Holum

1.0

Generated by Doxygen 1.8.11

Mon May 2 2016 23:02:55

Contents

Chapter 1

Holum

Obiettivi del progetto

Holum è un progetto che focalizza i suoi obiettivi su usabilità, intuitività ed efficienza. È progettato per utenti di ogni età e possiede una varietà di caratteristiche che possono essere utilizzate per soddisfare una vasta gamma di interessi ed esigenze. Si rivolge sia a utenti non specializzati o solo in parte avvertiti delle attività di programmazione informatica, sia a vari tipi di utenti specializzati, senza precludere nessuna tipologia.

L'obiettivo primario è quindi quello di sviluppare un software in grado di offrire varie funzionalità, tra cui la possibilità di **riproduzione multimediale** di filmati, sia pre-caricati appositamente dagli sviluppatori, che tramite la scelta dell'utente. Inoltre, l'utilizzatore potrà caricare e visualizzare **modelli tridimensionali**, con la possibilità di ingrandire, ruotare e traslare il modello in questione, tramite alcuni sistemi di interazione. Un'altra funzionalità riguarda la creazione di un insieme di **videogiochi**, realizzati con lo scopo di intrattenimento e svago dell'utente.

Ogni funzionalità del programma dovrà essere personalizzabile tramite la selezione di impostazioni adeguate alle preferenze e alle esigenze dell'utente. L'insieme di questi sottoprogrammi verrà poi raccolto e reso accessibile in un apposito **menù**.

L'interazione con il software dovrà avvenire tramite alcuni dispositivi in grado di raccogliere informazioni sulle scelte dell'utente, per poi tradurle in un linguaggio comprensibile dal calcolatore. Queste informazioni verrano quindi utilizzate per navigare all'interno del menù e del programma in generale.

Il software avrà anche il principale compito di suddividere l'immagine da visualizzare in 4 "parti" uguali, a due a due simmetriche e speculari. Il software dovrà poi essere accompagnato da un'appropriato **sistema piramidale**, sostenuto da una struttura di supporto. Questa struttura dovrà sorreggere un monitor, il quale si occuperà di visualizzare l'immagine prodotta dal software. Infine, la superficie riflettente della piramide, assieme alla suddivisione dell'immagine da visualizzare, dovrà creare il desiderato **effetto di tridimensionalità e profondità della proiezione nello spazio**.

Specifiche tecniche

L'effetto utilizzato per la riproduzione delle immagini è conosciuto come **"Pepper's Ghost"**, ovvero il "Fantasma di Pepper": una tecnica illusoria utilizzata in teatro e in vari trucchi magici che consiste nell'utilizzo di una o più lastre di vetro, posizionate ad un angolo di 45 tra lo spettatore e la scena, unite a una particolare illuminazione, per dare l'illusione della rappresentazione nello spazio delle immagini proiettate. Il contenuto che verrà proiettato sarà riprodotto da un software appositamente programmato, utilizzando il linguaggio C++ e le librerie **SFML, sfeMovie, OpenGL, assimp** e altre.

L'utente può quindi, come accennato sopra, interagire con il programma in diversi modi:

- **Utility Android**, sviluppata in Java, attualmente strutturata per la sola navigazione all'interno del software su PC, tramite connessione bluetooth;
- **Myo Armband** (https://www.myo.com), un braccialetto "smart" che tramite la la lettura dei movimenti muscolari permette il controllo wireless del software, utilizzando appositi gesti e movimenti. Per realizzare il collegamento con il programma in C++ abbiamo utilizzato la libreria myo, personalizzandone alcune funzioni per adattarla alle nostre esigenze;
- **Leap Motion** (https://www.leapmotion.com), un dispositivo in grado di rilevare accuratamente il movimento di dita e mani dell'utente nello spazio sovrastante al dispositivo stesso, permettendo l'interazione con il software in C++ tramite l'apposito SDK offerto da Leap Motion.

Il programma offrirà la riproduzione di una serie di mini-giochi, pensati opportunamente per sfruttare al meglio le potenzialità dei dispositivi in questione. Tramite questi controller è quindi possibile spostarsi all'interno dei menù e del programma, il quale offre varie categorie di intrattenimento.

La schermata iniziale, successiva allo splash screen di benvenuto, presenta il menù, suddiviso nelle seguenti sezioni:

- **Video**: Archivio di filmati pensati appositamente per la rappresentazione nella piramide. I video pre-caricati sono caratterizzati da uno sfondo nero e da una riproduzione tridimensionale dei caratteri raffigurati, per garantirne l'ottimale visualizzazione e favorire l'illusione della raffigurazione di un simil-ologramma;
- **Games**: Sezione dedicata ai giochi, sia inerenti al Leap Motion che non. I giochi presenti potranno essere selezionati dall'utente tramite un menù, simile a quello principale;
- **3D**: Sezione dedicata al caricamento e alla rappresentazione di un archivio di modelli 3D in formato OBJ, tramite le funzionalità offerte da OpenGL. L'utente potrà interagire con il modello tramite un controller a sua scelta;
- **Settings**: Gestione delle preferenze utente riguardanti l'interfaccia grafica, audio e dei controller.

Ruoli e responsabilità

- **Eusebio Apopei** (Costruttore, Programmatore): Ha il ruolo di realizzare, sia in fase prototipale che definitiva, la struttura piramidale e quella reggente. Si occuperà anche di svolgere il compito di programmatore di parte del software, tra cui l'utility per sistema operativo Android.
- **Davide Cosentino** (Grafico, Programmatore): Ha il ruolo di realizzare l'interfaccia grafica del programma e di sviluppare una parte del software, tra cui l'utility per sistema operativo Android.
- **Alessio Falai** (Project Manager, Sviluppatore Software): Ha il ruolo di coordinare tutte le attività del progetto e di supervisionare il corretto svolgimento dei lavori. Si occuperà anche di svolgere il compito di analista e programmatore di parte del software e di interfacciare i dispositivi di HID (Human Interface Device) con il software generale.
- **Lorenzo Pratesi** (Sviluppatore Software): Svolge il principale compito di analista e programmatore software del progetto. Si occuperà anche di interfacciare i dispositivi di HID (Human Interface Device) con il software generale.
- **Vittorio Sodini** (Grafico, Programmatore): Ha il ruolo di realizzare l'interfaccia grafica del programma e di sviluppare una parte del software, tra cui l'utility per sistema operativo Android.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 sh Namespace Reference

It is used to distinguish Holum shaders from SFML ones.

Data Structures

- class Shader

  The Shader class describes the little programs, which rest on the GPU, used in the rendering operation next to an OpenGL drawing command execution.

Chapter 7

Data Structure Documentation

7.1 Bluetooth Class Reference

The Bluetooth class provides wireless communication between this software and the Android application.

```
#include <Bluetooth.h>
```

Public Member Functions

- Bluetooth ()
- Bluetooth (const Bluetooth &b)
- bool acceptSocket ()
- bool bindSocket ()
- bool checkMessage ()
- void closeSocket ()
- int getDirection ()
- bool init ()
- bool isAvailable ()
- void isAvailable (bool available)
- bool listenSocket ()
- void manageBluetooth ()
- int readMessage ()
- bool registerService ()
- void startCommunication ()

Private Attributes

- atomic_bool available
- Clock clock
- Clock clock2
- atomic_int direction
- bool exit
- Time time

7.1.1 Constructor & Destructor Documentation

7.1.1.1 Bluetooth::Bluetooth ( )

7.1.1.2 Bluetooth::Bluetooth (const Bluetooth & *b*)

7.1.2 Member Function Documentation

7.1.2.1 bool Bluetooth::acceptSocket ( )

7.1.2.2 bool Bluetooth::bindSocket ( )

7.1.2.3 bool Bluetooth::checkMessage ( )

7.1.2.4 void Bluetooth::closeSocket ( )

7.1.2.5 int Bluetooth::getDirection ( )

7.1.2.6 bool Bluetooth::init ( )

7.1.2.7 bool Bluetooth::isAvailable ( )

7.1.2.8 void Bluetooth::isAvailable (bool *available*)

7.1.2.9 bool Bluetooth::listenSocket ( )

7.1.2.10 void Bluetooth::manageBluetooth ( )

7.1.2.11 int Bluetooth::readMessage ( )

7.1.2.12 bool Bluetooth::registerService ( )

7.1.2.13 void Bluetooth::startCommunication ( )

7.1.3 Field Documentation

7.1.3.1 atomic_bool Bluetooth::available [private]

7.1.3.2 Clock Bluetooth::clock [private]

7.1.3.3 Clock Bluetooth::clock2 [private]

7.1.3.4 atomic_int Bluetooth::direction [private]

7.1.3.5 bool Bluetooth::exit [private]

7.1.3.6 Time Bluetooth::time [private]

The documentation for this class was generated from the following file:

- Bluetooth.h

7.2 File Class Reference

The File class describes resources relative to the Video and ThreeD sections.

