Phantom Graph Demo Nachschlagewerk

Erzeugt von Doxygen 1.3.8

Sun Feb 5 22:12:11 2006

Inhaltsverzeichnis

Kapitel 1

Phantom Graph Demo Hierarchie-Verzeichnis

1.1 Phantom Graph Demo Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

Kapitel 2

Phantom Graph Demo Klassen-Verzeichnis

2.1 Phantom Graph Demo Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

Kapitel 3

Phantom Graph Demo Datei-Verzeichnis

3.1 Phantom Graph Demo Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 4

Phantom Graph Demo Seitenindex

4.1 Phantom Graph Demo Zusätzliche Informationen

Hier folgt eine Liste mit zusammengehörigen Themengebieten:

Kapitel 5

Phantom Graph Demo Klassen-Dokumentation

5.1 AppConfiguration Klassenreferenz

Klasse, die für projektübergreifende Aufgaben zuständig ist und static Member zur Verfügung stellt, Singelton.

`#include <AppConfiguration.h>`

Öffentliche Methoden

- void **setProjectLines** (int lines)

  setzt die Anzahl maximaler paralleler Aufgaben

- void **initTasks** ()

  hier werden alle Aufgaben initialisiert momentan hard codiert, als Erwiterung ist das einlesen aus XML vorgesehen

- void **setProjectDuration** (int days)

  setzt die Gesamtdauer des Projekts in Tagen

- int **getProjectDuration** ()

  Gibt die gesamt darzustellenden Dauer des Projekts zurück.

- **IBusinessAdapter** ∗ **getRootTask** ()

  liefert einen Pointer auf die Anfangsaufgabe

- int **getProjectLines** ()

  liefert die max. Anzahl paralleler Aufgaben

- void **setDebugState** (bool state)

  setzt den Debugstatus zur Anzeige von Infos während der Programmausführung

- bool **getDebugState** ()

liefert den aktuellen Debugstatus

- **AppConfiguration** ()

Öffentliche Attribute

- list< **BusinessTask** * > **m_BusinessTasks**

  Liste aller Aufgaben.

Private Attribute

- bool **m_debug**

  hält den aktuellen Debugstatus default=false, wird im Konstruktor gesetzt gesetzt

- int **m_ProjectDuration**

  Dauer des Projektzeitraums.

- int **m_ProjectLines**

  Anzahl maximaler paralleler Aufgaben.

- **IBusinessAdapter** * **m_rootTask**

5.1.1 Ausführliche Beschreibung

Klasse, die für projektübergreifende Aufgaben zuständig ist und static Member zur Verfügung stellt, Singelton.

Autor:
Carsten Arnold

5.1.2 Beschreibung der Konstruktoren und Destruktoren

5.1.2.1 AppConfiguration::AppConfiguration ()

Ausgabe von Debuginfos ein-/ausschalten

Debugausgabe Projektdauer

5.1.3 Dokumentation der Elementfunktionen

5.1.3.1 void AppConfiguration::setDebugState (bool *state*)

setzt den Debugstatus zur Anzeige von Infos während der Programmausführung

Parameter:
state true/false

5.1.3.2 void AppConfiguration::setProjectDuration (int *days*)

setzt die Gesamtdauer des Projekts in Tagen

Parameter:
days Gesamtdauer des Projekts in Tagen

5.1.3.3 void AppConfiguration::setProjectLines (int *lines*)

setzt die Anzahl maximaler paralleler Aufgaben

Parameter:
lines Anzahl maximaler paralleler Aufgaben

5.1.4 Dokumentation der Datenelemente

5.1.4.1 IBusinessAdapter* AppConfiguration::m_rootTask [private]

@ brief Pointer auf die fiktive Startaufgabe, die nur Startpunkt und selbst keine wirkliche Aufgabe ist Aufgabe

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- businesslogic/**AppConfiguration.h**
- businesslogic/**AppConfiguration.cpp**

5.2 BusinessTask Klassenreferenz

diese KLasse verwaltet alle Aufgaben, die im Graph dargestellt werden sollen.

`#include <BusinessTask.h>`

Klassendiagramm für BusinessTask::

Öffentliche Methoden

- virtual void **moveToLaterPosition** (int new_begin)
  bewegt eine Aufgabe zu einem späteren Startpunkt
- virtual void **moveToEarlierPosition** (int end)
  bewegt eine Aufgabe zu einem früheren Startpunkt
- void **calcForceMedium0** ()
  berechnet das Datum nach Links zum Ende der nächsten vorhergehenden Aufgabe
- void **calcForceMedium1** ()
  berechnet das Datum nach Rechts zum Begin der nächsten folgenden Aufgabe
- virtual int **calcForceInc0** ()
  berechnet das Datum über das nicht nach Links verschoben werden kann
- virtual int **calcForceInc1** ()
  berechnet das Datum über das nicht nach Rechts verschoben werden kann
- void **calcRanges** ()
  Aufruf zum berechnen der Kraftpositionen der Aufgabe.
- void **moveAllToFront** ()
  schiebt alle Aufgaben so weit wie möglich an den Anfang des Projekts
- void **moveFollowingToFront** (int earliest)
  schiebt nachfolgende Aufgaben so weit wie möglich an den Anfang des Projekts
- void **movePreviousToFront** ()
  schiebt vorhergehende Aufgaben so weit wie möglich an den Anfang des Projekts
- int **getEnd** ()
  Liefert das Ende der Aufgabe.
- int **getBegin** ()
  Liefert den Anfang der Aufgabe.
- bool **setBegin** (float begin)
  setzt den Anfang einer Aufgabe

- void **addTaskPrevious** (**BusinessTask** *follows)

  fügt eine Aufgabe zu der Liste der Vorgänger hinzu

- void **addTaskFollowing** (**BusinessTask** *followed_by)

  fügt eine Aufgabe zu der Liste der Nachfolger hinzu

- string **getName** ()

  liefert den Namen der Aufgabe

- void **printInfo** ()

  gibt einige Infos der Aufgabe auf stdout aus

- void **setLine** (int line)

  setzt die Eben der Aufgabe, erhöt sich, wenn die Aufgabe parallel zu einer anderen Aufgabe durchgeführt werden kann

- int **getLine** ()

  liefert die Eben der Aufgabe

- int **getWidth** ()
- virtual **force getForce** (float x, float y)

  Liefert die Kraft, die notwendig ist, um eine Aufgabe zu verschieben.

- **BusinessTask** (string taskname, int day_duration, int day_final, bool **isMilestone**)

  überschreibt den StandardKonstruktor mit der Übergabe folgender Parameter

- virtual list< **IBusinessAdapter** * > & **getNextTasks** ()

  Liefert die Nachfolger.

- virtual list< **IBusinessAdapter** * > & **getPreviousTasks** ()

  Liefert die Vorgänger.

Öffentliche Attribute

- int **m_ForceRangeMedium0**

  Datum vom Ende der nächsten vorhergehenden Aufgabe.

- int **m_ForceRangeMedium1**

  Datum vom Anfang der nächsten nachfolgenden Aufgabe.

- int **m_ForceRangeIncredible0**

  Datum über das die Aufgabe nicht weiter nach Links verschoben werden kann.

- int **m_ForceRangeIncredible1**

  Datum über das die Aufgabe nicht weiter nach Rechts verschoben werden kann.

Geschützte Attribute

- float **m_Movement**
  akkumuliert die übergebenen Differenzen von x-Axis Bewegungen der View

Private Methoden

- float **runden** (float value, int nachkommastellen)
  rundet floats auf beliebige nachkommastellen ab
- int **calcBegin** (int end, int duration)
  berechnet den Anfangspunkt anhand von Endpunkt und Dauer
- int **calcEnd** (int begin, int duration)
  berechnet den Endpunkt anhand von Anfangspunkt und Dauer

Private Attribute

- **force m_Force**
  Rückgabewerte für **getForce()**(S. 17)*, enum deklariert in* **IBusinessAdapter**(S. 61).
- int **m_Line**
  Darstellungsebene.
- float **width**
  Dauer der Aufgabe.
- bool **isMilestone**
  ist die Aufgabe Milestone
- int **m_Begin**
  Anfangsdatum der Aufgabe.
- int **m_End**
  Enddatum der Aufgabe.
- int **m_Final**
  Deadline der Aufgabe.
- int **m_Width**
  Dauer der Aufgabe.
- string **m_Name**
  Name der Aufgabe.
- list< **IBusinessAdapter** * > **m_TasksFollowing**
  Liste aller direkter nachfolgenden Aufgaben.

- list< **IBusinessAdapter** * > **m_TasksPrevious**

  Liste aller direkter vorhergehender Aufgaben.

5.2.1 Ausführliche Beschreibung

diese KLasse verwaltet alle Aufgaben, die im Graph dargestellt werden sollen.

5.2.2 Beschreibung der Konstruktoren und Destruktoren

5.2.2.1 BusinessTask::BusinessTask (string *taskname*, int *day_duration*, int *day_final*, bool *isMilestone*)

überschreibt den StandardKonstruktor mit der Übergabe folgender Parameter

Parameter:

taskname Name der Aufgabe

day_duration Dauer der Aufgabe

day_final Datum an dem die Aufgabe fertig gestellt sein muss

isMilestone ist die Aufgabe ein Milestone

5.2.3 Dokumentation der Elementfunktionen

5.2.3.1 int BusinessTask::calcBegin (int *end*, int *duration*) [private]

berechnet den Anfangspunkt anhand von Endpunkt und Dauer

Parameter:

end Endpunkt

duration Dauer

Rückgabe:

Dauer der Aufgabe

5.2.3.2 int BusinessTask::calcEnd (int *begin*, int *duration*) [private]

berechnet den Endpunkt anhand von Anfangspunkt und Dauer

Parameter:

begin Anfangspunkt

duration Dauer

Rückgabe:

Ende der Aufgabe

5.2.3.3 int BusinessTask::calcForceInc0 () [virtual]

berechnet das Datum über das nicht nach Links verschoben werden kann

Rückgabe:
gibt das Datum zurück

Implementiert **IBusinessAdapter** (S. 61).

5.2.3.4 int BusinessTask::calcForceInc1 () [virtual]

berechnet das Datum über das nicht nach Rechts verschoben werden kann

Rückgabe:
gibt das Datum zurück

Implementiert **IBusinessAdapter** (S. 61).

5.2.3.5 void BusinessTask::calcForceMedium0 ()

berechnet das Datum nach Links zum Ende der nächsten vorhergehenden Aufgabe

Rückgabe:
gibt das Datum zurück

5.2.3.6 void BusinessTask::calcForceMedium1 ()

berechnet das Datum nach Rechts zum Begin der nächsten folgenden Aufgabe

Rückgabe:
gibt das Datum zurück

5.2.3.7 int BusinessTask::getBegin () [virtual]

Liefert den Anfang der Aufgabe.

Rückgabe:
Anfang der Aufgabe

Implementiert **IBusinessAdapter** (S. 62).

5.2.3.8 int BusinessTask::getEnd ()

Liefert das Ende der Aufgabe.

Rückgabe:
Ende der Aufgabe

5.2.3.9 force BusinessTask::getForce (float x, float y) [virtual]

Liefert die Kraft, die notwendig ist, um eine Aufgabe zu verschieben.

Parameter:
- x aktueller x-Wert des Knoten in Business Einheiten
- y aktueller y-Wert des Knoten in Business Einheiten

Rückgabe:
m_Force (siehe oben)

Implementiert **IBusinessAdapter** (S. 62).

5.2.3.10 int BusinessTask::getLine () [virtual]

liefert die Eben der Aufgabe

Rückgabe:
Ebene der Aufgabe

Implementiert **IBusinessAdapter** (S. 62).

5.2.3.11 string BusinessTask::getName () [virtual]

liefert den Namen der Aufgabe

Rückgabe:
Name der Aufgabe

Implementiert **IBusinessAdapter** (S. 63).

5.2.3.12 list< IBusinessAdapter * > & BusinessTask::getNextTasks () [virtual]

Liefert die Nachfolger.

Rückgabe:
Liste mit Nachfolgern

Implementiert **IBusinessAdapter** (S. 63).

5.2.3.13 list< IBusinessAdapter * > & BusinessTask::getPreviousTasks () [virtual]

Liefert die Vorgänger.

Rückgabe:
Liste mit Vorgängern

Implementiert **IBusinessAdapter** (S. 63).

5.2.3.14 int BusinessTask::getWidth () [virtual]

@ brief liefert die Dauer der Aufgabe

Rückgabe:
Dauer der Aufgabe

Implementiert **IBusinessAdapter** (S. 63).

5.2.3.15 void BusinessTask::moveToEarlierPosition (int *end*) [virtual]

bewegt eine Aufgabe zu einem früheren Startpunkt

Parameter:
end erhält denneuen Endtermin von Nachfolgender Aufgabe

Implementiert **IBusinessAdapter** (S. 64).

5.2.3.16 void BusinessTask::moveToLaterPosition (int *new_begin*) [virtual]

bewegt eine Aufgabe zu einem späteren Startpunkt

Parameter:
new_begin erhält seinen neuen Anfang

Implementiert **IBusinessAdapter** (S. 64).

5.2.3.17 float BusinessTask::runden (float *value*, int *nachkommastellen*) [private]

rundet floats auf beliebige nachkommastellen ab

Parameter:
value float, das gerundet werden soll
nachkommastellen - anzugeben in 1 für keine, 10 für 1, 100 für 2 ... Nachkommastellen

5.2.3.18 bool BusinessTask::setBegin (float *begin*) [virtual]

setzt den Anfang einer Aufgabe

Parameter:
begin Anhand des Übergabewertes entscheidet die Aufgabe den genauen Begin

Rückgabe:
bool true=Begin erfolgreich geändert, false=keine Änderung durchgeführt

Implementiert **IBusinessAdapter** (S. 64).

5.2.3.19 void BusinessTask::setLine (int *line*)

setzt die Eben der Aufgabe, erhöt sich, wenn die Aufgabe parallel zu einer anderen Aufgabe durchgeführt werden kann

Parameter:
line zu setzende Eben [0..[

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- businesslogic/**BusinessTask.h**
- businesslogic/**BusinessTask.cpp**

5.3 Camera Klassenreferenz

Eine Klasse, die unter Verwendung von OpenGL eine minimale Kameraführung erlaubt.

`#include <Camera.h>`

Öffentliche Methoden

- **Camera** (double fovY, int viewportWidth, int viewportHeight, **HapticDevice** *pHd=NULL)

  Konstruktor: Initialisiert die Kamera mit einem Sichtwinkel, der danach nicht mehr verändert werden kann, der Höhe und Breite des Fensters und einem **HapticDevice**(S. 52)*, das sich dem Sichtvolumen der Kamera anpassen soll.*

- virtual ~**Camera** ()

  Destruktor: Gibt die Resourcen des Objektes frei.

- void **translateView** (double x, double y)

  Verschiebt die Kamera auf der x-y-Ebene um den Vektor (x, y).

- void **recalculateView** (int viewportWidth, int viewportHeight)

  Berechnet das Sichtvolumen der Kamera anhand der gegebenen Fensterbreite und -höhe neu.

- double **getRatio** ()

  Gibt das Verhältnis zwischen Fensterbreite und -höhe zurück.

Geschützte Attribute

- double **m_Ratio**

  Verhältnis der Fensterbreite zur Fensterhöhe.

- double **m_NearDistance**

  Abstand zur Kamera, ab dem man Objekte sehen kann.

- double **m_FovY**

  Sehwinkel in y-Richtung.

