# Roombots Simulator

1.0

Generated by Doxygen 1.8.11

# Contents

# Chapter 1

# Roombots Simulator

## 1.1 Introduction

"Roombots Simulator" is a software an immersive interface to interact with virtual Roombots.
Roombots are self-reconfigurable robots developed by the BioRobotics Laboratory (BioRob) at EPFL.
For more information about Roombots or BioRob, please visit their website : http://biorob.epfl.ch/cms/page-36376.html
This software uses the Leapmotion, a gesture-recognition device, as input and the Oculus Rift, a Virtual Reality (VR) device.
Both those devices MUST be used in order to enjoy the full experience.

## 1.2 How to run

First, make sure the Leapmotion AND the Oculus Rift are both ready for use (drivers installed, runtimes installed, etc.).
Then, open the "release" folder and run "RoombotsSimulator.exe".
It is important not to modify any file contained in this folder.
For more information about the folders and files used by the software see section "File hierarchy"
The software uses two windows : the main window showing the scene and the information window,
a terminal-like window showing information about the current state of the software.

## 1.3 How to use

### 1.3.1 The Graphic User Interface (GUI)

The GUI consists of blue holders containing Roombots Structures.
After grabbing and dropping a Structure coming from the holders, a new one will pop inside the corresponding holder.
On the right side of the room, there is a big grey cube representing a trash can. By dropping a Structure inside it, it will be deleted.

### 1.3.2 Controls

"Roombots Simulator" uses gesture-recognition. With the right-most hand above the Leapmotion, the user can grab Structures and drop them.
To do so, the user must "pinch" the Structures and then move the hand around. Opening the hand while holding a Structure will drop it.
One does not have to pinch exactly on the Structure, it is easier to grab Structure when keeping the hand pinching until the Structure is grabbed.
software also uses the keyboard. The 'wasd' keys are used to move around,'q' and 'e' can rotate the Structure currently held, clock-wise
or counter-clock-wise respectively and the space bar allows to switch between the Room-View and the Box-View.
The Room-View puts the user inside the room, while the Box-View puts the user above the room, as if he or she was sitting in front of a box containing the room.
In Room-view, it is not possible to reach for the holders and thus add new Structures, but allows to be more precise when moving Structures.
In Box-View, it is not possible to move around with 'wasd'

### 1.3.3 Closing the software.

To close the software, first close the main window showing the scene and only then the information window.

## 1.4 Details

### 1.4.1 File Hierarchy

The file hierachy of the "Release" folder is the following :
Release
-—Libraries //Contains library-related files. DO NOT MODIFY
-—Models //Contains the various .obj (wavefront) files used to import models
-—Shaders //Contains all the .glsl program files used by the software
-—Structures //Contains all the .rbs (RoomBotStructure) files representing the Structures
-—Textures //Contains all the texture images used to draw the scene

### 1.4.2 File details

Models : There are two models used in this software : a hemi-sphere and a circle.
There are different version of both, each version having a different number of vertices.
For instance, circle_100.obj is the base circle with 100% of its vertices, while hemisphere_10 is the hemi-sphere with 10% of its vertices

Structures : There are a few .rbs files available, the most important ones being "chair.rbs", "table.rbs" and "stool.rbs"

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

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

# Chapter 3

# Class Index

## 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

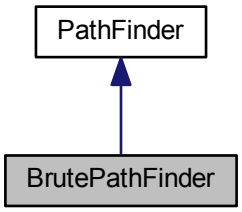
# Chapter 5

# Class Documentation

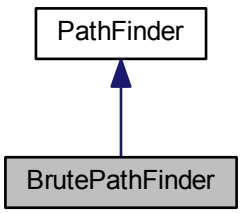
## 5.1 BrutePathFinder Class Reference

`#include <BrutePathFinder.hh>`

Inheritance diagram for BrutePathFinder:




Collaboration diagram for BrutePathFinder:



### Public Member Functions

- virtual void Run (Path &path, const Position &start, const Position &finish) const
- virtual std::string Name () const

### 5.1.1 Detailed Description

This PathFinder uses a very simple path-finding algorithm : It simply goes to the same z-coordinate than the final position, then to the same x-coordinate and finally to the same y-coordinate, to end up at the same Position

### 5.1.2 Member Function Documentation

#### 5.1.2.1 std::string BrutePathFinder::Name ( ) const [virtual]

This method simply returns the name of the path-finding algorithm

Implements PathFinder.

#### 5.1.2.2 void BrutePathFinder::Run ( Path & *path,* const Position & *start,* const Position & *finish* ) const [virtual]

Creates a Path, a succession of Position from a Position to another. It uses an external Path and fills it

- `path` A reference to the Path to fill
- `start` The starting Position of the Path
- `finish` The finishing Position of the Path

Implements PathFinder.

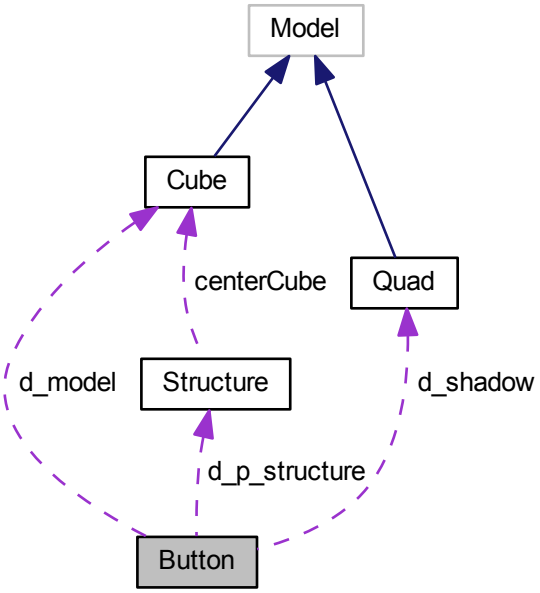
The documentation for this class was generated from the following files:

- RoombotsSimulator/BrutePathFinder.hh
- RoombotsSimulator/BrutePathFinder.cc

## 5.2 Button Class Reference

`#include <Button.hh>`

Collaboration diagram for Button:




### Public Member Functions

- Button (glm::vec3 position, unsigned int ID, Structure *p_structure)
- void Draw (const glm::mat4 &VP) const
- void CleanUp () const
- glm::vec3 Position () const
- unsigned int ID () const
- Structure * AssignedStructure () const

