Dokumentation der didaktischen Bibliothek GLOOP

(Version 3.7, Februar 2014)

Volker Quade

Zusammenfassung

Bei der Java-Bibliothek GLOOP (Graphics Library for object oriented programming) handelt es sich um eine didaktische Lernumgebung zur Vermittlung der objektorientierten Programmierung. Die Bibliothek bietet die Möglichkeit eines didaktisch vereinfachten und intuitiven objektorientierten Zugangs zur dreidimensionalen Grafikprogrammierung mit OpenGL. Sie wurde speziell für den Einstieg in die Objektorientierung im Unterricht der gymnasialen Einführungsphase entwickelt und soll helfen, den Schülerinnen und Schülern in optisch ansprechender und motivierender Weise objektorientierte Zusammenhänge zu verdeutlichen.

Inhaltsverzeichnis

1 Allgemeine Funktionsweise

Besteht das Ziel darin, einen dreidimensionalen Gegenstandsbereich in eine Simulation zu überführen, so braucht man einen virtuellen Raum. Dieser ist zunächst leer und wird mittels eines dreidimensionalen, rechtshändischen Koordinatensystems strukturiert. Der Raum erstreckt sich in die positive und in die negative Richtung aller Koordinatenachsen bis in die Unendlichkeit und soll später geometrische Objekte aufnehmen, aus denen eine Szene zusammengesetzt ist.

Um diese geometrischen Objekte zu realisieren, stehen vorgefertigte Klassen zur Verfügung, bei deren Instanziierung jeweils die Position des Mittelpunktes des Objektes und seine Ausdehnung im Raum angegeben werden muss. Der Bezeichner jeder Klasse beginnt mit dem Präfix `GL`, um später die Bezeichner der vorgefertigten Klassen besser von denen eigener unterscheiden zu können. Es stehen die folgenden Klassen für geometrische Objekte zur Verfügung: `GLKugel`, `GLQuader`, `GLZylinder`, `GLWuerfel`, `GLKegel`, `GLKegelstumpf`, `GLTorus` und `GLPrismoid`.

Versucht man zunächst einfach eine Kugel im dreidimensionalen Raum darzustellen, so muss man ein Objekt vom Typ `GLKugel` instanziieren. In der Deklaration wird ein Objekt vom Typ `GLKugel` angemeldet und im Konstruktor wird die Kugel z.B. an der Stelle `(0,0,0)` mit dem Radius `50` instanziiert.

Im Prinzip ist ein solches Programm lauffähig, wird aber nicht das erhoffte Resultat erzielen. Zwar gibt es nun eine Kugel im dreidimensionalen Raum, es gibt aber keine Möglichkeit, sie zu beobachten. Wollen wir die Kugel sehen, bedarf es einer Kamera und einer Lichtquelle, die unsere Szene ausleuchtet.

```
1  import GLOOP.*;
2  class Simulation {
3    GLKamera meineKamera;
4    GLLicht meinLicht;
5    GLKugel meineKugel;
6
7    Simulation() {
8      meineKamera = new GLKamera();
9      meinLicht   = new GLLicht();
10     meineKugel  = new GLKugel (0,0,0, 50);
11   }
12 }
```

Das Instanziieren einer Kamera führt dazu, dass ein Fenster aufspringt, in dem der virtuelle Raum in Echtzeit dargestellt wird. Die Standardposition der Kamera ist bei `(0,0,500)` mit Blick auf den Ursprung des Koordinatensystems. Die Lichtquelle leuchtet die Szene aus, so dass nun eine weiße Kugel zu sehen ist (vgl. Abbildung 1).

Alle sichtbaren geometrischen Objekte, die im Raum platziert werden können, sind Spezialisierungen der Klasse `GLObjekt` und verfügen somit über den gleichen Methodensatz. Zum Beispiel kann jedes Objekt vom Typ `GLObjekt` mit einer Textur überzogen werden.

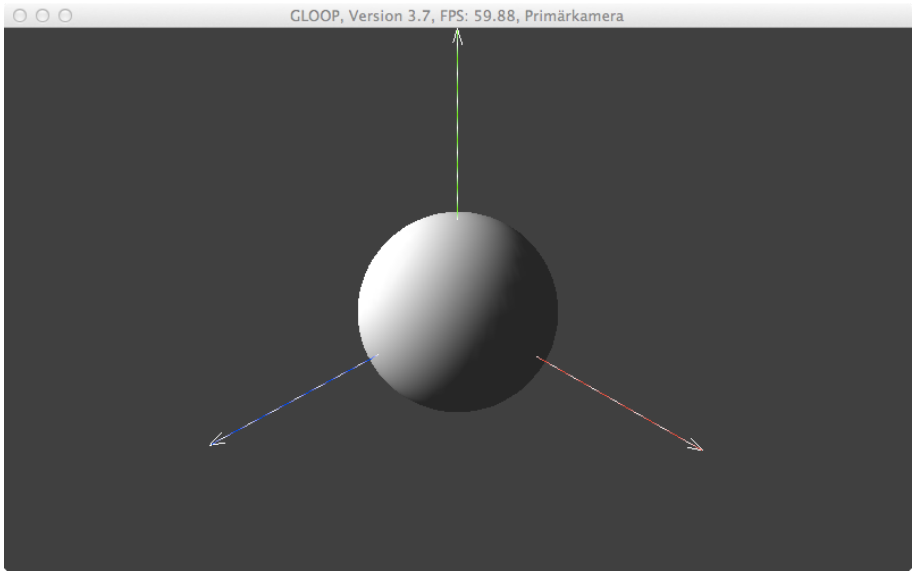

Abbildung 1: Kugel mit eingeblendeten Koordinatenachsen

Die folgende Erweiterung macht aus der Kugel einen Globus, indem eine Textur der Erde[1] ergänzt wird (vgl. Abbildung 2).

```
1 [...]
2 meineKugel = new GLKugel (0,0,0, 50);
3 meineKugel.setzeTextur("Erde.jpg");
4 [...]
```

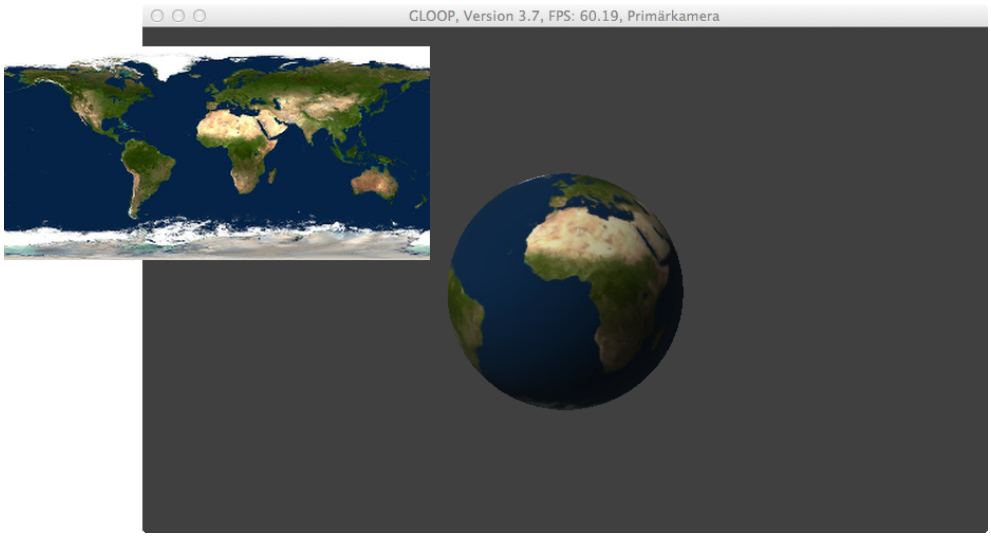

Abbildung 2: Kugel mit Erdtextur

Möchte man mehr als ein stehendes Bild zeigen, so muss der oben gegebene Quellcode um eine Animationschleife ergänzt werden. Der folgende Quellcode stellt eine einfache bewegte Simulation der Erde und des Mondes dar (vgl. Abbildung 3). Die Mondkugel ist ebenfalls mit einer entsprechenden Textur[2] versehen.