- double **m_FarDistance**

  Abstand zur Kamera, bis zu dem man Objekte sehen kann.

- double **m_LastX**

  Letzte x-Koordinate der Kamera.

- double **m_LastY**

  Letzte y-Koordinate der Kamera.

- **HapticDevice** * **m_pHapticDevice**

  HapticDevice(S. 52)*, das sich dem Sichtvolumen der Kamera anpassen soll. Wird nicht von der Camera freigegeben!*

5.3.1 Ausführliche Beschreibung

Eine Klasse, die unter Verwendung von OpenGL eine minimale Kameraführung erlaubt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Die Sichtrichtung der Kamera ist immer parallel zur z-Achse (Sicht in Richtung der negativen z-Achse), oben ist immer in Richtung der positiven y-Achse. Codebasis für die Klasse ist das von SensAble zu Verfügung gestellte Constraints-Beispiel.

5.3.2 Beschreibung der Konstruktoren und Destruktoren

5.3.2.1 Camera::Camera (double *fovY*, int *viewportWidth*, int *viewportHeight*, HapticDevice * *pHd* = NULL)

Konstruktor: Initialisiert die Kamera mit einem Sichtwinkel, der danach nicht mehr verändert werden kann, der Höhe und Breite des Fensters und einem **HapticDevice**(S. 52), das sich dem Sichtvolumen der Kamera anpassen soll.

Parameter:
fovY Sehwinkel in y-Richtung.

viewportWidth Fensterbreite.

viewportHeight Fensterhöhe.

pHd Pointer auf das **HapticDevice**(S. 52), das sich dem Sichtvolumen der Kamera anpassen soll. Wird nicht von der Camera freigegeben!

5.3.3 Dokumentation der Elementfunktionen

5.3.3.1 double Camera::getRatio ()

Gibt das Verhältnis zwischen Fensterbreite und -höhe zurück.

Rückgabe:
Verhältnis zwischen Fensterbreite und -höhe.

5.3.3.2 void Camera::recalculateView (int *viewportWidth*, int *viewportHeight*)

Berechnet das Sichtvolumen der Kamera anhand der gegebenen Fensterbreite und -höhe neu.

Parameter:
viewportWidth Fensterbreite.

viewportHeight Fensterhöhe.

5.3.3.3 void Camera::translateView (double *x*, double *y*)

Verschiebt die Kamera auf der x-y-Ebene um den Vektor (x, y).

Parameter:

x x-Komponente des Translationsvektors.

y y-Komponente des Translationsvektors.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Camera.h**
- hapticgraphclasses/**Camera.cpp**

5.4 ConstantForceEffect Klassenreferenz

Klasse zur Ausgabe einer konstanten Kraft in Richtung des Direction- Vektors (**setDirection()**(S. 24)) auf dem Phantom Device.

`#include <ConstantForceEffect.h>`

Klassendiagramm für ConstantForceEffect::

Öffentliche Methoden

- **ConstantForceEffect** (const double direction[3], double magnitude)
  Konstruktor: Initialisiert das Objekt mit den angegebenen Werten.
- **ConstantForceEffect** ()
  Standardkonstruktor: Initialisiert das Objekt mit Defaultwerten.
- virtual ~**ConstantForceEffect** ()
  Destruktor: Gibt die Resourcen des Objektes frei.
- virtual void **setMagnitude** (double value)
  Setzt die Länge des Kraftvektors.
- virtual void **setDirection** (double x, double y, double z)
  Setzt die Richtung der Kraft.

Geschützte Methoden

- virtual void **renderProperties** ()
  Setzt die entsprechenden Eigenschaften beim Rendern des Effekts in HLAPI-Aufrufe um.

Geschützte Attribute

- hduVector3Dd **m_Direction**
  Richtung (Vektor), in der die Kraft wirken soll.
- double **m_Magnitude**
  "Größe" der Kraft.

5.4.1 Ausführliche Beschreibung

Klasse zur Ausgabe einer konstanten Kraft in Richtung des Direction- Vektors (**setDirection()**(S. 24)) auf dem Phantom Device.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.4.2 Beschreibung der Konstruktoren und Destruktoren

5.4.2.1 ConstantForceEffect::ConstantForceEffect (const double *direction*[3], double *magnitude*)

Konstruktor: Initialisiert das Objekt mit den angegebenen Werten.

Parameter:

direction Richtungsvektor der Kraft.

magnitude "Größe" der Kraft.

5.4.2.2 ConstantForceEffect::ConstantForceEffect ()

Standardkonstruktor: Initialisiert das Objekt mit Defaultwerten.

Defaultwerte:

- direction: 0.0, 0.0, 0.0
- magnitude: 0.0

5.4.3 Dokumentation der Elementfunktionen

5.4.3.1 void ConstantForceEffect::setDirection (double *x*, double *y*, double *z*) [virtual]

Setzt die Richtung der Kraft.

Parameter:

x x-Komponente des Richtungsvektors.

y y-Komponente des Richtungsvektors.

z z-Komponente des Richtungsvektors.

5.4.3.2 void ConstantForceEffect::setMagnitude (double *value*) [virtual]

Setzt die Länge des Kraftvektors.

Parameter:

value "Größe" der Kraft.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**ConstantForceEffect.h**
- hapticgraphclasses/**ConstantForceEffect.cpp**

5.5 DragNodeOnGridHandler Klassenreferenz

Eine Eventhandlerklasse die es ermöglicht, haptische Nodes mit dem Phantom in x-Richtung auf einem **Grid**(S. 46) zu bewegen.

`#include <DragNodeOnGridHandler.h>`

Klassendiagramm für DragNodeOnGridHandler::

Öffentliche Methoden

- **DragNodeOnGridHandler** (**Node** *pNode, **Grid** *pGrid)

  Konstruktor, initialisiert das Eventhandler-Objekt mit dem zugehörigen **Node**(S. 68) *und dem* **Grid**(S. 46)*, auf dem dieser bewegt werden soll.*

- virtual ~**DragNodeOnGridHandler** ()

  Destruktor: Gibt die Resourcen des Objektes frei.

- void **initAction** (HLcache *pCache)

  Nimmt die Proxy-Position beim Starten des Drag-Vorgangs auf.

- void **handleDrag** (HLcache *pCache)

  Veranlasst das haptische Objekt, sich mit dem Proxy zu bewegen.

- void **finishAction** ()

  Sorgt dafür, dass der **Node**(S. 68) *am Ende des Dragvorgangs auf den nächstgelegenen gültigen Gitterpunkt des* **Grid**(S. 46) *gesetzt wird.*

- virtual void **unregisterAction** (HLuint shapeID)

  Registriert die Aktion für eine Shape bei HLAPI.

- virtual void **registerAction** (HLuint shapeID)

  Meldet die Aktion für eine Shape bei HLAPI ab.

Geschützte, statische Methoden

- void HLCALLBACK **OnButtonDown** (HLenum event, HLuint shapeID, HLenum thread, HLcache *cache, void *pHandlerObject)

  (HLAPI-Callbackfunktion) Started das Draggen des Objekts.

- void HLCALLBACK **OnButtonUp** (HLenum event, HLuint shapeID, HLenum thread, HLcache *cache, void *pHandlerObject)

  (HLAPI-Callbackfunktion) Beendet das Draggen des Objekts.

- void HLCALLBACK **OnDrag** (HLenum event, HLuint shapeID, HLenum thread, HLcache *cache, void *pHandlerObject)

  (HLAPI-Callbackfunktion) Steuert das Draggen des Objekts.

Geschützte Attribute

- **Node * m_pDragObj**
  Der **Node**(S. 68)*, dem der Eventhandler zugeordnet ist.*
- **Grid * m_pGrid**
  Das **Grid**(S. 46)*, auf dem der* **Node**(S. 68) *verschoben werden soll.*
- hduVector3Dd **m_LastProxyPos**
  Position(S. 75) *des Proxy beim letzten Aufruf des Draghandlers. Dient zur Berechnung des Vektors um den das Objekt verschoben werden soll.*

5.5.1 Ausführliche Beschreibung

Eine Eventhandlerklasse die es ermöglicht, haptische Nodes mit dem Phantom in x-Richtung auf einem **Grid**(S. 46) zu bewegen.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Der Eventhandler reagiert auf die folgende Events:

- der vordere Phantom-Button wird gedrückt, wenn ein Objekt mit dem Phantom berührt wird
- das Phantom wird mit gedrücktem Button bewegt
- der vordere Phantom-Button wird losgelassen Wirkung: Der **Node**(S. 68) folgt den Bewegungen des Phantom in x-Richtung. Die Bewegung des Phantom in y- und z-Richtung wird von dem **Node**(S. 68) ignoriert. Beim Loslassen des Buttons rastet der **Node**(S. 68) an dem nächstgelegenen Gitterpunkt ein.

5.5.2 Beschreibung der Konstruktoren und Destruktoren

5.5.2.1 DragNodeOnGridHandler::DragNodeOnGridHandler (Node * *pNode*, Grid * *pGrid*)

Konstruktor, initialisiert das Eventhandler-Objekt mit dem zugehörigen **Node**(S. 68) und dem **Grid**(S. 46), auf dem dieser bewegt werden soll.

Parameter:
pNode Pointer auf den **Node**(S. 68) für den der Eventhandler zuständig sein soll.
pGrid Pointer auf das **Grid**(S. 46), auf dem der **Node**(S. 68) bewegt werden soll.

5.5.3 Dokumentation der Elementfunktionen

5.5.3.1 void DragNodeOnGridHandler::handleDrag (HLcache * *pCache*)

Veranlasst das haptische Objekt, sich mit dem Proxy zu bewegen.

Parameter:
pCache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

5.5.3.2 void DragNodeOnGridHandler::initAction (HLcache * *pCache*)

Nimmt die Proxy-Position beim Starten des Drag-Vorgangs auf.

Parameter:

pCache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

5.5.3.3 void HLCALLBACK DragNodeOnGridHandler::OnButtonDown (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Started das Draggen des Objekts.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_1BUTTONDOWN.

shapeID Die ShapeID des Objekts, das bewegt werden soll.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragNodeOnGridHandler-Objekt, das das Event verarbeiten soll.

5.5.3.4 void HLCALLBACK DragNodeOnGridHandler::OnButtonUp (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Beendet das Draggen des Objekts.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_1BUTTONUP.

shapeID Hier soll HL_OBJECT_ANY angegeben werden.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragNodeOnGridHandler-Objekt, das das Event verarbeiten soll.

5.5.3.5 void HLCALLBACK DragNodeOnGridHandler::OnDrag (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Steuert das Draggen des Objekts.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_MOTION

shapeID Hier soll HL_OBJECT_ANY angegeben werden.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragNodeOnGridHandler-Objekt, das das Event verarbeiten soll.

5.5.3.6 void DragNodeOnGridHandler::registerAction (HLuint *shapeID*) [virtual]

Meldet die Aktion für eine Shape bei HLAPI ab.

Parameter:

shapeID ID der Shape, für die die Aktion registriert wurde.

Implementiert **IHapticAction** (S. 65).

5.5.3.7 void DragNodeOnGridHandler::unregisterAction (HLuint *shapeID*) [virtual]

Registriert die Aktion für eine Shape bei HLAPI.

Parameter:

shapeID ID der Shape, für die die Aktion registriert werden soll.

Implementiert **IHapticAction** (S. 65).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**DragNodeOnGridHandler.h**
- hapticgraphclasses/**DragNodeOnGridHandler.cpp**

5.6 DragObjectHandler Klassenreferenz

Eine Eventhandlerklasse die es ermöglicht, haptische Objekte mit dem Phantom zu bewegen.

`#include <DragObjectHandler.h>`

Klassendiagramm für DragObjectHandler::

Öffentliche Methoden

- **DragObjectHandler** (**HapticObject** ∗pObj)
  Konstruktor, initialisiert das Eventhandler-Objekt mit dem zugehörigen haptischen Objekt.
- void **initAction** (HLcache ∗pCache)
  Nimmt die Proxy-Position beim Starten des Drag-Vorgangs auf.
- void **handleDrag** (HLcache ∗pCache)
  Veranlasst das haptische Objekt, sich mit dem Proxy zu bewegen.
- virtual void **registerAction** (HLuint shapeID)
  Registriert die Aktion für eine Shape bei HLAPI.
- virtual void **unregisterAction** (HLuint shapeID)
  Meldet die Aktion für eine Shape bei HLAPI ab.

Geschützte, statische Methoden

- void HLCALLBACK **OnButtonDown** (HLenum event, HLuint shapeID, HLenum thread, HLcache ∗cache, void ∗pHandlerObject)
  (HLAPI-Callbackfunktion) Started das Draggen des Objekts
- void HLCALLBACK **OnButtonUp** (HLenum event, HLuint shapeID, HLenum thread, HLcache ∗cache, void ∗unused)
  (HLAPI-Callbackfunktion) Beendet das Draggen des Objekts.
- void HLCALLBACK **OnDrag** (HLenum event, HLuint shapeID, HLenum thread, HLcache ∗cache, void ∗pHandlerObject)
  (HLAPI-Callbackfunktion) Steuert das Draggen des Objekts.

Geschützte Attribute

- **HapticObject** ∗ **m_pDragObj**
  Das Objekt, dem der Eventhandler zugeordnet ist. Wird NICHT vom EventHandler freigegeben!
- hduVector3Dd **m_LastProxyPos**
  Position(S. 75) *des Proxy beim letzten Aufruf des Draghandlers. Dient zur Berechnung des Vektors um den das Objekt verschoben werden soll.*

5.6.1 Ausführliche Beschreibung

Eine Eventhandlerklasse die es ermöglicht, haptische Objekte mit dem Phantom zu bewegen.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Der Eventhandler reagiert auf die folgende Events:

- der vordere Phantom-Button wird gedrückt, wenn ein Objekt mit dem Phantom berührt wird
- das Phantom wird mit gedrücktem Button bewegt
- der vordere Phantom-Button wird losgelassen Wirkung: Solange der Button gedrückt gehalten wird, folgt das registrierte Objekt der Bewegung des Phantom

Noch zu erledigen
Objekte lassen sich noch nicht nach hinten (in neg. z-Richtung) verschieben.

5.6.2 Beschreibung der Konstruktoren und Destruktoren

5.6.2.1 DragObjectHandler::DragObjectHandler (HapticObject * *pObj*)

Konstruktor, initialisiert das Eventhandler-Objekt mit dem zugehörigen haptischen Objekt.

Parameter:
pObj Pointer auf das haptische Objekt für das der Eventhandler zuständig sein soll. Wird NICHT vom EventHandler freigegeben!

5.6.3 Dokumentation der Elementfunktionen

5.6.3.1 void DragObjectHandler::handleDrag (HLcache * *pCache*)

Veranlasst das haptische Objekt, sich mit dem Proxy zu bewegen.

Parameter:
pCache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

5.6.3.2 void DragObjectHandler::initAction (HLcache * *pCache*)

Nimmt die Proxy-Position beim Starten des Drag-Vorgangs auf.

Parameter:
pCache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

5.6.3.3 void HLCALLBACK DragObjectHandler::OnButtonDown (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Started das Draggen des Objekts

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_1BUTTONDOWN.

shapeID Die ShapeID des Objekts, das bewegt werden soll.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragObjectHandler-Objekt, das das Event verarbeiten soll.

5.6.3.4 void HLCALLBACK DragObjectHandler::OnButtonUp (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *unused*) [static, protected]

(HLAPI-Callbackfunktion) Beendet das Draggen des Objekts.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_1BUTTONUP.

shapeID Hier soll HL_OBJECT_ANY angegeben werden.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

unused Wird von dieser Funktion nicht benötigt.

