UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

4. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

Aufgabe 1: Portierung des bestehenden Codes nach C++ (35 Punkte)

Im ersten Teil der Übung machen Sie sich mit dem Erstellen und Verwenden von Klassen in C++ vertraut. Dazu sollen Sie zunächst den letzten Stand der Software nach C++ portieren. Diesen finden Sie im Ordner Musterlösungen Ihres Repositorys. Zum Laden und Verwalten der 3D-Modelle wird Ihnen die Klasse `Model` zur Verfügung gestellt, die in der Lage ist, .ply-Dateien zu lesen. Ihre Aufgabe besteht darin, den Code zum Erstellen des Hauptfensters und den Main-Loop in eine Klasse `MainWindow` zu transferieren. Diese soll die in `MainWindow.hpp` vorgegebene Struktur haben und neben dem Destruktur die folgenden Funktionalitäten bereitstellen:

```
MainWindow(std::string title, std::string plyname, int w, int h);
```

Der Konstruktor erstellt ein Hautfenster mit dem gegebenen Titel und lädt erstellt eine interne Modellinstanz, die in der Klassenvariablen `m_model` gespeichert wird. Fensterhöhe und Breite werden durch die Parameter `w` und `h` bestimmt. Beachten Sie, dass diese in der Musterlösung hardcodiert waren. Ersetzen Sie die entsprechenden Konstanten.

```
void execute();
```

Diese Methode enthält den Main-Loop des Programms. Dieser rendert das gegebene Modell und reagiert auf Events. Implementieren Sie Ihre Lösung in der Datei `MainWindow.cpp`. Instanzieren Sie ein `MainWindow` in der main-Funktion, die Sie in der Datei `Main.cpp` finden und starten Sie dort die Hauptschleife.

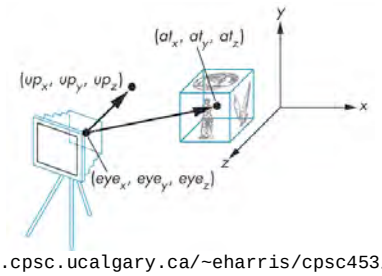
Aufgabe 2: Implementierung einer virtuellen Camera (65 Punkte)

Aufgabenstellung:

In dieser Aufgabe erweitern wir unser Programm um eine virtuelle Kamera. Mit dieser soll der Benutzer in die Lage versetzt werden, sich in der Szene zu bewegen. Zur Definition einer virtuellen Kamera steht in OpenGL Utility Toolkit die Funktion

```
gluLookAt(
        float eyex, float eyex, float eye,          /* Camera position */
        float atx, float aty, float atz,            /* Point to look at */
        float upx, float upy, float upz);           /* Upward direction of the camera */
```

zur Verfügung. Diese Funktion definiert die drei Vektoren, durch welche die Orientierung der Kamera festgelegt wird. Die drei ersten `float`-Werte bestimmen die Position der Kamera in der Szene. Die nächsten drei Werte legen fest, auf welchen Punkt gerade geschaut wird. Der letzten drei Parameter definieren einen Vektor, der die Orientierung festlegt, also angibt wo "oben" ist:



Quelle: `http://pages.cpsc.ucalgary.ca/~eharris/cpsc453/tut16/camera2.gif`

Diese Werte definieren Position und Ausrichtung der Kamera im globalen Koordinatensystem. Wird die Kamera durch Benutzereingaben bewegt, müssen sie entsprechend transformiert werden. Dies lässt sich am einfachsten realisieren, indem das von den Kameraparametern aufgespannte lokale Bezugssystem modifiziert und anschließend ins globale Koordinatensystem transformiert wird. Dazu werden zusätzlich die Drehwinkel des lokalen Systems um die globalen Koordinatenachsen ("Euler-Winkel") sowie ein Translationsvektor benötigt. Die Kamera soll in unserem Programm die Blickrichtung des Benutzers repräsentieren. Dreht sich der virtuelle Benutzer um die y-Achse, muss der Betrachtungspunkt entsprechend transformiert werden. In unserem Fall ist der Betrachtungspunkt immer der Punkt, der sich auf dem Einheitskreis um die Kameraposition ***p*** befindet. Bei der Drehung um die y-Achse um den Winkel β muss der Betrachtungspunkt ***l*** also passend in der x-z-Ebene rotiert werden und als Offset auf die aktuelle Kameraposition addiert werden. Die Formel für eine Linksdrehung lautet daher:

$$\begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \begin{pmatrix} \sin\beta \\ 0 \\ -\cos\beta \end{pmatrix}$$

Zur Simulation einer realistischen Bewegung soll die Bewegung der Kamera immer relativ zur aktuellen Blickrichtung erfolgen. Dazu wird bei jeder entsprechenden Eingabe der Translations-Offset ***t*** der Kamera von der aktuellen Position *p* berechnet. Diese Bewegungen erfolgen immer parallel zur x-z-Ebene, daher muss auch hier nur der globale Drehwinkel β um die y-Achse betrachtet werden. Bewegt sich der Benutzer mit Geschwindigkeit *s*, lautet die Formel für das *Update* der Translation (***p*** + ***t***) bei einer Vorwärtsbewegung:

$$\begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = \begin{pmatrix} s \cdot \sin\beta \\ 0 \\ s \cdot \cos\beta \end{pmatrix}$$