[1]The Celestia Motherlode, URL: www.celestiamotherlode.net/catalog/earth.php, abgerufen: Januar 2014
[2]The Celestia Motherlode, URL: www.celestiamotherlode.net/catalog/moon.php, abgerufen: Januar 2014

```
1  import GLOOP.*;
2  class ErdeMond {
3    GLKamera   dieKamera;
4    GLLicht    dasLicht;
5    GLKugel    dieErde, derMond;
6    GLTastatur dieTastatur;
7
8    ErdeMond(){
9      dieKamera   = new GLSchwenkkamera();
10     dasLicht    = new GLLicht();
11     dieTastatur = new GLTastatur();
12
13     dieErde = new GLKugel (0,0,0,150);
14     dieErde.setzeTextur ("Erde.jpg");
15     dieErde.drehe(90,0,0);
16
17     derMond = new GLKugel (250,0,0,50);
18     derMond.setzeTextur ("Mond.jpg");
19     derMond.drehe(90,0,0);
20
21     while (!dieTastatur.Esc()){
22       dieErde.drehe(0,1,0);
23       derMond.drehe(0,-0.25,0,  0,0,0);
24       Sys.warte();
25     }
26     Sys.beenden();
27   }
28 }
```

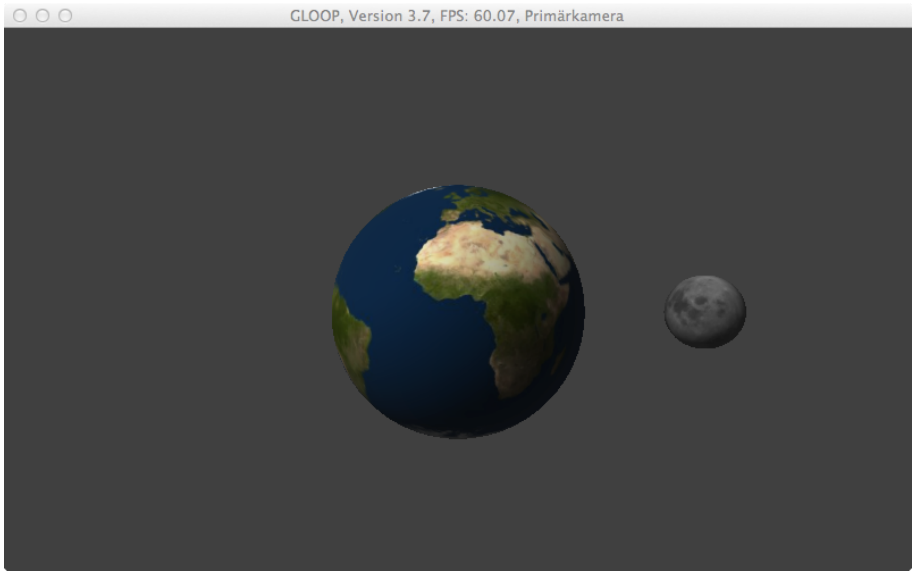

Abbildung 3: Erde und Mond als Animation

Zunächst werden zwei Kugel erstellt und mit den entsprechenden Texturen versehen. Anschließend werden sie aus optischen Gründen um 90 Grad gedreht. Die darauf folgende Animationschleife läuft, solange nicht die Taste `ESC` gedrückt wurde, und lässt beide Kugeln rotieren.

Die Methode `drehe` rotiert das Objekt um drei Drehachsen, die zu den Koordinatenachsen parallel sind. Im Fall der Kugel wird für die X-, Y- und Z-Drehachse jeweils ein Winkel angegeben. Alle drei Drehachsen laufen durch den Mittelpunkt des Objekts, so dass eine Eigenrotation realisiert wird. Im Fall des Mondes wird eine Variante der Methode `drehe` verwendet, die zusätzlich zu den drei Winkeln noch einen Punkt übergeben bekommt, durch den die Drehachsen laufen sollen. Auf diese Weise werden Drehungen realisiert, die keine Eigenrotationen des Objekts sind. Im vorliegenden Beispiel dreht sich die Erde um `1` Grad um ihren eigenen Mittelpunkt, wohingegeben der Mond um `-0.25` Grad um den Mittelpunkt der Erde rotiert (siehe auch Abschnitt 3.1).

Möchte man komplexere Simulationen realisieren, so wird man nicht mehr damit auskommen, einfachen geometrischen Objekten Aufträge zu erteilen. Vielmehr wir man dazu übergehen, mehrere Objekte vom Typ `GLObjekt` zu neuen Klassen zusammenzufassen oder Objekte mittels Vererbung zu spezialisieren. Die in Abbildung 4 gezeigte Teilmodellierung einer Analoguhr gibt einen ersten Eindruck davon.

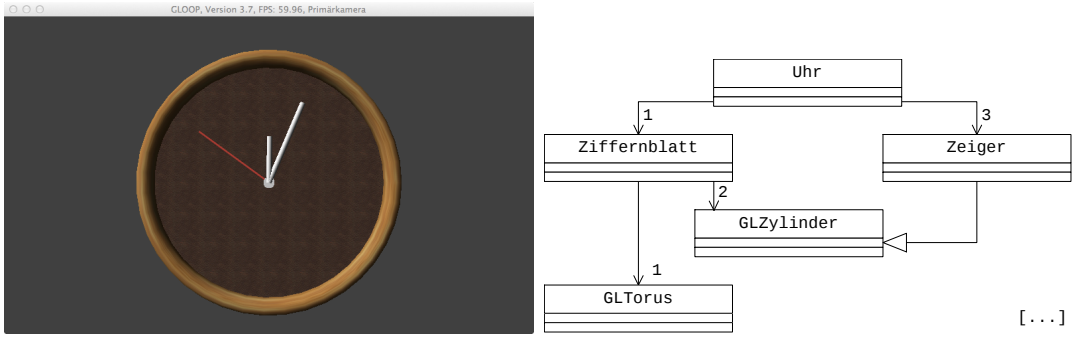

Abbildung 4: Analoguhr mit Teilmodellierung

Im Folgenden sollen zunächst die grundlegenden Möglichkeiten zur Konstruktion und Manipulation einer Szene mit GLOOP aufgezeigt werden.

2 Konstruktion einer Szene

Um die dreidimensionale Szene zu konstruieren, stehen eine Reihe von Objekten zur Verfügung, die im Folgenden mit ihrem Standardkonstruktor vorgestellt werden sollen. Sie alle erben von der abstrakten Klasse `GLObjekt` und verfügen somit – abgesehen von wenigen Ausnahmen – über den gleichen Methodensatz, auf den im Anschluss ebenfalls eingegangen wird (siehe Abschitt 3).

2.1 Standardobjekte als Grundbausteine

Die Konstruktoren der Standardobjekte folgen immer der gleichen Logik. Zuerst wird mit den Parametern `pX`, `pY` und `pZ` vom Typ `double` der Mittelpunkt des zu erstellenden Objektes übergeben. Anschließend werden die Ausdehnungen des Objektes abhängig vom speziellen Typ angegeben.

In den folgenden Beispielen wird das Objekt immer im Ursprung, also bei `(0,0,0)` erstellt.