5.6.3.5 void HLCALLBACK DragObjectHandler::OnDrag (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Steuert das Draggen des Objekts.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_MOTION.

shapeID Hier soll HL_OBJECT_ANY angegeben werden.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragObjectHandler-Objekt, das das Event verarbeiten soll.

5.6.3.6 void DragObjectHandler::registerAction (HLuint *shapeID*) [virtual]

Registriert die Aktion für eine Shape bei HLAPI.

Parameter:
shapeID ID der Shape, für die die Aktion registriert werden soll.

Implementiert **IHapticAction** (S. 65).

5.6.3.7 void DragObjectHandler::unregisterAction (HLuint *shapeID*) [virtual]

Meldet die Aktion für eine Shape bei HLAPI ab.

Parameter:
shapeID ID der Shape, für die die Aktion registriert wurde.

Implementiert **IHapticAction** (S. 65).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**DragObjectHandler.h**
- hapticgraphclasses/**DragObjectHandler.cpp**

5.7 DragSceneHandler Klassenreferenz

Eine Eventhandlerklasse die es ermöglicht, die gesamte Szene (Kameraführung) mit dem Phantom zu bewegen.

`#include <DragSceneHandler.h>`

Klassendiagramm für DragSceneHandler::

Öffentliche Methoden

- **DragSceneHandler** (**GraphScene** *pScene)

  Konstruktor: Initialisiert das Eventhandler-Objekt mit der Szene, die bewegt werden soll.

- void **initAction** (HLcache *pCache)

  Nimmt die Proxy-Position beim Starten des Drag-Vorgangs auf.

- void **handleDrag** (HLcache *pCache)

  Veranlasst die Szene, sich mit dem Proxy zu bewegen.

- virtual void **registerAction** (HLuint shapeID)

  Registriert die Aktion für eine Shape bei HLAPI.

- virtual void **unregisterAction** (HLuint shapeID)

  Meldet die Aktion für eine Shape bei HLAPI ab.

Geschützte, statische Methoden

- void HLCALLBACK **OnButtonDown** (HLenum event, HLuint shapeID, HLenum thread, HLcache *cache, void *pHandlerObject)

  (HLAPI-Callbackfunktion) Started das Draggen der Szene.

- void HLCALLBACK **OnButtonUp** (HLenum event, HLuint shapeID, HLenum thread, HLcache *cache, void *unused)

  (HLAPI-Callbackfunktion) Beendet das Draggen der Szene.

- void HLCALLBACK **OnDrag** (HLenum event, HLuint shapeID, HLenum thread, HLcache *cache, void *pHandlerObject)

  (HLAPI-Callbackfunktion) Steuert das Draggen der Szene.

Geschützte Attribute

- **GraphScene** * **m_pDragScene**

  Die Szene, die der Eventhandler bewegen soll. Wird NICHT vom EventHandler freigegeben!

- hduVector3Dd **m_LastProxyPos**

Position(S. 75) *des Proxy beim letzten Aufruf des Draghandlers. Dient zur Berechnung des Vektors um den die Szene verschoben werden soll.*

5.7.1 Ausführliche Beschreibung

Eine Eventhandlerklasse die es ermöglicht, die gesamte Szene (Kameraführung) mit dem Phantom zu bewegen.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Der Eventhandler reagiert auf die folgende Events:

- der hintere Phantom-Button wird gedrückt, wenn das registrierte Objekt mit dem Phantom berührt wird
- das Phantom wird mit gedrücktem Button bewegt
- der vordere Phantom-Button wird losgelassen Wirkung: Solange der Button gedrückt gehalten wird, folgt die Szene der Bewegung des Phantom (nur die x-Richtung wird berücksichtigt) (Kameraführung in entgegengesetzter x-Richtung).

5.7.2 Beschreibung der Konstruktoren und Destruktoren

5.7.2.1 DragSceneHandler::DragSceneHandler (GraphScene * *pScene*)

Konstruktor: Initialisiert das Eventhandler-Objekt mit der Szene, die bewegt werden soll.

Parameter:
pScene Pointer auf die Szene für die der Eventhandler zuständig sein soll. Wird NICHT vom EventHandler freigegeben!

5.7.3 Dokumentation der Elementfunktionen

5.7.3.1 void DragSceneHandler::handleDrag (HLcache * *pCache*)

Veranlasst die Szene, sich mit dem Proxy zu bewegen.

Parameter:
pCache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

5.7.3.2 void DragSceneHandler::initAction (HLcache * *pCache*)

Nimmt die Proxy-Position beim Starten des Drag-Vorgangs auf.

Parameter:
pCache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

5.7.3.3 void HLCALLBACK DragSceneHandler::OnButtonDown (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Started das Draggen der Szene.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_2BUTTONDOWN.

shapeID Die ShapeID des Objekts, auf dessen Berührung hin die Szene bewegt werden soll.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragSceneHandler-Objekt, das das Event verarbeiten soll.

5.7.3.4 void HLCALLBACK DragSceneHandler::OnButtonUp (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *unused*) [static, protected]

(HLAPI-Callbackfunktion) Beendet das Draggen der Szene.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_2BUTTONUP.

shapeID Hier soll HL_OBJECT_ANY angegeben werden.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

unused Wird von dieser Funktion nicht benötigt.

5.7.3.5 void HLCALLBACK DragSceneHandler::OnDrag (HLenum *event*, HLuint *shapeID*, HLenum *thread*, HLcache * *cache*, void * *pHandlerObject*) [static, protected]

(HLAPI-Callbackfunktion) Steuert das Draggen der Szene.

Parameter:

event Gibt an, auf welches HLAPI-Event hin die Callback- Funktion aufgerufen werden soll, hier HL_EVENT_MOTION.

shapeID Hier soll HL_OBJECT_ANY angegeben werden.

thread Gibt an, in welchem HLAPI-Thread das Event behandelt werden soll, in diesem Fall HL_CLIENT_THREAD.

cache HLAPI-State Schnappschuss in dem Moment, in dem das Event feuert.

pHandlerObject Pointer auf das DragSceneHandler-Objekt, das das Event verarbeiten soll.

5.7.3.6 void DragSceneHandler::registerAction (HLuint *shapeID*) [virtual]

Registriert die Aktion für eine Shape bei HLAPI.

Parameter:
shapeID ID der Shape, für die die Aktion registriert werden soll.

Implementiert **IHapticAction** (S. 65).

5.7.3.7 void DragSceneHandler::unregisterAction (HLuint *shapeID*) [virtual]

Meldet die Aktion für eine Shape bei HLAPI ab.

Parameter:
shapeID ID der Shape, für die die Aktion registriert wurde.

Implementiert **IHapticAction** (S. 65).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**DragSceneHandler.h**
- hapticgraphclasses/**DragSceneHandler.cpp**

5.8 Edge Klassenreferenz

Haptisches Objekt, das eine Kante in einem Graphen darstellt.

`#include <Edge.h>`

Klassendiagramm für Edge::

Öffentliche Methoden

- **Edge** (**Position** start=**Position**(0.0, 0.0, 0.0), **Position** end=**Position**(0.0, 0.0, 0.0))

  Konstruktor: Initialisiert die Edge mit Werten für Anfangs- und Endpunkt.

- virtual ~**Edge** ()

  Destruktor: Gibt die Resourcen des Objektes frei.

- void **setStartPosition** (**Position** value)

  Setzt den Anfangspunkt der Edge.

- void **setEndPosition** (**Position** value)

  Setzt den Endpunkt der Edge.

- virtual void **renderShape** ()

  Legt die Geometrie aller Objekte dieser Klasse fest.

Geschützte Methoden

- void **releaseDisplayList** ()

  Gibt die Displayliste frei und weist ihr einen ungültigen Wert zu.

- virtual void **renderDefaultGraphicProperties** ()

  Legt die graphischen Eigenschaften für Edges fest.

Geschützte Attribute

- **Position m_StartPosition**

  Anfangspunkt der Kante.

- **Position m_EndPosition**

  Endpunkt der Kante.

- GLuint **m_DisplayList**

  ID der OpenGL-Displayliste, mit der die Edge gezeichnet wird.

5.8.1 Ausführliche Beschreibung

Haptisches Objekt, das eine Kante in einem Graphen darstellt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.8.2 Beschreibung der Konstruktoren und Destruktoren

5.8.2.1 Edge::Edge (Position *start* = Position(0.0, 0.0, 0.0), Position *end* = Position(0.0, 0.0, 0.0))

Konstruktor: Initialisiert die Edge mit Werten für Anfangs- und Endpunkt.

Parameter:
start **Position**(S. 75) des Anfangspunktes. Default: [0.0, 0.0, 0.0]
end **Position**(S. 75) des Endpunktes. Default: [0.0, 0.0, 0.0]

5.8.3 Dokumentation der Elementfunktionen

5.8.3.1 void Edge::setEndPosition (Position *value*)

Setzt den Endpunkt der Edge.

Parameter:
value Neue **Position**(S. 75) des Endpunktes.

5.8.3.2 void Edge::setStartPosition (Position *value*)

Setzt den Anfangspunkt der Edge.

Parameter:
value Neue **Position**(S. 75) des Anfangspunktes.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Edge.h**
- hapticgraphclasses/**Edge.cpp**

5.9 FrictionForceEffect Klassenreferenz

Klasse zur Ausgabe einer "ambienten" Reibungskraft auf dem Phantom Device.

`#include <FrictionForceEffect.h>`

Klassendiagramm für FrictionForceEffect::

Öffentliche Methoden

- **FrictionForceEffect** (double magnitude=0.0, double gain=0.0)

  Konstruktor: Initialisiert den Effekt mit den angegebenen Werten.

- virtual ~**FrictionForceEffect** ()

  Destruktor: Gibt die Resourcen des Objekts frei.

- void **setMagnitude** (double value)

  Setzt den Wert für die Magnitude der Kraft.

- virtual void **setGain** (double value)

  Setzt den Wert für die Zunahme der Kraft.

Geschützte Methoden

- virtual void **renderProperties** ()

  Setzt die entsprechenden Eigenschaften beim Rendern des Effekts in HLAPI-Aufrufe um.

Geschützte Attribute

- double **m_Magnitude**

  "Größe" der Kraft.

- double **m_Gain**

  Zunahme der Kraft.

5.9.1 Ausführliche Beschreibung

Klasse zur Ausgabe einer "ambienten" Reibungskraft auf dem Phantom Device.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.9.2 Beschreibung der Konstruktoren und Destruktoren

5.9.2.1 FrictionForceEffect::FrictionForceEffect (double *magnitude* = 0.0, double *gain* = 0.0)

Konstruktor: Initialisiert den Effekt mit den angegebenen Werten.

Parameter:

magnitude "Größe" der Kraft. Default: 0.0

gain Zunahme der Kraft. Default: 0.0

5.9.3 Dokumentation der Elementfunktionen

5.9.3.1 void FrictionForceEffect::setGain (double *value*) [virtual]

Setzt den Wert für die Zunahme der Kraft.

Parameter:

value Neuer Wert für die Zunahme der Kraft.

5.9.3.2 void FrictionForceEffect::setMagnitude (double *value*)

Setzt den Wert für die Magnitude der Kraft.

Parameter:

value Neuer Wert für die Magnitude der Kraft.

5.9.4 Dokumentation der Datenelemente

5.9.4.1 double FrictionForceEffect::m_Gain [protected]

Zunahme der Kraft.

Für weiterführende Erkärungen s. "API Reference" des OpenHaptics Toolkit.

5.9.4.2 double FrictionForceEffect::m_Magnitude [protected]

"Größe" der Kraft.

Für weiterführende Erkärungen s. "API Reference" des OpenHaptics Toolkit.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**FrictionForceEffect.h**
- hapticgraphclasses/**FrictionForceEffect.cpp**

5.10 GlutString Klassenreferenz

Einfache Klasse um mit glut einen Text auszugeben.

`#include <Utilities.h>`

Öffentliche, statische Methoden

- void **write** (char *string, **Position** pos)
  Gibt den String string an der **Position**(S. 75) *pos aus.*

5.10.1 Ausführliche Beschreibung

Einfache Klasse um mit glut einen Text auszugeben.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Noch zu erledigen
Textausgabe funktioniert noch nicht.

5.10.2 Dokumentation der Elementfunktionen

5.10.2.1 void GlutString::write (char * *string*, Position *pos*) [static]

Gibt den String string an der **Position**(S. 75) pos aus.

Parameter:
string Zeiger auf einen C-String der ausgegeben werden soll.
pos **Position**(S. 75), an der der Text ausgegeben werden soll.

Noch zu erledigen
Funktioniert noch nicht.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Utilities.h**
- hapticgraphclasses/**Utilities.cpp**

5.11 GraphScene Klassenreferenz

Klasse, die alle haptischen Objekte der Scene verwaltet.

`#include <GraphScene.h>`

Öffentliche Methoden

- **GraphScene** (**UnitConversionInfo** &unitInfo)

  Konstruktor: Initialisiert das Objekt.

- virtual ~**GraphScene** ()

  Destruktor: Gibt die Resourcen des Objektes frei.

- void **renderScene** (bool bHapticsEnabled)

  Rendert die Haptik und Graphik der Szene.

- virtual void **initScene** (int viewportWidth, int viewportHeight, **HapticDevice** *pHd, int gridColumns, int gridRows, **IBusinessAdapter** *rootNode)

  Initialisiert die graphische/haptische Szene.

- void **addObject** (**HapticObject** *obj)

  Fügt der Szene ein neues Objekt hinzu.

- **Camera** * **getView** ()

  Gibt das Camera-Objekt zurück, mit dem die Szene betrachtet wird.

Öffentliche, statische Methoden

- float **getGraphPlaneZ** ()

  Gibt die z-Koordinate der Ebene parallel zur x-y-Ebene auf der der Graph dargestellt wird zurück.

Geschützte Methoden

- void **renderSceneHaptics** (bool bHapticsEnabled)

  Fordert alle Objekte auf, ihre haptische Beschaffenheit zu rendern.

- void **renderSceneGraphics** ()

  Fordert alle Objekte auf, ihre graphische Beschaffenheit zu rendern.

- **Node** * **createObjects** (**IBusinessAdapter** *businessObj, **Grid** *pGrid)

  Erzeugt aus einem Graphen, der durch IBusinessAdapter-Objekte beschrieben wird rekursiv die zugehörigen Darstellungsobjekte und fügt diese der Liste von Szenenelementen hinzu.

Geschützte Attribute

- vector< **HapticObject** * > **m_SceneElements**

  Liste aller haptischen Objekte der Scene. Alle Elemente der Szene werden auch von ihr freigegeben.

- **Camera** * **m_pCamera**

  Camera-Objekt, von dem aus die Szene beobachtet wird. Wird von der GraphScene erzeugt und freigegeben.

- **UnitConversionInfo** & **m_rUnitInfo**

  Referenz auf das Einheitenobjekt auf dessen Basis die Szene dargestellt werden soll. Wird NICHT von der GraphScene freigegeben!

- **IHapticAction** * **m_pDragSceneHandler**

  Eventhandlerobjekt, dass für das Bewegen der Szene mit dem Phantom zuständig ist. Wird von der GraphScene erzeugt und freigegeben.

5.11.1 Ausführliche Beschreibung

Klasse, die alle haptischen Objekte der Scene verwaltet.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Sie ist sowohl für die Erzeugung der Objekte verantwortlich, als auch dafür, dass sie sich zur richtigen Zeit rendern.