### Private Attributes

- Cube * d_model
  *the cube model used to represent the button*
- Quad * d_shadow
  *a shadow of the cube projected on the floor*
- const glm::vec3 d_position
  *its position within the scene*
- const unsigned int d_ID
  *its unique ID. Used to pop new Structures on the button*
- Structure * d_p_structure
  *a pointer to the Structure that pops from the button*

### 5.2.1 Detailed Description

A Button is a holder for MovableStructure to pop.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 Button::Button ( glm::vec3 *position,* unsigned int *ID,* Structure * *p_structure* )

Creates a new button containing the structure passed as argument

- `position` the position of the new Button
- `ID` the unique ID of the new Button
- `a` pointer to the Structure that will be used to pop new MovableStructures

### 5.2.3 Member Function Documentation

#### 5.2.3.1 Structure * Button::AssignedStructure ( ) const

Returns a pointer to the assigned Structure

#### 5.2.3.2 void Button::CleanUp ( ) const

Cleans up all the models (cube, shadow and the structure's) and deletes the pointer to the Structure, meaning it absolutely shouldn't be accessed once this method has been called

#### 5.2.3.3 void Button::Draw ( const glm::mat4 & *VP* ) const

Draws the button and its shadow

#### 5.2.3.4 unsigned int Button::ID ( ) const [inline]

Returns the button's ID

#### 5.2.3.5 glm::vec3 Button::Position ( ) const [inline]

Returns the button's position

### 5.2.4 Member Data Documentation

#### 5.2.4.1 const unsigned int Button::d_ID [private]

its unique ID. Used to pop new Structures on the button

#### 5.2.4.2 Cube* Button::d_model [private]

the cube model used to represent the button

#### 5.2.4.3 Structure* Button::d_p_structure [private]

a pointer to the Structure that pops from the button

#### 5.2.4.4 const glm::vec3 Button::d_position [private]

its position within the scene

#### 5.2.4.5 Quad* Button::d_shadow [private]

a shadow of the cube projected on the floor

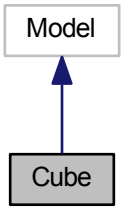
The documentation for this class was generated from the following files:

- RoombotsSimulator/Button.hh
- RoombotsSimulator/Button.cc

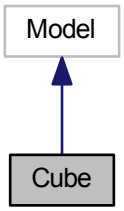
## 5.3 Cube Class Reference

`#include <Cube.hh>`

Inheritance diagram for Cube:



Collaboration diagram for Cube:




### Public Member Functions

- Cube (const std::string vShaderFileName, const std::string fShaderFileName, const std::string textureFile↩ Name, const glm::vec4 &color)
- Cube (const char ∗vShaderFileName, const char ∗fShaderFileName, const char ∗textureFileName, const glm::vec4 &color)

### Protected Member Functions

- virtual void SetVertices (std::vector< glm::vec3 > ∗vertices)
- virtual void SetUVs (std::vector< glm::vec2 > ∗uvs)

## 5.3.1 Constructor & Destructor Documentation

#### 5.3.1.1 Cube::Cube ( const std::string *vShaderFileName,* const std::string *fShaderFileName,* const std::string *textureFileName,* const glm::vec4 & *color* ) [inline]

#### 5.3.1.2 Cube::Cube ( const char ∗ *vShaderFileName,* const char ∗ *fShaderFileName,* const char ∗ *textureFileName,* const glm::vec4 & *color* ) [inline]

## 5.3.2 Member Function Documentation

#### 5.3.2.1 void Cube::SetUVs ( std::vector< glm::vec2 > ∗ *uvs* ) [protected], [virtual]

#### 5.3.2.2 void Cube::SetVertices ( std::vector< glm::vec3 > ∗ *vertices* ) [protected], [virtual]

Creates and adds vertices to the Model to create a one-meter cube

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

- RoombotsSimulator/Cube.hh
- RoombotsSimulator/Cube.cc

## 5.4 DepthBuffer Struct Reference

`#include <DepthBuffer.hh>`

### Public Member Functions

- DepthBuffer (OVR::Sizei size)

### Public Attributes

- GLuint texId

### 5.4.1 Constructor & Destructor Documentation

#### 5.4.1.1 DepthBuffer::DepthBuffer ( OVR::Sizei *size* ) [inline]

### 5.4.2 Member Data Documentation

#### 5.4.2.1 GLuint DepthBuffer::texId

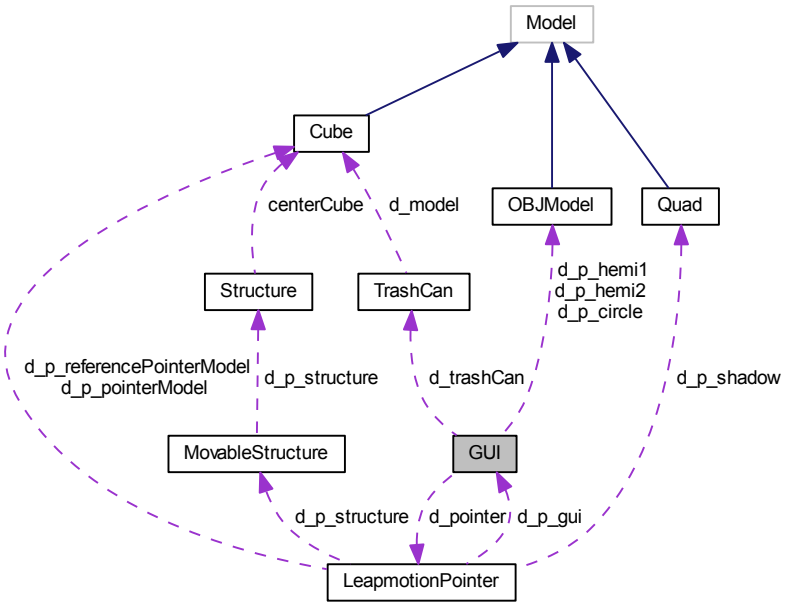
The documentation for this struct was generated from the following file:

- RoombotsSimulator/DepthBuffer.hh

## 5.5 GUI Class Reference

`#include <GUI.hh>`

Collaboration diagram for GUI:



### Public Member Functions

- void Init ()
- void Update (bool mode)
- void UpdateWorldMatrix (const glm::mat4 &worldMatrix)
- void const Render (const glm::mat4 &VP)
- size_t NButtons ()
- void DroppedStructure (unsigned int buttonID)
- void RotateStructure (bool clockWise)
- std::vector< Position > GetAllRoombotsPositions ()
- void CleanUp ()

### Private Member Functions

- void AddButton (Structure *p_structure)
- void CheckForPinchedStructure ()
- void UpdatePointer (bool mode)
- void PopStructure (unsigned int buttonID)

### Private Attributes

- std::vector< const Button * > d_buttons

  *The Buttons contained in the scene.*
- size_t d_nButtons = 0

  *The number of Buttons in the scene.*
- std::vector< MovableStructure * > d_structures

  *The Structures contained in the scene.*
- size_t d_nStructures = 0

  *The number of Structures in the scene.*
- LeapmotionPointer d_pointer

  *The pointer using the Leapmotion device.*
- OBJModel * d_p_hemi1

  *The pointer to the first hemisphere Model.*
- OBJModel * d_p_hemi2

  *The pointer to the second hemisphere Model.*
- OBJModel * d_p_circle

  *The pointer to the circle Model.*
- TrashCan * d_trashCan

#### 5.5.1 Detailed Description

The Graphic User Interface (GUI) regroups everything related to the manipulation of the environment. It notably handles the LeapmotionPointer, the Buttons, the TrashCan and all MovableStructures. It is also the link between all those elements

### 5.5.2 Member Function Documentation

#### 5.5.2.1 void GUI::AddButton ( Structure * *p_structure* ) [private]

Adds a button to the interface (max 3 for now)

- `p_structure` A pointer to the Structure that will be used to pop new MovableStructures from the added Button

#### 5.5.2.2 void GUI::CheckForPinchedStructure ( ) [private]

Checks for every Structure if it is being pinched by the LeapmotionPointer

#### 5.5.2.3 void GUI::CleanUp ( )

Cleans up everything

#### 5.5.2.4 void GUI::DroppedStructure ( unsigned int *buttonID* )

Notifies the GUI a Structure has been dropped and pops a new one on the corresponding button if the ID is valid

- `buttonID` The ID of the Button from which the dropped MovableStructure comes

#### 5.5.2.5 std::vector< Position > GUI::GetAllRoombotsPositions ( )

Returns a vector of the positions of all the RoomBots

#### 5.5.2.6 void GUI::Init ( )

Initializes the Graphic User Interface by initializing the LeapmotionPointer and adding Buttons

#### 5.5.2.7 size_t GUI::NButtons ( )

Returns the number of Buttons in the GUI

#### 5.5.2.8 void GUI::PopStructure ( unsigned int *buttonID* ) [private]

Pops a new structure in a Button

- `buttonID` The ID of the Button where to pop a new MovableStructure

#### 5.5.2.9 void const GUI::Render ( const glm::mat4 & *VP* )

Renders (draws) the Buttons, the Structures and the LeapmotionPointer

#### 5.5.2.10 void GUI::RotateStructure ( bool *clockWise* )

Rotates the MovableStructure currently assigned to the LeapmotionPointer

#### 5.5.2.11 void GUI::Update ( bool *mode* )

Updates the positions of the Structures and the LeapmotionPointer

- `mode` The current Simulator mode.

#### 5.5.2.12 void GUI::UpdatePointer ( bool *mode* ) [private]

Updates the LeapmotionPointer's position depending on the Leapmotion Controller's data

- `mode` the current Simulator mode

#### 5.5.2.13 void GUI::UpdateWorldMatrix ( const glm::mat4 & *worldMatrix* )

Updates the scene's world matrix. Used to change the behaviour of the LeapmotionPointer depending on the current mode

### 5.5.3 Member Data Documentation

#### 5.5.3.1 std::vector<const Button*> GUI::d_buttons [private]

The Buttons contained in the scene.

#### 5.5.3.2 size_t GUI::d_nButtons = 0 [private]

The number of Buttons in the scene.

#### 5.5.3.3 size_t GUI::d_nStructures = 0 [private]

The number of Structures in the scene.

#### 5.5.3.4 OBJModel* GUI::d_p_circle [private]

The pointer to the circle Model.

#### 5.5.3.5 OBJModel* GUI::d_p_hemi1 [private]

The pointer to the first hemisphere Model.

#### 5.5.3.6 OBJModel* GUI::d_p_hemi2 [private]

The pointer to the second hemisphere Model.

#### 5.5.3.7 LeapmotionPointer GUI::d_pointer [private]

The pointer using the Leapmotion device.

#### 5.5.3.8 std::vector<MovableStructure*> GUI::d_structures [private]

The Structures contained in the scene.

#### 5.5.3.9 TrashCan* GUI::d_trashCan [private]

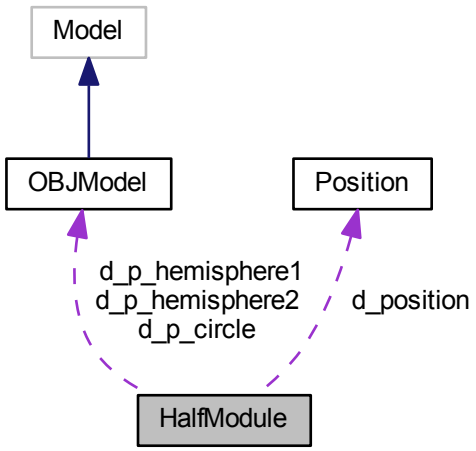
The documentation for this class was generated from the following files:

- RoombotsSimulator/GUI.hh
- RoombotsSimulator/GUI.cc

## 5.6 HalfModule Class Reference

`#include <HalfModule.hh>`

Collaboration diagram for HalfModule:



### Public Member Functions

- HalfModule (Position position, OBJModel *p_h1, OBJModel *p_h2, OBJModel *p_circle)
- HalfModule (int, int, int, OBJModel *p_h1, OBJModel *p_h2, OBJModel *p_circle)
- void Draw (const glm::mat4 &VP) const
- Position GetPosition () const
- void SetPosition (const Position &position)
- void CleanUp ()