`#include <File.h>`

Public Member Functions

- File ()
- File (string path, string title)
- string getPath ()
- RectangleShape * getThumbnail ()
- string getThumbnailPath ()
- Vector2f getThumbnailPosition ()
- Vector2f getThumbnailSize ()
- Texture * getThumbnailTexture ()
- string getTitle ()
- void moveThumbnail (float x, float y)
- void setPath (string path)
- void setThumbnailPath (string thumbnailPath)
- void setThumbnailPosition (float x, float y)
- void setThumbnailScale (float x, float y)
- void setThumbnailSize (float x, float y)
- void setTitle (string title)

Private Member Functions

- void init ()

Private Attributes

- string path
- float rectHeight
- float rectWidth
- RectangleShape thumbnail
- string thumbnailPath
- Texture thumbnailTexture
- string title

7.2.1 Constructor & Destructor Documentation

7.2.1.1 File::File ( )

Constructs a File object, without setting any parameter.

7.2.1.2 File::File (string *path*, string *title*)

Constructs a File object, by setting the path and title variables to the given values.

7.2.2 Member Function Documentation

7.2.2.1 string File::getPath ( )

Returns the file path.

7.2.2.2 RectangleShape* File::getThumbnail ( )

Returns the thumbnail object.

7.2.2.3 string File::getThumbnailPath ( )

Returns the thumbnail's path.

7.2.2.4 Vector2f File::getThumbnailPosition ( )

Returns the thumbnail's current position.

7.2.2.5 Vector2f File::getThumbnailSize ( )

Returns the thumbnail's size.

7.2.2.6 Texture* File::getThumbnailTexture ( )

Returns the thumbnail's texture object.

7.2.2.7 string File::getTitle ( )

Returns the image's name.

7.2.2.8 void File::init ( ) [private]

Called by the constructor, it is used to load a specific thumbnail, using the constructor's parameters.

7.2.2.9 void File::moveThumbnail (float *x,* float *y*)

Sets the thumbnail's position, by adding the given offset to the current position.

7.2.2.10 void File::setPath (string *path*)

Sets the file path.

7.2.2.11 void File::setThumbnailPath (string *thumbnailPath*)

Sets the thumbnail's path.

7.2.2.12 void File::setThumbnailPosition (float *x*, float *y*)

Sets the thumbnail's position.

7.2.2.13 void File::setThumbnailScale (float *x*, float *y*)

Sets the thumbnail's scale factor.

7.2.2.14 void File::setThumbnailSize (float *x*, float *y*)

Sets the thumbnail's size.

7.2.2.15 void File::setTitle (string *title*)

Sets the image's name.

7.2.3 Field Documentation

7.2.3.1 string File::path [private]

This path is relative to the file which is being processed, like an OBJ or an MP4.

7.2.3.2 float File::rectHeight [private]

The thumbnail's rectangle height, measured in pixels.

Default = 720px.

7.2.3.3 float File::rectWidth [private]

The thumbnail's rectangle width, measured in pixels.

Default = 480px.

7.2.3.4 RectangleShape File::thumbnail [private]

The actual thumbnail's rectangle, whose perimeter is rectWidth $*$ rectHeight, drawn using SFML functions.

7.2.3.5 string File::thumbnailPath [private]

The thumbnail's absolute path.

7.2.3.6 Texture File::thumbnailTexture [private]

The actual thumbnail's texture, which contains the image specified in the title variable.

7.2.3.7 string File::title [private]

The name of the image file used to identify a particular resource.

The documentation for this class was generated from the following file:

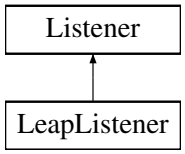
- File.h

7.3 LeapListener Class Reference

The LeapListener class provides data transfers between this software and the Leap Motion device.

`#include <LeapListener.h>`

Inheritance diagram for LeapListener:



Public Member Functions

- Leap::Vector getHandDirection ()
- Leap::HandList getHandsList ()
- Leap::Vector getLeapTranslation ()
- Leap::Vector getPalmNormal ()
- virtual void onConnect (const Leap::Controller &)
- virtual void onDisconnect (const Leap::Controller &)
- virtual void onExit (const Leap::Controller &)
- virtual void onFrame (const Leap::Controller &)
- virtual void onInit (const Leap::Controller &)
- virtual void onServiceConnect (const Leap::Controller &)
- virtual void onServiceDisconnect (const Leap::Controller &)

Private Attributes

- Leap::Arm arm
- Leap::Frame currentFrame
- Leap::FingerList fingers
- Leap::GestureList gestures
- Leap::Vector handDirection
- Leap::HandList hands
- Leap::Vector leapTranslation
- Leap::Vector palmNormal
- Leap::Frame previousFrame

7.3.1 Detailed Description

To handle Leap Motion events, create an instance of a Listener subclass and assign it to the Controller instance. The Controller calls the relevant Listener callback function when an event occurs, passing in a reference to itself. You do not have to implement callbacks for events you do not want to handle.

The Controller object calls these Listener functions from a thread created by the Leap Motion library, not the thread used to create or set the Listener instance.

7.3.2 Member Function Documentation

7.3.2.1 Leap::Vector LeapListener::getHandDirection ( )

Returns the direction from the palm position toward the fingers.

The direction is expressed as a unit vector pointing in the same direction as the directed line from the palm position to the fingers.

You can use the palm direction vector to compute the pitch and and yaw angles of the palm with respect to the horizontal plane.

7.3.2.2 Leap::HandList LeapListener::getHandsList ( )

Returns the list of Hand objects detected in the frame.

The HandList class acts like a vector-style array and supports iterators. You cannot remove or alter the member objects of a hand lists received from the Leap Motion API, but you can combine lists of the same object type.

The HandList class defines additional functions for getting a member of the list based on its relative position within the Leap coordinate system. These functions include leftmost(), rightmost(), and frontmost().

7.3.2.3 Leap::Vector LeapListener::getLeapTranslation ( )

Returns the change of position derived from the overall linear motion between the current frame and the specified frame. The returned translation vector provides the magnitude and direction of the movement in millimeters.

The Leap Motion software derives frame translation from the linear motion of all objects detected in the field of view.

If either the current or the previous frame is an invalid Frame object, then this method returns a zero vector.

7.3.2.4 Leap::Vector LeapListener::getPalmNormal ( )

Returns the normal vector to the palm.

If your hand is flat, this vector will point downward, or “out” of the front surface of your palm.

The direction is expressed as a unit vector pointing in the same direction as the palm normal (that is, a vector orthogonal to the palm).

You can use the palm normal vector to compute the roll angle of the palm with respect to the horizontal plane.

7.3.2.5 virtual void LeapListener::onConnect (const Leap::Controller &) [virtual]

Called when the Controller object connects to the Leap Motion software and the Leap Motion hardware device is plugged in, or when this Listener object is added to a Controller that is already connected.

7.3.2.6 virtual void LeapListener::onDisconnect (const Leap::Controller &) [virtual]

Called when the Controller object disconnects from the Leap Motion software or the Leap Motion hardware is unplugged.