5.11.2 Beschreibung der Konstruktoren und Destruktoren

5.11.2.1 GraphScene::GraphScene (UnitConversionInfo & *unitInfo*)

Konstruktor: Initialisiert das Objekt.

Parameter:
unitInfo Referenz auf das Einheitenobjekt auf dessen Basis die Szene dargestellt werden soll. Wird NICHT von der GraphScene freigegeben!

5.11.3 Dokumentation der Elementfunktionen

5.11.3.1 void GraphScene::addObject (HapticObject * *obj*)

Fügt der Szene ein neues Objekt hinzu.

Parameter:
obj Pointer auf das Objekt, das hinzugefügt werden soll.

5.11.3.2 Node * GraphScene::createObjects (IBusinessAdapter * *businessObj*, Grid * *pGrid*) [protected]

Erzeugt aus einem Graphen, der durch IBusinessAdapter-Objekte beschrieben wird rekursiv die zugehörigen Darstellungsobjekte und fügt diese der Liste von Szenenelementen hinzu.

Parameter:

businessObj Knoten, für den ein Darstellungsobjekt erzeugt werden soll. Auch für dessen Nachfolger wird createObjects aufgerufen. Die Knoten werden mit Kanten verbunden.

pGrid Nötiger Parameter um die Darstellungsobjekte mit dem **Grid**(S. 46) zu verknüpfen.

Rückgabe:

Das Darstellungsobjekt, das das businessObj repräsentiert.

Noch zu erledigen

Bisher kann mit dieser Methode nur ein Baum erzeugt werden, nicht jeder beliebige Graph.

5.11.3.3 float GraphScene::getGraphPlaneZ () [static]

Gibt die z-Koordinate der Ebene parallel zur x-y-Ebene auf der der Graph dargestellt wird zurück.

Rückgabe:

z-Koordinate der Ebene parallel zur x-y-Ebene auf der der Graph dargestellt wird.

5.11.3.4 Camera * GraphScene::getView ()

Gibt das Camera-Objekt zurück, mit dem die Szene betrachtet wird.

Rückgabe:

Zeiger auf das Camera-Objekt, mit dem die Szene betrachtet wird.

5.11.3.5 void GraphScene::initScene (int *viewportWidth*, int *viewportHeight*, HapticDevice * *pHd*, int *gridColumns*, int *gridRows*, IBusinessAdapter * *rootNode*) [virtual]

Initialisiert die graphische/haptische Szene.

Parameter:

viewportWidth Fensterbreite.

viewportHeight Fensterhöhe.

pHd Pointer auf das **HapticDevice**(S. 52), das sich dem Sichtvolumen der Kamera anpassen soll.

gridColumns Anzahl der Spalten, die das **Grid**(S. 46) haben soll, auf dem der Graph dargestellt wird.

gridRows Anzahl der Zeilen, die das **Grid**(S. 46) haben soll, auf dem der Graph dargestellt wird.

rootNode Wurzelknoten eines Graphen, der durch IBusinessAdapter-Objekte beschrieben wird.

5.11.3.6 void GraphScene::renderScene (bool *bHapticsEnabled*)

Rendert die Haptik und Graphik der Szene.

Parameter:
bHapticsEnabled Gibt an, ob die Haptik gerendert werden kann.

5.11.3.7 void GraphScene::renderSceneHaptics (bool *bHapticsEnabled*) [protected]

Fordert alle Objekte auf, ihre haptische Beschaffenheit zu rendern.

Parameter:
bHapticsEnabled Gibt an, ob die Haptik gerendert werden kann.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**GraphScene.h**
- hapticgraphclasses/**GraphScene.cpp**

5.12 Grid Klassenreferenz

Klasse, die Gitterraster darstellt, auf dem die Elemente eines Graphen Angeordnet werden können. Die **Position**(S. 75) des Grid wird durch die linke untere Ecke festgelegt.

`#include <Grid.h>`

Klassendiagramm für Grid::

Öffentliche Methoden

- **Grid** (**UnitConversionInfo** &unitInfo, int cols=1, int rows=1)

  Konstruktor: Initialisiert das Grid mit einer bestimmten Anzahl an Zeilen und Spalten.

- virtual ~**Grid** ()

  Destruktor: Gibt die Resourcen des Objektes frei.

- bool **isGridPoint** (**Position** pos)

  Stellt fest, ob eine gegebene **Position**(S. 75) *mit einem gültigen Gitterpunkt überein stimmt (Toleranz 0.01).*

- **Position nearestGridPoint** (**Position** pos)

  Ermittelt zu einer gegebenen **Position**(S. 75) *den nächstgelegenen gültigen Gitterpunkt.*

- virtual void **renderShape** ()

  Legt die Geometrie aller Objekte dieser Klasse fest.

Geschützte Methoden

- virtual void **renderDefaultGraphicProperties** ()

  Legt die graphischen Eigenschaften für Grids fest.

Geschützte Attribute

- int **m_Rows**

  Anzahl der Zeilen des Grid.

- int **m_Columns**

  Anzahl der Spalten des Grid.

- GLuint **m_DisplayList**

  ID der OpenGL-Displayliste, mit der das Grid gezeichnet wird.

- **UnitConversionInfo** & **m_rUnitInfo**

  Referenz auf das Einheitenobjekt auf dessen Basis das Grid dargestellt werden soll. Wird NICHT vom Grid freigegeben!

5.12.1 Ausführliche Beschreibung

Klasse, die Gitterraster darstellt, auf dem die Elemente eines Graphen Angeordnet werden können. Die **Position**(S. 75) des Grid wird durch die linke untere Ecke festgelegt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.12.2 Beschreibung der Konstruktoren und Destruktoren

5.12.2.1 Grid::Grid (UnitConversionInfo & *unitInfo*, int *cols* = 1, int *rows* = 1)

Konstruktor: Initialisiert das Grid mit einer bestimmten Anzahl an Zeilen und Spalten.

Parameter:
unitInfo Referenz auf das Einheitenobjekt auf dessen Basis das Grid dargestellt werden soll. Wird NICHT vom Grid freigegeben!

cols Anzahl der Spalten des Grid. Default: 1

rows Anzahl der Zeilen des Grid. Default: 1

5.12.3 Dokumentation der Elementfunktionen

5.12.3.1 bool Grid::isGridPoint (Position *pos*)

Stellt fest, ob eine gegebene **Position**(S. 75) mit einem gültigen Gitterpunkt überein stimmt (Toleranz 0.01).

Parameter:
pos Die zu überprüfende **Position**(S. 75).

Rückgabe:
true, wenn pos ein gültiger Gitterpunkt ist, sonst false.

5.12.3.2 Position Grid::nearestGridPoint (Position *pos*)

Ermittelt zu einer gegebenen **Position**(S. 75) den nächstgelegenen gültigen Gitterpunkt.

Parameter:
pos Die zu überprüfende **Position**(S. 75).

Rückgabe:
Position(S. 75) des zu pos nächstgelegenen Gitterpunktes.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Grid.h**
- hapticgraphclasses/**Grid.cpp**

5.13 HapticConstraint Klassenreferenz

Klasse, die ein HLAPI-Constraint zu einem **HapticObject**(S. 56) darstellen kann.

`#include <HapticConstraint.h>`

Öffentliche Methoden

- **HapticConstraint** (HLfloat snapDist=0.0)

  Konstruktor: Registriert die Constraint-Shape bei HLAPI und initialisiert das Objekt mit den angegebenen Werten. Der Constraint ist defaultmäßig aktiviert.

- virtual ~**HapticConstraint** ()

  Destruktor: Meldet die Constraint-Shape bei HLAPI ab.

- void **setSnapDistance** (HLfloat value)

  Setzt den Wert der m_SnapDistance.

- void **renderConstraint** (**HapticObject** *pObj)

  Rendert ein Constraint für das mit pObj bezeichnete Objekt als eigenständige HLAPI-Shape. Darf nicht innerhalb eines hlBeginShape()/hlEndShape-Blockes aufgerufen werden.

- bool **enable** ()

  Sorgt dafür, dass das Constraint bei einem Aufruf von **renderConstraint()**(S. 49) *gerendert wird.*

- bool **disable** ()

  Verhindert, dass das Constraint bei einem Aufruf von **renderConstraint()**(S. 49) *gerendert wird.*

Geschützte Attribute

- HLfloat **m_SnapDistance**

  Der Abstand zwischen der Oberfläche und der **Position**(S. 75) *des Phantom der überschritten werden muss, um das Phantom von der Constraint- Oberfläche zu lösen in Millimetern in Workspace-Koordinaten.*

- const HLuint **m_HLConstraintID**

  ID, mit der die Constraint-Shape bei HLAPI bekannt ist.

- bool **m_Enabled**

  Flag, das angibt, ob der Constraint gerendert werden soll.

5.13.1 Ausführliche Beschreibung

Klasse, die ein HLAPI-Constraint zu einem **HapticObject**(S. 56) darstellen kann.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Das Objekt wirkt auf das Phantom-Device je nach dem Wert der snapDistance mehr oder weniger magnetisch, wenn ihm ein HapticConstraint zugeordnet ist.

5.13.2 Beschreibung der Konstruktoren und Destruktoren

5.13.2.1 HapticConstraint::HapticConstraint (HLfloat *snapDist* = 0.0)

Konstruktor: Registriert die Constraint-Shape bei HLAPI und initialisiert das Objekt mit den angegebenen Werten. Der Constraint ist defaultmäßig aktiviert.

Parameter:

snapDist Abstand von der Oberfläche, den das Phantom überschreiten muss, um sich von ihr zu lösen. Defaultwert ist 0.0, was bedeutet, dass kein Constraint wirkt.

5.13.3 Dokumentation der Elementfunktionen

5.13.3.1 bool HapticConstraint::disable ()

Verhindert, dass das Constraint bei einem Aufruf von **renderConstraint()**(S. 49) gerendert wird.

Rückgabe:

Gibt den vorherigen Aktivierungsstatus zurück.

5.13.3.2 bool HapticConstraint::enable ()

Sorgt dafür, dass das Constraint bei einem Aufruf von **renderConstraint()**(S. 49) gerendert wird.

Rückgabe:

Gibt den vorherigen Aktivierungsstatus zurück.

5.13.3.3 void HapticConstraint::renderConstraint (HapticObject * *pObj*)

Rendert ein Constraint für das mit pObj bezeichnete Objekt als eigenständige HLAPI-Shape. Darf nicht innerhalb eines hlBeginShape()/hlEndShape-Blockes aufgerufen werden.

Parameter:

pObj **HapticObject**(S. 56), für das ein Constraint gerendert werden soll.

5.13.3.4 void HapticConstraint::setSnapDistance (HLfloat *value*)

Setzt den Wert der m_SnapDistance.

Parameter:

value Neuer Wert der m_SnapDistance.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**HapticConstraint.h**
- hapticgraphclasses/**HapticConstraint.cpp**

5.14 HapticCursor Klassenreferenz

Kapselt ein Zeige-Widget, das den Bewegungen des Phantom-Proxy folgt.

`#include <HapticCursor.h>`

Öffentliche Methoden

- **HapticCursor** (int sizePix=DEFAULT_CURSOR_SIZE_PIX)

  Konstuktor: Initialisiert das Cursor-Objekt mit Defaultwerten, sofern nicht eine Größe für den Cursor angegeben wird.

- virtual ~**HapticCursor** ()

  Destuktor: Löscht das Cursor-Objekt.

- void **scale** ()

  Skaliert den Cursor je nach der aktuellen Projektion.

- virtual void **render** ()

  Zeichnet den Cursor so dass er mit der **Position**(S. 75) *des haptischen Gerätes übereinstimmt.*

Geschützte Attribute

- const int **m_SizePixels**

  Größe des Cursors in Pixeln.

- double **m_Scale**

  Skalierungsfaktor, der je nach Projektion berechnet wird.

- GLuint **m_DisplayList**

  ID der OpenGL-Displayliste, mit der der Cursor gezeichnet wird.

5.14.1 Ausführliche Beschreibung

Kapselt ein Zeige-Widget, das den Bewegungen des Phantom-Proxy folgt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Codebasis für die Klasse ist das von SensAble zu Verfügung gestellte Constraints-Beispiel.

5.14.2 Beschreibung der Konstruktoren und Destruktoren

5.14.2.1 HapticCursor::HapticCursor (int *sizePix* = DEFAULT_CURSOR_SIZE_PIX)

Konstuktor: Initialisiert das Cursor-Objekt mit Defaultwerten, sofern nicht eine Größe für den Cursor angegeben wird.

Parameter:

sizePix Größe des Cursors in Pixeln (optional).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**HapticCursor.h**
- hapticgraphclasses/**HapticCursor.cpp**

5.15 HapticDevice Klassenreferenz

Eine Klasse zur Verwaltung des Haptischen Gerätes.

`#include <HapticDevice.h>`

Öffentliche Methoden

- **HapticDevice** ()

  Konstruktor: Initialisiert die Membervariablen mit Defaultwerten Das Gerät wird erst intialisiert, wenn **initialize()**(S. 52) *aufgerufen wird.*

- ~**HapticDevice** ()

  Gibt den Rendering Context und das haptische Device frei.

- void **updateWorkspace** ()

  Passt den Workspace des Gerätes an das Sichtvolumen an.

- bool **isActive** ()

  Mit dieser Methode kann festgestellt werden, ob das haptische Gerät aktiv ist.

- void **initialize** ()

  Initialisiert das haptische Gerät und den haptischen Rendering Context.

Geschützte Attribute

- HHD **m_hHapticDevice**

  Handle, mit dem auf das haptische Gerät zugegriffen werden kann.

- HHLRC **m_hHLRenderingContext**

  Handle zum haptischen Rendering Context.

5.15.1 Ausführliche Beschreibung

Eine Klasse zur Verwaltung des Haptischen Gerätes.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Übernimmt die Initialisierung des Haptischen Gerätes und die Anpassung des Workspaces des Gerätes an das Sichtvolumen.

5.15.2 Dokumentation der Elementfunktionen

5.15.2.1 bool HapticDevice::isActive ()

Mit dieser Methode kann festgestellt werden, ob das haptische Gerät aktiv ist.

Rückgabe:
true, wenn das Gerät benutzt werden kann, sonst false.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**HapticDevice.h**
- hapticgraphclasses/**HapticDevice.cpp**

5.16 HapticEffect Klassenreferenz

Abstrakte Klasse zur Kapselung von HLAPI Force Effects.

`#include <HapticEffect.h>`

Klassendiagramm für HapticEffect::

Öffentliche Methoden

- **HapticEffect** (HLenum type=HL_EFFECT_CONSTANT)

  Konstruktor: Initialisiert das Objekt mit Defaultwerten und macht den Effekt bei HLAPI bekannt.

- virtual ~**HapticEffect** ()

  Destuktor: Gibt den Effekt bei HLAPI frei.

- virtual void **startEffect** ()

  Startet den Effekt. Der Effekt wird so lange gerendert, bis stopEffekt() aufgerufen wird. Wurde der Effekt bereits gestartet, hat der Aufruf von **startEffect()**(S. 54) *keinerlei Auswirkung.*

- virtual void **stopEffect** ()

  Stoppt einen bereits gestarteten Effekt. Ist der Effekt inaktiv, hat der Aufruf von **stopEffect()**(S. 54) *keinerlei Auswirkung.*

- virtual void **triggerEffect** (double duration=100.0)

  Löst den Effekt einmalig aus, er dauert dann für die mit duration spezifizierte Zeit an.

