



Trackball mit *OpenGL* und *GLUT*

Manfred Brill

Oktober 2005

Inhaltsverzeichnis

1	Examine	1
1.1	Modellierung	1
1.2	Implementierung	1
1.3	Keyboard-Shortcuts für Examine	3
1.4	Anbindung an die Maustasten	3
1.5	Abschließende Bemerkungen	5
	Literatur	6

Zusammenfassung

In allen 3D-Anwendungen kann mit Hilfe der Maus die Ansicht in der dreidimensionalen Welt verändert werden.

Dieser Text beschreibt die Implementierung einer `examine` und einer `pilotsview`-Funktion in *OpenGL* und die Anbindung als *GLUT*-Callback. Dabei wird davon ausgegangen, dass Sie entsprechendes Wissen über die Sichtdefinition in *OpenGL* haben, wie es beispielsweise in [BB05] zu finden ist.

Kapitel 1

Examine

Sie kennen dieses Verhalten sicher als ein Standard-Modus einer *VRML*-Datei. Mit *Examine* wird eine Sichtdefinition eines Betrachters realisiert, der sich immer um ein Objekt bewegt. Der Blickpunkt liegt dabei im Objektzentrum, der Augpunkt hat eine veränderbare Distanz zu diesem Zentrum.

1.1 Modellierung

Der Einfachheit halber gehen wir in der Implementierung von einem Objekt aus, dessen Objektzentrum oder Pivot-Punkt im Ursprung unserer Szene liegt. Dann beschreiben wir die Position der Kamera durch Kugelkoordinaten. Der Radius entspricht dann dem Abstand zwischen Objektzentrum und Betrachter, die beiden Winkel *Azimuth* und *Elevation* legen dann die eindeutige Position des Betrachters auf der Kugel mit dem gegebenen Radius fest.

1.2 Implementierung

Für die Kugelkoordinaten gibt es die globalen Variablen *RADIUS*, *AZIMUTH* und *ELEVATION* und entsprechende Inkremente, die zur Zeit nicht veränderbar sind. Als Default wird dabei die *OpenGL*-Sicht aus der in [Bri05] beschriebenen *OpenGL*-Schablone, entlang der negativen *z*-Achse aus einer Distanz von 5 Einheiten eingestellt.

Darüber hinaus sehen wir eine globale Variable *ORTHO* vor, mit deren Hilfe zwischen einer Parallel- und einer Zentralprojektion umgeschaltet werden kann. Den Öffnungswinkel, der einer Brennweite entspricht wird in der globalen Variable *FOVY* abgelegt. Für die Definition der Zentralprojektion und des Viewports wird auch das Breiten/Höhe-Verhältnis in der globalen Variable *ASPECT* definiert.

```
// Globale Größen für die Kamera
// Ein Winkel von 76.0 entspricht einem 24 mm Objektiv
// ORTHO = true entspricht Parallelprojektion ...
bool ORTHO = false;

GLfloat FOVY = 78.0, ASPECT = 1.0;
GLfloat AZIMUTH = 0.0, ELEVATION = 0.0, RADIUS = 5.0;
GLfloat AZIMUTH_INC = 1.0, ELEVATION_INC = 1.0, RADIUS_INC = 0.2;

// Pan-Variablen
GLfloat PAN_X = 0.0, PAN_Y = 0.0, PAN_INC = 0.05;
```

Neben der *Examine*-Funktionalität soll noch ein Verschieben der Sicht realisiert werden. Dazu werden die globalen Variablen `PAN_X`, `PAN_Y` und `PAN_INC` definiert. Damit soll es möglich werden, einen *Pan* mit einer festgehaltenen Maustaste zu implementieren. Durch Veränderung des Radius oder des Öffnungswinkels der eingestellten Kamera kann ein *Zoom* angeboten werden.