The controller can disconnect when the Leap Motion device is unplugged, the user shuts the Leap Motion software down, or the Leap Motion software encounters an unrecoverable error.

7.3.2.7 virtual void LeapListener::onExit (const Leap::Controller &) [virtual]

Called when this Listener object is removed from the Controller or the Controller instance is destroyed.

7.3.2.8 virtual void LeapListener::onFrame (const Leap::Controller &) [virtual]

Called when a new frame of hand and finger tracking data is available. Access the new frame data using the Controller::frame() function.

Note, the Controller skips any pending onFrame events while your onFrame handler executes. If your implementation takes too long to return, one or more frames can be skipped. The Controller still inserts the skipped frames into the frame history. You can access recent frames by setting the history parameter when calling the Controller↩ ::frame() function. You can determine if any pending onFrame events were skipped by comparing the ID of the most recent frame with the ID of the last received frame.

7.3.2.9 virtual void LeapListener::onInit (const Leap::Controller &) [virtual]

Called once, when this Listener object is newly added to a Controller.

7.3.2.10 virtual void LeapListener::onServiceConnect (const Leap::Controller &) [virtual]

Called when the Leap Motion daemon/service connects to your application Controller.

7.3.2.11 virtual void LeapListener::onServiceDisconnect (const Leap::Controller &) [virtual]

Called if the Leap Motion daemon/service disconnects from your application Controller.

Normally, this callback is not invoked. It is only called if some external event or problem shuts down the service or otherwise interrupts the connection.

7.3.3 Field Documentation

7.3.3.1 Leap::Arm LeapListener::arm [private]

The valid Arm object, which represents the forearm, obtained from a Hand object.

7.3.3.2 Leap::Frame LeapListener::currentFrame [private]

The newest frame of tracking data from the Leap Motion software.

7.3.3.3 Leap::FingerList LeapListener::fingers [private]

The FingerList containing all the tracked fingers in the current frame.

7.3.3.4 Leap::GestureList LeapListener::gestures [private]

The GestureList containing all gestures that have occurred since the previous frame.

7.3.3.5 Leap::Vector LeapListener::handDirection [private]

The direction from the palm position toward the fingers.

7.3.3.6 Leap::HandList LeapListener::hands [private]

The list of Hand objects detected in the frame.

7.3.3.7 Leap::Vector LeapListener::leapTranslation [private]

The vector of translations occurred since the previous frame.

7.3.3.8 Leap::Vector LeapListener::palmNormal [private]

The normal vector to the palm.

7.3.3.9 Leap::Frame LeapListener::previousFrame [private]

The previous frame of tracking data from the Leap Motion software, retrieved in the stored frames by the controller.

The documentation for this class was generated from the following file:

- LeapListener.h

7.4 Manager Class Reference

```
#include <Manager.h>
```

Public Member Functions

- Manager ()

Private Member Functions

- void changeStatus ()
- void checkErrors ()
- void drawGL ()
- void drawObjects (vector< Drawable * > toDraw)
- void drawOn (vector< Drawable * > toDraw)
- void init ()
- void initLeap ()
- void initMyo ()
- mat4 leapTransform (mat4 modelMatrix)
- void manageBluetooth ()
- void manageGames ()
- void manageMenu ()
- void manageMyo ()
- void manageSettings ()
- void manageThreeD ()
- void manageVideos ()
- void playVideo (sfe::Movie *movie)
- void run ()
- void splashScreen ()
- void windowEvents ()

Private Attributes

- float angleX
- float angleY
- float angleZ
- Bluetooth bluetooth
- Clock cDeb
- MANAGER_STATUS currentStatus
- bool drawWithGL
- bool enterPressed
- bool escapePressed
- bool firstMyoPose
- bool fullscreen
- float height3D
- Hub * hub
- Leap::Controller leapController
- LeapListener leapListener
- bool loadCheck
- Menu menu
- Myo * myoArmband
- MyoConnector myoConnector
- string myoCurrentPose
- vec3 myoDirections
- string myoLastPose
- thread myoManager
- Settings settings
- Clock tDeb
- ThreeD threeD
- Video video
- float VIEW_DIMENSION
- float VIEW_DIMENSION_X
- float VIEW_DIMENSION_Y
- float VIEW_POSITION_BOTTOM_X
- float VIEW_POSITION_BOTTOM_Y
- float VIEW_POSITION_LEFT_X
- float VIEW_POSITION_LEFT_Y
- float VIEW_POSITION_RIGHT_X
- float VIEW_POSITION_RIGHT_Y
- float VIEW_POSITION_TOP_X
- float VIEW_POSITION_TOP_Y
- View viewBottom
- float viewHeight
- View viewLeft
- View viewRight
- View viewTop
- float viewWidth
- float width3D
- RenderWindow * window
- float zoom

7.4.1 Constructor & Destructor Documentation

7.4.1.1 Manager::Manager ( )

7.4.2 Member Function Documentation

7.4.2.1 void Manager::changeStatus ( ) [private]

7.4.2.2 void Manager::checkErrors ( ) [private]

7.4.2.3 void Manager::drawGL ( ) [private]

7.4.2.4 void Manager::drawObjects (vector< Drawable * > *toDraw*) [private]

7.4.2.5 void Manager::drawOn (vector< Drawable * > *toDraw*) [private]

7.4.2.6 void Manager::init ( ) [private]

7.4.2.7 void Manager::initLeap ( ) [private]

7.4.2.8 void Manager::initMyo ( ) [private]

7.4.2.9 mat4 Manager::leapTransform (mat4 *modelMatrix*) [private]

7.4.2.10 void Manager::manageBluetooth ( ) [private]

7.4.2.11 void Manager::manageGames ( ) [private]

7.4.2.12 void Manager::manageMenu ( ) [private]

7.4.2.13 void Manager::manageMyo ( ) [private]

7.4.2.14 void Manager::manageSettings ( ) [private]

7.4.2.15 void Manager::manageThreeD ( ) [private]

7.4.2.16 void Manager::manageVideos ( ) [private]

7.4.2.17 void Manager::playVideo (sfe::Movie * *movie*) [private]

7.4.2.18 void Manager::run ( ) [private]

7.4.2.19 void Manager::splashScreen ( ) [private]

7.4.2.20 void Manager::windowEvents ( ) [private]

7.4.3 Field Documentation

7.4.3.1 float Manager::angleX [private]

7.4.3.2 float Manager::angleY [private]

7.4.3.3 float Manager::angleZ [private]

7.4.3.4 Bluetooth Manager::bluetooth [private]

7.4.3.5 Clock Manager::cDeb [private]

7.4.3.6 MANAGER_STATUS Manager::currentStatus [private]

7.4.3.7 bool Manager::drawWithGL [private]

7.4.3.8 bool Manager::enterPressed [private]

7.4.3.9 bool Manager::escapePressed [private]

7.4.3.10 bool Manager::firstMyoPose [private]

7.4.3.11 bool Manager::fullscreen [private]

7.4.3.12 float Manager::height3D [private]

7.4.3.13 Hub* Manager::hub [private]

7.4.3.14 Leap::Controller Manager::leapController [private]

7.4.3.15 LeapListener Manager::leapListener [private]

7.4.3.16 bool Manager::loadCheck [private]

7.4.3.17 Menu Manager::menu [private]

7.4.3.18 Myo* Manager::myoArmband [private]

7.4.3.19 MyoConnector Manager::myoConnector [private]

7.4.3.20 string Manager::myoCurrentPose [private]

7.4.3.21 vec3 Manager::myoDirections [private]

7.4.3.22 string Manager::myoLastPose [private]

7.4.3.23 thread Manager::myoManager [private]

7.4.3.24 Settings Manager::settings [private]

7.4.3.25 Clock Manager::tDeb [private]

7.4.3.26 ThreeD Manager::threeD [private]

7.4.3.27 Video Manager::video [private]

7.4.3.28 float Manager::VIEW_DIMENSION [private]