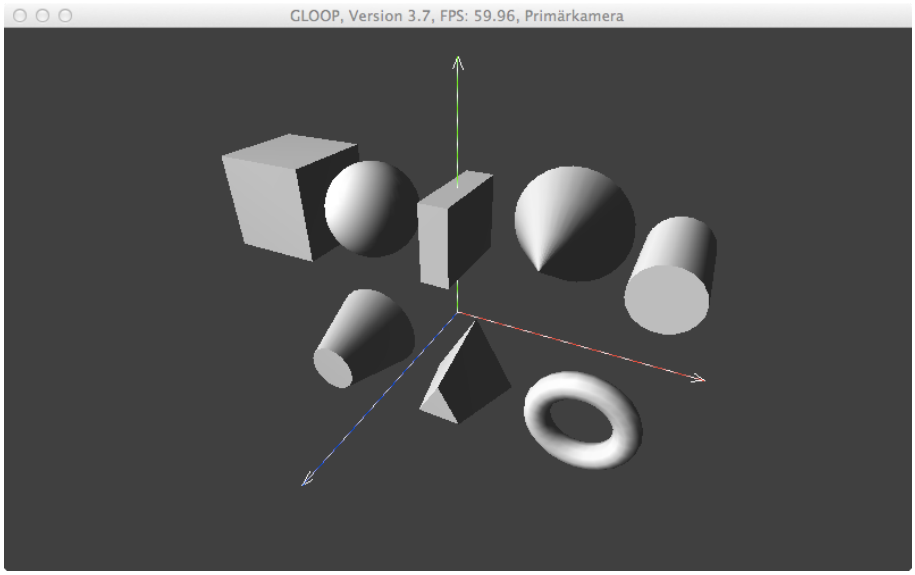

Abbildung 5: Alle Standardobjekte im Überblick

Die Klasse GLKugel (Oberklasse `GLObjekt`)

Konstr. `GLKugel(double pX, double pY, double pZ, double pRadius)`
Erstellt eine Kugel mit dem Radius `pRadius`.

Beispiel:

```
meineKugel = new GLKugel(0,0,0, 40);
```

Die Klasse GLWuerfel (Oberklasse `GLObjekt`)

Konstr. `GLWuerfel(double pX, double pY, double pZ, double pSeitenlaenge)`
Erstellt einen Würfel mit der Seitenlänge `pSeitenlaenge`.

Beispiel:

```
meinWuerfel = new GLWuerfel (0,0,0, 40);
```

Die Klasse GLQuader (Oberklasse `GLObjekt`)

Konstr. `GLQuader(double pX, double pY, double pZ, double pLX, double pLY, double pLZ)`
Erstellt einen Quader mit den Abmessungen `pLX`, `pLY`, `pLZ` bzgl. der drei Raumdimensionen. D.h. `pLX` ist die Breite, `pLY` die Höhe und `pLZ` die Tiefe.

Beispiel:

```
meinQuader = new GLQuader(0,0,0, 30,30,80);
```

Die Klasse GLKegel (Oberklasse GLObjekt)

Konstr. `GLKegel(double pX, double pY, double pZ, double pRadius, double pHoehe)`

Erstellt einen Kegel mit der Höhe `pHoehe` und mit dem Grundflächenradius `pRadius`. Das Objekt ist parallel zur Z-Achse ausgerichtet. Die Spitze weist in Richtung der Z-Achse.

Beispiel:

```
meinKegel = new GLKegel(0,0,0, 20,80);
```

Die Klasse GLZylinder (Oberklasse GLObjekt)

Konstr. `GLZylinder(double pX, double pY, double pZ, double pRadius, double pHoehe)`

Erstellt einen Zylinder mit der Höhe `pHoehe` und mit dem Grundflächenradius `pRadius`. Das Objekt ist parallel zur Z-Achse ausgerichtet.

Beispiel:

```
meinZylinder = new GLZylinder(0,0,0, 20,80);
```

Die Klasse GLKegelstumpf (Oberklasse GLObjekt)

Konstr. `GLKegelstumpf(double pX, double pY, double pZ, double pRadius1,double pRadius2, double pHoehe)`

Erstellt einen Kegelstumpf mit der Höhe `pHoehe` und den Radien `pRadius1` und `pRadius2`. Das Objekt ist parallel zur Z-Achse ausgerichtet.

Beispiel:

```
meinKegelstumpf = new GLKegelstumpf(0,0,0, 10,20,80);
```

Die Klasse GLTorus (Oberklasse GLObjekt)

Konstr. `GLTorus(double pX, double pY,double pZ, double pRadius, double pDicke)`

Erstellt einen Torus mit dem Radius `pRadius` und der Dicke `pDicke`. Der Torus liegt in einer zur XY-Ebene parallelen Ebene.

Beispiel:

```
meinTorus = new GLTorus(0,0,0, 80,20);
```

Die Klasse GLPrismoid (Oberklasse GLObjekt)

Konstr. `GLPrismoid(double pX, double pY, double pZ, double pRadius1, double pRadius2, int pEckenanzahl, double pHoehe)`

Erstellt einen Prismoiden entlang der Z-Achse. Die vordere und die hintere Fläche des Prismoiden haben `pEckenanzahl` Ecken. Der Radius des Umkreises der vorderen Fläche beträgt `pRadius1`, der der hinteren `pRadius2`. Die Höhe des Prismoiden entlang der Z-Achse wird mit `pHoehe` angegeben.

Mit Hilfe der richtigen Parameterwahl können spezielle Prismoide wie z.B. Prismen, Pyramidenstümpfe oder Pyramiden erzeugt werden.

Beispiel:

```
meinPrismoid = new GLPrismoid(0,0,0, 50,50,3,80);
```

Hier wird ein Prisma mit dreieckiger Grund- bzw. Deckelfläche erzeugt.

Auftrag `void setzeMantelglaettung(boolean pGlaettung)`

Bietet die Möglichkeit, die Glättung der Kanten zwischen den Mantelflächen des Prismoiden ein- bzw. auszuschalten. Bei eingeschalteter Kantenglättung und hoher Eckenzahl kann so z.B. der Eindruck eines Zylinders entstehen.

Des Weiteren sind alle Konstruktoren überladen, so dass gleich beim Erstellen ein Texturobjekt oder eine Bilddatei übergeben werden kann. Die Oberfläche des Objektes wird dann entsprechend gestaltet. Das folgende Beispiel zeigt das:

```
1  //Uebergabe einer Bilddatei
2  meineKugel1 = new GLKugel (0,0,0, 50, "Holz.jpg");
3  [...]
4  //Uebergabe eines Texturobjektes
5  GLTextur texHolz = new GLTextur ("Holz.jpg");
6  meineKugel2 = new GLKugel (0,0,0,50,texHolz);
```

Des Weiteren steht die Klasse `GLLicht` zur Verfügung, um die Szene zu erhellen.

Die Klasse GLLicht (Oberklasse GLObjekt)

Konstr. `GLLicht()`

`GLLicht(double pX, double pY, double pZ)`

Erstellt eine weiße Lichtquelle an der Stelle `(-10000, 10000, 10000)`, wenn kein Parameter übergeben wird, bzw. an der Stelle `(pX, pY, pZ)`.

Beispiel:

```
meinLicht = new GLLicht();
```

Auftrag `void setzeAbschwaechung(double pAbschwaechung)`

Setzt, wie stark sich das Licht in der Entfernung abschwächt.

Auftrag `void setzeFarbe(double pR, double pG, double pB)`

Setzt die Farbe der Lichtquelle (vgl. Methode von `GLObjekt`).

Auftrag `void setzeGlanzlicht(double pR, double pG, double pB)`

Setzt Farbe und Intensität des Glanzlichtanteils der Lichtquelle.

Auftrag `void setzeHintergrundlicht(double pR, double pG, double pB)`

Setzt den Hintergrundlichtanteil der Lichtquelle.

2.2 Spezialobjekte als Ergänzung

Neben den in Abschnitt 2.1 aufgeführten geometrischen Objekten existieren die Klassen `GLBoden`, `GLHimmel`, `GLNebel`, `GLTerrain` und `GLTafel`. Objekte dieser Klassen können ebenfalls in die Szene eingefügt werden, stellen aber keine eigentlichen dreidimensionalen Körper dar, sondern ermöglichen es, die Umgebung ansprechender zu gestalten bzw. im Fall von `GLTafel` Beschriftungen in die Welt zu setzen. Die Klassen `GLTerrain` und `GLTafel` erben ebenfalls von `GLObjekt` (siehe Abschnitt 3).

