UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

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

Wintersemester 2014 / 2015

Aufgabe: Implementierung einer virtuellen Camera

Ziel der Übung:

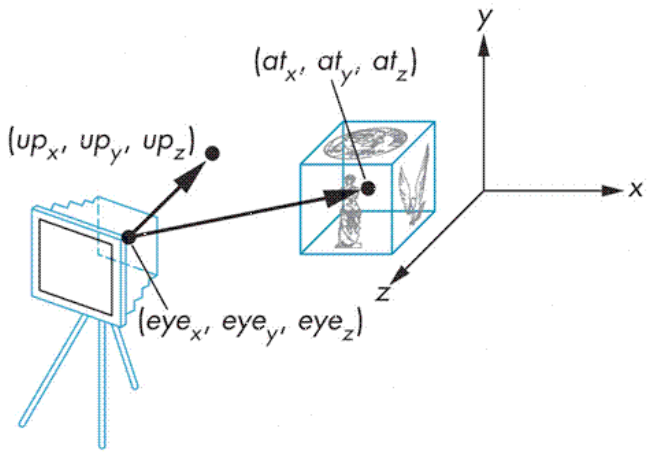
In dieser Übung werden wir den Umgang mit komplexen Datentypen üben. Dazu implementieren wir eine virtuelle Kamera, die durch eine passende `struct` repräsentiert wird.

Aufgabenstellung:

In StudIP finden Sie das Programmgerüst für diese Aufgabe. Es erweitert die Musterlösung des letzten Aufgabenblattes durch Aufrufe von Manipulationsfunktionen für eine virtuelle Kamera. Mit diesen soll der Benutzer in die Lage versetzt werden, sich in der Szene zu bewegen. Zur Definition einer virtuellen Kamera steht in OpenGL 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 Kamera befindet. Bei der Drehung um die y-Achse um den Winkel β muss der Betrachtungspunkt also passend in der x-z-Ebene rotiert werden. Die Formel für eine Linksdrehung lautet daher:

$$\begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} = \begin{pmatrix} l_x + \sin(\beta) \\ l_y \\ -l_z - \cos(\beta) \end{pmatrix}$$

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

$$\begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = \begin{pmatrix} s\sin(\beta) \\ t_y \\ s\cos(\beta) \end{pmatrix}$$

Erklären Sie Ihrem Tutor diese Formeln *(10 Punkte)*. 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. Das neue Programmgerüst nutzt für die Repräsentation der Kamera die `struct camera`, sowie die Funktionen, die die ihre Ausrichtung manipulieren. Deren Definition finden Sie im Header `camera.h`. Die `struct camera` ist wie folgt definiert:

```
struct camera
{
  struct vector up;             /* Up vector */
  struct vector trans;          /* Camera position */
  struct vector lat;            /* Lookt at vector */
  struct vector rot;            /* Current rotation state */
  struct vector initial;        /* Initial position of the camera */
  float turn_speed;             /* Turning speed (in radians) */
  float move_speed;             /* Moving speed in coordinate units */
};
```

Die von `camera` benutzte `struct vector` definiert einen Punkt im 3D-Raum:

```
struct vector
{
  float x;
  float y;
  float z;
 };
```

Erklären Sie Ihrem Tutor die Benutzung von Strukturen in C *(25 Punkte)*. Die ersten drei Vektoren der Kamera enthalten Translation der Kamera, den Betrachtungspunkt und die Orientierung der Kamera. Im Vektor `rot` sind die Drehwinkel um die globalen Achsen gespeichert. Der Vektor `initial` enthält die initiale Position der Kamera im globalen Koordinatensystem. Die Wert `turn_speed` und `move_speed` geben an, um wie viele Einheiten die Kamera gedreht und bewegt werden soll, wenn eine Mauseingabe erfolgt. Darüber hinaus enthält `camera.h` noch die folgenden Manipulationsfunktionen:

```
void init_camera(struct camera *cam, struct vector pos, float rot_speed, float speed);
void move_camera(struct camera *cam, enum direction dir);
void turn_camera(struct camera *cam, enum direction ax);
void apply_camera(struct camera *cam);
void print_camera(struct camera *cam);
```

Diese Funktionen nutzen die `enum direction`:

```
enum direction {FORWARD, BACKWARD, LEFT, RIGHT, UP, DOWN};
```

Erklären Sie Ihrem Tutor die Verwendung von Aufzählungen in C *(10 Punkte)*. Implementieren Sie die oben

genannten Funktionen mit den im Folgenden spezifizierten Funktionalitäten in der Datei `camera.c`.

Beschreibung der geforderten Funktionalität:

`void init_camera(struct camera *cam, struct vector pos, float rot_speed, float speed)`:

Diese Funktion initialisiert die Felder der gegebenen Kamera-Struktur. Die Drehwinkel sowie der Translationsoffset sollen mit `0` initialisiert werden. Die initiale Kameraposition ist durch den Parameter `pos` gegeben. Rotations und Translationsgeschwindgkeit werden durch `rot_speed` und `speed` gesetzt *(5 Punkte)*.

`void move_camera(struct camera *cam, enum direction dir)`:

Diese Funktion bewegt die Kamera um `move_speed` Einheiten. Für die Fälle `UP` und `DOWN` soll die y-Koordinate des Translationsvektors angehoben bzw. abgesenkt werden. 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. 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 *(10 Punkte)*.

`void turn_camera(struct camera *cam, enum direction ax)`:

Diese Funktion soll eine Bewegung des Kopfes des Benutzers simulieren. In den Fällen `UP` und `DOWN` wird dazu der Rotationswinkel um die x-Achse verkleinert bzw. erhöht. In den Fällen `LEFT` und `RIGHT` wird der Drehwinkel um die y-Achse verringert bzw. angehoben *(10 Punkte)*.

`void print_camera(struct camera *cam)`:

Implementieren Sie diese Funktion für Debugzwecke. Wird sie aufgerufen, sollen die aktuellen Kameraparameter formatiert auf der Konsole ausgegeben werden *(5 Punkte)*.

`void apply_camera(struct camera *cam);`

Wird `apply_camera` aufgerufen, werden die internen Werte der Kamera-Struktur in die entsprechenden Parameter für `gluLookAt(...)` umgerechnet *(10 Punkte)*. 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). Berücksichtigen Sie allerdings zusätzlich die Neigung des Kopfes, d.h. den Drehwinkel um die x-Achse. Da wir in unserer Kamera keine Rollbewegungen zulassen, ist der View-Up-Vektor der Kamera immer identisch mit der y-Achse. Implementieren Sie die Kamerasteueung per Maus durch Überladen der passenden Callbacks *(20 Punkte)*. Denken sie daran, vor `gluLookAt` die Funktion `glLoadIdentity` aufzurufen.

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.

Abgabe

Mailen Sie pro Gruppe Ihre Dateien bis Montag den 17. November 08:00 Uhr mit der Angabe Ihres Tutors an die Adresse `cpp@informatik.uni-osnabrueck.de`. Bringen Sie jeweils pro Person einen Ausdruck Ihrer Abgabe zum Testat mit.