7.4.3.29 float Manager::VIEW_DIMENSION_X [private]

7.4.3.30 float Manager::VIEW_DIMENSION_Y [private]

7.4.3.31 float Manager::VIEW_POSITION_BOTTOM_X [private]

7.4.3.32 float Manager::VIEW_POSITION_BOTTOM_Y [private]

7.4.3.33 float Manager::VIEW_POSITION_LEFT_X [private]

7.4.3.34 float Manager::VIEW_POSITION_LEFT_Y [private]

7.4.3.35 float Manager::VIEW_POSITION_RIGHT_X [private]

7.4.3.36 float Manager::VIEW_POSITION_RIGHT_Y [private]

7.4.3.37 float Manager::VIEW_POSITION_TOP_X [private]

7.4.3.38 float Manager::VIEW_POSITION_TOP_Y [private]

7.4.3.39 View Manager::viewBottom [private]

7.4.3.40 float Manager::viewHeight [private]

7.4.3.41 View Manager::viewLeft [private]

7.4.3.42 View Manager::viewRight [private]

7.4.3.43 View Manager::viewTop [private]

7.4.3.44 float Manager::viewWidth [private]

7.4.3.45 float Manager::width3D [private]

7.4.3.46 RenderWindow* Manager::window [private]

7.4.3.47 float Manager::zoom [private]

The documentation for this class was generated from the following file:

- Manager.h

7.5 Menu Class Reference

`#include <Menu.h>`

Public Member Functions

- Menu ()
- void checkPositions ()
- MANAGER_STATUS getCurrentStatus ()
- bool getDownAnimation ()
- bool getLeftAnimation ()
- vector< Drawable * > getObjectsVector ()
- bool getRightAnimation ()
- bool getUpAnimation ()
- void menuEvents ()
- void setDownAnimation (bool downAnimation)
- void setLeftAnimation (bool leftAnimation)
- void setRightAnimation (bool rightAnimation)
- void setUpAnimation (bool upAnimation)

Private Member Functions

- void animateDown ()
- void animateLeft ()
- void animateRight ()
- void animateUp ()
- void init ()
- void setPositions ()

Private Attributes

- float animationSpeed
- float animationTime
- bool downAnimation
- bool first
- int firstTextPosition
- bool leftAnimation
- int leftPosition
- Font menuFont
- vector< Text > menuTexts
- int nText
- int outPosition
- bool rightAnimation
- int rightPosition
- float scaleFactor
- vector< float > speeds
- int stepCounter
- RectangleShape strip
- unsigned int textSize
- vector< Drawable * > toDraw
- bool upAnimation

7.5.1 Constructor & Destructor Documentation

7.5.1.1 Menu::Menu ( )

7.5.2 Member Function Documentation

7.5.2.1 void Menu::animateDown ( ) [private]

7.5.2.2 void Menu::animateLeft ( ) [private]

7.5.2.3 void Menu::animateRight ( ) [private]

7.5.2.4 void Menu::animateUp ( ) [private]

7.5.2.5 void Menu::checkPositions ( )

7.5.2.6 MANAGER_STATUS Menu::getCurrentStatus ( )

7.5.2.7 bool Menu::getDownAnimation ( )

7.5.2.8 bool Menu::getLeftAnimation ( )

7.5.2.9 vector<Drawable*> Menu::getObjectsVector ( )

7.5.2.10 bool Menu::getRightAnimation ( )

7.5.2.11 bool Menu::getUpAnimation ( )

7.5.2.12 void Menu::init ( ) [private]

7.5.2.13 void Menu::menuEvents ( )

7.5.2.14 void Menu::setDownAnimation (bool *downAnimation*)

7.5.2.15 void Menu::setLeftAnimation (bool *leftAnimation*)

7.5.2.16 void Menu::setPositions ( ) [private]

7.5.2.17 void Menu::setRightAnimation (bool *rightAnimation*)

7.5.2.18 void Menu::setUpAnimation (bool *upAnimation*)

7.5.3 Field Documentation

7.5.3.1 float Menu::animationSpeed [private]

7.5.3.2 float Menu::animationTime [private]

7.5.3.3 bool Menu::downAnimation [private]

7.5.3.4 bool Menu::first [private]

7.5.3.5 int Menu::firstTextPosition [private]

7.5.3.6 bool Menu::leftAnimation [private]

7.5.3.7 int Menu::leftPosition [private]

7.5.3.8 Font Menu::menuFont [private]

7.5.3.9 vector<Text> Menu::menuTexts [private]

7.5.3.10 int Menu::nText [private]

7.5.3.11 int Menu::outPosition [private]

7.5.3.12 bool Menu::rightAnimation [private]

7.5.3.13 int Menu::rightPosition [private]

7.5.3.14 float Menu::scaleFactor [private]

7.5.3.15 vector<float> Menu::speeds [private]

7.5.3.16 int Menu::stepCounter [private]

7.5.3.17 RectangleShape Menu::strip [private]

7.5.3.18 unsigned int Menu::textSize [private]

7.5.3.19 vector<Drawable*> Menu::toDraw [private]

7.5.3.20 bool Menu::upAnimation [private]

The documentation for this class was generated from the following file:

- Menu.h

7.6 Mesh Class Reference

The Mesh class defines each single drawable entity, in a format that OpenGL uses to render the objects.

`#include <Mesh.h>`

Data Structures

- struct texture

  It is used to organize the material data, in the form of textures.
- struct vertex

  It is a set of vectors which contains a position vector, a normal vector and a texture coordinate vector.

Public Member Functions

- Mesh (vector< vertex > vertices, vector< GLuint > indices, vector< texture > textures)
- void draw (sh::Shader shader)

Data Fields

- vector< GLuint > indices
- vector< texture > textures
- vector< vertex > vertices

Private Member Functions

- void init ()

Private Attributes

- GLuint EBO
- GLuint VAO
- GLuint VBO

7.6.1 Constructor & Destructor Documentation

7.6.1.1 Mesh::Mesh (vector< **vertex** > *vertices*, vector< GLuint > *indices*, vector< **texture** > *textures*)

The constructor loads Assimp's data structures and transforms that data to a format that OpenGL understands.

7.6.2 Member Function Documentation

7.6.2.1 void Mesh::draw (**sh::Shader** *shader*)

It renders the mesh, after binding the appropriate textures.

7.6.2.2 void Mesh::init ( ) [private]

This initialization function is used to setup the appropriate buffers and specify the vertex shader layout via vertex attribute pointers.

It is called after the constructor, which provides us large lists of mesh data that we can use for rendering.

7.6.3 Field Documentation

7.6.3.1 GLuint Mesh::EBO [private]

An EBO is a buffer, just like a vertex buffer object, that stores indices that OpenGL uses to decide what vertices to draw.

7.6.3.2 vector<GLuint> Mesh::indices

The set of indices which defines a particular mesh.

7.6.3.3 vector<texture> Mesh::textures

The set of textures which defines a particular mesh.

7.6.3.4 GLuint Mesh::VAO [private]

The so called vertex array object (VAO) can be bound just like a vertex buffer object and any subsequent vertex attribute calls from that point on will be stored inside the VAO.

This has the advantage that when configuring vertex attribute pointers you only have to make those calls once and whenever we want to draw the object, we can just bind the corresponding VAO. This makes switching between different vertex data and attribute configurations as easy as binding a different VAO.

Core OpenGL requires that we use a VAO so it knows what to do with our vertex inputs. If we fail to bind a VAO, OpenGL will most likely refuse to draw anything.

A vertex array object stores the following:

- Calls to glEnableVertexAttribArray or glDisableVertexAttribArray.
- Vertex attribute configurations via glVertexAttribPointer.
- Vertex buffer objects associated with vertex attributes by calls to glVertexAttribPointer.

7.6.3.5 GLuint Mesh::VBO [private]

The so called vertex buffer objects (VBO) can store a large number of vertices in the GPU's memory.