Die Klasse GLBoden (Oberklasse `Object`)

Konstr. `GLBoden(String pBilddatei)`
`GLBoden(GLTextur pTextur)`
Erstellen eine endlose Ebene in der Szene, die mit der im Parameter übergebenen Textur gekachelt ist. Die Ebene entspricht immer der XZ-Ebene. Des Weiteren kann die Kamera nicht mehr unter die Ebene bewegt werden.

Beispiel:
```
meinBoden = new GLBoden("Sand.jpg");
```

Anfrage `GLTextur gibTextur()`
Liefert das Texturobjekt, an welches das Objekt aktuell gebunden ist.

Auftrag `void loesche()`
Löscht den Boden aus der Szene.

Auftrag `void setzeFarbe(double pR, double pG, double pB)`
Setzt die Farbe des Objektes. pR = Rotanteil, pG = Grünanteil, pB = Blauanteil (vgl. Methode von `GLObjekt`).

Auftrag `void setzeSichtbarkeit(boolean pS)`
Macht das Objekt sichtbar bzw. unsichtbar.

Auftrag `void setzeTextur(GLTextur pTextur)`
Überzieht das Objekt mit der übergebenen Textur.

Auftrag `void setzeTextur(String pDateiname)`
Erstellt aus der übergebenen Datei ein Texturobjekt und überzieht das Objekt mit dieser Textur.

Die Klasse GLHimmel (Oberklasse `Object`)

Konstr. `GLHimmel(String pBilddatei)`
`GLHimmel(GLTextur pTextur)`
Erstellt eine Himmelssphäre, die auf der Innenseite die im Parameter übergebene Bildtextur zeigt. Auf diese Weise kann ein Hintergrundbild der gesamten Szene erzeugt werden.

Beispiel:
```
meinHimmel = new GLHimmel("Sterne.jpg");
```

Anfrage `GLTextur gibTextur()`
Liefert das Texturobjekt, an welches das Objekt aktuell gebunden ist.

Auftrag `void loesche()`
Löscht den Himmel aus der Szene.

Auftrag `void setzeFarbe(double pR, double pG, double pB)`
Setzt die Farbe des Objektes. pR = Rotanteil, pG = Grünanteil, pB = Blauanteil (vgl. Methode von `GLObjekt`).

Auftrag `void setzeSichtbarkeit(boolean pS)`
Macht das Objekt sichtbar bzw. unsichtbar.

Auftrag `void setzeTextur(GLTextur pTextur)`
Überzieht das Objekt mit der übergebenen Textur.

Auftrag `void setzeTextur(String pDateiname)`

Erstellt aus einer Datei ein Texturobjekt und überzieht das Objekt mit dieser Textur.

Die Klasse GLTerrain (Oberklasse `GLObjekt`)

Konstr. `GLTerrain(double pX, double pY, double pZ, String pMap)`

Erstellt eine Landschaftsfläche der Größe 512x512 in der Szene, deren Mittelpunkt bei `(pX,pY,pZ)` ist. Diese Fläche zeigt eine Hügelung, die der in `pMap` übergebenen *Heightmap* entspricht. Die Heightmap `pMap` muss ein Graustufenbild mit den Abmessungen 512x512 sein. Je heller ein Pixel im Bild ist, umso höher wird im Terrain diese Stelle sein. Schwarz entspricht dabei der Höhe 0 und Weiß der Höhe 255.

Beispiel:

```
dasTerrain = new GLTerrain(0,0,0,"Heightmap.jpg");
```

Anfrage `double gibHoehe(double pX, double pZ)`

Liefert die Höhe des Terrains an der Stelle `(pX,pY)`. Die Parameter stellen dabei relative Koordinaten auf der Oberfläche des Terrains dar. Der Punkt `(0,0)` entspricht der Mitte des Terrains.

Auftrag `void setzeAbmessungen(double pBreite, double pHoehe, double pTiefe)`

Setzt die Abmessungen des Terrains neu. `pHoehe` entspricht der maximal möglichen Höhe des Terrains.

Beispiel:

```
dasTerrain.setzeAbmessungen(1024,255,1024);
```

Setzt das Terrain auf die doppelte Größe ohne die Höhen zu verändern.

Auftrag `void setzeHoehen(float[][] pHoehen)`

Die Höhen des Terrains können mit Hilfe eines Feldes (`float[512][512]`) übergeben werden. Die einzelnen Werte des Feldes müssen zwischen 0 und 1 liegen.

Auftrag `void zeigeUnterseite(boolean pU)`

Die Unterseite des Terrains wird zur Leistungssteigerung im Normalfall nicht vollständig dargestellt (Backfaceculling). Mit dieser Methode kann die vollständige Darstellung der Unterseite an- bzw. ausgeschaltet werden.

Die Klasse GLTafel (Oberklasse `GLObjekt`)

Konstr. `GLTafel(double pX, double pY, double pZ, double pLX, double pLY)`

Erstellt eine zweidimensionale, beschriftbare Tafel in der Szene. Sie hat die Breite pLX und die Höhe pLY. Ein Objekt dieser Klasse kann dazu verwendet werden, Textinformationen in der Szene zu realisieren. Des Weiteren kann die Tafel sich automatisch in Richtung Kamera drehen (Billboarding) oder auch im Kamerabild fixiert werden.

Beispiel:

```
meineTafel = new GLTafel(0,0,0, 50,30);
meineTafel.setzeText("Hallo Welt!", 16);
```

Auftrag `void setzeText(String pText, double pGroesse)`

Setzt die Aufschrift der Tafel auf `pText`. Die Darstellungsgröße der Zeichenkette entspricht `pGroesse`. Ist der Text größer als die Tafel, so wird sie automatisch entsprechend vergrößert.

Anfrage `String gibText()`

Liefert den auf der Tafel angezeigten Text.

Auftrag `void setzeTextfarbe(double pR, double pG, double pB)`

Setzt die Farbe des Textes (siehe `setzeFarbe` der Klasse `GLObjekt`).

Auftrag `void setzeAutodrehung(boolean pD)`

Schaltet die automatische Drehung der Tafel in Richtung der Kamera ein bzw. aus (Billboarding).

Auftrag `void setzeAutodrehung(boolean pD, int pAchsenfixierung)`

Schaltet die automatische Drehung der Tafel in Richtung der Kamera ein bzw. aus. Der Parameter pAchsenfixierung kann die Werte 1-3 annehmen und schränkt die automatische Ausrichtung auf die Drehung um nur eine Achse ein. (1 = nur um X-Achse drehen; 2 = nur um Y-Achse drehen; 3 = nur um Z-Achse drehen)

Auftrag `void setzeKamerafixierung(boolean pF)`

Schaltet die Fixierung der Tafel im Kamerabild ein bzw. aus. Wird die Fixierung eingeschaltet, so bleibt die Tafel immer an der gleichen Stelle im Kamerabild zu sehen wie im Augenblick der Fixierung, selbst dann, wenn die Kamera bewegt wird.

Auftrag `void setzeFaecherung(int pFaecheranzahl)`

Stellt die Tafel in Form von mehreren Fächern dar. Die Anzahl der Fächer wird mit `pFaecherzahl` angegeben. Bei mehrfacher Fächerung ist eine Autodrehung oder Kamerafixierung nicht mehr möglich.

Auftrag `void setzeBeleuchtung(boolean pB)`

Schaltet die Beleuchtung durch Lichtquellen für die Tafel an bzw. aus. Wird die Beleuchtung ausgeschaltet, so wird die Tafel unabhängig von Lichtquellen erhellt dargestellt.

