Aufgabe 19

Generell sind homogene Koordinaten von Vorteil, weil dann die Projektionsmatrix problemlos mit den Transformationsmatrizen verknüpft werden kann.
Speziell bei der Zentralprojektion kann man die Eigenschaft ausnutzen, dass die sogenannte w-Koordinate der Überführung in den R^3 dient:

$$x'' = \frac{x'}{w'},\ y'' = \frac{y'}{w'},\ z'' = \frac{z'}{w'}$$

Da bei einer Matrizenmultiplikation nur Additionen und Multiplikationen auftauchen, wird an dieser Stelle die Division genutzt, um die bei der Zentralprojektion notwendige Verkürzung durchzuführen. Sie wird als letztes durchgeführt:

$$\begin{pmatrix} x'' \\ y'' \\ z'' \\ w'' \end{pmatrix} = M_{per} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

mit

$$M_{per} = D(near,\ far) \cdot S_{xyz}\left(\frac{1}{far}\right) \cdot S_{xy}(\theta_W, \theta_H) \cdot R(Look, Up)^T \cdot T(-P)$$

Unter Ausnutzung der Tatsache, dass sich für die Matrix

$$k = \frac{near}{far}$$

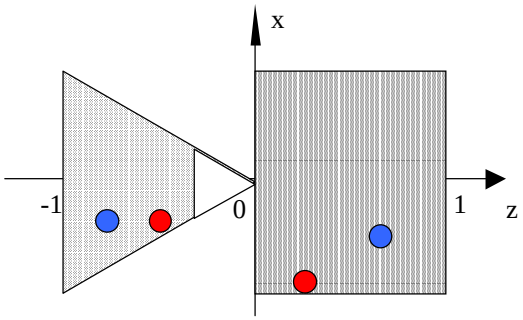
$$D(near,\ far) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(k-1) & k/(k-1) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$w'' = -z'$ ergibt, bekommt man in R^3:

$$(x' \quad y' \quad z' \quad w')^T = \left(\frac{x'}{w'} \quad \frac{y'}{w'} \quad \frac{z'}{w'}\right)^T = \left(-\frac{x'}{z'} \quad -\frac{y'}{z'} \quad -\frac{z'+k}{z' \cdot (k-1)}\right)^T$$

Die Projektionsmatrix gehört zur Transformationsklasse der Scherungen.

In einer Prinzipskizze wird die perspektivische Verkürzung dadurch sichtbar, dass aus einer Sichtpyramide (symbolisiert durch ein Dreieck) ein Quader wird (als Rechteck gezeichnet), die beiden Objekte – rot und blau – ändern entsprechend der Verzerrung ihre x- und y-Koordinate.



Aufgabe 20

Die allgemeine Transformationsfolge hat die Gestalt:

$$M_{per} = D(near, far) \cdot S_{xyz}\left(\frac{1}{far}\right) \cdot S_{xy}(\theta_W, \theta_H) \cdot R(Look, Up)^T \cdot T(-P)$$

a) Fast alle Parameter sind explizit bekannt, nur θ_H muss noch berechnet werden. Dazu nutze ich das bereits bekannte Aspect Ratio:

$$\begin{aligned}\theta_H &= \frac{1}{3:2} \cdot \theta_W \\ &= \frac{2}{3} \cdot 90° \\ &= 60°\end{aligned}$$

Einsetzen aller Werte in die entsprechenden 4x4-Matrizen liefert:

$$D(near, far) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(k-1) & k/(k-1) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$k = \frac{1}{6}$$

$$D(1,6) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{6}{5} & -1/5 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$S_{xyz}\left(\frac{1}{far}\right) = \begin{pmatrix} 1/far & 0 & 0 & 0 \\ 0 & 1/far & 0 & 0 \\ 0 & 0 & 1/far & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S_{xyz}\left(\frac{1}{6}\right) = \begin{pmatrix} 1/6 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 \\ 0 & 0 & 1/6 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S_{xy}(\theta_W,\theta_H)=\begin{pmatrix}\cot\frac{\theta_W}{2} & 0 & 0 & 0\\ 0 & \cot\frac{\theta_H}{2} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$\cot\frac{90°}{2}=1$$

$$\cot\frac{60°}{2}=\frac{\cos 30°}{\sin 30°}=\frac{\sqrt{3}}{2}:\frac{1}{2}=\sqrt{3}$$