In [BB05] wird in der Fallstudie 2.6.3. „Die virtuelle Kamera in OpenGL“ der Dualismus zwischen Modelltransformationen und Kameratransformationen in *OpenGL* dargestellt. *examine* realisiert alle Transformationen im *MODELVIEW*-Stack; der *PROJECTION*-Stack dient ausschließlich zur Einstellung von Parallel- bzw. Zentralprojektion. Denken Sie bei der Interpretation des folgenden Quelltexts an die Reihenfolge der Transformationen! Die Parallelprojektion hat zur Zeit ein festes, nicht veränderbares Sichtvolumen.

```
void examine(void)
{
    // Die Projektion festlegen
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (ortho)
        glOrtho(-1.0, 1.0, -1.0, 1.0, 0.5, 20.0);
    else
        gluPerspective(FOVY, ASPECT, 0.5, 20.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(PAN_X, PAN_Y, -RADIUS);
    glRotatef(-ELEVATION, 1.0, 0.0, 0.0);
    glRotatef(AZIMUTH, 0.0, 1.0, 0.0);
}
```

Auch bei der Frage, wo die Funktion *examine* in die grafische Ausgabe integriert wird muss wieder an die Reihenfolge der Transformationen erinnert werden. Hilfreich dabei ist, die in [WND00] dargestellte Metapher zu verwenden: wir stellen die Szene dar, in dem wir zuerst die Kamera und anschließend die Objekte positionieren. Dies stimmt mit der Reihenfolge überein, wie die Transformation in einem *OpenGL*-Programm auftreten.

```
void display(void)
{
    /* Virtuelle Kamera in Funktion examine() */
    examine();

    /* Buffer zurücksetzen */
    glClear(GL_COLOR_BUFFER_BIT);
    /* Ein Gitter ausgeben */
    grid(10);
    /* Den Utah Teapot ausgeben */
    glColor3f(0.5, 0.5, 0.5);
    glutSolidTeapot(1.0);

    /* Die Buffer austauschen */
    glutSwapBuffers();
}
```

Falls sich die Seitenverhältnisse des Ausgabe-Fensters verändern und Zentralprojektion eingestellt ist muss die globale Variable `ASPECT` in *reshape* verändert werden:

```
void reshape(int w, int h)
{
    // Viewport
    glViewport(0,0, (GLsizei) w, (GLsizei) h);
}
```

```
ASPECT = (float)w/(float)h;
}
```

1.3 Keyboard-Shortcuts für Examine

Neben der Anbindung der Maus an diese Funktionalität existieren auch Shortcuts für die Tastatur. Diese sind ausschließlich in der `special`-Funktion, also auf nicht-ASCII-Zeichen gelegt. In Tabelle 1.1 finden Sie die zur gültige Belegung.

Tabelle 1.1: Die Belegung der Tastatur für die Examine-Funktion

←	Azimuth um 1° vergrößern
→	Azimuth um 1° verkleinern
↑	Elevation um 1° vergrößern
↓	Elevation um 1° verkleinern
Bild auf	Betrachterabstand um 0,2 verkleinern
bild ab	Betrachterabstand um 0,2 vergrößern
Home	Verkleinern des Öffnungswinkels um 1°
Pos1	Vergrößern des Öffnungswinkels um 1°

Ein Reset auf die Ausgangssicht ist durch die Funktionstaste F10 realisiert. Mit F9 kann zwischen Parallel- und Zentralprojektion umgeschaltet werden. Die Defaulteinstellung für die Projektionsart ist Zentralprojektion. In Tabelle 1.2 finden Sie die aktuellen Ausgangswerte für die Sicht in der `examine`-Funktion.

Tabelle 1.2: Die Ausgangswerte für die `examine`-Funktion

Betrachterabstand	5,0
Azimuth-Winkel	0°
Elevations-Winkel	0°
Öffnungswinkel der Kamera	78°

1.4 Anbindung an die Maustasten

OpenGL bietet *immer* die Möglichkeit, drei Tasten einer Maus zu verwenden. Das stammt aus X11; dort hatten die Mäuse immer drei Tasten. Mit der linken Maustaste wird die eigentliche Trackball-Funktion aufgerufen, die rechte Maustaste dient zu einem Zoom (durch Veränderung des Betrachterabstands); die mittlere Maustaste realisiert ein Pan, ein Verschieben der Szene. In GLUT ist mit `glutMouseFunc(mouse)` die folgende Funktion registriert:

```
void mouse(int button,int state,int x,int y)
{
    MOUSE_BUTTON_PRESSED=button;
}
```

Auf der Variable `button` übergibt GLUT, welche Maustaste gedrückt wurde; dazu gibt es die drei Konstanten `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` und `GLUT_RIGHT_BUTTON`.