Die Klasse GLNebel (Oberklasse `Object`)

Konstr. `GLNebel()`

Erstellt ein Nebelobjekt, das die Szene mit gleichmäßigem Dunst ausfüllt. Der Nebel hat weder eine Position noch eine Größe, sondern ermöglicht es vielmehr, die Atmosphärenbeschaffenheit zu bestimmen. Die Klasse erbt daher auch nicht von `GLObjekt`.

Auftrag `void loesche()`

Löscht den Nebel aus der Szene.

Auftrag `void setzeNebelbereich(double pAnfang, double pEnde)`

Der Nebelbereich wird gesetzt. Dazu wird angegeben, ab welcher Entfernung von der Kamera der Nebel anfängt (`pAnfang`) und ab welcher Entfernung ein Gegenstand vollständig vom Nebel verschluckt wurde (`pEnde`). Zwischen `pAnfang` und `pEnde` nimmt die Nebeldichte linear zu.

Beispiel:

```
GLNebel lN = new GLNebel();
lN.setzeNebelbereich(1000, 2000);
```

Ab der Entfernung `1000` vor der Kamera fängt der Nebel an. Objekte, die weiter als `2000` von der Kamera entfernt sind, sind nicht mehr zu sehen.

Auftrag `void setzeFarbe(double pR, double pG, double pB)`

Setzt die Farbe des Nebels. Die Standardfarbe ist weiß. Blauer Nebel wirkt wie Wasser, schwarzer wie Dunkelheit und roter wie Feuer.

2.3 Alternative Konstruktoren

Alle Klassen, die Unterklassen von `GLObjekt` sind, verfügen neben den bereits vorgestellten Standardkonstruktoren über weitere Konstruktorvarianten. Die Parameterliste jedes Standardkonstruktors kann mit der Angabe einer Textur ergänzt werden, so dass das neue Objekt sofort mit dem entsprechenden Bild überzogen wird. Die Textur kann als Objekt vom Typ `GLTextur` übergeben werden, oder es kann eine Bilddatei als `String` übergeben werden.

Beispiel

```
1  //Variante 1: Erstellen einer Kugel mit einem Texturobjekt
2  //als Uebergabe
3  GLTextur meineTextur = new GLTextur("Erde.jpg");
4  GLKugel meineKugel1 = new GLKugel (0,0,0, 50, meineTextur);
5
6  [...]
7
8  //Variante 2: Erstellen einer Kugel mit einer Bilddatei
9  //als Uebergabe
10 GLKugel meineKugel2 = new GLKugel (0,0,0, 50, "Erde.jpg");
```

Des Weiteren gibt es Konstruktorvarianten, bei denen die Position des Objekts als Objekt vom Typ `GLVektor` übergeben werden kann (vgl. Abschnitt 7).

3 Manipulation von Objekten

Die Klasse `GLObjekt` dient als Oberklasse für alle im Raum zu platzierenden Objekte, mit Ausnahme der Kamera, des Nebels und der Klassen für Boden und Himmel. Sie ist abstrakt und kann somit nicht selbst instanziiert werden. Auch Lichtquellen sind Unterklassen von `GLObjekt`. Die Klasse stellt eine Reihe von Methoden zur Manipulation des Objektes zur Verfügung, die im Folgenden in zwei Kategorien eingeordnet werden sollen. Zum einen stehen sogenannte Basisbefehle zur Verfügung, die in fast jedem Unterrichtsprojekt zum Einsatz kommen und somit jedem Lernenden bekannt sein sollten. Zum anderen werden Spezialbefehle unterstützt, die nur im Kontext spezieller Unterrichtsprojekte benötigt werden und somit nur bei Bedarf vermittelt werden müssen. Die meisten Unterrichtsprojekte sind allein mit Basisbefehlen zu realisieren, welche im Folgenden kurz beschrieben werden:

3.1 Basismethoden

Der Basisbefehlssatz der Klasse `GLObjekt` lässt sich in mehrere Bereiche gliedern. Für die Bereiche der *Positionierung*, *Drehung* und *Skalierung* stehen jeweils Befehle in der absoluten Variante und der relativen Variante zur Verfügung. In weiteren Bereichen sind Befehle zur *Oberflächenmanipulation* und zur *Positionsabfrage* zu finden. Des Weiteren können Objekte wieder aus der Szene entfernt werden.

3.1.1 Positionierung

Zur Positionierung des Objektes im Raum umfasst der Basisbefehlssatz zwei Methoden.

Auftrag `void setzePosition(double pX, double pY, double pZ)`

Die Position des Objektes wird auf den Punkt `(pX,pY,pZ)` gesetzt. Es wird eine absolute Positionierung durchgeführt, d.h. die vorherige Position des Objekts ist nicht von Bedeutung. Des Weiteren wird die Drehung des Objekts nicht verändert.

Auftrag `void verschiebe(double pX, double pY, double pZ)`

Das Objekt wird um `pX`, `pY` und `pZ` entlang der entsprechenden Koordinatenachsen verschoben. Die Endposition des Objektes ist abhängig von der vorherigen Position. D.h. es handelt sich um eine relative Positionierung.

3.1.2 Drehung

Zur Drehung des Objektes umfasst der Basisbefehlssatz ebenfalls zwei Methoden. Die Methode zur relativen Drehung ist überladen und kann mit zwei Parametersätzen aufgerufen werden.

Auftrag `void setzeDrehung(double pWX, double pWY, double pWZ)`

Dreht das Objekt um durch den Mittelpunkt des Objektes gehende Parallelen der Koordinatenachsen, unabhängig von der vorangegangenen Ausrichtung des Objektes auf die angegebenen Drehwinkel. D.h. das Objekt wird um seinen eigenen Mittelpunkt gedreht. Die Drehung wird absolut durchgeführt. Die Parameter `pWX`, `pWY` und `pWZ` sind die Drehwinkel um die jeweiligen Achsen.

Beispiel:

```
meinWuerfel.setzeDrehung(45,0,45);
```

Der Würfel wird auf eine Spitze gestellt.

Auftrag `void drehe(double pWX, double pWY, double pWZ)`

Das Objekt wird wie zuvor um durch den Mittelpunkt des Objektes gehende Parallelen der Koordinatenachsen gedreht. Es wird jedoch nicht vom Ursprungszustand des Objektes, sondern von seiner aktuellen Drehung im Raum ausgegangen. Es handelt sich also um eine relative Drehung. Gedreht wird ebenfalls um den Mittelpunkt des Objektes.

Beispiel:

```
meineKugel.drehe(0,1,0);
```

In einer Schleife aufgerufen wird die Kugel in 360 Schritten eine Eigenrotation durchgeführt haben.

Auftrag `void drehe(double pWX, doube pWY, double pWZ, double pX, double pY, double pZ)`

Diese Syntaxvariante wird verwendet, wenn keine Eigenrotation realisiert werden soll, sondern stattdessen um Achsen rotiert werden soll, die durch einen bestimmten Punkt gehen. Dieser Punkt `(pX, pY, pZ)` wird nach den drei Winkeln angegeben.

Beispiel:

```
meineKugel.drehe(0,1,0, 0,0,0);
```

Dreht die Kugel um den Koordinatenursprung. In einer Schleife aufgerufen könnte so z.B. eine Planetenrotation um die Sonne realisiert werden.

3.1.3 Skalierung

Mit Hilfe der Skalierungsbefehle kann die Größe und auch die Form eines Objektes nach seiner Instanziierung geändert werden. Auch hier stehen wieder absolute und relative Befehlsvarianten zur Verfügung.

Auftrag `void setzeSkalierung(double pG)`

Die Größe des Objektes wird ausgehend von der Größe zum Instanzierungszeitpunkt um den Faktor `pG` skaliert.