### Private Attributes

- Position d_position

  *The relative coordinates within the Structure.*
- OBJModel * d_p_hemisphere1

  *A pointer to a up-oriented hemisphere Model.*
- OBJModel * d_p_hemisphere2

  *A pointer to a down-oriented hemisphere Model.*
- OBJModel * d_p_circle

  *A pointer to a circle model.*

## 5.6.1 Detailed Description

A HalfModule is the half of every RoomBot module

## 5.6.2 Constructor & Destructor Documentation

### 5.6.2.1 HalfModule::HalfModule ( Position *position*, OBJModel * *p_h1*, OBJModel * *p_h2*, OBJModel * *p_circle* )

Creates a new HalfModule

- `position` The position of the new HalfModule
- `p_h1` A pointer to the first hemi-sphere OBJModel used to draw the HalfModule
- `p_h2` A pointer to the second hemi-sphere OBJModel used to draw the HalfModule
- `p_circle` A pointer to the circle OBJModel used to draw all six faces of the HalfModule

### 5.6.2.2 HalfModule::HalfModule ( int *x*, int *y*, int *z*, OBJModel * *p_h1*, OBJModel * *p_h2*, OBJModel * *p_circle* )

Same as the first constructor, but with all three `int` used to create a new Position

## 5.6.3 Member Function Documentation

### 5.6.3.1 void HalfModule::CleanUp ( )

Cleans up the three Models used in the HalfModule

#### 5.6.3.2 void HalfModule::Draw ( const glm::mat4 & *VP* ) const

Draws the two hemispheres and the circle six times in different positions and orientations.

#### 5.6.3.3 Position HalfModule::GetPosition ( ) const

Returns the relative Position of the halfModule within its Structure

#### 5.6.3.4 void HalfModule::SetPosition ( const Position & *position* )

Directly sets the position of the Module. This should only be used when running the simulation or when the Module is free from any Structure

### 5.6.4 Member Data Documentation

#### 5.6.4.1 OBJModel* HalfModule::d_p_circle [private]

A pointer to a circle model.

#### 5.6.4.2 OBJModel* HalfModule::d_p_hemisphere1 [private]

A pointer to a up-oriented hemisphere Model.

#### 5.6.4.3 OBJModel* HalfModule::d_p_hemisphere2 [private]

A pointer to a down-oriented hemisphere Model.

#### 5.6.4.4 Position HalfModule::d_position [private]

The relative coordinates within the Structure.

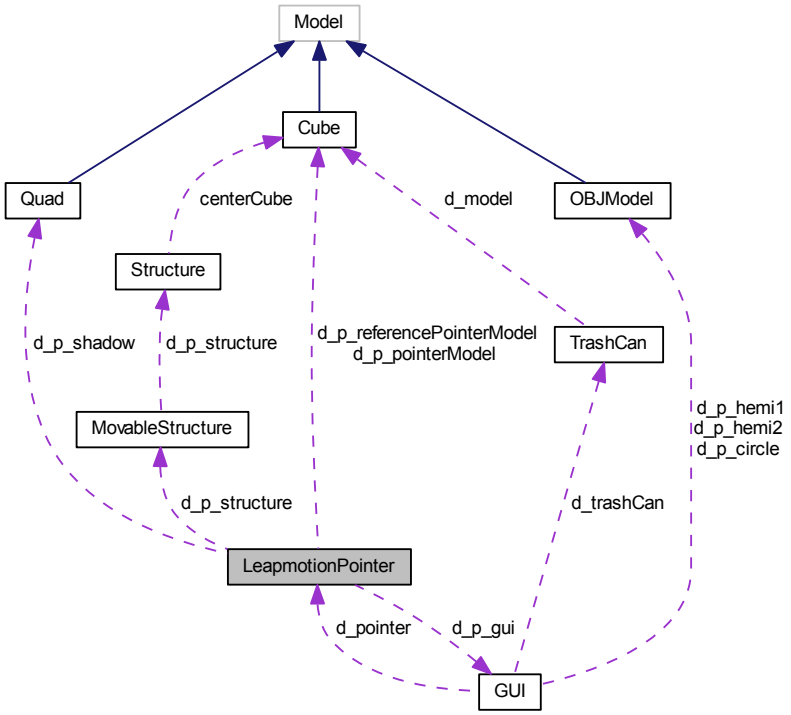
The documentation for this class was generated from the following files:

- RoombotsSimulator/HalfModule.hh
- RoombotsSimulator/HalfModule.cc

## 5.7 LeapmotionPointer Class Reference

`#include <LeapmotionPointer.hh>`

Collaboration diagram for LeapmotionPointer:




### Public Member Functions

- void Init (GUI *p_gui)
- void update (bool mode)
- void UpdateWorldMatrix (const glm::mat4 &worldMatrix)
- void Draw (const glm::mat4 &VP) const
- bool Pinching () const
- glm::vec3 Position () const
- void AssignStructure (MovableStructure *p_structure)
- MovableStructure * AssignedStructure () const
- void RotateStructure (bool clockWise)
- void CleanUp ()

### Private Member Functions

- Leap::Vector AdaptToMode (Leap::Vector right_hand_pos, bool mode)

### Private Attributes

- Leap::Controller d_controller

  *The object allowing us to get data from the Leapmotion device.*
- glm::vec3 d_position

  *The position of the LeapmotionPointer.*
- MovableStructure * d_p_structure = NULL
- glm::mat4 d_invertedWorldMatrix = glm::mat4()
- bool d_init = false

  *Set to true once initialized.*
- Cube * d_p_pointerModel
- Cube * d_p_referencePointerModel

  *the reference Cube that doesn't change size*
- Quad * d_p_shadow

  *the Quad projected on the floor, helping perceive depth*
- GUI * d_p_gui

## 5.7.1 Detailed Description

The LeapmotionPointer is an object allowing to interact with the virtual environment by using the Leapmotion, a gesture-recognition device. It uses the pinching gesture to grab MovableStructures

## 5.7.2 Member Function Documentation

#### 5.7.2.1 Leap::Vector LeapmotionPointer::AdaptToMode ( Leap::Vector *right_hand_pos,* bool *mode* ) [private]

Adapts the offset, the limits and the sensitivity of the pointer depending on the current mode

- `right_hand_pos` The current position of the user's right-most hand
- `mode` The current Simulator mode