Geschützte Methoden

- virtual void **renderProperties** ()=0

  Muss von abgeleiteten Klassen implementiert werden, um die gewünschten Eigenschaften des Effektes zu spezifizieren.

Geschützte Attribute

- HLenum **m_EffectType**

  Effekttyp.

- HLuint **m_EffectID**

  ID, mit der der Effekt bei HLAPI bekannt ist.

- bool **m_IsActive**

  Flag, das angibt, ob der Effekt aktiv (gestartet) ist.

5.16.1 Ausführliche Beschreibung

Abstrakte Klasse zur Kapselung von HLAPI Force Effects.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Dies ist die Basisklasse für alle "ambienten" Force Effects. Sie ist für das Registrieren und Deregistrieren des Effektes bei HLAPI zuständig und stellt Methoden zum Starten, Stoppen und Triggern des Effektes zur Verfügung. Abgeleitete Klassen, die instanziiert werden sollen, müssen die abstrakte Methode **renderProperties()**(S. 54) implementieren.

Fehler
Manchmal wird zwar **startEffect()**(S. 54) aufgerufen, der Effekt aber nicht gerendert. Das gleiche gilt für **stopEffect()**(S. 54): Manchmal wird der Effekt nicht mehr angehalten. Ursache ist bisher ungekärt.

5.16.2 Dokumentation der Elementfunktionen

5.16.2.1 void HapticEffect::triggerEffect (double *duration* = `100.0`) `[virtual]`

Löst den Effekt einmalig aus, er dauert dann für die mit duration spezifizierte Zeit an.

Parameter:
duration Dauer des Effektes in Millisekunden. Default: 100.0 ms

5.16.3 Dokumentation der Datenelemente

5.16.3.1 HLenum HapticEffect::m_EffectType `[protected]`

Effekttyp.

Zulässige Werte:

- HL_EFFECT_CONSTANT
- HL_EFFECT_SPRING
- HL_EFFECT_VISCOUS
- HL_EFFECT_FRICTION Für eine detailiertere Beschreibung der Effekttypen, siehe 3D TOUCH SDK - API REFERENCE

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**HapticEffect.h**
- hapticgraphclasses/**HapticEffect.cpp**

5.17 HapticObject Klassenreferenz

Basisklasse aller haptischen Objekte. Stellt grundlegende Funktionalität zum Fühlbarmachen und Bewegen von Objekten zur Verfügung. Kümmert sich Um die Registrierung von Eventhandlern.

`#include <HapticObject.h>`

Klassendiagramm für HapticObject::

Öffentliche Methoden

- **HapticObject** ()

  Konstruktor: Initialisiert ein HapticObject mit Defaultwerten: Die Standardposition eines Objektes ist [0.0, 0.0, 0.0], es wird ein inaktiver Constraint für das Objekt definiert.

- virtual ~**HapticObject** ()

  Destruktor: Gibt die Resourcen des Objektes frei. Zugeordnete Constraints und HapticActions werden dabei mit freigegeben.

- virtual void **renderShape** ()=0

  Hier wird die Geometrie des Objekts festgelegt.

- virtual void **renderShapeAtPosition** ()

  Rendert die Geometrie des Objekts an der richtigen **Position**(S. 75) *im Raum.*

- void **renderHaptics** ()

  Rendert das Objekt als Contact-Shape auf dem Phantom. Falls ein Constraint definiert ist, wird auch der gerendert.

- void **renderGraphics** ()

  Rendert das Objekt mit seinen graphischen Eigenschaften.

- void **setHapticConstraint** (**HapticConstraint** *value)

  Setzt ein Constraint für das HapticObject. Das übergebene Constraint-Objekt wird beim Deleten des HapticObjects mit freigegeben.

- **HapticConstraint** * **getHapticConstraint** ()

  Gibt das HapticConstraint-Objekt, das mit dem HapticObject verbunden ist, zurück.

- void **addHapticAction** (**IHapticAction** *act)

  Fügt dem Objekt einen Eventhandler hinzu und sorgt dafür, dass die entprechenden Events für das Objekt registriert werden. Das übergebene Eventhandler-Objekt wird beim Deleten des HapticObjects mit freigegeben.

- virtual void **translate** (const double x, const double y, const double z)

  Verschiebt das Objekt um den Vektor [x, y, z].

- virtual void **setPosition** (const double x, const double y, const double z)

  Platziert das Objekt an der Stelle mit dem Ortsvektor [x, y, z].

- **Position getPosition** ()
  Ermittelt den Ortsvektor der aktuellen **Position**(S. 75) *des Objekts und gibt diesen zurück.*

Geschützte Methoden

- virtual void **renderDefaultHapticProperties** ()
  Hier werden die Standard-Haptikeigenschaften eines Objektes festgelegt.
- virtual void **renderDefaultGraphicProperties** ()
  Hier werden die Standard-Graphikeigenschaften eines Objektes festgelegt.

Geschützte Attribute

- const HLuint **m_HLShapeID**
  ID, mit der das Objekt bei HLAPI bekannt ist.
- hduMatrix **m_transformMatrix**
  Die aktuelle Transform-Matrix des HapticObject.
- **HapticConstraint** $*$ **m_pHapticConstraint**
  Sorgt dafür, dass sich das HapticObject für das Phantom magnetisch anfühlt.
- vector< **IHapticAction** $*$ > **m_HapticActions**
  Liste der Eventhandler, die das Objekt betreffen.

5.17.1 Ausführliche Beschreibung

Basisklasse aller haptischen Objekte. Stellt grundlegende Funktionalität zum Fühlbarmachen und Bewegen von Objekten zur Verfügung. Kümmert sich Um die Registrierung von Eventhandlern.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Da die Klasse HapticObject abstrakt ist, kann sie nicht direkt instanziiert werden. Abgeleitete Klassen müssen die Geometrie für das Objekt erzeugen.

5.17.2 Dokumentation der Elementfunktionen

5.17.2.1 void HapticObject::addHapticAction (IHapticAction $*$ *act*)

Fügt dem Objekt einen Eventhandler hinzu und sorgt dafür, dass die entprechenden Events für das Objekt registriert werden. Das übergebene Eventhandler-Objekt wird beim Deleten des Haptic-Objects mit freigegeben.

Parameter:
act Pointer auf den Actionhandler, der dem Objekt zugeordnet werden soll. Wird von dem HapticObject freigegeben.

5.17.2.2 HapticConstraint * HapticObject::getHapticConstraint ()

Gibt das HapticConstraint-Objekt, das mit dem HapticObject verbunden ist, zurück.

Rückgabe:
value Pointer auf ein Constraint-Objekt, das mit dem HapticObject verbunden ist.

5.17.2.3 Position HapticObject::getPosition ()

Ermittelt den Ortsvektor der aktuellen **Position**(S. 75) des Objekts und gibt diesen zurück.

Rückgabe:
Ortsvektor der aktuellen **Position**(S. 75) des Objekts.

5.17.2.4 void HapticObject::renderDefaultGraphicProperties () [protected, virtual]

Hier werden die Standard-Graphikeigenschaften eines Objektes festgelegt.

Die Standardeigenschaften können für abgeleitete Klassen neu festgelegt werden.

Erneute Implementation in **Edge** (S. 37) und **Grid** (S. 46).

5.17.2.5 void HapticObject::renderDefaultHapticProperties () [protected, virtual]

Hier werden die Standard-Haptikeigenschaften eines Objektes festgelegt.

Die Standardeigenschaften können für abgeleitete Klassen neu festgelegt werden.

5.17.2.6 virtual void HapticObject::renderShape () [pure virtual]

Hier wird die Geometrie des Objekts festgelegt.

Am besten wird dafür eine Displayliste verwendet. Diese Methode muss von allen abgeleiteten Klassen, die instanziiert werden sollen, implementiert werden.

Implementiert in **Edge** (S. 37), **Grid** (S. 46) und **Node** (S. 68).

5.17.2.7 void HapticObject::setHapticConstraint (HapticConstraint * *value*)

Setzt ein Constraint für das HapticObject. Das übergebene Constraint-Objekt wird beim Deleten des HapticObjects mit freigegeben.

Parameter:
value Pointer auf ein Constraint-Objekt, das von jetzt an mit dem HapticObject verbunden ist. Wird von dem HapticObject freigegeben.

5.17.2.8 void HapticObject::setPosition (const double *x*, const double *y*, const double *z*) [virtual]

Platziert das Objekt an der Stelle mit dem Ortsvektor [x, y, z].

Parameter:

x x-Koordinate des Ortsvektors.

y y-Koordinate des Ortsvektors.

z z-Koordinate des Ortsvektors.

Erneute Implementation in **Node** (S. 72).

5.17.2.9 void HapticObject::translate (const double *x*, const double *y*, const double *z*) [virtual]

Verschiebt das Objekt um den Vektor [x, y, z].

Parameter:

x x-Koordinate des Translationsvektors.

y y-Koordinate des Translationsvektors.

z z-Koordinate des Translationsvektors.

Erneute Implementation in **Node** (S. 72).

5.17.3 Dokumentation der Datenelemente

5.17.3.1 vector<IHapticAction*> HapticObject::m_HapticActions [protected]

Liste der Eventhandler, die das Objekt betreffen.

Die zugeordneten Actions werden vom HapticObject freigegeben.

5.17.3.2 HapticConstraint* HapticObject::m_pHapticConstraint [protected]

Sorgt dafür, dass sich das HapticObject für das Phantom magnetisch anfühlt.

Wird vom HapticObject freigegeben.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**HapticObject.h**
- hapticgraphclasses/**HapticObject.cpp**

5.18 HDInitialisationFailedException Klassenreferenz

Exceptionklasse, die einen Fehler bei der Initialisierung des haptischen Geräts anzeigt.

`#include <HapticsExceptions.h>`

Öffentliche Methoden

- **HDInitialisationFailedException** (const string &message)
  Konstruktor: Erzeugt ein Exception-Objekt mit einer Nachricht.

5.18.1 Ausführliche Beschreibung

Exceptionklasse, die einen Fehler bei der Initialisierung des haptischen Geräts anzeigt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.18.2 Beschreibung der Konstruktoren und Destruktoren

5.18.2.1 HDInitialisationFailedException::HDInitialisationFailedException (const string & *message*) [inline]

Konstruktor: Erzeugt ein Exception-Objekt mit einer Nachricht.

Parameter:
message Nachricht, die beim Exceptionhandling abgefragt werden kann.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- exceptionclasses/**HapticsExceptions.h**

5.19 IBusinessAdapter Klassenreferenz

Adapterklasse der eine Referenz zwischen Grafikobjekten der haptischen Szene und den Aufgaben herstellt.

`#include <IBusinessAdapter.h>`

Klassendiagramm für IBusinessAdapter::

Öffentliche Methoden

- virtual **force getForce** (float x, float y)=0
  Liefert die Kraft, die notwendig ist, um eine Aufgabe zu verschieben.
- virtual int **getWidth** ()=0
  Ermittelt die Breite des Darstellungsobjektes in View-Einheiten.
- virtual int **getLine** ()=0
  Ermittelt die Höhe des Darstellungsobjektes in View-Einheiten.
- virtual int **getBegin** ()=0
  Ermittelt den Starttag.
- virtual string **getName** ()=0
  Liefert den Namen des Tasks.
- virtual list< **IBusinessAdapter** * > & **getNextTasks** ()=0
  Liefert die Nachfolger.
- virtual list< **IBusinessAdapter** * > & **getPreviousTasks** ()=0
  Liefert die Vorgänger.
- virtual bool **setBegin** (float begin)=0
  setzt den Anfang einer Aufgabe
- virtual void **moveAllToFront** ()=0
  schiebt alle Aufgaben an den Projektanfang
- virtual void **moveFollowingToFront** (int earliest)=0
  schiebt nachfolgende Aufgaben so weit wie möglich nach vorne
- virtual void **movePreviousToFront** ()=0
  schiebt vorhergehende Aufgaben so weit wie möglich an den Anfang des Projekts
- virtual int **calcForceInc0** ()=0
  berechnet den Punkt über den nicht nach Links verschoben werden kann
- virtual int **calcForceInc1** ()=0
  berechnet den Punkt über den nicht nach Rechts verschoben werden kann

- virtual void **moveToEarlierPosition** (int end)=0
  bewegt eine Aufgabe zu einem früheren Startpunkt

- virtual void **moveToLaterPosition** (int new_begin)=0
  bewegt eine Aufgabe zu einem späteren Startpunkt

5.19.1 Ausführliche Beschreibung

Adapterklasse der eine Referenz zwischen Grafikobjekten der haptischen Szene und den Aufgaben herstellt.

Jeder **Node**(S. 68) enthält eine Referenz auf seine entsprechende Aufgabe um relevante Daten mit Ihr auszutauschen

Noch zu erledigen
Definition der abstrakten Adapter Methoden

5.19.2 Dokumentation der Elementfunktionen

5.19.2.1 virtual int IBusinessAdapter::getBegin () [pure virtual]

Ermittelt den Starttag.

Rückgabe:
Starttag

Implementiert in **BusinessTask** (S. 16).

5.19.2.2 virtual force IBusinessAdapter::getForce (float *x*, float *y*) [pure virtual]

Liefert die Kraft, die notwendig ist, um eine Aufgabe zu verschieben.

Parameter:
x aktueller x-Wert des Knoten in Business Einheiten
y aktueller y-Wert des Knoten in Business Einheiten

Rückgabe:
m_Force (siehe oben)

Implementiert in **BusinessTask** (S. 17).

5.19.2.3 virtual int IBusinessAdapter::getLine () [pure virtual]

Ermittelt die Höhe des Darstellungsobjektes in View-Einheiten.

Rückgabe:
Höhe des Darstellungsobjektes in View-Einheiten.

Implementiert in **BusinessTask** (S. 17).

5.19.2.4 virtual string IBusinessAdapter::getName () [pure virtual]

Liefert den Namen des Tasks.

Rückgabe:
Name als string

Implementiert in **BusinessTask** (S. 17).

5.19.2.5 virtual list<IBusinessAdapter*>& IBusinessAdapter::getNextTasks () [pure virtual]

Liefert die Nachfolger.

Rückgabe:
Liste mit Nachfolgern

Implementiert in **BusinessTask** (S. 17).

5.19.2.6 virtual list<IBusinessAdapter*>& IBusinessAdapter::getPreviousTasks () [pure virtual]

Liefert die Vorgänger.

Rückgabe:
Liste mit Vorgängern

Implementiert in **BusinessTask** (S. 17).

5.19.2.7 virtual int IBusinessAdapter::getWidth () [pure virtual]

Ermittelt die Breite des Darstellungsobjektes in View-Einheiten.

Rückgabe:
Breite des Darstellungsobjektes in View-Einheiten.

Implementiert in **BusinessTask** (S. 18).

5.19.2.8 virtual void IBusinessAdapter::moveFollowingToFront (int *earliest*) [pure virtual]

schiebt nachfolgende Aufgaben so weit wie möglich nach vorne

Parameter:
earliest frühester Anfang nachfolgender Aufgaben

Implementiert in **BusinessTask** (S. 12).

5.19.2.9 virtual void IBusinessAdapter::moveToEarlierPosition (int *end*) [pure virtual]

bewegt eine Aufgabe zu einem früheren Startpunkt

Parameter:
end erhält sein neues Ende

Implementiert in **BusinessTask** (S. 18).