Beispiel:

```
meineKugel.setzeSkalierung(2);
```

Die Kugel ist nun doppelt so groß wie zum Instanziierungszeitpunkt. Die Skalierung erfolgt absolut, d.h. vorangegangene Skalierungen sind ohne Bedeutung.

Auftrag `void skaliere(double pG)`

Die Größe des Objektes wird um den Faktor `pG` skaliert.

Beispiel:

```
meineKugel.skaliere(2);
```

Die Kugel ist nun doppelt so groß wie zuvor. Wird der Befehl zweimal ausgeführt, ist das Objekt viermal so groß.

Auftrag `void skaliere(double pGX, double pGY, double pGZ)`

Mit Hilfe dieser beiden Befehle kann in Richtung der drei Koordinatenachsen unterschiedlich skaliert und somit eine Verzerrung des Objektes erreicht werden. Ansonsten entsprechen sie den zuvor genannten.

Auftrag `void setzeSkalierung(double pGX, double pGY, double pGZ)`

Die Methode stellt die absolute Variante der Methode `skaliere` mit drei Parametern dar.

Beispiel:

```
meineKugel.setzeSkalierung(2,1,2)
```

Die Kugel wird in X-Richtung und in Z-Richtung um den Faktor 2 skaliert, in Y-Richtung wird ihre Ausdehnung nicht verändert. Die Kugel wird also zu einem Ellipsoiden verzerrt.

3.1.4 Oberflächenmanipulation

Im Basisbefehlssatz gibt es zwei Befehle, welche die Oberfläche eines Objektes verändern können.

Auftrag `void setzeFarbe(double pR, double pG, double pB)`
Die Methode setzt die Farbe des Objektes entsprechend der Parameter `pR`, `pG` und `pB`, wobei es sich um eine RGB-Farbmischung handelt. Der Wertebereich der drei Parameter entspricht dem Intervall `[0;1]`.

Beispiel:
```
meineKugel.setzeFarbe(0,1,0);
```
Die Kugel wird grün.

Auftrag `void setzeTextur(String pDateiname)`
Das Objekt wird mit einer Textur überzogen, welche aus der im Parameter übergebenen Bilddatei erstellt wird. Dabei sind prinzipiell alle gängigen Dateiformate erlaubt. In der Regel sollte eine Datei im JPG- oder PNG-Format verwendet werden. Abhängig von der zum Einsatz kommenden Grafikkarte sind jedoch Anforderungen an die Abmessungen der Bilddatei zu stellen. Universell einsetzbar sind quadratische Bilder, deren Abmessungen Zweierpotenzen entsprechen. Je kleiner das verwendete Bild ist, umso verschwommener bzw. "pixeliger"wird die Darstellung auf der Oberfläche des Objektes sein. Sind die Bildabmessungen zu groß, dauert das Laden der Textur und somit der Programmstart recht lange. Des Weiteren wird der Graphikspeicher stark belastet. Für normale Objekttexturen sollte eine Abmessung von `512x512` Bildpunkten nicht überschritten werden.

Beispiel:
```
meineKugel.setzeTextur("Erde.jpg");
```
Die Kugel wird mit einer Textur aus dem Bild `Erde.jpg` überzogen. So kann eine Weltkugel visualisiert werden.

Auftrag `void setzeTextur(GLTextur pTextur)`
Der Befehl zum Setzen einer Textur ist überladen und kann statt eines Dateinamens auch ein Objekt von Typ `GLTextur` als Parameter erhalten. Diese Variante sollte bei umfangreicheren Projekten Verwendung finden, damit bei vielen Objekte, die alle dieselbe Textur erhalten sollen, die Bilddatei nicht für jedes Objekt separat in den Speicher geladen werden muss.

Beispiel:
```
GLTextur lT = new GLTextur("Felsen.jpg");
meineKugel1.setzeTextur(lT);
meineKugel2.setzeTextur(lT);
```

3.1.5 Positionsabfrage

Die Position des Objekts kann mit den folgenden Methoden abgefragt werden.

Anfragen `double gibX()`
`double gibY()`
`double gibZ()`
Liefert die entsprechende Koordinate der Position des Objekts.

3.1.6 Löschen

Das Objekt kann mit Hilfe der folgenden Methode aus der Szene entfernt werden.

Auftrag `void loesche()`

Das Objekt wird aus der Szene entfernt, verbleibt aber im Speicher. Um es vollständig zu entfernen, muss zusätzlich die entsprechende Referenz auf `null` gesetzt werden.

Beispiel:

```
meineKugel.loesche();
meineKugel = null;
```

Die Kugel wird erst aus der Szene gelöscht und anschließend wird die Referenz darauf auf `null` gesetzt. Existiert keine weitere Referenz auf das Kugelobjekt, wird es von Java aus dem Speicher entfernt.

3.2 Spezialmethoden

Der Spezialbefehlssatz von `GLObjekt` wird in der Regel nur selten verwenden, kann aber bei machen Projekten hilfreich sein. Folgende Methoden stehen zur Verfügung:

Auftrag `void rotiere(double pWinkel, double pRX, double pRY, double pRZ, double pX, double pY, double pZ)`

Rotiert das Objekt um die angegebene Achse im Raum. Die Richtung der Achse wird mit `(pRX, pRY, pRZ)` angegeben (Richtungsvektor). Der Punkt `(pX, pY, pZ)` ist derjenige Punkt, durch den die Achse läuft (Ortsvektor).

Auftrag `void setzeGlanz(double pR, double pG, double pB, int pHaerte)`

Setzt die Farbe `(pR, pG, pB)` und die Intensität (`pHaerte`) des Glanzes des Objektes.

Auftrag `void setzeSelbstleuchten (double pR, double pG, double pB)`

Setzt das Selbstleuchten eines Objektes auf die angegebene Farbe. Das Objekt wirkt dann, als würde es von innen heraus leuchten. Es wird jedoch nicht zur Lichtquelle.

Auftrag `void setzeMaterial(float[][] pM)`

Setzt die Materialbeschaffenheit des Objektes. Für `pM` können die folgenden Konstanten aus der statischen Klasse `GLMaterial` eingesetzt werden:
`GOLD, JADE, RUBIN, GLAS, ROTGLAS, GRUENGLAS, BLAUGLAS, MESSING, KUPFER, KUPFER_POLIERT, BRONZE, BRONZE_POLIERT, SILBER, SILBER_POLIERT, CHROM, ZINN, SMARAGD, OBSIDIAN, PERLMUTT,TUERKIS, PLASTIK, GUMMI`

Beispiel:

```
meineKugel.setzeMaterial(GLMaterial.GLAS);
```

Auftrag `void setzeSichtbarkeit(boolean pS)`

Ermöglicht es, das Objekt unsichtbar bzw. sichtbar zu machen.

Auftrag `GLTextur gibTextur()`

Liefert das aktuell verwendete Texturobjekt.

Auftrag `void setzeQualitaet(int pQ)`

Setzt die Darstellungsqualität des Objektes. Je höher der Wert von pQ ist, umso präziser wird das Objekt dargestellt. Hohe Einstellungen gehen auf Kosten der Systemleistung.

4 Kamerasteuerung

Um den im Rechner simulierten Raum zu visualisieren, bedarf es einer Kamera. Sie soll die Objekte des Raumes in Echtzeit darstellen. Dazu muss ein Objekt der Klasse `GLKamera` erstellt werden.

4.1 Standardkamera

Für den normalen Programmablauf steht eine Kamera in Form der Klasse `GLKamera` zu Verfügung. Sie wird normalerweise am Punkt `(0,0,500)` positioniert, kann mit einer Reihe von Befehlen aber im Raum bewegt werden. Diese Befehle orientieren sich teilweise an ihren Entsprechungen in der Klasse `GLObjekt`, sind aber nicht ganz mit ihnen identisch. Des Weiteren gibt es eine Reihe von kameraspezifischen Methoden.