#### 5.7.2.2 MovableStructure * LeapmotionPointer::AssignedStructure ( ) const

Returns a pointer to the currently assigned MovableStructure

#### 5.7.2.3 void LeapmotionPointer::AssignStructure ( MovableStructure * *p_structure* )

Assigns a MovableStructure to the pointer

- `p_structure` A pointer to the desired MovableStructure

#### 5.7.2.4 void LeapmotionPointer::CleanUp ( )

Cleans up the three models used to represent the Pointer

#### 5.7.2.5 void LeapmotionPointer::Draw ( const glm::mat4 & *VP* ) const

Draws the LeapmotionPointer's Model

#### 5.7.2.6 void LeapmotionPointer::Init ( GUI * *p_gui* )

Initializes the LeapmotionPointer Its offset, its position (set to (0,0,0)) and its Model

- `p_gui` A pointer to the GUI that uses the pointer

#### 5.7.2.7 bool LeapmotionPointer::Pinching ( ) const

returns whether or not the user's rightmost hand is pinching

#### 5.7.2.8 glm::vec3 LeapmotionPointer::Position ( ) const

Returns the position of the user's rightmost hand or the last captured position if the hand is out of reach

#### 5.7.2.9 void LeapmotionPointer::RotateStructure ( bool *clockWise* )

Rotates the MovableStructure currently assigned to the pointer

#### 5.7.2.10 void LeapmotionPointer::update ( bool *mode* )

Updates the the LeapmotionPointer's position and drags the pinched Structure if there is one

- `mode` The current Simulator mode

#### 5.7.2.11 void LeapmotionPointer::UpdateWorldMatrix ( const glm::mat4 & *worldMatrix* )

Updates the _invertedWorldMatrix with the one from the main

### 5.7.3 Member Data Documentation

#### 5.7.3.1 Leap::Controller LeapmotionPointer::d_controller [private]

The object allowing us to get data from the Leapmotion device.

#### 5.7.3.2 bool LeapmotionPointer::d_init = false [private]

Set to true once initialized.

#### 5.7.3.3 glm::mat4 LeapmotionPointer::d_invertedWorldMatrix = glm::mat4() [private]

This matrix is the inverse from the Scene's WorldMatrix. It allows the user to move around in the room and move Structures more easily

#### 5.7.3.4 GUI* LeapmotionPointer::d_p_gui [private]

A pointer to the GUI so it can tell it to add a new Structure to a button once it's dropped

#### 5.7.3.5 Cube* LeapmotionPointer::d_p_pointerModel [private]

the Cube representing the LeapmotionPointer and changing size depending on its pinching value

#### 5.7.3.6 Cube* LeapmotionPointer::d_p_referencePointerModel [private]

the reference Cube that doesn't change size

#### 5.7.3.7 Quad* LeapmotionPointer::d_p_shadow [private]

the Quad projected on the floor, helping perceive depth

#### 5.7.3.8 MovableStructure* LeapmotionPointer::d_p_structure = NULL [private]

A pointer to the Structure being dragged (NULL if no Structure is being dragged)

#### 5.7.3.9 glm::vec3 LeapmotionPointer::d_position [private]

The position of the LeapmotionPointer.

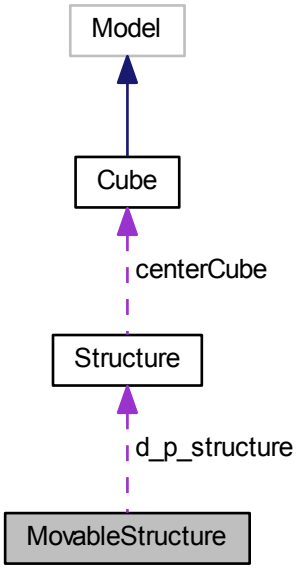
The documentation for this class was generated from the following files:

- RoombotsSimulator/LeapmotionPointer.hh
- RoombotsSimulator/LeapmotionPointer.cc

## 5.8 MovableStructure Class Reference

`#include <MovableStructure.hh>`

Collaboration diagram for MovableStructure:




### Public Member Functions

- MovableStructure (Structure *p_structure, glm::vec3 position, int ID, unsigned int buttonID)
- void Rotate (bool clockWise)
- bool CloseEnough (glm::vec3 position) const
- bool CloseEnough (glm::vec3 position, float distance) const
- void Drop ()
- void Drag (const glm::vec3 &position)
- void Draw (const glm::mat4 &VP) const
- Position GetPosition () const
- unsigned int LinkedButtonID () const
- std::vector< Position > RoombotsPositions () const

### Private Member Functions

- void SetCenterOffset ()
- void SetPosition (glm::vec3 position)

### Private Attributes

- const Structure * d_p_structure

  *The pointer to the Structure to be moved.*
- glm::vec3 d_position

  *The structure's position.*
- int d_ID

  *Its ID.*
- unsigned int d_buttonID
- bool d_moving = false

  *Whether the Structure is moving or not. (used in Drop())*
- int d_rotation = 0

## 5.8.1 Detailed Description

This class encapsulates a Roombot Structure as something that interact with the buttons and the Leapmotion↩ Pointer. Indeed, Structures are static sets of Roombots Module and are not meant to be moved around.

This encapsulation allows to use the same Structure for all similar MovableStructures. For instance, all 'chairs' MovableStructures use the same Structure through a pointer.

All the visible Structures when running the software are actually MovableStructures.

## 5.8.2 Constructor & Destructor Documentation

### 5.8.2.1 MovableStructure::MovableStructure ( Structure * *p_structure,* glm::vec3 *position,* int *ID,* unsigned int *buttonID* )

## 5.8.3 Member Function Documentation

### 5.8.3.1 bool MovableStructure::CloseEnough ( glm::vec3 *position* ) const

Checks if a position is close enough from the MovableStructure's Structure's center

- `position` The reference position

**Returns**

true if `position` is within the default drag radius

### 5.8.3.2 bool MovableStructure::CloseEnough ( glm::vec3 *position,* float *distance* ) const

Checks if a position is at most at a certain distance from the MovableStructure's Structure's center

- `position` The position used to check the distance

**Returns**

true if `position` is within `distance` from the center

#### 5.8.3.3 void MovableStructure::Drag ( const glm::vec3 & *position* )

Moves the Structure to the position passed in argument

- `position` The target position where to drag the currently addigned MovableStructure

#### 5.8.3.4 void MovableStructure::Draw ( const glm::mat4 & *VP* ) const

Draws the Structure

#### 5.8.3.5 void MovableStructure::Drop ( )

Imidiately Drops the Structure where the shadow is drawn There is no movement of the Model, it simply "teleports" on the ground

