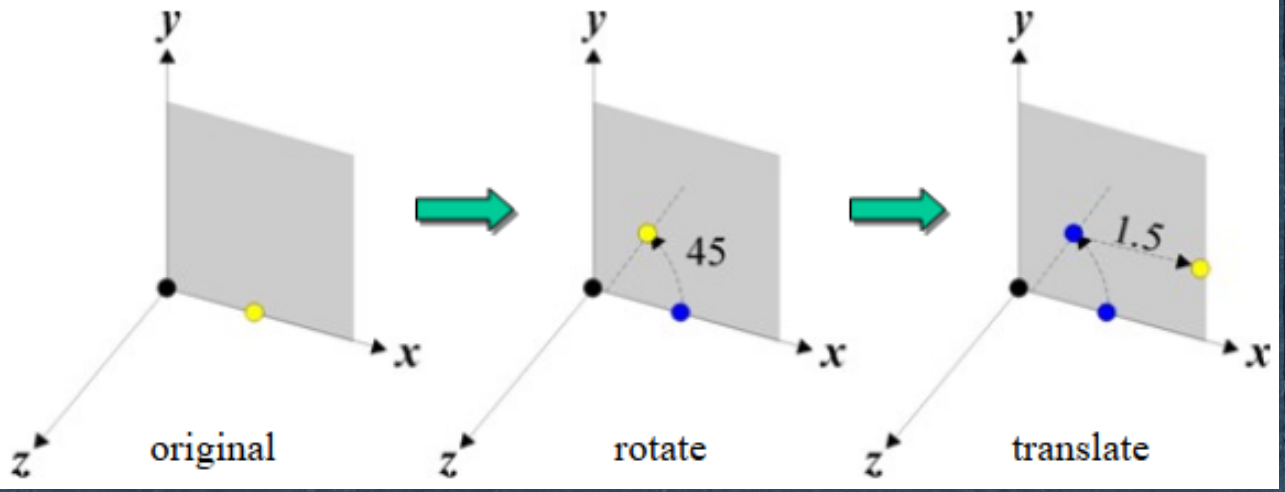



- Wie sieht eine äquivalente Darstellung mithilfe der Befehle glMultMatrix und glTranslate aus?
  Erstellt entsprechende Rotations- und Verschiebungsmatrizen (die obigen 3 Vektoren sind die Eingabedaten).

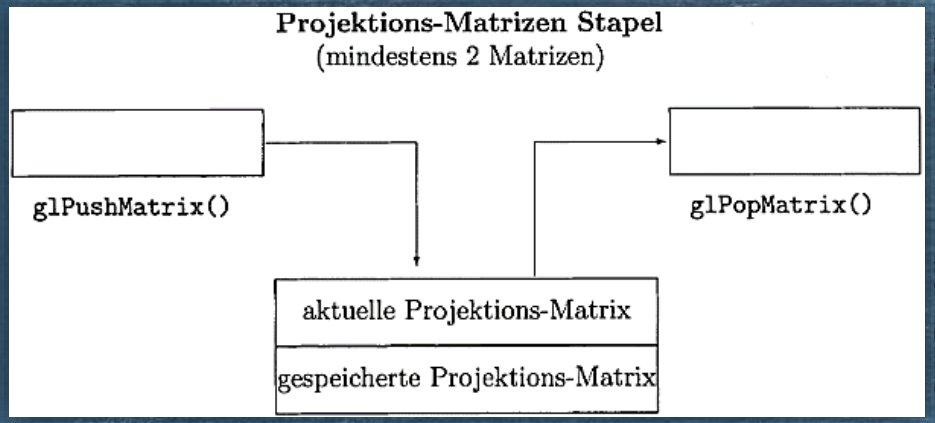
# Transformationen: die **Reihenfolge** beim Angeben