The advantage of using those buffer objects is that we can send large batches of data all at once to the graphics card, without having to send data a vertex a time. Sending data to the graphics card from the CPU is relatively slow, so wherever we can we try to send as much data as possible at once. Once the data is in the graphics card's memory the vertex shader has almost instant access to the vertices making it extremely fast.

7.6.3.6 vector<vertex> Mesh::vertices

The set of vertices which defines a particular mesh.

The documentation for this class was generated from the following file:

- Mesh.h

7.7 Mesh::texture Struct Reference

It is used to organize the material data, in the form of textures.

```
#include <Mesh.h>
```

Data Fields

- GLuint id
- aiString path
- string type

7.7.1 Field Documentation

7.7.1.1 GLuint Mesh::texture::id

The id of the texture, which is just an incremental identifier.

7.7.1.2 aiString Mesh::texture::path

The actual path of a particular texture.

7.7.1.3 string Mesh::texture::type

The type of the texture (e.g. a diffuse texture or a specular texture).

The documentation for this struct was generated from the following file:

- Mesh.h

7.8 Mesh::vertex Struct Reference

It is a set of vectors which contains a position vector, a normal vector and a texture coordinate vector.

```
#include <Mesh.h>
```

Data Fields

- vec3 normal
- vec3 position
- vec2 texCoords

7.8.1 Field Documentation

7.8.1.1 vec3 Mesh::vertex::normal

When lighting is enabled in OpenGL, the normal vectors are used to determine how much light is received at the specified vertex or surface.

This lighting processing is performed at eye coordinate space, therefore, normal vectors in object coordinates are also transformed to eye coordinates with GL_MODELVIEW matrix, but in a different way as vertices do.

7.8.1.2 vec3 Mesh::vertex::position

The actual point represented in the tridimensional space (x, y, z).

7.8.1.3 vec2 Mesh::vertex::texCoords

Texture coordinates specify the point in the texture image that will correspond to the vertex you are specifying them for.

The documentation for this struct was generated from the following file:

- Mesh.h

7.9 Model Class Reference

```
#include <Model.h>
```

Public Member Functions

- Model ()
- Model (GLchar *path)
- void draw (sh::Shader shader)

Data Fields

- float XMAX
- float XMIN
- float YMAX
- float YMIN
- float ZMAX
- float ZMIN

Private Member Functions

- vector< Mesh::texture > loadMaterialTextures (aiMaterial *mat, aiTextureType type, string typeName)
- void loadModel (string path)
- Mesh processMesh (aiMesh *mesh, const aiScene *scene)
- void processNode (aiNode *node, const aiScene *scene)
- GLint textureFromFile (const char *path)

Private Attributes

- string directory
- vector< Mesh::texture > loadedTextures
- vector< Mesh > meshes

7.9.1 Constructor & Destructor Documentation

7.9.1.1 Model::Model ( )

7.9.1.2 Model::Model (GLchar * *path*)

7.9.2 Member Function Documentation

7.9.2.1 void Model::draw (sh::Shader *shader*)

7.9.2.2 vector<Mesh::texture> Model::loadMaterialTextures (aiMaterial * *mat,* aiTextureType *type,* string *typeName*) [private]

7.9.2.3 void Model::loadModel (string *path*) [private]

7.9.2.4 Mesh Model::processMesh (aiMesh * *mesh,* const aiScene * *scene*) [private]

7.9.2.5 void Model::processNode (aiNode * *node,* const aiScene * *scene*) [private]

7.9.2.6 GLint Model::textureFromFile (const char * *path*) [private]

7.9.3 Field Documentation

7.9.3.1 string Model::directory [private]

7.9.3.2 vector<Mesh::texture> Model::loadedTextures [private]

7.9.3.3 vector<Mesh> Model::meshes [private]

7.9.3.4 float Model::XMAX

7.9.3.5 float Model::XMIN

7.9.3.6 float Model::YMAX

7.9.3.7 float Model::YMIN

7.9.3.8 float Model::ZMAX

7.9.3.9 float Model::ZMIN

The documentation for this class was generated from the following file:

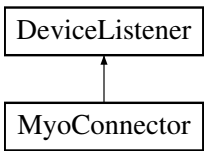
- Model.h

7.10 MyoConnector Class Reference

A MyoConnector object receives and processes events about a Myo.

`#include <MyoConnector.h>`

Inheritance diagram for MyoConnector:



Public Member Functions

- MyoConnector ()
- string getCurrentPose ()
- vec3 getDirections ()
- void onArmSync (Myo ∗myo, uint64_t timestamp, Arm arm, XDirection xDirection, float rotation, WarmupState warmupState)
- void onArmUnsync (Myo ∗myo, uint64_t timestamp)
- void onLock (Myo ∗myo, uint64_t timestamp)
- void onOrientationData (Myo ∗myo, uint64_t timestamp, const Quaternion< float > &quat)
- void onPose (Myo ∗myo, uint64_t timestamp, Pose pose)
- void onUnlock (Myo ∗myo, uint64_t timestamp)
- void onUnpair (Myo ∗myo, uint64_t timestamp)
- void print ()

Private Attributes

- Pose currentPose
- bool isUnlocked
- bool onArm
- int pitch_w
- int roll_w
- Arm whichArm
- int yaw_w

7.10.1 Detailed Description

Classes that inherit from myo::DeviceListener can be used to receive events from Myo devices. DeviceListener provides several virtual functions for handling different kinds of events. If you do not override an event, the default behavior is to do nothing.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 MyoConnector::MyoConnector ( )

It is used to generate an instance of the MyoConnector class and link it to the already created Hub object.

7.10.3 Member Function Documentation

7.10.3.1 string MyoConnector::getCurrentPose ( )

It returns the current pose, read by the onPose function whenever the Myo detects that the person wearing it has changed their pose.

7.10.3.2 vec3 MyoConnector::getDirections ( )

It returns the current pitch, roll and yaw values.

7.10.3.3 void MyoConnector::onArmSync (Myo * *myo,* uint64_t *timestamp,* Arm *arm,* XDirection *xDirection,* float *rotation,* WarmupState *warmupState*)

Called when a paired Myo recognizes that it is on an arm.

7.10.3.4 void MyoConnector::onArmUnsync (Myo * *myo,* uint64_t *timestamp*)

Called when a paired Myo is moved or removed from the arm.

7.10.3.5 void MyoConnector::onLock (Myo * *myo,* uint64_t *timestamp*)

Called when a paired Myo becomes locked.

7.10.3.6 void MyoConnector::onOrientationData (Myo * *myo,* uint64_t *timestamp,* const Quaternion< float > & *quat*)

Called when a paired Myo has provided new orientation data.

7.10.3.7 void MyoConnector::onPose (Myo * *myo,* uint64_t *timestamp,* Pose *pose*)

Called when a paired Myo has provided a new pose.

7.10.3.8 void MyoConnector::onUnlock (Myo * *myo,* uint64_t *timestamp*)

Called when a paired Myo becomes unlocked.

7.10.3.9 void MyoConnector::onUnpair (Myo * *myo,* uint64_t *timestamp*)

Called when a Myo has been unpaired.

7.10.3.10 void MyoConnector::print ( )

It prints the current values that were updated by the overriden virtual functions.

7.10.4 Field Documentation

7.10.4.1 Pose MyoConnector::currentPose [private]

It stores the value returned by the onPose funtion, which represents the pose detected in the current frame.

7.10.4.2 bool MyoConnector::isUnlocked [private]

It stores the value returned by the onLock and onUnlock funtions, which represents wheter or not the user has unlocked the paired Myo armband.

7.10.4.3 bool MyoConnector::onArm [private]

It stores the value returned by the onArmSync and onArmUnsync funtions, which represents wheter or not the user is wearing the paired Myo armband.

7.10.4.4 int MyoConnector::pitch_w [private]

It stores the value returned by the onOrientationData funtion, which represents the pitch Euler angle, obtained from the unit quaternion.