#### 5.8.3.6 Position MovableStructure::GetPosition ( ) const

Returns the reference position of the Structure

#### 5.8.3.7 unsigned int MovableStructure::LinkedButtonID ( ) const

Returns the ID of the button from which the MovableStructure comes

#### 5.8.3.8 std::vector< Position > MovableStructure::RoombotsPositions ( ) const

Returns the positions of all the Roombots of its Structure

#### 5.8.3.9 void MovableStructure::Rotate ( bool *clockWise* )

Rotates the MovableStructure

- `clockWise` True if the rotation must clock-wise, false otherwise

#### 5.8.3.10 void MovableStructure::SetCenterOffset ( ) [private]

Computes the center's offset from the lower left corner

#### 5.8.3.11 void MovableStructure::SetPosition ( glm::vec3 *position* ) [private]

Sets the new position of the the MovableStructure and ensures that it stays on the grid

### 5.8.4 Member Data Documentation

#### 5.8.4.1 unsigned int MovableStructure::d_buttonID [private]

The ID of the button from which the Structure was created Once dropped, it is set to -1 to "unlink" it from the button

#### 5.8.4.2 int MovableStructure::d_ID [private]

Its ID.

#### 5.8.4.3 bool MovableStructure::d_moving = false [private]

Whether the Structure is moving or not. (used in Drop())

#### 5.8.4.4 const Structure* MovableStructure::d_p_structure [private]

The pointer to the Structure to be moved.

#### 5.8.4.5 glm::vec3 MovableStructure::d_position [private]

The structure's position.

#### 5.8.4.6 int MovableStructure::d_rotation = 0 [private]

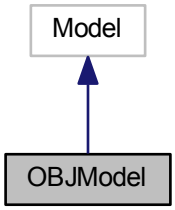
The documentation for this class was generated from the following files:

- RoombotsSimulator/MovableStructure.hh
- RoombotsSimulator/MovableStructure.cc

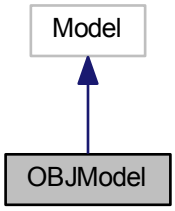
## 5.9 OBJModel Class Reference

`#include <OBJModel.hh>`

Inheritance diagram for OBJModel:



Collaboration diagram for OBJModel:




### Public Member Functions

- OBJModel (const std::string OBJFilename, const char ∗vShaderFileName, const char ∗fShaderFileName, const char ∗textureFileName, const glm::vec4 &color)

### Protected Member Functions

- virtual void SetVertices (std::vector< glm::vec3 > ∗vertices)
- virtual void SetUVs (std::vector< glm::vec2 > ∗uvs)

### Protected Attributes

- const std::string _objfilename

  *The name of the .obj file to import.*

## 5.9.1 Detailed Description

An OBJModel is a Model that gets its vertices from a .obj file. .obj (wavefront) files are standard graphics models files and are pretty easy to parse. But they also contain shading and texturing information that are not used in this software. Furthermore, .obj files sometimes contain polygons which would be ignored by the parser. However, quads would be interpreted as two triangles

This class allows to use complex models pretty easily.

## 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 OBJModel::OBJModel ( const std::string *OBJFilename*, const char ∗ *vShaderFileName*, const char ∗ *fShaderFileName*, const char ∗ *textureFileName*, const glm::vec4 & *color* )

Same constructor as the Model class but with the .obj filename added to it

- `OBJFilename` The name of the .obj file containing the OBJModel

### 5.9.3 Member Function Documentation

#### 5.9.3.1 void OBJModel::SetUVs ( std::vector< glm::vec2 > * *uvs* ) [protected], [virtual]

#### 5.9.3.2 void OBJModel::SetVertices ( std::vector< glm::vec3 > * *vertices* ) [protected], [virtual]

This method parses the .obj filename of the OBJModel and creates the corresponding vertices

### 5.9.4 Member Data Documentation

#### 5.9.4.1 const std::string OBJModel::_objfilename [protected]

The name of the .obj file to import.

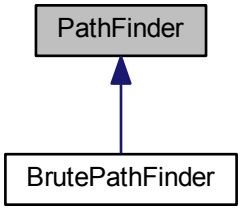
The documentation for this class was generated from the following files:

- RoombotsSimulator/OBJModel.hh
- RoombotsSimulator/OBJModel.cc

## 5.10 PathFinder Class Reference

```
#include <PathFinder.hh>
```

Inheritance diagram for PathFinder:




### Public Member Functions

- virtual void Run (Path &path, const Position &start, const Position &finish) const =0
- virtual std::string Name () const =0

### 5.10.1 Detailed Description

This class represents a path-finding algorithm. It is abstract, as we can design a lot of different pathi-finding algorithm, and all must implement the 'run' method described below.

The Path represents a series of successive Positions a Roombot Module must pass through to get to its final position

### 5.10.2 Member Function Documentation

#### 5.10.2.1 virtual std::string PathFinder::Name ( ) const [pure virtual]

This method simply returns the name of the path-finding algorithm

Implemented in BrutePathFinder.

#### 5.10.2.2 virtual void PathFinder::Run ( Path & *path,* const Position & *start,* const Position & *finish* ) const [pure virtual]

Creates a Path, a succession of Position from a Position to another. It uses an external Path and fills it

- `path` A reference to the Path to fill
- `start` The starting Position of the Path
- `finish` The finishing Position of the Path

Implemented in BrutePathFinder.

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

- RoombotsSimulator/PathFinder.hh

## 5.11 Position Class Reference

`#include <Position.hh>`

### Public Member Functions

- Position ()
- Position (int x, int y, int z)
- Position (Position ∗)
- Position (glm::vec3)
- Position operator+ (Position other) const
- Position operator- (Position other) const
- Position operator∗ (int factor) const
- void operator+= (Position other)
- void operator-= (Position other)
- void operator∗= (int factor)
- bool operator== (Position other) const
- bool operator!= (Position other) const
- int distanceTo (Position other) const
- void Print () const
- glm::vec3 ToGLM () const
- int x () const
- int y () const
- int z () const

### Private Attributes

- int d_x
- int d_y
- int d_z

## 5.11.1 Detailed Description

This class is a simple triplet of integers that represent a discrete Position. It allows easier handling of grid-bound objects. The grid is also Roombot-sized. It has basic operators and methods, as expected from such object

## 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 Position::Position ( )

This constructor creates a new Position at (0,0,0)