$$S_{xy}(\theta_W,\theta_H)=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \sqrt{3} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$R(Look,Up)^T=\begin{pmatrix}r_{ux} & r_{vx} & r_{nx} & 0\\ r_{uy} & r_{vy} & r_{ny} & 0\\ r_{uz} & r_{vz} & r_{nz} & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$n=-\frac{Look}{\|Look\|}=\begin{pmatrix}-3\\-4\\0\end{pmatrix}\cdot\left(-\frac{1}{5}\right)=\begin{pmatrix}3/5\\4/5\\0\end{pmatrix}$$

$$v=\frac{Up-(Look\bullet Up)\cdot Look}{\|Up-(Look\bullet Up)\cdot Look\|}=\begin{pmatrix}-4\\3\\0\end{pmatrix}\cdot\frac{1}{5}=\begin{pmatrix}-4/5\\3/5\\0\end{pmatrix}$$

$$u=\frac{v\times n}{\|v\times n\|}=\begin{pmatrix}-4/5\\3/5\\0\end{pmatrix}\times\begin{pmatrix}3/5\\4/5\\0\end{pmatrix}=\begin{pmatrix}0\\0\\-1\end{pmatrix}$$

$$R\left(\begin{pmatrix}-3\\-4\\0\end{pmatrix},\begin{pmatrix}-4\\3\\0\end{pmatrix}\right)^T=\begin{pmatrix}0 & -4/5 & 3/5 & 0\\ 0 & 3/5 & 4/5 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$T(-P)=\begin{pmatrix}1 & 0 & 0 & -x_p\\ 0 & 1 & 0 & -y_p\\ 0 & 0 & 1 & -z_p\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$T(-O)=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Die Orientierungsmatrix ergibt sich für die Parameter der Aufgabenstellung als:

$$M_{orientation}(P,Look,Up)=R(Look,Up)^T\cdot T(-P)$$

$$=\begin{pmatrix}0 & -4/5 & 3/5 & 0\\ 0 & 3/5 & 4/5 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$=\begin{pmatrix}0 & -4/5 & 3/5 & 0\\ 0 & 3/5 & 4/5 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Für die Projektionsmatrix gilt im Beispiel:

$$M_{proj}(near,\ far,\theta_W,\theta_H)$$

$$=D(near,\ far)\cdot S_{xyz}\left(\frac{1}{far}\right)\cdot S_{xy}(\theta_W,\theta_H)$$

$$=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & -\frac{6}{5} & -1/5\\ 0 & 0 & -1 & 0\end{pmatrix}\cdot\begin{pmatrix}1/6 & 0 & 0 & 0\\ 0 & 1/6 & 0 & 0\\ 0 & 0 & 1/6 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \sqrt{3} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$=\begin{pmatrix}1/6 & 0 & 0 & 0\\ 0 & \sqrt{3}/6 & 0 & 0\\ 0 & 0 & -1/5 & -1/5\\ 0 & 0 & -1/6 & 0\end{pmatrix}$$

Die Verknüpfung von Orientierungs- und Projektionsmatrix führt zu:

$$M_{pers}(near,\ far, \theta_W, \theta_H, P, Look, Up) = M_{proj}(near,\ far, \theta_W, \theta_H) \cdot M_{orientation}(P, Look, Up)$$

Ausmultipliziert bekommt man für die Aufgabe:

$$M_{pers} = \begin{pmatrix} 1/6 & 0 & 0 & 0 \\ 0 & \sqrt{3}/6 & 0 & 0 \\ 0 & 0 & -1/5 & -1/5 \\ 0 & 0 & -1/6 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & -4/5 & 3/5 & 0 \\ 0 & 3/5 & 4/5 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -2/15 & 1/10 & 0 \\ 0 & \sqrt{3}/10 & 2\sqrt{3}/15 & 0 \\ 1/5 & 0 & 0 & -1/5 \\ 1/6 & 0 & 0 & 0 \end{pmatrix}$$

b) Wenn man die oben angegebenen Matrizen symbolisch auswertet, so hat die finale Matrix die Form:

$$M_{pers} = \begin{pmatrix} \frac{u_x \cdot \cot\frac{\theta_W}{2}}{far} & \frac{v_x \cdot \cot\frac{\theta_W}{2}}{far} & \frac{n_x \cdot \cot\frac{\theta_W}{2}}{far} & -\frac{u_x p_x + v_x p_y + n_x p_z}{far} \cdot \cot\frac{\theta_W}{2} \\ \frac{u_y \cdot \cot\frac{\theta_H}{2}}{far} & \frac{v_y \cdot \cot\frac{\theta_H}{2}}{far} & \frac{n_y \cdot \cot\frac{\theta_H}{2}}{far} & -\frac{u_y p_x + v_y p_y + n_y p_z}{far} \cdot \cot\frac{\theta_H}{2} \\ \frac{u_z}{near - far} & \frac{v_z}{near - far} & \frac{n_z}{near - far} & \frac{u_z p_x + v_z p_y + n_z p_z - near}{far - near} \\ -\frac{u_z}{far} & -\frac{v_z}{far} & -\frac{n_z}{far} & \frac{u_z p_x + v_z p_y + n_z p_z}{far} \end{pmatrix}$$

Um aus der gegebenen Matrix die einzelnen Variablen extrahieren zu können, müssen für die einzelnen Positionen in der Matrix die entsprechenden Gleichungen gelöst werden:

$$M_{pers} = \begin{pmatrix} -1/2 & 0 & 0 & -1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1/2 \end{pmatrix}$$

Die Betrachtung der vielen Nullen lässt Rückschlüsse auf die Koordinaten von u, v und n zu:

$$u = \begin{pmatrix} -\frac{1}{2}\, far \cdot \tan\frac{\theta_W}{2} \\ 0 \\ 0 \end{pmatrix},\ v = \begin{pmatrix} 0 \\ \frac{1}{2}\, far \cdot \tan\frac{\theta_H}{2} \\ 0 \end{pmatrix},\ n = \begin{pmatrix} 0 \\ 0 \\ -\frac{far}{2} \end{pmatrix}$$

Da alle drei Vektoren per Definition normiert sind, kann n ermittelt werden:

$$n = \begin{pmatrix} 0 \\ 0 \\ -\frac{far}{2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

$$far = 2$$

Es sollte erwähnt werden, dass auch $far = -2$ möglich wäre, allerdings liefern dann die weiteren Rechenschritte unzulässige Ergebnisse.

Eine erneute Ausnutzung der Normiertheit liefert für u:

$$1 = \left| -\frac{1}{2}\, far \cdot \tan\frac{\theta_W}{2} \right|$$
$$1 = \tan\frac{\theta_W}{2}$$
$$\theta_W = 90°$$

Und für v:

$$1 = \left| \frac{1}{2}\, far \cdot \tan\frac{\theta_H}{2} \right|$$
$$1 = \tan\frac{\theta_H}{2}$$
$$\theta_H = 90°$$

Aus m_{33} folgt:

$$-\frac{1}{near - far} = -1$$
$$near = 1$$

Anschließend muss die Kameraposition p ermittelt werden, dazu eignet sich die 4.Spalte der Matrix:

$$-\frac{1}{2} = -\frac{u_x p_x + v_x p_y + n_x p_z}{far} \cdot \cot\frac{\theta_W}{2}$$
$$p_x = -1$$
$$\frac{1}{2} = -\frac{u_y p_x + v_y p_y + n_y p_z}{far} \cdot \cot\frac{\theta_H}{2}$$
$$p_y = -1$$
$$\frac{1}{2} = \frac{u_z p_x + v_z p_y + n_z p_z}{far}$$
$$p_z = -1$$

Für die Kameravektoren gilt (unter Verwendung der Tatsache, dass u, v und n jeweils senkrecht aufeinander stehen):

$$Look = -n = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$
$$Up = v = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Zusammengefasst ermittelte ich folgende Ergebnisse:

- Position der Kamera: $(-1,-1,-1)^T$
- Blickrichtungsvektor: $Look = (0,0,1)^T$
- Aufwärtsvektor: $Up = (0,1,0)^T$
- Öffnungswinkel: $\theta_W = \theta_H = 90°$
- Clipping-Planes: $near = 1,\ far = 2$

Aufgabe 21

Der Abstand *distance* der Kamera vom Ursprung definiert den Radius einer Kugel. Auf deren Oberfläche wird mit den Winkeln *azimuth* (geographische Länge) und *elevation* (geographische Breite) ein Punkt eindeutig bestimmt. Dessen Rotation um die Sichtrichtung bestimmt *twist*.