7.10.4.5 int MyoConnector::roll_w [private]

It stores the value returned by the onOrientationData funtion, which represents the roll Euler angle, obtained from the unit quaternion.

7.10.4.6 Arm MyoConnector::whichArm [private]

It stores the value returned by the onArmSync and onArmUnsync funtions, which represents on which arm the user is wearing the paired Myo armband.

7.10.4.7 int MyoConnector::yaw_w [private]

It stores the value returned by the onOrientationData funtion, which represents the yaw Euler angle, obtained from the unit quaternion.

The documentation for this class was generated from the following file:

- MyoConnector.h

7.11 Settings Class Reference

`#include <Settings.h>`

Public Member Functions

- Settings ()
- bool getDownAnimation ()
- bool getFadeLeftAnimation ()
- bool getFadeRightAnimation ()
- vector< Drawable * > getObjectsVector ()
- bool getScrollDownAnimation ()
- bool getScrollUpAnimation ()
- bool getUpAnimation ()
- void setDownAnimation (bool downAnimation)
- void setFadeLeftAnimation (bool fadeLeftAnimation)
- void setFadeRightAnimation (bool fadeRightAnimation)
- void setScrollDownAnimation (bool scrollDownAnimation)
- void setScrollUpAnimation (bool scrollUpAnimation)
- void settingsEvents ()
- void setUpAnimation (bool upAnimation)
- void test ()

Private Member Functions

- void animateDown ()
- void animateUp ()
- void fadeLeft ()
- void fadeRight ()
- void init ()
- void pageDown ()
- void pageUp ()
- void scrollDown ()
- void scrollUp ()

Private Attributes

- float alpha
- float animationSpeed
- float animationTime
- RectangleShape background
- bool downAnimation
- bool fadeLeftAnimation
- bool fadeRightAnimation
- float fadeSpeed
- float fadeTime
- int nSettings
- vector< int > optionsPositions
- vector< vector< Text > > optionsTexts
- bool pageDownAnimation
- int pageHeaderPosition

- bool pageUpAnimation
- bool scrollDownAnimation
- bool scrollUpAnimation
- bool secondFade
- RectangleShape selector
- int selectorPosition
- Font settingsFont
- vector< Text > settingsTexts
- int stepCounter
- float textMargin
- unsigned int textSize
- float thickness
- vector< Drawable * > toDraw
- float udAnimationSpeed
- bool upAnimation

7.11.1 Constructor & Destructor Documentation

7.11.1.1 Settings::Settings ( )

7.11.2 Member Function Documentation

7.11.2.1 void Settings::animateDown ( ) [private]

7.11.2.2 void Settings::animateUp ( ) [private]

7.11.2.3 void Settings::fadeLeft ( ) [private]

7.11.2.4 void Settings::fadeRight ( ) [private]

7.11.2.5 bool Settings::getDownAnimation ( )

7.11.2.6 bool Settings::getFadeLeftAnimation ( )

7.11.2.7 bool Settings::getFadeRightAnimation ( )

7.11.2.8 vector<Drawable*> Settings::getObjectsVector ( )

7.11.2.9 bool Settings::getScrollDownAnimation ( )

7.11.2.10 bool Settings::getScrollUpAnimation ( )

7.11.2.11 bool Settings::getUpAnimation ( )

7.11.2.12 void Settings::init ( ) [private]

7.11.2.13 void Settings::pageDown ( ) [private]

7.11.2.14 void Settings::pageUp ( ) [private]

7.11.2.15 void Settings::scrollDown ( ) [private]

7.11.2.16 void Settings::scrollUp ( ) [private]

7.11.2.17 void Settings::setDownAnimation (bool *downAnimation*)

7.11.2.18 void Settings::setFadeLeftAnimation (bool *fadeLeftAnimation*)

7.11.2.19 void Settings::setFadeRightAnimation (bool *fadeRightAnimation*)

7.11.2.20 void Settings::setScrollDownAnimation (bool *scrollDownAnimation*)

7.11.2.21 void Settings::setScrollUpAnimation (bool *scrollUpAnimation*)

7.11.2.22 void Settings::settingsEvents ( )

7.11.2.23 void Settings::setUpAnimation (bool *upAnimation*)

7.11.2.24 void Settings::test ( )

7.11.3 Field Documentation

7.11.3.1 float Settings::alpha [private]

7.11.3.2 float Settings::animationSpeed [private]

7.11.3.3 float Settings::animationTime [private]

7.11.3.4 RectangleShape Settings::background [private]

7.11.3.5 bool Settings::downAnimation [private]

7.11.3.6 bool Settings::fadeLeftAnimation [private]

7.11.3.7 bool Settings::fadeRightAnimation [private]

7.11.3.8 float Settings::fadeSpeed [private]

7.11.3.9 float Settings::fadeTime [private]

7.11.3.10 int Settings::nSettings [private]

7.11.3.11 vector<int> Settings::optionsPositions [private]

7.11.3.12 vector<vector<Text> > Settings::optionsTexts [private]

7.11.3.13 bool Settings::pageDownAnimation [private]

7.11.3.14 int Settings::pageHeaderPosition [private]

7.11.3.15 bool Settings::pageUpAnimation [private]

7.11.3.16 bool Settings::scrollDownAnimation [private]

7.11.3.17 bool Settings::scrollUpAnimation [private]

7.11.3.18 bool Settings::secondFade [private]

7.11.3.19 RectangleShape Settings::selector [private]

7.11.3.20 int Settings::selectorPosition [private]

7.11.3.21 Font Settings::settingsFont [private]

7.11.3.22 vector<Text> Settings::settingsTexts [private]

7.11.3.23 int Settings::stepCounter [private]

7.11.3.24 float Settings::textMargin [private]

7.11.3.25 unsigned int Settings::textSize [private]

7.11.3.26 float Settings::thickness [private]

7.11.3.27 vector<Drawable*> Settings::toDraw [private]

7.11.3.28 float Settings::udAnimationSpeed [private]

7.11.3.29 bool Settings::upAnimation [private]

The documentation for this class was generated from the following file:

- Settings.h

7.12 sh::Shader Class Reference

The Shader class describes the little programs, which rest on the GPU, used in the rendering operation next to an OpenGL drawing command execution.

```
#include <Shader.h>
```

Public Member Functions

- Shader ()
- Shader (const GLchar *vertexPath, const GLchar *fragmentPath)
- void init ()
- void use ()

Data Fields

- const GLchar * fragmentPath
- GLuint program
- const GLchar * vertexPath

7.12.1 Detailed Description

A Shader is a user-defined program designed to run on some stage of a graphics processor. Its purpose is to execute one of the programmable stages of the rendering pipeline and it represents compiled GLSL code. In a basic sense, shaders are nothing more than programs transforming inputs to outputs. Shaders are also very isolated programs in that they are not allowed to communicate with each other: the only communication they have is via their inputs and outputs. The Shader class handles both the vertex and the fragment shaders.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 sh::Shader::Shader ( )

Constructs a Shader object, without setting any parameter.

7.12.2.2 sh::Shader::Shader (const GLchar * *vertexPath*, const GLchar * *fragmentPath*)

Constructs a Shader object, by setting the vertexPath and fragmentPath variables to the given values.

7.12.3 Member Function Documentation

7.12.3.1 void sh::Shader::init ( )

Called by the constructor, it opens the data stream to the GLSL source files, creates the shader objects, compiles the obtained GLSL source code and links it to the newly generated shader objects. It also attachs both the GL_V↩ ERTEX_SHADER and the GL_FRAGMENT_SHADER to the program object.

7.12.3.2 void sh::Shader::use ( )

Called to use the defined shaders program object, which is needed by the OpenGL drawing functions.

7.12.4 Field Documentation

7.12.4.1 const GLchar* sh::Shader::fragmentPath

It contains the path to a GLSL source file, filled with a set of strings, which represents the core of the GL_FRAG↩
MENT_SHADER stage.