5.19.2.10 virtual void IBusinessAdapter::moveToLaterPosition (int *new_begin*) [pure virtual]

bewegt eine Aufgabe zu einem späteren Startpunkt

Parameter:
new_begin erhält seinen neuen Anfang

Implementiert in **BusinessTask** (S. 18).

5.19.2.11 virtual bool IBusinessAdapter::setBegin (float *begin*) [pure virtual]

setzt den Anfang einer Aufgabe

Parameter:
begin Anhand des Übergabewertes entscheidet die Aufgabe den genauen Begin

Implementiert in **BusinessTask** (S. 18).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- businesslogic/**IBusinessAdapter.h**

5.20 IHapticAction Klassenreferenz

Schnittstelle für haptische Eventhandler.

`#include <HapticAction.h>`

Klassendiagramm für IHapticAction::

Öffentliche Methoden

- virtual void **unregisterAction** (HLuint shapeID)=0

  Registriert ein Event für das haptische Objekt, dem es zugeordnet ist.

- virtual void **registerAction** (HLuint shapeID)=0

  Löscht ein Event für das haptische Objekt, dem es zugeordnet ist.

5.20.1 Ausführliche Beschreibung

Schnittstelle für haptische Eventhandler.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Implementierende Klassen müssen darauf achten, dass alle HLAPI-Events sauber registriert und deregistriert werden.

5.20.2 Dokumentation der Elementfunktionen

5.20.2.1 virtual void IHapticAction::registerAction (HLuint *shapeID*) [pure virtual]

Löscht ein Event für das haptische Objekt, dem es zugeordnet ist.

Parameter:
shapeID Die ID des Objektes, für das das Event deregistriert werden soll.

Implementiert in **DragNodeOnGridHandler** (S. 28), **DragObjectHandler** (S. 32) und **DragSceneHandler** (S. 36).

5.20.2.2 virtual void IHapticAction::unregisterAction (HLuint *shapeID*) [pure virtual]

Registriert ein Event für das haptische Objekt, dem es zugeordnet ist.

Parameter:
shapeID Die ID des Objektes, für das das Event registriert werden soll.

Implementiert in **DragNodeOnGridHandler** (S. 28), **DragObjectHandler** (S. 32) und **DragSceneHandler** (S. 36).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- hapticgraphclasses/**HapticAction.h**

5.21 IObserver Klassenreferenz

Schnittstelle für das Observer-Pattern.

`#include <IObserver.h>`

Klassendiagramm für IObserver::

Öffentliche Methoden

- virtual void **Update** (**Observable** *pObservable)=0

  Veranlasst den Observer, sich die benötigten Informationen vom **Observable**(S. 73) *zu holen.*

5.21.1 Ausführliche Beschreibung

Schnittstelle für das Observer-Pattern.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Klassen, die diese Schnittstelle implementieren, können sich bei Observables anmelden um über Änderungen informiert zu werden.

5.21.2 Dokumentation der Elementfunktionen

5.21.2.1 virtual void IObserver::Update (Observable * *pObservable*) [pure virtual]

Veranlasst den Observer, sich die benötigten Informationen vom **Observable**(S. 73) zu holen.

Parameter:
pObservable Das observierte Objekt, von dem der Observer Informationen braucht.

Implementiert in **Node** (S. 69).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- businesslogic/**IObserver.h**

5.22 Node Klassenreferenz

Haptisches Objekt, das einen Knoten in einem Graphen als Rechteck darstellt.

`#include <Node.h>`

Klassendiagramm für Node::

Öffentliche Methoden

- **Node** (**IBusinessAdapter** *businessObj, **UnitConversionInfo** &unitInfo)

  Konstruktor: Initialisiert das Objekt mit dem dazugehörigen Business-Objekt. Alle Darstellungsinformationen werden vom Business-Objekt angefordert.

- virtual ~**Node** ()

  Destruktor: Gibt die Ressourcen des Objektes frei.

- void **setWidth** (float value)

  Ändert die Breite des Node.

- void **setHeight** (float value)

  Ändert die Höhe des Node.

- float **getWidth** ()

  Gibt die aktuelle Breite des Node zurück.

- float **getHeight** ()

  Gibt die aktuelle Höhe des Node zurück.

- void **setImpossibleToMoveEffect** (**HapticEffect** *value)

  Weist dem Node einen neuen Effekt für das Verhindern von Bewegung zu.

- void **setHardToMoveEffect** (**HapticEffect** *value)

  Weist dem Node einen neuen Effekt für die eigeschränkte Bewegung zu.

- void **addIncomingEdge** (**Edge** *pEdge)

  Fügt dem Knoten eine eingehende Kante hinzu.

- void **addOutgoingEdge** (**Edge** *pEdge)

  Fügt dem Knoten eine ausgehende Kante hinzu.

- virtual void **renderShape** ()

  Legt die Geometrie aller Objekte dieser Klasse fest.

- virtual void **translate** (const double x, const double y, const double z)

  Verschiebt den Node um den Vektor [x, y, z] unter Beachtung durch die Businesslogik definierten Restriktionen.

- virtual void **setPosition** (const double x, const double y, const double z)

Platziert das Objekt an der Stelle mit dem Ortsvektor [x, y, z], falls das Business-Objekt es erlaubt. Ansonsten wird der Node an einer vom Business-Objekt berechneten **Position**(S. 75) *platziert.*

- virtual void **Update** (**Observable** *pObservable)

  Veranlasst den Observer, sich die benötigten Informationen vom **Observable**(S. 73) *zu holen.*

Geschützte Methoden

- void **updateIncomingEdge** (**Edge** *pEdge)

  Setzt den Endpunkt einer eingehenden Kante neu.

- void **updateOutgoingEdge** (**Edge** *pEdge)

  Setzt den Anfangspunkt einer ausgehenden Kante neu.

- void **updateEdges** ()

  Aktualisiert die Anfangs- und Endpunkte der Kanten, die mit dem Knoten verbunden sind.

- void **releaseDisplayList** ()

  Gibt die Displayliste frei und weist ihr einen ungültigen Wert zu.

Geschützte Attribute

- float **m_Width**

  Breite des Node in View-Koordinaten.

- float **m_Height**

  Höhe des Node in View-Koordinaten.

- **IBusinessAdapter** * **m_pBusinessObject**

  Zeiger auf ein Businesslogik-Objekt, das der Node darstellen soll. Die Eigenschaften und das Verhalten des Node werden von diesem Objekt abgefragt. Wird NICHT vom Node freigegeben!

- **UnitConversionInfo** & **m_rUnitInfo**

  Referenz auf das Einheitenobjekt auf dessen Basis der Node dargestellt werden soll. Wird NICHT vom Node freigegeben!

- **HapticEffect** * **m_pHardToMoveEffect**

  Effekt, der aktiviert wird, wenn sich der Node nur eingeschränkt bewegen lassen soll. Wird vom Node erzeugt und freigegeben.

- **HapticEffect** * **m_pImpossibleToMoveEffect**

  Effekt, der aktiviert wird, wenn sich der Node gar nicht bewegen lassen soll. Wird vom Node erzeugt und freigegeben.

- list< **Edge** * > **m_IncomingEdges**

  Liste von Verbindungen zu Vorgängerknoten.

- list< **Edge** * > **m_OutgoingEdges**

  Liste von Verbindungen zu Nachfolgerknoten.

- GLuint **m_DisplayList**

  ID der OpenGL-Displayliste, mit der der Node gezeichnet wird.

5.22.1 Ausführliche Beschreibung

Haptisches Objekt, das einen Knoten in einem Graphen als Rechteck darstellt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Der Node lässt sich mit verschieden starken Restriktionen bewegen, je nach Definition der Businesslogik:

- Frei bewegbar
- eingeschränkt bewegbar
- gar nicht bewegbar In der Rolle des Observers beobachtet der Node sein Businesslogik-Objekt und reagiert auf dessen Änderungen.

Noch zu erledigen
Der Effekt für "gar nicht bewegbar" muss noch richtig eingestellt werden.

5.22.2 Beschreibung der Konstruktoren und Destruktoren

5.22.2.1 Node::Node (IBusinessAdapter * *businessObj*, UnitConversionInfo & *unitInfo*)

Konstruktor: Initialisiert das Objekt mit dem dazugehörigen Business-Objekt. Alle Darstellungsinformationen werden vom Business-Objekt angefordert.

Parameter:
businessObj Zeiger auf ein Businesslogik-Objekt, das der Node darstellen soll. Wird NICHT vom Node freigegeben!

unitInfo Referenz auf das Einheitenobjekt auf dessen Basis der Node dargestellt werden soll. Wird NICHT vom Node freigegeben!

5.22.3 Dokumentation der Elementfunktionen

5.22.3.1 void Node::addIncomingEdge (Edge * *pEdge*)

Fügt dem Knoten eine eingehende Kante hinzu.

Parameter:
pEdge Kante, die dem Knoten hinzugefügt werden soll. Die Kante wird beim Löschen des Knotens nicht freigegeben!

5.22.3.2 void Node::addOutgoingEdge (Edge * *pEdge*)

Fügt dem Knoten eine ausgehende Kante hinzu.

Parameter:

pEdge Kante, die dem Knoten hinzugefügt werden soll. Die Kante wird beim Löschen des Knotens nicht freigegeben!

5.22.3.3 float Node::getHeight ()

Gibt die aktuelle Höhe des Node zurück.

Rückgabe:

Die aktuelle Höhe des Node.

5.22.3.4 float Node::getWidth ()

Gibt die aktuelle Breite des Node zurück.

Rückgabe:

Die aktuelle Breite des Node.

5.22.3.5 void Node::setHardToMoveEffect (HapticEffect * *value*)

Weist dem Node einen neuen Effekt für die eigeschränkte Bewegung zu.

Parameter:

value Zeiger auf den neuen Effekt für die eigeschränkte Bewegung. Wird vom Node freigegeben.

5.22.3.6 void Node::setHeight (float *value*)

Ändert die Höhe des Node.

Parameter:

value Neue Höhe des Node.

5.22.3.7 void Node::setImpossibleToMoveEffect (HapticEffect * *value*)

Weist dem Node einen neuen Effekt für das Verhindern von Bewegung zu.

Parameter:

value Zeiger auf den neuen Effekt für das Verhindern von Bewegung. Wird vom Node freigegeben.

5.22.3.8 void Node::setPosition (const double *x*, const double *y*, const double *z*) [virtual]

Platziert das Objekt an der Stelle mit dem Ortsvektor [x, y, z], falls das Business-Objekt es erlaubt. Ansonsten wird der Node an einer vom Business-Objekt berechneten **Position**(S. 75) platziert.

Parameter:

x x-Koordinate des Ortsvektors.

y y-Koordinate des Ortsvektors.

z z-Koordinate des Ortsvektors.

Erneute Implementation von **HapticObject** (S. 59).

5.22.3.9 void Node::setWidth (float *value*)

Ändert die Breite des Node.

Parameter:

value Neue Breite des Node.

5.22.3.10 void Node::translate (const double *x*, const double *y*, const double *z*) [virtual]

Verschiebt den Node um den Vektor [x, y, z] unter Beachtung durch die Businesslogik definierten Restriktionen.

Parameter:

x x-Koordinate des Translationsvektors.

y y-Koordinate des Translationsvektors.

z z-Koordinate des Translationsvektors.

Erneute Implementation von **HapticObject** (S. 59).

5.22.3.11 void Node::updateIncomingEdge (Edge * *pEdge*) [protected]

Setzt den Endpunkt einer eingehenden Kante neu.

Parameter:

pEdge **Edge**(S. 37), deren Endpunkt verändert werden soll.

5.22.3.12 void Node::updateOutgoingEdge (Edge * *pEdge*) [protected]

Setzt den Anfangspunkt einer ausgehenden Kante neu.

Parameter:

pEdge **Edge**(S. 37), deren Anfangspunkt verändert werden soll.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Node.h**
- hapticgraphclasses/**Node.cpp**

5.23 Observable Klassenreferenz

Implementiert das Observer-Pattern.

`#include <Observable.h>`

Klassendiagramm für Observable::

Öffentliche Methoden

- **Observable** ()
  Konstrktor: Initialisiert das Objekt.
- void **AddObserver** (**IObserver** *pObserver)
  Registriert einen neuen Obersver.
- void **RemoveObserver** (**IObserver** *pObserver)
  Meldet einen Observer beim Observable ab. Das Observer-Objekt wird dabei nicht gelöscht.
- void **NotifyAll** ()
  Informiert alle Observer darüber, dass sich der Zustand des Observables geändert hat.

Geschützte Attribute

- list< **IObserver** * > **m_Observers**
  Liste aller angemeldeten Observer.

5.23.1 Ausführliche Beschreibung

Implementiert das Observer-Pattern.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Klassen, die von Observable erben, können Objekte, die an ihrem Zustand interessiert sind über Änderungen informieren ohne eine enge Bindung zu ihnen zu haben.

5.23.2 Dokumentation der Elementfunktionen

5.23.2.1 void Observable::AddObserver (IObserver * *pObserver*)

Registriert einen neuen Obersver.

Parameter:
pObserver Zeiger auf ein Observer-Objekt, das den Observable von jetzt an beobachten will.

5.23.2.2 void Observable::RemoveObserver (IObserver * *pObserver*)

Meldet einen Observer beim Observable ab. Das Observer-Objekt wird dabei nicht gelöscht.

Parameter:

pObserver Zeiger auf das Observer-Objekt, das von jetzt an den Observable nicht mehr beobachten will.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- businesslogic/**Observable.h**
- businesslogic/**Observable.cpp**

5.24 Position Strukturreferenz

Stellt einen Punkt im 3D-Raum dar.

`#include <Utilities.h>`

Öffentliche Methoden

- **Position** (double **x**=0.0, double **y**=0.0, double **z**=0.0)

  Konstruktor: Initialisiert die Position mit ihren x-, y- und z-Koordinaten.

- virtual ~**Position** ()

  Destruktor: Gibt die Resourcen des Objekts frei.

- **Position operator**+ (**Position** pos)

  Addiert die Koordinaten der gegebenen Position pos zu den Koordinaten des aktuellen Objektes und gibt das Ergebnis als Positon zurück. Das aktuelle Objekt wird dabei nicht verändert.

- **Position operator**+ (double value)

  Addiert einen gegebenen Wert zu den Koordinaten des aktuellen Objektes und gibt das Ergebnis als Positon zurück. Das aktuelle Objekt wird dabei nicht verändert.

Öffentliche Attribute

- double **x**

  x-Koordinate der Position.

- double **y**

  y-Koordinate der Position.

- double **z**

  z-Koordinate der Position.

5.24.1 Ausführliche Beschreibung

Stellt einen Punkt im 3D-Raum dar.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.24.2 Beschreibung der Konstruktoren und Destruktoren

5.24.2.1 Position::Position (double x = 0.0, double y = 0.0, double z = 0.0)

Konstruktor: Initialisiert die Position mit ihren x-, y- und z-Koordinaten.

Werden beim Erzeugen eines Position-Objektes keine Koordinaten angegeben, wird es per default mit [0.0, 0.0, 0.0] initialisiert.

Parameter:

x x-Koordinate der Position.

y y-Koordinate der Position.

z z-Koordinate der Position.

5.24.3 Dokumentation der Elementfunktionen

5.24.3.1 Position Position::operator+ (double *value*)

Addiert einen gegebenen Wert zu den Koordinaten des aktuellen Objektes und gibt das Ergebnis als Positon zurück. Das aktuelle Objekt wird dabei nicht verändert.

Parameter:

value Wert, der auf die aktuelle Position addiert werden soll.

Rückgabe:

Position mit den addierten Koordinaten.