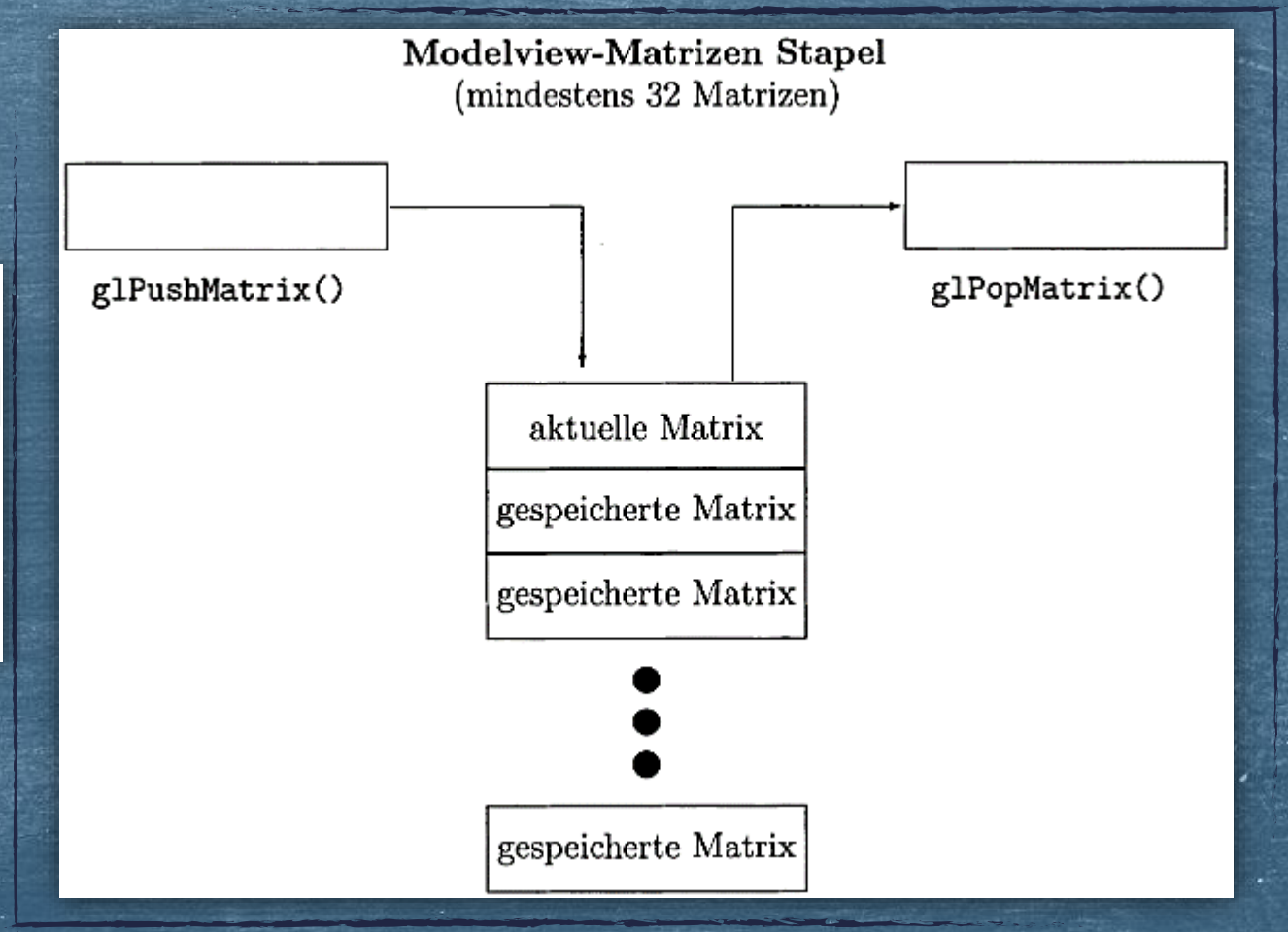
- Der letzte im Quelltext angegebene Befehl wird zuerst ausgeführt

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.5, 0.0, 0.0);
glRotatef(45.0, 0.0, 0.0, 1.0);
glVertex3f(1.0, 0.0, 0.0);
```



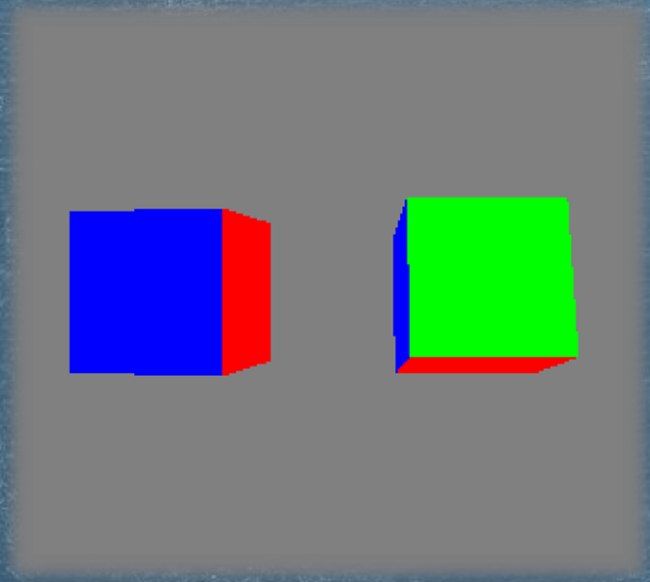
# Matrizen-Stapel: glPushMatrix() & glPopMatrix()




*Bildquelle: https://doi.org/10.1007/978-3-322-92916-7*

# Matrizen-Stapel: Beispiel



```
public void display(GLAutoDrawable drawable) {
    ....
    gl.glLoadIdentity(); gl.glTranslatef(0.0f,
    0.0f, -10.0f);

    // save matrix state
    gl.glPushMatrix();
        gl.glTranslatef(-2.0f, 0.0f, 0.0f);
        gl.glRotatef(angle, 0.0f, 1.0f, 0.0f);
        drawCube(gl);
    // restore matrix state
    gl.glPopMatrix();
    gl.glTranslatef(2.0f, 0.0f, 0.0f);
    gl.glRotatef(angle, 1.0f, 0.0f, 0.0f);
    drawCube(gl);
}
```

# MODELVIEW, Pushs & Pops: Schritt-für-Schritt Schema