7.12.4.2 GLuint sh::Shader::program

A program object represents fully processed executable code, in the OpenGL Shading Language (GLSL), for one or more Shader stages. Empty program objects must be filled in by compiling and linking shaders into the program itself.

7.12.4.3 const GLchar* sh::Shader::vertexPath

It contains the path to a GLSL source file, filled with a set of strings, which represents the core of the GL_VERTE↩
X_SHADER stage.

The documentation for this class was generated from the following file:

- Shader.h

7.13 ThreeD Class Reference

`#include <ThreeD.h>`

Public Member Functions

- ThreeD ()
- void checkPositions ()
- float getCameraDistance ()
- bool getDownAnimation ()
- float getHorizontalK ()
- bool getLeftAnimation ()
- Model * getModel ()
- float getModelDepthOffset ()
- float getModelHorizontalOffset ()
- float getModelVerticalOffset ()
- vector< Drawable * > getObjectsVector ()
- bool getRightAnimation ()
- sh::Shader getShader ()
- bool getUpAnimation ()
- float getVerticalK ()
- void loadModel ()
- void setDownAnimation (bool downAnimation)
- void setLeftAnimation (bool leftAnimation)
- void setRightAnimation (bool rightAnimation)
- void setUpAnimation (bool upAnimation)
- MANAGER_STATUS threeDEvents ()

Private Member Functions

- void animateDown ()
- void animateLeft ()
- void animateRight ()
- void animateUp ()
- bool checkExtension (string modelName, int modelNameLen)
- void init ()
- void loadFiles ()

Private Attributes

- float animationSpeed
- float animationTime
- bool downAnimation
- bool first
- int firstModelPosition
- string fragmentShaderPath = workingPath + "3D/Shaders/MacOS/fragmentShader.frag"
- bool leftAnimation
- int leftPosition
- Model model
- vector< File > modelFiles
- GLchar * modelPath
- int nModel
- int outPosition
- bool rightAnimation
- int rightPosition
- float scaleFactor
- sh::Shader shader
- int stepCounter
- vector< Drawable * > toDraw
- bool upAnimation
- string vertexShaderPath = workingPath + "3D/Shaders/MacOS/vertexShader.vert"
- float verticalK = 4.8f / (3.7f * verticalAspectRatio)
- float xAxisK = 4.8f / 3.7f
- float yAxisK = 4.8f / 2.f
- float zAxisK = 4.8f / 0.78f

7.13.1 Constructor & Destructor Documentation

7.13.1.1 ThreeD::ThreeD ( )

7.13.2 Member Function Documentation

7.13.2.1 void ThreeD::animateDown ( ) [private]

7.13.2.2 void ThreeD::animateLeft ( ) [private]

7.13.2.3 void ThreeD::animateRight ( ) [private]

7.13.2.4 void ThreeD::animateUp ( ) [private]

7.13.2.5 bool ThreeD::checkExtension (string *modelName,* int *modelNameLen*) [private]

7.13.2.6 void ThreeD::checkPositions ( )

7.13.2.7 float ThreeD::getCameraDistance ( )

7.13.2.8 bool ThreeD::getDownAnimation ( )

7.13.2.9 float ThreeD::getHorizontalK ( )

7.13.2.10 bool ThreeD::getLeftAnimation ( )

7.13.2.11 **Model**∗ ThreeD::getModel ( )

7.13.2.12 float ThreeD::getModelDepthOffset ( )

7.13.2.13 float ThreeD::getModelHorizontalOffset ( )

7.13.2.14 float ThreeD::getModelVerticalOffset ( )

7.13.2.15 vector<Drawable∗> ThreeD::getObjectsVector ( )

7.13.2.16 bool ThreeD::getRightAnimation ( )

7.13.2.17 **sh::Shader** ThreeD::getShader ( )

7.13.2.18 bool ThreeD::getUpAnimation ( )

7.13.2.19 float ThreeD::getVerticalK ( )

7.13.2.20 void ThreeD::init ( ) [private]

7.13.2.21 void ThreeD::loadFiles ( ) [private]

7.13.2.22 void ThreeD::loadModel ( )

7.13.2.23 void ThreeD::setDownAnimation (bool *downAnimation*)

7.13.2.24 void ThreeD::setLeftAnimation (bool *leftAnimation*)

7.13.2.25 void ThreeD::setRightAnimation (bool *rightAnimation*)

7.13.2.26 void ThreeD::setUpAnimation (bool *upAnimation*)

7.13.2.27 MANAGER_STATUS ThreeD::threeDEvents ( )

7.13.3 Field Documentation

7.13.3.1 float ThreeD::animationSpeed [private]

7.13.3.2 float ThreeD::animationTime [private]

7.13.3.3 bool ThreeD::downAnimation [private]

7.13.3.4 bool ThreeD::first [private]

7.13.3.5 int ThreeD::firstModelPosition [private]

7.13.3.6 string ThreeD::fragmentShaderPath = workingPath + "3D/Shaders/MacOS/fragmentShader.frag" [private]

7.13.3.7 bool ThreeD::leftAnimation [private]

7.13.3.8 int ThreeD::leftPosition [private]

7.13.3.9 Model ThreeD::model [private]

7.13.3.10 vector<File> ThreeD::modelFiles [private]

7.13.3.11 GLchar* ThreeD::modelPath [private]

7.13.3.12 int ThreeD::nModel [private]

7.13.3.13 int ThreeD::outPosition [private]

7.13.3.14 bool ThreeD::rightAnimation [private]

7.13.3.15 int ThreeD::rightPosition [private]

7.13.3.16 float ThreeD::scaleFactor [private]

7.13.3.17 sh::Shader ThreeD::shader [private]

7.13.3.18 int ThreeD::stepCounter [private]

7.13.3.19 vector<Drawable*> ThreeD::toDraw [private]

7.13.3.20 bool ThreeD::upAnimation [private]

7.13.3.21 string ThreeD::vertexShaderPath = workingPath + "3D/Shaders/MacOS/vertexShader.vert" [private]

7.13.3.22 float ThreeD::verticalK = 4.8f / (3.7f * verticalAspectRatio) [private]

7.13.3.23 float ThreeD::xAxisK = 4.8f / 3.7f [private]

7.13.3.24 float ThreeD::yAxisK = 4.8f / 2.f [private]

7.13.3.25 float ThreeD::zAxisK = 4.8f / 0.78f [private]

The documentation for this class was generated from the following file:

- ThreeD.h

7.14 Video Class Reference

`#include <Video.h>`

Public Member Functions

- Video ()
- void checkPositions ()
- bool getDownAnimation ()
- bool getLeftAnimation ()
- vector< Drawable * > getObjectsVector ()
- bool getRightAnimation ()
- bool getUpAnimation ()
- sfe::Movie * getVideoToPlay ()
- void setDownAnimation (bool downAnimation)
- void setLeftAnimation (bool leftAnimation)
- void setPositions ()
- void setRightAnimation (bool rightAnimation)
- void setUpAnimation (bool upAnimation)
- MANAGER_STATUS videoEvents ()

Private Member Functions

- void animateDown ()
- void animateLeft ()
- void animateRight ()
- void animateUp ()
- bool checkExtension (string videoName, int videoNameLen)
- void init ()
- void loadVideos ()

Private Attributes

- float animationSpeed
- float animationTime
- bool downAnimation
- bool first
- int firstVideoPosition
- bool leftAnimation
- int leftPosition
- sfe::Movie movie
- int nVideo
- int outPosition
- bool rightAnimation
- int rightPosition
- float scaleFactor
- int stepCounter
- vector< Drawable * > toDraw
- bool upAnimation
- vector< File > videoFiles

7.14.1 Constructor & Destructor Documentation

7.14.1.1 Video::Video ( )

7.14.2 Member Function Documentation

7.14.2.1 void Video::animateDown ( ) [private]

7.14.2.2 void Video::animateLeft ( ) [private]

7.14.2.3 void Video::animateRight ( ) [private]

7.14.2.4 void Video::animateUp ( ) [private]

