Einführung in C++ – Übung 5
Testatgruppe A (Isaak)

Rasmus Diederichsen

12. November 2014

Aufgabe 5.1 Die Formeln

Hier gibt es nix zu erklären, da keine der beiden Formeln sinnhaft ist.

Aufgabe 5.2 Strukturen

`structs` dienen in C dazu, mehrere Datenfelder in einem neuen Datentyp zusammenzufassen.

Aufgabe 5.3 Enumerations

`enums` haben den Zweck, schnell eine große Anzahl Konstanten mit sinnvollen Bezeichnern einzuführen, die alle zur selben Kategorie gehören. Der Datentyp einer `enum`-Konstante ist ein Ganzzahltyp (vermutlich `int`).

Listing 1: camera.c

```
#include <camera.h>
#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <glut.h>
#else
#include <GL/gl.h>
#include <GL/glut.h>
#endif
#include <stdio.h>
#include <math.h>

typedef struct vector vector;
typedef struct camera camera;

/* Functions to do multiply with rotation matrix */
void rotateX(vector *v, float angle);
void rotateY(vector *v, float angle);

void init_camera(camera *cam, vector pos,
```

```
        float rot_speed, float move_speed)
{
    vector up = { .x = 0, .y = 1, .z = 0 },
           trans = { .x = 0, .y = 0, .z = 0 },
           rot = { .x = 0, .y = 0, .z = 0 },
           look_at = { .x = 0, .y = 0, .z = -1 };
    cam->up = up;
    cam->trans = trans;
    cam->l = look_at;
    cam->rot = rot;
    cam->initial = pos;
    cam->turn_speed = rot_speed;
    cam->move_speed = move_speed;
}

void move_camera(camera *cam, enum direction dir)
{
    /* Vector for direction of gaze */
    vector forward = {.x = 0, .y = 0, .z = 1};
    rotateY(&forward,cam->rot.y);

    /* This vector will be perpendicular, but with the same y-value
        */
    vector perp  = { .x = -forward.z,
                             .y = forward.y,
                             .z = forward.x
                           };

    switch (dir)
    {
        case FORWARD:
            /* Move camera */
            cam->trans.x += cam->move_speed * forward.x;
            cam->trans.y += cam->move_speed * forward.y;
            cam->trans.z += cam->move_speed * forward.z;
            /* Move view point since its supposed to be at distance 1
                */
            cam->l.x += cam->move_speed * forward.x;
            cam->l.y += cam->move_speed * forward.y;
            cam->l.z += cam->move_speed * forward.z;
            break;
        case BACKWARD:
            cam->trans.x -= cam->move_speed * forward.x;
            cam->trans.y -= cam->move_speed * forward.y;
            cam->trans.z -= cam->move_speed * forward.z;
            cam->l.x -= cam->move_speed * forward.x;
            cam->l.y -= cam->move_speed * forward.y;
            cam->l.z -= cam->move_speed * forward.z;
            break;
        case LEFT:
            /* Move along perpendicular */
            cam->trans.x -= cam->move_speed * perp.x;
            cam->trans.z -= cam->move_speed * perp.z;
            break;
        case RIGHT:
            cam->trans.x += cam->move_speed * perp.x;
            cam->trans.z += cam->move_speed * perp.z;
```

```
75            break;
76        case UP:
77            cam->trans.y += cam->move_speed;
78            break;
79        case DOWN:
80            cam->trans.y -= cam->move_speed;
81            break;
82    }
83 }
84
85 void turn_camera(camera *cam, enum direction ax)
86 {
87          switch (ax)
88          {
89                  case LEFT:      cam->rot.y -= (cam->turn_speed);
90                                              break;
91
92                  case RIGHT:     cam->rot.y += (cam->turn_speed);
93                                              break;
94
95                  case UP:                cam->rot.x += (cam->
                        turn_speed);
96                                              break;
97
98                  case DOWN:      cam->rot.x -= (cam->turn_speed);
99                                              break;
100       default: fprintf(stderr, "turn_camer␣error:␣invalid␣axis␣%d.\
              n", ax);
101                 break;
102         }
103
104 }
105
106 void apply_camera(camera *cam)
107 {
108    glLoadIdentity();
109    vector pos;
110    pos.x = cam->initial.x + cam->trans.x;
111    pos.y = cam->initial.y + cam->trans.y;
112    pos.z = cam->initial.z + cam->trans.z;
113
114    vector forward = { .x = 0, .y = 0, .z = 1. };
115
116    /* lookAt vector can be rotated in xz-plane or up/downwards */
117    rotateX(&forward, cam->rot.x);
118    rotateY(&forward, cam->rot.y);
119
120    cam->l.x = forward.x + pos.x;
121    cam->l.y = forward.y + pos.y;
122    cam->l.z = forward.z + pos.z;
123
124    gluLookAt(pos.x, pos.y, pos.z,
125          cam->l.x, cam->l.y, cam->l.z,
126          cam->up.x, cam->up.y, cam->up.z);
127 }
128
129 void print_camera(camera *cam)
```

```
{
    printf("%-20s [%5.4f,%5.4f,%5.4f]\n", "up:",cam->up.x,cam->up.y,
        cam->up.z);
    printf("%-20s [%5.4f,%5.4f,%5.4f]\n", "trans:",cam->trans.x,cam
        ->trans.y,cam->trans.z);
    printf("%-20s [%5.4f,%5.4f,%5.4f]\n", "l:",cam->l.x,cam->l.y,cam
        ->l.z);
    printf("%-20s [%5.4f,%5.4f,%5.4f]\n", "rot:",cam->rot.x,cam->rot
        .y,cam->rot.z);
    printf("%-20s [%5.4f,%5.4f,%5.4f]\n", "initial:",cam->initial.x,
        cam->initial.y,cam->initial.z);
    printf("%-20s %5.4f\n","turn_speed:", cam->turn_speed);
    printf("%-20s %5.4f\n","move_speed:", cam->move_speed);
}

void rotateX(vector *v, float angle)
{
        v->y = (float)cos(angle) * v->y  - (float)sin(angle) * v->z
            ;
        v->z = (float)sin(angle) * v->y  + (float)cos(angle) * v->z
            ;
}

void rotateY(vector *v, float angle)
{
        v->x = (float)cos(angle) * v->x  - (float)sin(angle) * v->z
            ;
        v->z = -(float)sin(angle) * v->x + (float)cos(angle) * v->z
            ;
}
```