5.24.3.2 Position Position::operator+ (Position *pos*)

Addiert die Koordinaten der gegebenen Position pos zu den Koordinaten des aktuellen Objektes und gibt das Ergebnis als Positon zurück. Das aktuelle Objekt wird dabei nicht verändert.

Parameter:

pos Position, die auf die aktuelle Position addiert werden soll.

Rückgabe:

Position mit den addierten Koordinaten.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Utilities.h**
- hapticgraphclasses/**Utilities.cpp**

5.25 SpringForceEffect Klassenreferenz

Klasse zur Ausgabe einer Federkraft auf dem Phantom Device.

`#include <SpringForceEffect.h>`

Klassendiagramm für SpringForceEffect::

Öffentliche Methoden

- **SpringForceEffect** (double magnitude, double gain, **Position** anchor)

  Konstruktor: Initialisiert den Effekt mit den angegebenen Werten.

- virtual ~**SpringForceEffect** ()

  Destruktor: Gibt die Resourcen des Objekts frei.

- void **setMagnitude** (double value)

  Setzt den Wert für die Magnitude der Kraft.

- virtual void **setGain** (double value)

  Setzt den Wert für die Zunahme der Kraft.

- void **setAnchorPosition** (double x, double y, double z)

  Ändert den Ankerpunkt der dargestellten Federkraft.

Geschützte Methoden

- virtual void **renderProperties** ()

  Spezifiziert die Eigenschaften des Effektes für HLAPI.

Geschützte Attribute

- double **m_Magnitude**

  "Größe" der Kraft.

- double **m_Gain**

  Zunahme der Kraft.

- hduVector3Dd **m_AnchorPosition**

  Ankerpunkt der dargestellten Federkraft.

5.25.1 Ausführliche Beschreibung

Klasse zur Ausgabe einer Federkraft auf dem Phantom Device.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.25.2 Beschreibung der Konstruktoren und Destruktoren

5.25.2.1 SpringForceEffect::SpringForceEffect (double *magnitude*, double *gain*, Position *anchor*)

Konstruktor: Initialisiert den Effekt mit den angegebenen Werten.

Parameter:
magnitude "Größe" der Kraft.
gain Zunahme der Kraft.
anchor Ankerpunkt der dargestellten Federkraft.

5.25.3 Dokumentation der Elementfunktionen

5.25.3.1 void SpringForceEffect::setAnchorPosition (double *x*, double *y*, double *z*)

Ändert den Ankerpunkt der dargestellten Federkraft.

Parameter:
x x-Koordinate des neuen Ankerpunktes.
y y-Koordinate des neuen Ankerpunktes.
z z-Koordinate des neuen Ankerpunktes.

5.25.3.2 void SpringForceEffect::setGain (double *value*) [virtual]

Setzt den Wert für die Zunahme der Kraft.

Parameter:
value Neuer Wert für die Zunahme der Kraft.

5.25.3.3 void SpringForceEffect::setMagnitude (double *value*)

Setzt den Wert für die Magnitude der Kraft.

Parameter:
value Neuer Wert für die Magnitude der Kraft.

5.25.4 Dokumentation der Datenelemente

5.25.4.1 double SpringForceEffect::m_Gain [protected]

Zunahme der Kraft.

Für weiterführende Erkärungen s. "API Reference" des OpenHaptics Toolkit.

5.25.4.2 double SpringForceEffect::m_Magnitude [protected]

"Größe" der Kraft.

Für weiterführende Erkärungen s. "API Reference" des OpenHaptics Toolkit.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**SpringForceEffect.h**
- hapticgraphclasses/**SpringForceEffect.cpp**

5.26 UnitConversionInfo Klassenreferenz

Klasse, die die Umwandlung zwischen View- und Businesskoordinaten übernimmt.

`#include <Utilities.h>`

Öffentliche Methoden

- **UnitConversionInfo** (int upsHorz, int upsVert, float paddingHorz, float paddingVert)

  Konstruktor: Initialisiert das Objekt mit den angegebenen Werten und berechnet daraus die Einheitengröße.

- virtual ~**UnitConversionInfo** ()

  Destruktor: Gibt die Resourcen des Objekts frei.

- int **getUnitsPerScreenVertical** ()

  Gibt die aktuell gesetzte Anzahl der Einheiten, die in vertikaler Richtung auf dem Bildschirm dargestellt werden sollen, zurück.

- int **getUnitsPerScreenHorizontal** ()

  Gibt die aktuell gesetzte Anzahl der Einheiten, die in horizontaler Richtung auf dem Bildschirm dargestellt werden sollen, zurück.

- float **getUnitWidth** ()

  Gibt die Breite einer Einheit in View-Einheiten zurück.

- float **getUnitHeight** ()

  Gibt die Höhe einer Einheit in View-Einheiten zurück.

- float **getHorizontalPadding** ()

  Gibt den horizontalen Abstand zwischen zwei Knoten zurück.

- float **getVerticalPadding** ()

  Gibt den vertikalen Abstand zwischen zwei Knoten zurück.

- float **xValueToUnit** (float xValue)

  Rechnet eine View-x-Koordinate in Business-Einheiten um.

- float **yValueToUnit** (float yValue)

  Rechnet eine View-y-Koordinate in Business-Einheiten um.

- float **unitToXValue** (float unit)

  Rechnet einen Business-Einheiten-Wert in View-x-Koordinaten um.

- float **unitToYValue** (float unit)

  Rechnet einen Business-Einheiten-Wert in View-y-Koordinaten um.

Geschützte Attribute

- const int **m_UnitsPerScreenVertical**

  Anzahl der Einheiten, die in vertikaler Richtung auf dem Bildschirm dargestellt werden sollen.

- const int **m_UnitsPerScreenHorizontal**

  Anzahl der Einheiten, die in horizontaler Richtung auf dem Bildschirm dargestellt werden sollen.

- float **m_UnitWidth**

  Breite einer Einheit in View-Einheiten.

- float **m_UnitHeight**

  Höhe einer Einheit in View-Einheiten.

- const float **m_HorizontalPadding**

  Horizontaler Abstand zwischen zwei Knoten.

- const float **m_VerticalPadding**

  Vertikaler Abstand zwischen zwei Knoten.

Statische geschützte Attribute

- const float **ScreenWidthInViewUnits** = 3.6f

  Bildschirmbreite in View-Einheiten.

- const float **ScreenHeihgtInViewUnits**

  Bildschirmhöhe in View-Einheiten.

5.26.1 Ausführliche Beschreibung

Klasse, die die Umwandlung zwischen View- und Businesskoordinaten übernimmt.

Autor:
Katharina Greiner, Matr.-Nr. 943471

Noch zu erledigen
Die Umrechnung erfolgt bisher mit festen Dummy-Werten. Die dynamische Berechnung der Einheiten aus der einstellbaren Anzahl der Einheiten pro Bildschirm muss noch implementiert werden.

5.26.2 Beschreibung der Konstruktoren und Destruktoren

5.26.2.1 UnitConversionInfo::UnitConversionInfo (int *upsHorz*, int *upsVert*, float *paddingHorz*, float *paddingVert*)

Konstruktor: Initialisiert das Objekt mit den angegebenen Werten und berechnet daraus die Einheitengröße.

Parameter:

upsHorz Anzahl der Einheiten, die in horizontaler Richtung auf dem Bildschirm dargestellt werden sollen.

upsVert Anzahl der Einheiten, die in vertikaler Richtung auf dem Bildschirm dargestellt werden sollen.

paddingHorz Horizontaler Abstand zwischen zwei Knoten.

paddingVert Vertikaler Abstand zwischen zwei Knoten.

Noch zu erledigen

Die dynamische Berechnung der Einheiten aus der Anzahl der Einheiten pro Bildschirm muss noch implementiert werden.

5.26.3 Dokumentation der Elementfunktionen

5.26.3.1 float UnitConversionInfo::getHorizontalPadding ()

Gibt den horizontalen Abstand zwischen zwei Knoten zurück.

Rückgabe:

Horizontaler Abstand zwischen zwei Knoten.

5.26.3.2 float UnitConversionInfo::getUnitHeight ()

Gibt die Höhe einer Einheit in View-Einheiten zurück.

Rückgabe:

Höhe einer Einheit in View-Einheiten.

5.26.3.3 int UnitConversionInfo::getUnitsPerScreenHorizontal ()

Gibt die aktuell gesetzte Anzahl der Einheiten, die in horizontaler Richtung auf dem Bildschirm dargestellt werden sollen, zurück.

Rückgabe:

Anzahl der Einheiten, die in horizontaler Richtung auf dem Bildschirm dargestellt werden.

5.26.3.4 int UnitConversionInfo::getUnitsPerScreenVertical ()

Gibt die aktuell gesetzte Anzahl der Einheiten, die in vertikaler Richtung auf dem Bildschirm dargestellt werden sollen, zurück.

Rückgabe:

Anzahl der Einheiten, die in vertikaler Richtung auf dem Bildschirm dargestellt werden.

5.26.3.5 float UnitConversionInfo::getUnitWidth ()

Gibt die Breite einer Einheit in View-Einheiten zurück.

Rückgabe:
Breite einer Einheit in View-Einheiten.

5.26.3.6 float UnitConversionInfo::getVerticalPadding ()

Gibt den vertikalen Abstand zwischen zwei Knoten zurück.

Rückgabe:
Vertikaler Abstand zwischen zwei Knoten.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**Utilities.h**
- hapticgraphclasses/**Utilities.cpp**

5.27 ViscousForceEffect Klassenreferenz

Klasse zur Ausgabe einer "ambienten" Viskostät auf dem Phantom Device.

`#include <ViscousForceEffect.h>`

Klassendiagramm für ViscousForceEffect::

Öffentliche Methoden

- **ViscousForceEffect** (double magnitude=0.0, double gain=0.0)

  Konstruktor: Initialisiert den Effekt mit den angegebenen Werten.

- virtual ~**ViscousForceEffect** ()

  Destruktor: Gibt die Resourcen des Objekts frei.

- void **setMagnitude** (double value)

  Setzt den Wert für die Magnitude der Kraft.

- virtual void **setGain** (double value)

  Setzt den Wert für die Zunahme der Kraft.

Geschützte Methoden

- virtual void **renderProperties** ()

  Spezifiziert die Eigenschaften des Effektes für HLAPI.

Geschützte Attribute

- double **m_Magnitude**

  "Größe" der Kraft.

- double **m_Gain**

  Zunahme der Kraft.

5.27.1 Ausführliche Beschreibung

Klasse zur Ausgabe einer "ambienten" Viskostät auf dem Phantom Device.

Autor:
Katharina Greiner, Matr.-Nr. 943471

5.27.2 Beschreibung der Konstruktoren und Destruktoren

5.27.2.1 ViscousForceEffect::ViscousForceEffect (double *magnitude* = 0.0, double *gain* = 0.0)

Konstruktor: Initialisiert den Effekt mit den angegebenen Werten.

Parameter:

magnitude "Größe" der Kraft. Default: 0.0

gain Zunahme der Kraft. Default: 0.0

5.27.3 Dokumentation der Elementfunktionen

5.27.3.1 void ViscousForceEffect::setGain (double *value*) [virtual]

Setzt den Wert für die Zunahme der Kraft.

Parameter:

value Neuer Wert für die Zunahme der Kraft.

5.27.3.2 void ViscousForceEffect::setMagnitude (double *value*)

Setzt den Wert für die Magnitude der Kraft.

Parameter:

value Neuer Wert für die Magnitude der Kraft.

5.27.4 Dokumentation der Datenelemente

5.27.4.1 double ViscousForceEffect::m_Gain [protected]

Zunahme der Kraft.

Für weiterführende Erkärungen s. "API Reference" des OpenHaptics Toolkit.

5.27.4.2 double ViscousForceEffect::m_Magnitude [protected]

"Größe" der Kraft.

Für weiterführende Erkärungen s. "API Reference" des OpenHaptics Toolkit.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- hapticgraphclasses/**ViscousForceEffect.h**
- hapticgraphclasses/**ViscousForceEffect.cpp**

Kapitel 6

Phantom Graph Demo Datei-Dokumentation

6.1 businesslogic/AppConfiguration.cpp Dateireferenz

```
#include "AppConfiguration.h"
```

6.1.1 Ausführliche Beschreibung

Autor:
Carsten, Arnold

Datum:
Erstellt am 30.11.2005
Letzte Änderung 27.01.2006

6.2 businesslogic/AppConfiguration.h Dateireferenz

```
#include "BusinessTask.h"
#include <list>
#include <iostream>
```

Namensbereiche

- namespace **std**

Klassen

- class **AppConfiguration**

  Klasse, die für projektübergreifende Aufgaben zuständig ist und static Member zur Verfügung stellt, Singelton.

6.2.1 Ausführliche Beschreibung

Autor:
Carsten, Arnold

Datum:
Erstellt am 30.11.2005
Letzte Änderung 27.01.2006

6.3 businesslogic/BusinessTask.cpp Dateireferenz

```
#include "BusinessTask.h"
#include "AppConfiguration.h"
#include <stdio.h>
```

Variablen

- **AppConfiguration appData**
  Objekt mit globale Konfigurations Daten CA.

6.3.1 Ausführliche Beschreibung

Autor:
Carsten Arnold

Datum:
Erstellt am 30.11.2005
Letzte Änderung 30.01.2006 CA

6.3.2 Variablen-Dokumentation

6.3.2.1 AppConfiguration appData

Objekt mit globale Konfigurations Daten CA.

Autor:
Carsten Arnold

6.4 businesslogic/BusinessTask.h Dateireferenz

```
#include <string>
#include <list>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include "IBusinessAdapter.h"
#include "IBusinessConverter.h"
#include "..\HAPTICGRAPHCLASSES\Utilities.h"
```

Klassen

- class **BusinessTask**

  diese KLasse verwaltet alle Aufgaben, die im Graph dargestellt werden sollen.

6.4.1 Ausführliche Beschreibung

Autor:
Carsten Arnold

Datum:
Erstellt am 30.11.2005
Letzte Änderung 30.01.2006 CA

6.5 businesslogic/IBusinessAdapter.h Dateireferenz

```
#include "../hapticgraphclasses/Utilities.h"
#include "Observable.h"
#include <string>
#include <stdlib.h>
#include <stdio.h>
```

Klassen

- class **IBusinessAdapter**

  Adapterklasse der eine Referenz zwischen Grafikobjekten der haptischen Szene und den Aufgaben herstellt.

Aufzählungen

- enum **force** { **none**, **medium**, **incredible** }

  Aufzählungstyp für die Kraft.

6.5.1 Ausführliche Beschreibung

Autor:
Carsten Arnold, Matr.-Nr. 232237

Datum:
Erstellt am 15.12.2005
Letzte Änderung 24.01.2006

6.6 businesslogic/IObserver.h Dateireferenz

```
#include "Observable.h"
```

Klassen

- class **IObserver**

  Schnittstelle für das Observer-Pattern.

6.6.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 28.12.2005
Letzte Änderung 30.12.2005

6.7 businesslogic/Observable.cpp Dateireferenz

```
#include "Observable.h"
```

6.7.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 28.12.2005
Letzte Änderung 28.12.2005

6.8 businesslogic/Observable.h Dateireferenz

```
#include "IObserver.h"
#include <list>
```

Klassen

- class **Observable**

  Implementiert das Observer-Pattern.

6.8.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 28.12.2005
Letzte Änderung 05.02.2006

6.9 exceptionclasses/HapticsExceptions.h Dateireferenz

```
#include <stdexcept>
```

Klassen

- class **HDInitialisationFailedException**

  Exceptionklasse, die einen Fehler bei der Initialisierung des haptischen Geräts anzeigt.