7.14.2.5 bool Video::checkExtension (string *videoName,* int *videoNameLen*) [private]

7.14.2.6 void Video::checkPositions ( )

7.14.2.7 bool Video::getDownAnimation ( )

7.14.2.8 bool Video::getLeftAnimation ( )

7.14.2.9 vector<Drawable*> Video::getObjectsVector ( )

7.14.2.10 bool Video::getRightAnimation ( )

7.14.2.11 bool Video::getUpAnimation ( )

7.14.2.12 sfe::Movie* Video::getVideoToPlay ( )

7.14.2.13 void Video::init ( ) [private]

7.14.2.14 void Video::loadVideos ( ) [private]

7.14.2.15 void Video::setDownAnimation (bool *downAnimation*)

7.14.2.16 void Video::setLeftAnimation (bool *leftAnimation*)

7.14.2.17 void Video::setPositions ( )

7.14.2.18 void Video::setRightAnimation (bool *rightAnimation*)

7.14.2.19 void Video::setUpAnimation (bool *upAnimation*)

7.14.2.20 MANAGER_STATUS Video::videoEvents ( )

7.14.3 Field Documentation

7.14.3.1 float Video::animationSpeed [private]

7.14.3.2 float Video::animationTime [private]

7.14.3.3 bool Video::downAnimation [private]

7.14.3.4 bool Video::first [private]

7.14.3.5 int Video::firstVideoPosition [private]

7.14.3.6 bool Video::leftAnimation [private]

7.14.3.7 int Video::leftPosition [private]

7.14.3.8 sfe::Movie Video::movie [private]

7.14.3.9 int Video::nVideo [private]

7.14.3.10 int Video::outPosition [private]

7.14.3.11 bool Video::rightAnimation [private]

7.14.3.12 int Video::rightPosition [private]

7.14.3.13 float Video::scaleFactor [private]

7.14.3.14 int Video::stepCounter [private]

7.14.3.15 vector<Drawable*> Video::toDraw [private]

7.14.3.16 bool Video::upAnimation [private]

7.14.3.17 vector<File> Video::videoFiles [private]

The documentation for this class was generated from the following file:

- Video.h

Chapter 8

File Documentation

8.1 Bluetooth.h File Reference

Data Structures

- class Bluetooth

  The Bluetooth class provides wireless communication between this software and the Android application.

8.2 Errors.txt File Reference

8.3 File.h File Reference

Data Structures

- class File

  The File class describes resources relative to the Video and ThreeD sections.

8.4 Global.h File Reference

```
#include <Libraries.h>
```

Macros

- #define DEBUG
- #define DIAGONAL
- #define DOWN 4
- #define LEAP_SCALE 15
- #define LEFT 1
- #define PI 3.14159265
- #define RIGHT 3
- #define SCREEN
- #define UP 2
- #define VERBOSE

Enumerations

- enum MANAGER_STATUS {
  MENU_STATUS, VIDEO_STATUS, THREED_STATUS, SETTINGS_STATUS,
  EXIT_STATUS }

Functions

- void initGlobal ()

Variables

- double diagonalAngle
- int frameRateLimit
- float height
- float horizontalAspectRatio
- RectangleShape mainDiagonal
- bool MYO
- double pit
- bool quit
- RectangleShape secondaryDiagonal
- float verticalAspectRatio
- float width
- string workingPath

8.4.1 Macro Definition Documentation

8.4.1.1 #define DEBUG

8.4.1.2 #define DIAGONAL

8.4.1.3 #define DOWN 4

8.4.1.4 #define LEAP_SCALE 15

8.4.1.5 #define LEFT 1

8.4.1.6 #define PI 3.14159265

8.4.1.7 #define RIGHT 3

8.4.1.8 #define SCREEN

8.4.1.9 #define UP 2

8.4.1.10 #define VERBOSE

8.4.2 Enumeration Type Documentation

8.4.2.1 enum MANAGER_STATUS

It describes the various status in which the program could be.

The default status is the MENU_STATUS, passed to the program at its execution.

Enumerator

MENU_STATUS

VIDEO_STATUS

THREED_STATUS

SETTINGS_STATUS

EXIT_STATUS

8.4.3 Function Documentation

8.4.3.1 void initGlobal ( )

It is used to initialize each and every global variable, specified in the header file.

8.4.4 Variable Documentation

8.4.4.1 double diagonalAngle

The angle of the program window's hypotenuse (diagonal).

8.4.4.2 int frameRateLimit

It sets the maximum frequency (rate) at which the program should display consecutive images, called frames.

8.4.4.3 float height

The height of the program window.

8.4.4.4 float horizontalAspectRatio

The horizontal aspect ratio describes the proportional relationship between the program's width and height.

It is calcuted by the following formula:

```
horizontalAspectRatio = width / height;
```

8.4.4.5 RectangleShape mainDiagonal

The actual main diagonal line, draw on the window.

8.4.4.6 bool MYO

The variable which defines the user's choice of using or not the Myo armband.

8.4.4.7 double pit

The length of the program window's hypotenuse (diagonal).

8.4.4.8 bool quit

The variable used by some functions to stop the program execution.

8.4.4.9 RectangleShape secondaryDiagonal

The actual secondary diagonal line, draw on the window.

8.4.4.10 float verticalAspectRatio

The vertical aspect ratio describes the proportional relationship between the program's height and width.

It is calcuted by the following formula:

```
verticalAspectRatio = height / width;
```

8.4.4.11 float width

The width of the program window.

8.4.4.12 string workingPath

It defines the project's working path, which contains resource files such as videos and 3D objects.

8.5 LeapListener.h File Reference

Data Structures

- class LeapListener

  The LeapListener class provides data transfers between this software and the Leap Motion device.

Macros

- #define NFINGERS 5
- #define NSTATES 4

8.5.1 Macro Definition Documentation

8.5.1.1 #define NFINGERS 5

8.5.1.2 #define NSTATES 4

8.6 Libraries.h File Reference

```
#include <iostream>
#include <thread>
#include <atomic>
#include <cstdlib>
#include <string>
#include <cmath>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>
#include <algorithm>
#include <iomanip>
#include <stdexcept>
#include <myo/myo.hpp>
#include <Leap.h>
#include <GL/glew.h>
#include <SOIL.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
#include <SFML/Graphics.hpp>
#include <SFML/OpenGL.hpp>
#include <SFML/Audio.hpp>
#include <sfeMovie/Movie.hpp>
```

8.7 Manager.h File Reference

```
#include <Menu.h>
#include <Video.h>
#include <ThreeD.h>
#include <Settings.h>
#include <MyoConnector.h>
#include <LeapListener.h>
#include <Bluetooth.h>
```

Data Structures

- class Manager

8.8 Menu.h File Reference

Data Structures

- class Menu

8.9 Mesh.h File Reference

```
#include <Shader.h>
```

Data Structures

- class Mesh

  The Mesh class defines each single drawable entity, in a format that OpenGL uses to render the objects.
- struct Mesh::texture

  It is used to organize the material data, in the form of textures.
- struct Mesh::vertex

  It is a set of vectors which contains a position vector, a normal vector and a texture coordinate vector.

8.10 Model.h File Reference

```
#include <Mesh.h>
```

Data Structures

- class Model

8.11 MyoConnector.h File Reference

Data Structures

- class MyoConnector

  A MyoConnector object receives and processes events about a Myo.

8.12 README.md File Reference

8.13 Settings.h File Reference

Data Structures

- class Settings

8.14 Shader.h File Reference

Data Structures

- class sh::Shader

  The Shader class describes the little programs, which rest on the GPU, used in the rendering operation next to an OpenGL drawing command execution.

Namespaces

- sh

  It is used to distinguish Holum shaders from SFML ones.

8.15 ThreeD.h File Reference

```
#include <Model.h>
#include <File.h>
```

Data Structures

- class ThreeD

8.16 Video.h File Reference

```
#include <File.h>
```

Data Structures

- class Video