Listing 2: mainwindow.c

```
/* Glut header */
#ifdef __APPLE__
#include <glut.h>
#else
#include <GL/glut.h>
#endif

/* Standard io header */
#include <stdio.h>
#include <math.h>

/* Camera manipulation functions */
#include "camera.h"

/* Extern declarations of used global variables */
extern float* rvBuffer;
extern int*   riBuffer;
extern int    riCount;
extern float  rvCount;
extern int    old_x;
extern int    old_y;
extern struct camera* cam;
extern int    mouse_button;
extern int    mouse_state;

```

```
26
27  void mousePressed(int button, int state, int x, int y)
28  {
29      /* Empty stub. Insert Code here */
30      mouse_button = button;
31      mouse_state = state;
32  }
33
34  void mouseMoved(int x, int y)
35  {
36      /* Empty stub. Insert Code here. Keep glut call on last line */
37      glutPostRedisplay();
38  }
39
40  void keyPressed(unsigned char key, int x, int y)
41  {
42
43      switch(key) {
44         case 'd':
45            move_camera(cam, RIGHT);
46            break;
47         case 'a':
48            move_camera(cam, LEFT);
49            break;
50         case 'w':
51            move_camera(cam, FORWARD);
52            break;
53         case 's':
54            move_camera(cam, BACKWARD);
55            break;
56         case 'f':
57            turn_camera(cam, LEFT);
58            break;
59         case 'h':
60            turn_camera(cam, RIGHT);
61            break;
62         case 't':
63            turn_camera(cam, UP);
64            break;
65         case 'g':
66            turn_camera(cam, DOWN);
67            break;
68      }
69      glutPostRedisplay();
70  }
71
72  void reshape(int w, int h)
73  {
74          if (h == 0)
75                  h = 1;
76
77          float ratio =  w * 1.0 / h;
78          glMatrixMode(GL_PROJECTION);
79          glLoadIdentity();
80          glViewport(0, 0, w, h);
81          gluPerspective(45,ratio,1,10000);
82          glMatrixMode(GL_MODELVIEW);
```

```
}

void render(void)
{
    int i;

    /* Clear backbuffer with black color */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    apply_camera(cam);


    for(i = 0; i < riCount; i++)
    {
        /* Get position og current triangle in buffer */
        int index = 3 * i;

        /* Get vertex indices of triangle vertices */
        int a = 3 * riBuffer[index];
        int b = 3 * riBuffer[index + 1];
        int c = 3 * riBuffer[index + 2];


        /* Render wireframe model */
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0, 1.0, 1.0);
        glVertex3f(rvBuffer[a], rvBuffer[a + 1], rvBuffer[a + 2]);
        glVertex3f(rvBuffer[b], rvBuffer[b + 1], rvBuffer[b + 2]);
        glVertex3f(rvBuffer[c], rvBuffer[c + 1], rvBuffer[c + 2]);
        glEnd();
    }

    /* Call backbuffer and replace screen contends */
    glutSwapBuffers();
}
```