Die Klasse GLKamera (Oberklasse `Object`)

Konstr. `GLKamera()`

`GLKamera(int pBreite, int pHoehe)`

Erstellt eine Kamera an der Stelle `(0,0,500)`, die auf den Punkt `(0,0,0)` blickt. Die Y-Achse zeigt nach oben. Werden keine Parameter übergeben, so wird das Kamerabild im Vollbildmodus dargestellt. Wird ein Wert in `pBreite` und ein Wert in `pHoehe` übergeben, so wird das Bild in einem Fenster der entsprechenden Abmessungen dargestellt.

Beispiel:

```
meineKamera = new GLKamera();
```

Erstellt eine Kamera im Vollbildmodus.

Auftrag `void setzePosition(double pX, double pY, double pZ)`

Setzt die Position der Kamera auf den Punkt `(pX, pY, pZ)`. Der Blickpunkt der Kamera wird dabei nicht geändert.

Auftrag `void setzeBlickpunkt(double pX,double pY,double pZ)`

Setzt den Blickpunkt der Kamera auf den Punkt `(pX, pY, pZ)`. Die Position der Kamera wird dabei nicht geändert.

Auftrag `void setzeScheitelrichtung(double pX, double pY, double pZ)`

Setzt die Scheitelrichtung der Kamera auf `(pX, pY, pZ)`. D.h. die Kamera wird so gedreht, dass die angegebene Richtung diejenige ist, die aus Kameraperspektive direkt nach oben zeigt. Position und Blickpunkt ändern sich dabei nicht.

Beispiel:

```
meineKamera.setzeScheitelrichtung(1,0,0);
```

Die Kamera liegt nun auf der Seite.

Auftrag `void an()`

Die Kamera wird eingeschaltet und erzeugt Bilder der Szene. Dies ist die Standardeinstellung.

Auftrag `void aus()`

Die Kamera wird ausgeschaltet. Es werden keine neuen Bilder mehr erzeugt. Das zuletzt erstellte Bild wird aber weiter angezeigt.

Auftrag `void erstelleEinzelbild()`

Ist die Kamera aus, kann mit diesem Befehl ein einzelnes Bild erstellt werden.

Auftrag `void rotiere(double pWinkel, double pRX, double pRY, double pRZ, double pX, double pY, double pZ)`

Rotiert die Kamera um die angegebene Achse im Raum. Die Richtung der Achse wird mit `(pRX,pRY,pRZ)` angegeben (Richtungsvektor). `(pX,pY,pZ)` ist ein Punkt, durch den die Achse läuft (Ortsvektor).

Auftrag `void verschiebe(double pX, double pY, double pZ)`

Verschiebt die Kamera um den Wert `pX` auf der X-Achse, `pY` auf der Y-Achse und `pZ` auf der Z-Achse. Position und Blickpunkt werden verschoben.

Auftrag `void vor(double pWeite)`

Lässt die Kamera in Richtung des Blickpunktes um `pWeite` vorfahren.

Auftrag `pubic void schwenkeHorizontal(double pWinkel)`

Dreht die Kamera in der Art eines Horizontalschwenks (links/rechts) um den Winkel `pWinkel`.

Anfrage `void schwenkeVertikal(double pWinkel)`

Dreht die Kamera in der Art eines Vertikalschwenks (oben/unten) um den Winkel `pWinkel`.

Anfragen `double gibX()`

`double gibY()`

`double gibZ()`

Gibt die X-Koordinate, Y-Koordinate oder Z-Koordinate der Position der Kamera zurück.

Anfragen `double gibBlickpunktX()`

`double gibBlickpunktY()`

`double gibBlickpunktZ()`

Gibt die X-Koordinate, Y-Koordinate oder Z-Koordinate des Blickpunktes der Kamera zurück.

Auftrag `void setzeStereomodus(boolean pM)`

Schalten den Stereomodus (Rot-Cyan Anaglyphenbilder) der Kamera ein bzw. aus.

Auftrag `void setzeAugendistanz(double pAugendistanz)`

Setzt die Distanz zwischen den beiden Augen des Betrachters. Diese Methode hat nur im Stereomodus Auswirkungen.

Auftrag `void zeigeAchsen(boolean pAn)`
Blendet die Koordinatenachsen im Kamerabild ein.

Auftrag `void setzeFensterposition(int pX, int pY)`
Setzt die Position des Fensters auf dem Bildschirm.

Anfrage `int gibBreite()`
Liefert die Breite des Kamerafensters.

Auftrag `int gibHoehe()`
Liefert die Höhe des Kamerafensters.

Auftrag `void loesche()`
Löscht die Kamera und schließt das entsprechende Fenster.

Auftrag `void zeigeFenster(boolean pB)`
Gibt die Möglichkeit, das Kamerafenster ein- bzw. auszublenden.

4.2 Schwenkkamera und Entwicklerkamera

Neben der normalen Kamera vom Typ `GLKamera` existieren zwei weitere Unterklassen, die beide über eine automatische Maussteuerung verfügen; die Klasse `GLSchwenkkamera` und die Klasse `GLEntwicklerkamera`.

Durch Klicken und Ziehen mit der Maus können Kameraobjekte dieser Klassen in alle Richtungen um den aktuellen Blickpunkt geschwenkt werden, so dass Objekte in diesem Bereich leicht von allen Seiten untersucht werden können.

Objekte vom Typ `GLEntwicklerkamera` verfügen darüber hinaus über eine Reihe von Tasten zur Steuerung von Spezialfunktionen:

A	Blendet die Koordinatenachsen ein bzw. aus.
G	Schaltet den Gitterdarstellungsmodus der Kamera ein bzw. aus.
S	Setzt die Kamera zurück auf ihre Ursprungsposition bei `(0, 0, 500)`.
3	Aktiviert bzw. deaktiviert den 3D-Stereomodus.
O	Verkleinert den Augenabstand im Stereomodus.
P	Vergrößert den Augenabstand im Stereomodus.

5 Abfrage von Tastatur und Maus

Um Eingaben von der Tastatur und der Maus abzufragen, können die Klassen `GLMaus` und `GLTastatur` verwendet werden.

Die Klasse GLMaus (Oberklasse `Object`)

Konstr. `GLMaus()`
Erstellt ein Mausobjekt.

Anfrage `boolean gedruecktLinks()`
Liefert `true`, sofern die linke Maustaste gerade gedrückt ist.

Anfrage `boolean gedruecktRechts()`
Liefert `true`, sofern die rechte Maustaste gerade gedrückt ist.

Anfrage `boolean wirdBewegt()`
Liefert `true`, sofern die Maus gerade bewegt wird.

Anfrage `boolean linksklick()`
Liefert `true`, sofern ein Linksklick erfolgt ist.

Anfrage `boolean rechtsklick()`
Liefert `true`, sofern ein Rechtsklick erfolgt ist.

Anfrage `boolean doppelklick()`
Liefert `true`, sofern ein Doppelklick erfolgt ist.

Anfrage `double gibX()`
`double gibY()`
Liefert die X- bzw. Y-Koordinate der Maus auf dem Kamerafenster. Der Nullpunkt ist in der linken oberen Ecke.

Die Klasse GLTastatur (Oberklasse `Object`)

Konstr. `GLTastatur()`
Erstellt ein Tastaturobjekt.

Anfrage `boolean istGedrueckt()`
Liefert `true`, sofern irgendeine Taste gerade gedrückt ist.

Anfrage `boolean istGedrueckt(char pT)`
Liefert `true`, sofern die dem Zeichen `pT` entsprechende Taste gerade gedrückt ist.

Anfragen `boolean alt()`
`boolean strg()`
`boolean shift()`
`boolean tab()`
`boolean links()`
`boolean rechts()`
`boolean oben()`
`boolean unten()`
`boolean esc()`
`boolean backspace()`
`boolean enter()`
Methoden, die `true` liefern, sofern die entsprechende Sondertaste gerade gedrückt ist.