Die Position der Kamera bestimmt man durch einfache Benutzung der Polarkoordinaten:

$$P = \begin{pmatrix} \cos(elevation) \cdot \cos(azimuth) \\ \cos(elevation) \cdot \sin(azimuth) \\ \sin(elevation) \end{pmatrix} \cdot distance$$

Sie schaut stets in den Ursprung:

$$Look = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Die Rotation um die Kameraachse erhält man, wenn man die y-Achse als den Standard-Up-Vektor annimmt und diesen entsprechend auch mitdreht, was sich auch gut vereinfachen lässt zu:

$$Up = \begin{pmatrix} \cos(elevation) \cdot \sin(twist) \\ \cos(elevation) \cdot \cos(twist) \\ \sin(elevation) \end{pmatrix}$$

Der OpenGL-Pseudocode kann ausnutzen, dass Rotationen um beliebige Achsen in diesem Grafiksystem bereits implementiert sind (`glRotate`) und sieht dann wie folgt aus:

```
// von Grad in Radiant umrechnen
elev = elevation*pi/180;
azim = azimuth*pi/180;
tw   = twist*pi/180;

// Kamerastandpunkt
px = distance*cos(elev)*cos(azim);
py = distance*cos(elev)*sin(azim);
pz = distance*sin(elev);

// Up-Vektor
ux = cos(elev)*sin(tw);
uy = cos(elev)*cos(tw);
uz = sin(elev);

// und einrichten
gluLookAt(px, py, pz, 0, 0, 0, ux, uy, uz);
```

Ein völlig anderes Ansatz beruht darauf, dass die Kamera im Ursprung bleibt und die gesamte Szene verschoben wird. Man kann sich sogar `gluLookAt` ersparen:

```
glLoadIdentity();                       // die negative z-Achse entlang
glScale(0, 0, -1);                      // Blickrichtung umkehren
glTranslatef(0, 0, distance);           // Abstand (Radius) herstellen
glRotatef(-twist, 0, 0, 1);             // um die eigene Achse drehen
glRotatef(-elevation, 1, 0, 0);         // „geogr. Breite“
glRotatef(azimuth, 0, 0, 1);            // „geogr. Länge“
```

Aufgabe 22

Die Darstellung eines Modells in mehreren Ansichten kann über mehrere Tasten gesteuert werden, wobei ich natürlich noch auf die Möglichkeit der Rotation ähnlich Drag'n'Drop hinweisen will:

Taste	Aktion
+	in die Hauptansicht hineinzoomen
-	aus der Hauptansicht herauszoomen
1	in die xy-Ansicht hineinzoomen
2	aus der xy-Ansicht herauszoomen
3	in die xz-Ansicht hineinzoomen
4	aus der xz-Ansicht herauszoomen
5	in die yz-Ansicht hineinzoomen
6	aus der yz-Ansicht herauszoomen
l	orthogonale Modelle ausfüllen ein/aus
L	Hauptmodell ausfüllen ein/aus
Leertaste	automatische Rotation
ESC	Programm beenden

Die rot markierten Einstellungen entsprechen den Standardwerten. Mit dem Porsche-Modell bekommt man dann sehr schöne Bilder (Achtung: da der Porsche zu groß modelliert wurde, muss man relativ stark herauszoomen, um das nachfolgende Bild zu erhalten):

Die Kernaufgabe erfüllt die Methode `OnDraw`. Sie muss für jede zu zeichnende Ansicht folgende Schritte durchlaufen:

1. Zeichenbereich innerhalb des Fensters festlegen (`glViewport`)
2. auf die Projektionsmatrix umschalten (`glMatrixMode`)
3. Einheitsmatrix laden (`glLoadIdentity`)
4. Projektionsmatrix generieren (`glOrtho` oder `gluPerspective`)
5. zur Model-View-Matrix wechseln (`glMatrixMode`)
6. Einheitsmatrix laden (`glLoadIdentity`)
7. Sichtvolumen setzen (`gluLookAt`)
8. Szene zeichnen