Die Trackballfunktionalität kommt erst zum Tragen, wenn die Maus bewegt wird. Dazu dient die Funktion `mouseMotion`, die mit `glutMotionFunc(mouseMotion)` registriert wird.

In dieser Funktion wird auf statischen Variablen die letzte Maus-Position gespeichert. Auf den beiden Übergabevariablen übergibt GLUT die aktuelle Maus-Position. Es ist eine „Bremse“ eingebaut, damit die Steuerung nicht zu schnell reagiert. Sollte auf Ihrem Rechner die Steuerung zu zäh sein, dann verringern Sie diesen Wert.

Die Entscheidung, in welche Richtung das Pan durchgeführt wird, oder ob der Azimuth- oder der Elevationswinkel verändert werden soll erfolgt durch Untersuchung, ob Sie die Maus mehr in die x -Richtung (links oder rechts) oder die y -Richtung (oben oder unten) bewegt haben. Für die Veränderung der Werte werden die gleichen Inkrement-Werte verwendet, die auch auf den Keyboard-Shortcuts liegen. Für den Zoom wird der Betrachterabstand verändert.

```
void mouseMotion(int x, int y)
{
    static int OLD_MOUSE_X = 0, OLD_MOUSE_Y = 0;
    int distx, disty; // Distanzen in x oder y-Koordinaten.
    int bremsen = 10; // Maus-Koordinaten skalieren, damit
                       // die Steuerung nicht so schnell reagiert!

    distx = x-OLD_MOUSE_X;
    disty = y-OLD_MOUSE_Y;
    switch(MOUSE_BUTTON_PRESSED) {
        case GLUT_LEFT_BUTTON: // Drehung
            // Unterschied in x größer als y
            // => Azimut verändern!
            if (abs(distx)>abs(disty)) {
                if (distx>0)
                    AZIMUTH += AZIMUTH_INC;
                else if (distx<0)
                    AZIMUTH -= AZIMUTH_INC;
            } else
                if (disty>0)
                    ELEVATION -= ELEVATION_INC;
                else if (disty<0)
                    ELEVATION += ELEVATION_INC;
            break;
        case GLUT_MIDDLE_BUTTON: // PAN ...
            if (abs(distx)>abs(disty)) {
                if (distx>0)
                    PAN_X += PAN_INC;
                else if (distx<0)
                    PAN_X -= PAN_INC;
            } else
                if (disty>0)
                    PAN_Y -= PAN_INC;
                else if (disty<0)
                    PAN_Y += PAN_INC;
        case GLUT_RIGHT_BUTTON: //Zoom
            if (disty > 0)
                RADIUS += (RADIUS_INC/bremsen);
            else if (disty < 0)
                RADIUS -= (RADIUS_INC/bremsen);
            break;
        default:
            break;
    }

    OLD_MOUSE_X=x;
    OLD_MOUSE_Y=y;

    glutPostRedisplay();
}
```

1.5 Abschließende Bemerkungen

Diese Implementierung ist kein echter Trackball. Es gibt im WWW entsprechende Implementierungen; im green book [Kil96] finden Sie eine Implementierung, die sehr stabil ist. Allerdings werden dafür noch einige Überlegungen benötigt, wie Rotationen sinnvoll realisiert werden. Kilgard und auch viele andere verwenden hier Quaternionen. Mehr zu Quaternionen finden Sie in [?]. Auch in [Ang05] finden Sie einige Bemerkungen zum Trackball.

Literaturverzeichnis

- [Ang05] ANGEL, EDWARD: *Interactive Computer Graphics – A Top-Down Approach with OpenGL*. Addison-Wesley, 2005.
- [BB05] BENDER, MICHAEL und BRILL, MANFRED: *Computergrafik – ein anwendungsbezogenes Lehrbuch*. 2. Auflage, Hanser, 2005.
- [Bri05] BRILL, MANFRED: *Programmieren mit OpenGL und GLUT*. Visualisierungslabor der FH Kaiserslautern, 2005.
- [Kil96] KILGARD, MARK: *OpenGL Programming for the X Windows Systems*. Addison-Wesley, 1996.
- [WND00] WOO, MASON, NEIDER, JACKIE und DAVIS, TOM: *OpenGL Programming Guide (2nd Ed.) – The Official Guide to Learning OpenGL*. Addison-Wesley, 2000.