Anfrage `boolean wurdeGedrueckt()`
Liefert `true`, sofern im Tastaturpuffer mindestens ein Zeichen vorliegt.

Anfrage `char gibZeichen()`
Liefert das erste Zeichen des Tastaturpuffers und löscht es daraus. Ist der Puffer leer, wird `char(0)` geliefert.

Auftrag `void loeschePuffer()`
Löscht den Tastaturpuffer.

6 Hilfsklassen

Neben den oben aufgeführten enthält das GLOOP-Paket noch einige Hilfsklassen, die im Folgenden kurz erläutert werden. Bei der Klasse `Sys` handelt es sich um eine statische Klasse, die einen beschränkten Einfluss auf das System selbst erlaubt. Die wichtigste Methode dieser Klasse ist `warte()`, da mit ihr die obligatorische Animationsschleife in ihrem Durchlauf gebremst werden kann, so dass eine Animation nicht zu schnell abläuft.

Die Klasse Sys (statische Klasse)

Da Klasse statisch ist, verfügt sie über keinen Konstruktor.

Auftrag `void warte()`
Lässt das System eine Millisekunde warten.

Auftrag `void warte(int pMS)`
Lässt das System für `pMS` Millisekunden warten.

Beispiel:
```
Sys.warte(5);
```

Anfrage `GLObjekt gibObjekt(double pX, double pY)`
Gibt das Objekt zurück, welches an der Stelle `(pX, pY)` im Kamerafenster zu sehen ist. Der Nullpunkt ist die linke obere Ecke des Fensters. Mit Hilfe dieses Befehls wird die Objektselektion in GLOOP realisiert.

Beispiel:
```
GLObjekt lA = Sys.gibObjekt(100,100);
lA.setzeFarbe(1,0,0);
```
Das Objekt an der Fensterposition `(100, 100)` wird rot gefärbt. Ist dort kein Objekt, kommt es zum Fehler, da `lA = null`.

Auftrag `void beenden()`
Beendet das Programm augenblicklich.

Auftrag `void erstelleAusgabe(String pM)`
Gibt den String `pM` auf einer am unteren Bildrand eingeblendeten Konsole aus.

Auftrag `void erstelleAusgabe(String pT, String pM)`
Gibt den String `pM` auf einer am unteren Bildrand eingeblendeten Konsole aus. Die Konsole wird mit der Zeichenkette in `pT` übertitelt.

Anfrage `String erwarteEingabe()`
Blendet am unteren Bildschirmrand eine Konsole ein und wartet auf die Eingabe eines String.

Anfrage `String erwarteEingabe(String pT)`
Blendet am unteren Bildschirmrand eine Konsole ein und wartet auf die Eingabe eines String. Die Konsole wird mit der Zeichenkette in `pT` übertitelt.

Die Klasse GLVektor (Oberklasse `Object`)

Konstr. `GLVektor()`
`GLVektor (double pX, double pY, double pZ)`
`GLVektor (double pX1, double pY1, double pZ1, double pX2, double pY2, double pZ2)`

Ein dreidimensionaler Vektor wird erstellt. Entweder der Vektor `(0,0,0)`, der Vektor `(pX,pY,pZ)` oder der Vektor `(pX2-pX1, pY2-pY1, pZ2-pZ1)`. Auf die Komponenten des Vektor kann direkt über die öffentlichen Variablen `x`, `y` und `z` zugegriffen werden.

Beispiel:

```
GLVektor lV = new GLVektor(1,0,1);
lV.x = lV.x * 5;
```

Ergebnisvektor ist `(5,0,1)`.

Auftrag `void normiere()`

Normiert den Vektor.

Anfrage `double gibBetrag()`

Liefert den Betrag des Vektors.

Auftrag `void addiere(GLVektor pV)`

Addiert `pV` auf den Vektor auf.

Auftrag `void subtrahiere(GLVektor pV)`

Subtrahiert `pV` von dem Vektor.

Auftrag `void multipliziere(double pS)`

Multipliziert den Skalar `pS` mit dem Vektor.

Anfrage `GLVektor gibKreuzprodukt(GlVektor pV)`

Errechnet das Kreuzprodukt aus dem Vektor und `pV` und liefert es als neues Objekt vom Typ `GLVektor` zurück.

Anfrage `double gibSkalarprodukt(GLVektor pV)`

Liefert das Skalarprodukt des Vektors und `pV`.

Auftrag `void skaliereAuf(double pB)`

Setzt den Betrag des Vektors auf den Wert von `pB`.

Auftrag `void drehe(double pWX, double pWY, double pWZ)`

Die Spitze des Vektors wird gedreht (vgl. die Methode `drehe` der Klasse `GLObjekt`).

Auftrag `void rotiere(double pWinkel, double pRX, double pRY, double pRZ)`

Rotiert die Spitze des Vektors um die durch `(pRX, pRY, pRZ)` gegebene Achse (vgl. Methode `rotiere` der Klasse `GLObjekt`).

Anfrage `double gibX()`
`double gibY()`
`double gibZ()`

Liefert die entsprechende Komponente des Vektors.

7 Vektorvarianten von Konstruktoren und Methoden

Für die Entwicklung weiterführender Projekt mit GLOOP steht für jede Methode, die in ihren Parametern einen Punkt oder eine Richtung übergeben bekommt, eine Variante zur Verfügung, die ein Objekt der Klasse `GLVektor` als Übergabe akzeptiert.

Beispiel

```
1  //Variante 1: Erstellen, Rotieren und Verschieben einer Kugel.
2  GLKugel meineKugel1 = new GLKugel (0,0,0, 50);
3  meineKugel1.verschiebe(100,0,0);
4  meineKugel1.rotiere(45,0,0,1,0,0,0);
5
6  [...]
7
8  //Variante 2: Erstellen, Rotieren und Verschieben einer Kugel
9  //mit Vektorobjekten.
10 GLVektor lUrsprung = new GLVektor(0,0,0);
11 GLVektor lVerschiebung = new GLVektor(100,0,0);
12 GLVektor lAchsenrichtung = new GLVektor (0,0,1);
13
14 GLKugel meineKugel2 = new GLKugel (lUrsprung, 50);
15 meineKugel2.verschiebe(lVerschiebung);
16 meineKugel2.rotiere(45,lAchsenrichtung,lUrsprung);
```

Beide Quellcodevarianten erfüllen dieselbe Aufgabe. Es wird eine Kugel im Ursprung `(0,0,0)` erstellt, um `100` auf der X-Achse verschoben und anschließend um 45 Grad um die Z-Achse gedreht.

Des Weiteren verfügen Objekte vom Typ `GLObjekt` und Objekte vom Typ `GLKamera` über eine Methode, welche die Position des Objektes im Raum als Vektor vom Typ `GLVektor` zurückliefert:

Anfrage `GLVektor gibPosition()`
Liefert die Position des Objekts als Vektor (vgl. dazu die Methoden `gibX()`, `gibY()` und `gibZ()`).

Objekte vom Typ `GLKamera` können darüber hinaus auch Blickpunkt, Blickrichtung und Scheitelrichtung als Vektor liefern:

Anfrage `GLVektor gibBlickpunkt()`
Liefert den Blickpunkt der Kamera.

Anfrage `GLVektor gibScheitelrichtung()`
Liefert die Scheitelrichtung der Kamera als normierten Vektor.

Anfrage `GLVektor gibBlickrichtung()`
Liefert die Blickrichtung der Kamera als normierten Vektor.

8 Schlusswort

Die Bibliothek GLOOP befindet sich, ebenso wie dazugehörige Unterrichtsmaterialien, noch im Prozess der Entwicklung. Es kann daher keine Garantie für Fehlerfreiheit oder Vollständigkeit übernommen werden.

Kommentare, Anregungen und auch Fragen sind natürlich immer willkommen. Bitte wenden Sie sich an: `volker.quade@br.nrw.de`