#### 5.11.2.2 Position::Position ( int *x*, int *y*, int *z* )

This constructor creates a new Position at (x,y,z)

#### 5.11.2.3 Position::Position ( Position * *other* )

Copy constructor

#### 5.11.2.4 Position::Position ( glm::vec3 *other* )

This constructor converts a 'glm::vec3' into a Position that is at the closest spot on the grid from the 'glm::vec3' passed as argument

## 5.11.3 Member Function Documentation

#### 5.11.3.1 int Position::distanceTo ( Position *other* ) const

Sums up the difference between all three coordinates of 'this' and 'other'

#### 5.11.3.2 bool Position::operator!= ( Position *other* ) const

#### 5.11.3.3 Position Position::operator* ( int *factor* ) const

#### 5.11.3.4 void Position::operator*= ( int *factor* )

#### 5.11.3.5 Position Position::operator+ ( Position *other* ) const

#### 5.11.3.6 void Position::operator+= ( Position *other* )

#### 5.11.3.7 Position Position::operator- ( Position *other* ) const

#### 5.11.3.8 void Position::operator-= ( Position *other* )

#### 5.11.3.9 bool Position::operator== ( Position *other* ) const

#### 5.11.3.10 void Position::Print ( ) const

Prints out the Position as (x,y,z)

#### 5.11.3.11 glm::vec3 Position::ToGLM ( ) const

Returns a 'glm::vec3' equivalent to the Position

#### 5.11.3.12 int Position::x ( ) const

#### 5.11.3.13 int Position::y ( ) const

#### 5.11.3.14 int Position::z ( ) const

### 5.11.4 Member Data Documentation

#### 5.11.4.1 int Position::d_x [private]

#### 5.11.4.2 int Position::d_y [private]

#### 5.11.4.3 int Position::d_z [private]

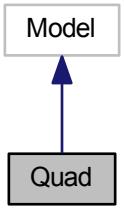
The documentation for this class was generated from the following files:

- RoombotsSimulator/Position.hh
- RoombotsSimulator/Position.cc

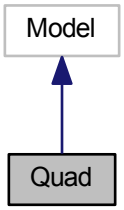
## 5.12 Quad Class Reference

`#include <Quad.hh>`

Inheritance diagram for Quad:




Collaboration diagram for Quad:




### Public Member Functions

- Quad (const std::string vShaderFileName, const std::string fShaderFileName, const std::string textureFile↩ Name, const glm::vec4 &color)
- Quad (const char ∗vShaderFileName, const char ∗fShaderFileName, const char ∗textureFileName, const glm::vec4 &color)

### Protected Member Functions

- virtual void SetVertices (std::vector< glm::vec3 > ∗vertices)
- virtual void SetUVs (std::vector< glm::vec2 > ∗uvs)

### 5.12.1 Constructor & Destructor Documentation

#### 5.12.1.1 Quad::Quad ( const std::string *vShaderFileName,* const std::string *fShaderFileName,* const std::string *textureFileName,* const glm::vec4 & *color* ) [inline]

#### 5.12.1.2 Quad::Quad ( const char * *vShaderFileName,* const char * *fShaderFileName,* const char * *textureFileName,* const glm::vec4 & *color* ) [inline]

### 5.12.2 Member Function Documentation

#### 5.12.2.1 void Quad::SetUVs ( std::vector< glm::vec2 > * *uvs* ) [protected], [virtual]

#### 5.12.2.2 void Quad::SetVertices ( std::vector< glm::vec3 > * *vertices* ) [protected], [virtual]

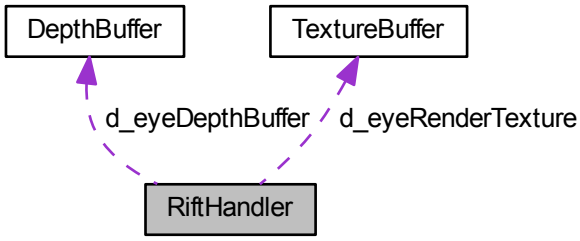
The documentation for this class was generated from the following files:

- RoombotsSimulator/Quad.hh
- RoombotsSimulator/Quad.cc

## 5.13 RiftHandler Class Reference

`#include <RiftHandler.hh>`

Collaboration diagram for RiftHandler:




### Public Member Functions

- void DisplayOnRift ()
- void Init (DisplayFunction)
- OVR::Matrix4f ovrViewProjMatrix ()
- glm::mat4 glmViewProjMatrix ()
- unsigned int ResolutionWidth ()
- unsigned int ResolutionHeight ()
- void CleanUp ()

### Private Attributes

- OVR::Matrix4f d_viewProjMatrix

  *THE viewProj matrix representing the orientation of the HMD.*
- ovrHmd d_hmd = nullptr

  *The code object representing the Rift's HMD.*
- ovrTrackingState d_trackingState

  *The tracking state of the HMD.*
- bool d_isVisible = false

  *Keeps the rendering to be displayed while the Rift hasn't been initialized.*
- ovrEyeRenderDesc d_EyeRenderDesc [2]

  *Store all the rendering information for both eyes.*
- TextureBuffer * d_eyeRenderTexture [2]

  *The textures where the image from both eyes will be stored.*
- DepthBuffer * d_eyeDepthBuffer [2]

  *The buffer where the depth information from both eyes will be stored.*
- GLuint d_mirrorFBO = 0

  *The FBO for the mirror display.*
- ovrGLTexture * d_mirrorTexture

  *The texture for the mirror display.*
- DisplayFunction d_displayFunction

  *the function that draws what will be displayed on the Rift's screen*

## 5.13.1 Member Function Documentation

#### 5.13.1.1 void RiftHandler::CleanUp ( )

Destroys and shuts the Rift virtual object down

#### 5.13.1.2 void RiftHandler::DisplayOnRift ( )

displays the rendering done in 'displayFunction' on the Rift's screen

#### 5.13.1.3 glm::mat4 RiftHandler::glmViewProjMatrix ( )

Same as 'ovrProjViewMatrix()' but returns a glm::mat4 instead

#### 5.13.1.4 void RiftHandler::Init ( DisplayFunction *disFunc* )

Initializes the Rift

- DisplayFunction the function that will do the actual rendering and will draw what is to be displayed on the Rift's screen

#### 5.13.1.5 OVR::Matrix4f RiftHandler::ovrViewProjMatrix ( )

Returns the view projection matrix that is built based on the Rift's orientation

#### 5.13.1.6 unsigned int RiftHandler::ResolutionHeight ( )

Returns the height of the Rift's resolution

#### 5.13.1.7 unsigned int RiftHandler::ResolutionWidth ( )

Returns the width of the Rift's resolution

### 5.13.2 Member Data Documentation

#### 5.13.2.1 DisplayFunction RiftHandler::d_displayFunction [private]

the function that draws what will be displayed on the Rift's screen

#### 5.13.2.2 DepthBuffer* RiftHandler::d_eyeDepthBuffer[2] [private]

The buffer where the depth information from both eyes will be stored.