Erklären Sie Ihrem Tutor diese Formeln. Das betrachtete Objekt liegt in unserem Fall im Ursprung des Koordinatensystems. Um es im Blickfeld der Kamera zu haben, muss diese also in Richtung der negativen z-Achse blicken, daher das entsprechende Minuszeichen. Schreiben Sie eine Klasse `Camera`, die mindestens die in der Vorgabe vorhandenen Methoden und Felder enthält. Das entsprechende Gerüst wird Ihnen in Ihrem Repository zur Verfügung gestellt. Dieses enthält die Definitionen aller erforderlichen Felder und Methoden, um eine Kamera wie oben beschrieben zu implementieren. Zu den einzelnen Methoden hier noch ein paar Hinweise:

Beschreibung der geforderten Funktionalität:

```
Camera(Vector position, float turnSpeed, float moveSpeed);
```

Der Konstruktor initialisiert eine Kamera an der gegebenen Position. Die Blickrichtung erfolgt in -z-Richtung. Die LookUp-Vektor zeigt nach oben.

```
void move(CameraMovement dir);
```

Diese Funktion bewegt die Kamera um `m_moveSpeed` Einheiten in die durch `dir` definierte Richtung. Für die Fälle `UP` und `DOWN` soll die y-Koordinate des Translationsvektors angehoben bzw. abgesenkt werden. Ist der Wert von `dir` `FORWARD` oder `BACKWARD`, soll die Kamera entlang der aktuellen Blickrichtung bewegt werden. Passen sie die x- und z-Koordinaten der aktuellen Translation an. In den Fällen `LEFT` und `RIGHT` soll sich die Kamera auf der aktuellen Höhe senkrecht zur Blickrichtung bewegen. *Hinweis*: So ein Vektor ist um 90° zur aktuellen Blickrichtung gedreht.

```
void turn(CameraMovement dir);
```

Diese Funktion soll eine Bewegung des Kopfes des Benutzers simulieren. In den Fällen `LEFT` und `RIGHT` wird der Drehwinkel um die y-Achse verringert bzw. angehoben. Alle anderen Bewegungen sollen derzeit ignoriert werden (diese werden auf einem der kommenden Aufgabenblätter implementiert. Derzeit wollen wir und auf die Bewegung in der x-z-Ebene beschränken).

`void apply();`

Wird `apply` aufgerufen, werden die internen Werte der Kamera-Struktur in die entsprechenden Parameter für `gluLookAt(...)` umgerechnet. Die aktuelle Kameraposition ist die Summe von initialer Position und Tanslationsvektor. Der Betrachtungspunkt lässt sich berechnen, indem auf die aktuelle Position der durch die internen Drehwinkel repräsentierte Punkt auf der Einheitskugel auf diesen Punkt addiert wird (siehe erste Formel oben). Da wir in unserer Kamera keine Rollbewegungen zulassen, ist der View-Up-Vektor der Kamera immer identisch mit der y-Achse.

Fügen Sie eine Kamerainstanz zu Ihrer `MainWindow`-Klasse hinzu. Implementieren Sie die folgenden Keybindungs zur Steuerung der Kamera, indem Sie die Hauptschleife wie folgt erweitern:

```
switch(event.type)
{
    case SDL_QUIT:
        loop = false;
        break;
    case SDL_KEYDOWN:
        switch(event.key.keysym.sym)
        {
            case SDLK_w:    /* Move camera forward      */ break;
            case SDLK_s:    /* Move camera backwards    */ break;
            case SDLK_a:    /* Move camera to the left  */ break;
            case SDLK_d:    /* Move camera to the right */ break;
            case SDLK_KP_4: /* Turn camera left         */ break;
            case SDLK_KP_6: /* Turn camera right        */ break;
        }
        break;
    default:
        break;
}
```

Strategischer Hinweis:

Bevor Sie mit der Implementierung beginnen, versuchen Sie die oben angegebenen Formeln selber herzuleiten. Überlegen Sie sich genau, wie die Vorzeichen bei der Umrechung auszusehen haben. So werden Sie bei der Implementierung allzu viele Trial- and Error-Aktionen vermeiden.

Achtung: Ab diesem Zettel werden wir ausschließlich in C++ programmieren! Vermeiden Sie jegliche Nutzung von Funktionalitäten aus der C-Standardbibliothek. Insbesondere sollen alle Dateioperationen über C++-Streams realisiert werden! Stellen Sie auch sicher, dass Ihre Programme mit dem g++-Parameter `–Wall` kompilieren, ohne dass Warnungen erzeugt werden. Stellen Sie darüber hinaus sicher, dass jeglicher Speicher, den Sie dynamisch anlegen auch wieder freigegeben wird. Wie werden das in den Testaten mit `valgrind` überprüfen.

Abgabe:

Checken Sie Ihre Lösung des Aufgabenblattes bis Montag den 19.11.2018, 8:00 Uhr in den Master-Branch des git-Repositorys Ihrer Gruppe ein. Bringen Sie pro Gruppe einen Ausdruck des von Ihnen erstellten Codes mit.