Die orthogonale Darstellung der xy-Ansicht erfolgt z.B. mit:

```
// zeichne Ansicht x,y
glViewport(maxx-orthosize, 2*orthosize, orthosize, orthosize);
glMatrixMode(GL_PROJECTION);
// orthogonale Sicht
glLoadIdentity();
glOrtho(-1/zoomxy_, 1/zoomxy_, -1/zoomxy_, 1/zoomxy_, 0.1, 100);
// entlang z-Achse
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0,0,10, 0,0,0, 0,1,0);
glColor3f(0,0,1);
drawScene();
```

Der Zoomfaktor kommt in Schritt 4 zum tragen, indem er entweder den Öffnungswinkel (perspektivisch) oder die Clippingebenen (orthogonal) beeinflusst.

Quellcode

Die Anwendungsrahmenklasse wurde dahingehend abgeändert, dass sie ein interaktives Drehen mit gedrückter Maustaste zulässt. Da dieser Code schon für das letzte Übungsblatt entwickelt wurde, gehe ich nicht näher darauf ein.

CGMultiview.h

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm zu Aufgabenzettel 7
//

#ifndef CG_MULTIVIEW_H
#define CG_MULTIVIEW_H

#include "cgapplication.h"

class CGMultiview : public CGApplication {
public:
    CGMultiview();
    virtual ~CGMultiview();

    // Ueberschreibe alle diese Ereignisse:
    virtual void onInit();
    virtual void onDraw();
    virtual void onIdle();
    virtual void onDrag(double dx, double dy);
    virtual void onKey(unsigned char key);
    virtual void onSize(unsigned int newWidth, unsigned int newHeight);

    // value Methode
    unsigned char value(int x, int z) const;

private:
    void drawScene();

    int width_;
    int height_;
    double mainzoom_;
    double zoomxy_;
    double zoomxz_;
    double zoomyz_;
    bool mainfilled_;
    bool childfilled_;
    bool run_;
    double rotate_;
    double rotate2_;
};