6.9.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 27.12.2005

6.10 hapticgraphclasses/Camera.cpp Dateireferenz

```
#include <math.h>
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include "Camera.h"
```

6.10.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.01.2006
Letzte Änderung 26.01.2006

6.11 hapticgraphclasses/Camera.h Dateireferenz

```
#include "HapticDevice.h"
```

Klassen

- class **Camera**

  Eine Klasse, die unter Verwendung von OpenGL eine minimale Kameraführung erlaubt.

6.11.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.01.2006
Letzte Änderung 26.01.2006

6.12 hapticgraphclasses/ConstantForceEffect.cpp Dateireferenz

```
#include "ConstantForceEffect.h"
#include <HL/hl.h>
```

6.12.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 05.01.2006
Letzte Änderung 05.01.2006

6.13 hapticgraphclasses/ConstantForceEffect.h Dateireferenz

```
#include <HDU/hduVector.h>
```

```
#include "HapticEffect.h"
```

Klassen

- class **ConstantForceEffect**

  *Klasse zur Ausgabe einer konstanten Kraft in Richtung des Direction- Vektors (***setDirection()**(S. 24)*) auf dem Phantom Device.*

6.13.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 05.01.2006
Letzte Änderung 05.01.2006

6.14 hapticgraphclasses/DragNodeOnGridHandler.cpp Dateireferenz

```
#include "DragNodeOnGridHandler.h"
```

6.14.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 23.01.2006
Letzte Änderung 26.01.2006

6.15 hapticgraphclasses/DragNodeOnGridHandler.h Dateireferenz

```
#include "HapticAction.h"
#include "Node.h"
#include "Grid.h"
#include "Utilities.h"
```

Klassen

- class **DragNodeOnGridHandler**

  Eine Eventhandlerklasse die es ermöglicht, haptische Nodes mit dem Phantom in x-Richtung auf einem **Grid**(S. 46) *zu bewegen.*

6.15.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 23.01.2006
Letzte Änderung 23.01.2006

6.16 hapticgraphclasses/DragObjectHandler.cpp Dateireferenz

```
#include "DragObjectHandler.h"
```

6.16.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 15.12.2005
Letzte Änderung 15.12.2005

6.17 hapticgraphclasses/DragObjectHandler.h Dateireferenz

```
#include <HL/hl.h>
#include <HDU/hduMatrix.h>
#include "HapticAction.h"
#include "HapticObject.h"
```

Klassen

- class **DragObjectHandler**

  Eine Eventhandlerklasse die es ermöglicht, haptische Objekte mit dem Phantom zu bewegen.

6.17.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 15.12.2005
Letzte Änderung 15.12.2005

6.18 hapticgraphclasses/DragSceneHandler.cpp Dateireferenz

```
#include "DragSceneHandler.h"
```

6.18.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 19.01.2006
Letzte Änderung 28.01.2006

6.19 hapticgraphclasses/DragSceneHandler.h Dateireferenz

```
#include <HL/hl.h>
#include <HDU/hduMatrix.h>
#include "HapticAction.h"
#include "GraphScene.h"
```

Klassen

- class **DragSceneHandler**

  Eine Eventhandlerklasse die es ermöglicht, die gesamte Szene (Kameraführung) mit dem Phantom zu bewegen.

6.19.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 19.01.2006
Letzte Änderung 04.02.2006

6.20 hapticgraphclasses/Edge.cpp Dateireferenz

```
#include "Edge.h"
#include "GraphScene.h"
```

6.20.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 27.01.2006
Letzte Änderung 27.01.2006

6.21 hapticgraphclasses/Edge.h Dateireferenz

```
#include <windows.h>
#include <GL/gl.h>
#include "HapticObject.h"
#include "Utilities.h"
```

Klassen

- class **Edge**

  Haptisches Objekt, das eine Kante in einem Graphen darstellt.

6.21.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 27.01.2006
Letzte Änderung 04.02.2006

6.22 hapticgraphclasses/FrictionForceEffect.cpp Dateireferenz

```
#include "FrictionForceEffect.h"
```

6.22.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 06.01.2006
Letzte Änderung 06.01.2006

6.23 hapticgraphclasses/FrictionForceEffect.h Dateireferenz

```
#include "HapticEffect.h"
```

Klassen

- class **FrictionForceEffect**

  Klasse zur Ausgabe einer "ambienten" Reibungskraft auf dem Phantom Device.

6.23.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 06.01.2006
Letzte Änderung 04.02.2006

6.24 hapticgraphclasses/GraphScene.cpp Dateireferenz

```
#include <HL/hl.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include "GraphScene.h"
#include "Node.h"
#include "DragObjectHandler.h"
#include "DragNodeOnGridHandler.h"
#include "DragSceneHandler.h"
#include "Edge.h"
```

6.24.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 04.02.2006

6.25 hapticgraphclasses/GraphScene.h Dateireferenz

```
#include <vector>
#include "HapticObject.h"
#include "Camera.h"
#include "Node.h"
#include "Grid.h"
#include "Utilities.h"
#include "HapticAction.h"
#include "../businesslogic/IBusinessAdapter.h"
```

Klassen

- class **GraphScene**

  Klasse, die alle haptischen Objekte der Scene verwaltet.

6.25.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 04.02.2006

6.26 hapticgraphclasses/Grid.cpp Dateireferenz

```
#include <windows.h>
#include <GL/glut.h>
#include <math.h>
#include "Grid.h"
#include "GraphScene.h"
```

6.26.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 17.01.2006
Letzte Änderung 26.01.2006

6.27 hapticgraphclasses/Grid.h Dateireferenz

```
#include <windows.h>
#include <GL/gl.h>
#include "HapticObject.h"
#include "Utilities.h"
```

Klassen

- class **Grid**

  Klasse, die Gitterraster darstellt, auf dem die Elemente eines Graphen Angeordnet werden können. Die **Position**(S. 75) *des Grid wird durch die linke untere Ecke festgelegt.*

6.27.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 17.01.2006
Letzte Änderung 04.02.2006

6.28 hapticgraphclasses/HapticAction.h Dateireferenz

```
#include <HL/hl.h>
```

Klassen

- class **IHapticAction**

  Schnittstelle für haptische Eventhandler.

6.28.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 15.12.2005
Letzte Änderung 27.12.2005

6.29 hapticgraphclasses/HapticConstraint.cpp Dateireferenz

```
#include "HapticConstraint.h"
#include "HapticObject.h"
```

6.29.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 02.01.2006
Letzte Änderung 23.01.2006

6.30 hapticgraphclasses/HapticConstraint.h Dateireferenz

```
#include <HL/hl.h>
```

Klassen

- class **HapticConstraint**

  Klasse, die ein HLAPI-Constraint zu einem **HapticObject**(S. 56) *darstellen kann.*

6.30.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 02.01.2006
Letzte Änderung 08.01.2006

6.31 hapticgraphclasses/HapticCursor.cpp Dateireferenz

```
#include <HL/hl.h>
#include <HLU/hlu.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include "HapticCursor.h"
```

6.31.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 27.12.2005

6.32 hapticgraphclasses/HapticCursor.h Dateireferenz

Klassen

- class **HapticCursor**

  Kapselt ein Zeige-Widget, das den Bewegungen des Phantom-Proxy folgt.

Makrodefinitionen

- #define **DEFAULT_CURSOR_SIZE_PIX** 20

  Standardgröße des HapticCursors.

6.32.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 05.02.2006

6.33 hapticgraphclasses/HapticDevice.cpp Dateireferenz

```
#include <HLU/hlu.h>
#include <GL/gl.h>
#include "HapticDevice.h"
#include "..\exceptionclasses\HapticsExceptions.h"
```

6.33.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 27.12.2005

6.34 hapticgraphclasses/HapticDevice.h Dateireferenz

```
#include <HL/hl.h>
```

Klassen

- class **HapticDevice**

  Eine Klasse zur Verwaltung des Haptischen Gerätes.

6.34.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 27.12.2005

6.35 hapticgraphclasses/HapticEffect.cpp Dateireferenz

```
#include "HapticEffect.h"
```

6.35.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 03.01.2006
Letzte Änderung 30.01.2006

6.36 hapticgraphclasses/HapticEffect.h Dateireferenz

```
#include <HL/hl.h>
```

Klassen

- class **HapticEffect**

  Abstrakte Klasse zur Kapselung von HLAPI Force Effects.

6.36.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 03.01.2006
Letzte Änderung 05.02.2006

6.37 hapticgraphclasses/HapticObject.cpp Dateireferenz

```
#include <HL/hl.h>
#include <GL/gl.h>
#include "HapticObject.h"
```

6.37.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 03.12.2005
Letzte Änderung 26.01.2006

6.38 hapticgraphclasses/HapticObject.h Dateireferenz

```
#include <HL/hl.h>
#include <HDU/hduMatrix.h>
#include <vector>
#include "HapticAction.h"
#include "HapticConstraint.h"
#include "Utilities.h"
```

Klassen

- class **HapticObject**

  Basisklasse aller haptischen Objekte. Stellt grundlegende Funktionalität zum Fühlbarmachen und Bewegen von Objekten zur Verfügung. Kümmert sich Um die Registrierung von Eventhandlern.

6.38.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 03.12.2005
Letzte Änderung 04.02.2006

6.39 hapticgraphclasses/Node.cpp Dateireferenz

```
#include <HL/hl.h>
#include "Node.h"
#include "GraphScene.h"
#include "FrictionForceEffect.h"
#include "SpringForceEffect.h"
```

6.39.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 30.12.2005
Letzte Änderung 28.01.2006

6.40 hapticgraphclasses/Node.h Dateireferenz

```
#include <windows.h>
#include <GL/gl.h>
#include <list>
#include "HapticObject.h"
#include "Edge.h"
#include "Utilities.h"
#include "HapticEffect.h"
#include "../businesslogic/IObserver.h"
#include "../businesslogic/IBusinessAdapter.h"
```

Klassen

- class **Node**

  Haptisches Objekt, das einen Knoten in einem Graphen als Rechteck darstellt.

6.40.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 30.12.2005
Letzte Änderung 05.02.2006

6.41 hapticgraphclasses/SpringForceEffect.cpp Dateireferenz

```
#include "SpringForceEffect.h"
```

6.41.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 28.01.2006
Letzte Änderung 28.01.2006

6.42 hapticgraphclasses/SpringForceEffect.h Dateireferenz

```
#include <HDU/hduVector.h>
#include "HapticEffect.h"
#include "Utilities.h"
```

Klassen

- class **SpringForceEffect**

  Klasse zur Ausgabe einer Federkraft auf dem Phantom Device.

6.42.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 28.01.2006
Letzte Änderung 05.02.2006

6.43 hapticgraphclasses/Utilities.cpp Dateireferenz

```
#include "Utilities.h"
```

```
#include <GL/glut.h>
```

6.43.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 03.01.2006
Letzte Änderung 28.01.2006

6.44 hapticgraphclasses/Utilities.h Dateireferenz

Klassen

- struct **Position**

  Stellt einen Punkt im 3D-Raum dar.

- class **UnitConversionInfo**

  Klasse, die die Umwandlung zwischen View- und Businesskoordinaten übernimmt.

- class **GlutString**

  Einfache Klasse um mit glut einen Text auszugeben.

6.44.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 03.01.2006
Letzte Änderung 05.02.2006

6.45 hapticgraphclasses/ViscousForceEffect.cpp Dateireferenz

```
#include "ViscousForceEffect.h"
```

6.45.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 08.01.2006
Letzte Änderung 08.01.2006

6.46 hapticgraphclasses/ViscousForceEffect.h Dateireferenz

```
#include "HapticEffect.h"
```

Klassen

- class **ViscousForceEffect**

  Klasse zur Ausgabe einer "ambienten" Viskostät auf dem Phantom Device.

6.46.1 Ausführliche Beschreibung

Autor:
Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 08.01.2006
Letzte Änderung 05.02.2006

6.47 Main.cpp Dateireferenz

```
#include <math.h>
#include <assert.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include <HL/hl.h>
#include <HDU/hduMatrix.h>
#include <HDU/hduError.h>
#include <HLU/hlu.h>
#include "hapticgraphclasses/HapticDevice.h"
#include "hapticgraphclasses/GraphScene.h"
#include "hapticgraphclasses/HapticCursor.h"
#include "exceptionclasses/HapticsExceptions.h"
#include "businesslogic/AppConfiguration.h"
#include "businesslogic/BusinessTask.h"
```

Funktionen

- void **glutDisplay** (void)
- void **glutReshape** (int width, int height)
- void **glutIdle** (void)
- void **exitHandler** (void)
- void **initGL** ()
- void **initHL** ()
- void **initScene** (int viewportWidth, int viewportHeight)
- void **setAppData** ()

  Initialisiert Konfigurationsdaten Objekt CA.

- int **main** (int argc, char *argv[])

Variablen

- **HapticDevice** * **pHapticDevice** = NULL
- **GraphScene** * **pScene** = NULL
- **HapticCursor cursor**
- **AppConfiguration appData**

  Objekt mit globale Konfigurations Daten CA.

6.47.1 Ausführliche Beschreibung

Autor:

angepasst von Katharina Greiner, Matr.-Nr. 943471

Datum:
Erstellt am 26.12.2005
Letzte Änderung 28.01.2006 (CA)

6.47.2 Dokumentation der Funktionen

6.47.2.1 void setAppData ()

Initialisiert Konfigurationsdaten Objekt CA.

Debuginfo zur Ausgabe aller Tasks

6.47.3 Variablen-Dokumentation

6.47.3.1 AppConfiguration appData

Objekt mit globale Konfigurations Daten CA.

Autor:
Carsten Arnold

Kapitel 7

Phantom Graph Demo Zusätzliche Informationen

7.1 Liste der zu erledigenden Dinge

Klasse DragObjectHandler(S. 29) Objekte lassen sich noch nicht nach hinten (in neg. z-Richtung) verschieben.

Klasse GlutString(S. 41) Textausgabe funktioniert noch nicht.

Element GlutString::write(S. 41)**(char *string, Position**(S. 75) **pos)** Funktioniert noch nicht.

Element GraphScene::createObjects(S. 44)**(IBusinessAdapter**(S. 61) ***businessObj, Grid**(S. 46) ***pGrid)** Bisher kann mit dieser Methode nur ein Baum erzeugt werden, nicht jeder beliebige Graph.

Klasse IBusinessAdapter(S. 61) Definition der abstrakten Adapter Methoden

Klasse Node(S. 68) Der Effekt für "gar nicht bewegbar" muss noch richtig eingestellt werden.

Klasse UnitConversionInfo(S. 80) Die Umrechnung erfolgt bisher mit festen Dummy-Werten. Die dynamische Berechnung der Einheiten aus der einstellbaren Anzahl der Einheiten pro Bildschirm muss noch implementiert werden.

Element UnitConversionInfo::UnitConversionInfo(S. 81)**(int upsHorz, int upsVert, float paddingHorz,** Die dynamische Berechnung der Einheiten aus der Anzahl der Einheiten pro Bildschirm muss noch implementiert werden.

7.2 Liste der bekannten Fehler

Klasse HapticEffect(S. 54) Manchmal wird zwar startEffect() aufgerufen, der Effekt aber nicht gerendert. Das gleiche gilt für stopEffect(): Manchmal wird der Effekt nicht mehr angehalten. Ursache ist bisher ungekärt.

Index