#endif
```

CGMultiview.cpp

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm zu Aufgabenzettel 7
//

#include "cgmultiview.h"
#include "vector.h"
#include <fstream.h>
#include <stdlib.h>
```

```
CGMultiview::CGMultiview() {
    run_ = false;

    mainfilled_ = true;
    childfilled_ = false;

    mainzoom_ = 1;
    zoomxy_ = zoomxz_ = zoomyz_ = 1;

    rotate_ = rotate2_ = 0;
}

CGMultiview::~CGMultiview( ) {
}

void CGMultiview::drawScene() {
    glRotatef(rotate_,-1,1,0);
    glRotatef(rotate2_,0,1,0);

    static GLuint cache = 0;
    if (cache == 0) {
        cache = glGenLists(1);
        glNewList(cache,GL_COMPILE_AND_EXECUTE);
        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
        glRotated(-90,1,0,0);

        //ifstream s("big.txt");  // fuer den Porsche
        ifstream s("small.txt");
        char buf[100];

        int nTriangleCount = 0;

        glBegin(GL_TRIANGLES);
        Vector v[3];
        while (!s.eof()) {
            s >> buf;

            while (s.good()) {
                s >> v[0] >> v[1] >> v[2];
                Vector n = (v[1]-v[0])*(v[2]-v[0]);
                glNormal3dv(n.rep());
                glVertex3dv(v[0].rep());
                glVertex3dv(v[1].rep());
                glVertex3dv(v[2].rep());

                nTriangleCount++;
            }
            if (s.rdstate() & ios::failbit) {
                s.clear(s.rdstate() & ~ios::failbit);
            }
        }
        glEnd();
        glPopMatrix();
        glEndList();

        cout << "triangles: " << nTriangleCount << endl;
    } else {
        glCallList(cache);
    }
}

void CGMultiview::onInit() {
    // OpenGL Lichtquelle
    static GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};  /* diffuse light. */
    static GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};  /* Infinite light location. */
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    // automatische Normalisierung
    glEnable(GL_NORMALIZE);
```

```
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);

    // Tiefen Test aktivieren
    glEnable(GL_DEPTH_TEST);

    // Smooth Schattierung aktivieren
    glShadeModel(GL_SMOOTH);

    // Projection
    glMatrixMode(GL_PROJECTION);
    gluPerspective(60.0,1.0,2,2000.0);

    // LookAt
    glMatrixMode(GL_MODELVIEW);
//    gluLookAt(0.0, 0.0, 4.0,  // from (0,0,4)
//        0.0, 0.0, 0.0,  // to (0,0,0)
//        0.0, 1.0, 0.);  // up

    glClearColor(0.95,0.95,0.95,1);
}

void CGMultiview::onSize(unsigned int newWidth,unsigned int newHeight) {
    width_ = newWidth;
    height_ = newHeight;

    glutPostRedisplay();
}

void CGMultiview::onKey(unsigned char key) {
    switch (key) {
    case 27:  { exit(0); break; }
    case ' ': { run_ = !run_; break; }

    case '+': { mainzoom_*= 1.1; break; }
    case '-': { mainzoom_*= 0.9; break; }

    case '1': { zoomxy_*= 1.1; break; }
    case '2': { zoomxy_*= 0.9; break; }
    case '3': { zoomxz_*= 1.1; break; }
    case '4': { zoomxz_*= 0.9; break; }
    case '5': { zoomyz_*= 1.1; break; }
    case '6': { zoomyz_*= 0.9; break; }

    case 'l': { childfilled_ = !childfilled_; break; }
    case 'L': { mainfilled_  = !mainfilled_; break; }
    }

    glutPostRedisplay();
}

void CGMultiview::onIdle() {
    if (run_) {
        rotate_ += 1;
        glutPostRedisplay();
    }
}

void CGMultiview::onDrag(double dx, double dy)
{
    rotate2_ += dx*20;
//    glRotatef(-dy*20, 1, 0, 0);

    glutPostRedisplay();
}

void CGMultiview::onDraw() {
    // Loesche den Farb- und Tiefenspeicher
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Fentergröße
    int maxx = width_;
    int maxy = height_;
    int orthosize = maxy/3;
```

```
    // zeichne perspektivische Hauptansicht
    glViewport(0,0, maxx-1, maxy-1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45/mainzoom_, (float)maxx/maxy, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0,0,10, 0,0,0, 0,1,0);

    // Füllmodus
    glPolygonMode(GL_FRONT_AND_BACK, mainfilled_ ? GL_FILL:GL_LINE);
    glColor3f(1,1,1);
    drawScene();


    glPolygonMode(GL_FRONT_AND_BACK, childfilled_ ? GL_FILL:GL_LINE);
    // zeichne Ansicht x,y
    glViewport(maxx-orthosize, 2*orthosize, orthosize, orthosize);
    glMatrixMode(GL_PROJECTION);
    // orthogonale Sicht
    glLoadIdentity();
    glOrtho(-1/zoomxy_, 1/zoomxy_, -1/zoomxy_, 1/zoomxy_, 0.1, 100);
    // entlang z-Achse
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0,0,10, 0,0,0, 0,1,0);
    glColor3f(0,0,1);
    drawScene();

    // zeichne Ansicht x,z
    glViewport(maxx-orthosize, orthosize, orthosize, orthosize);
    glMatrixMode(GL_PROJECTION);
    // orthogonale Sicht
    glLoadIdentity();
    glOrtho(-1/zoomxz_, 1/zoomxz_, -1/zoomxz_, 1/zoomxz_, 0.1, 100);
    // entlang y-Achse
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0,10,0, 0,0,0, -1,0,0);
    glColor3f(0,1,0);
    drawScene();

    // zeichne Ansicht y,z
    glViewport(maxx-orthosize, 0, orthosize, orthosize);
    glMatrixMode(GL_PROJECTION);
    // orthogonale Sicht
    glLoadIdentity();
    glOrtho(-1/zoomyz_, 1/zoomyz_, -1/zoomyz_, 1/zoomyz_, 0.1, 100);
    // entlang x-Achse
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(10,0,0, 0,0,0, 0,1,0);
    glColor3f(1,0,0);
    drawScene();

    // Nicht vergessen! Front- und Back-Buffer tauschen:
    swapBuffers();
}

// Hauptprogramm
int main(int argc, char* argv[]) {
    // Erzeuge eine Instanz der Beispiel-Anwendung:
    CGMultiview sample;

    // Starte die Beispiel-Anwendung:
    sample.start("CGMultiview, Stephan Brumme, 702544");
    return(0);
}
```