#### 5.13.2.3 ovrEyeRenderDesc RiftHandler::d_EyeRenderDesc[2] [private]

Store all the rendering information for both eyes.

#### 5.13.2.4 TextureBuffer* RiftHandler::d_eyeRenderTexture[2] [private]

The textures where the image from both eyes will be stored.

#### 5.13.2.5 ovrHmd RiftHandler::d_hmd = nullptr [private]

The code object representing the Rift's HMD.

#### 5.13.2.6 bool RiftHandler::d_isVisible = false [private]

Keeps the rendering to be displayed while the Rift hasn't been initialized.

#### 5.13.2.7 GLuint RiftHandler::d_mirrorFBO = 0 [private]

The FBO for the mirror display.

#### 5.13.2.8 ovrGLTexture* RiftHandler::d_mirrorTexture [private]

The texture for the mirror display.

#### 5.13.2.9 ovrTrackingState RiftHandler::d_trackingState [private]

The tracking state of the HMD.

#### 5.13.2.10 OVR::Matrix4f RiftHandler::d_viewProjMatrix [private]

THE viewProj matrix representing the orientation of the HMD.

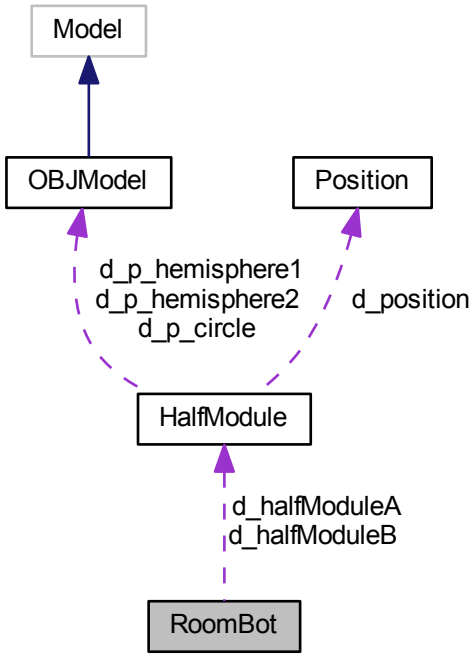
The documentation for this class was generated from the following files:

- RoombotsSimulator/RiftHandler.hh
- RoombotsSimulator/RiftHandler.cc

## 5.14 RoomBot Class Reference

`#include <RoomBot.hh>`

Collaboration diagram for RoomBot:



### Public Member Functions

- RoomBot (Position A, Position B, OBJModel *p_h1, OBJModel *p_h2, OBJModel *p_circle)
- RoomBot (int Ax, int Ay, int Az, int Bx, int By, int Bz, OBJModel *p_h1, OBJModel *p_h2, OBJModel *p_circle)
- glm::vec3 MiddlePosition () const
- void Draw (const glm::mat4 &VP) const
- Position PositionA () const
- Position PositionB () const

### Private Attributes

- HalfModule d_halfModuleA

  *The first half of the Roombot.*
- HalfModule d_halfModuleB

  *The second half of the Roombot.*

## 5.14.1 Detailed Description

The Roombot class is a simple class that encapsulates two halfModules. It is mainly an interface class between the HalfModules and other parts of the software

## 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 RoomBot::RoomBot ( Position *A*, Position *B*, OBJModel * *p_h1*, OBJModel * *p_h2*, OBJModel * *p_circle* )

Creates a new RoomBot

- `A` The Position of the first HalfModule
- `B` The Position of the second HalfModule
- p_h1,p_h2_p_circle Pointers to OBJModel used to construct the two HalfModule

#### 5.14.2.2 RoomBot::RoomBot ( int *Ax*, int *Ay*, int *Az*, int *Bx*, int *By*, int *Bz*, OBJModel * *p_h1*, OBJModel * *p_h2*, OBJModel * *p_circle* )

Same as the first constructor but with the six `int` used to create the two necessary `Position`

## 5.14.3 Member Function Documentation

#### 5.14.3.1 void RoomBot::Draw ( const glm::mat4 & *VP* ) const

Draws the RoomBot

#### 5.14.3.2 glm::vec3 RoomBot::MiddlePosition ( ) const [inline]

Returns the position of the middle between the two HalfModules

#### 5.14.3.3 Position RoomBot::PositionA ( ) const

Returns the Position of the first HalfModule

#### 5.14.3.4 Position RoomBot::PositionB ( ) const

Returns the Position of the second HalfModule

### 5.14.4 Member Data Documentation

#### 5.14.4.1 HalfModule RoomBot::d_halfModuleA [private]

The first half of the Roombot.

#### 5.14.4.2 HalfModule RoomBot::d_halfModuleB [private]

The second half of the Roombot.

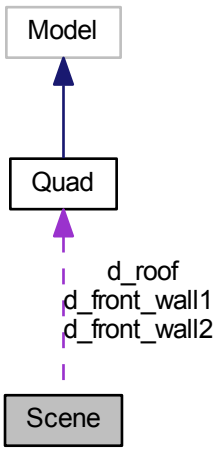
The documentation for this class was generated from the following files:

- RoombotsSimulator/RoomBot.hh
- RoombotsSimulator/RoomBot.cc

## 5.15 Scene Class Reference

`#include <Scene.hh>`

Collaboration diagram for Scene:



### Public Member Functions

- void Init (float roomSize)
- void const Render (const glm::mat4 &VP, bool drawRoof)
- void CleanUp ()

### Private Member Functions

- void AddModel (Model *sourceModel)

### Private Attributes

- std::vector< const Model * > d_models

  *The models contained in the scene.*
- size_t d_nModels

  *The number of models in the scene.*
- Quad * d_roof

  *The roof.*
- Quad * d_front_wall1

  *The first half of the back wall.*
- Quad * d_front_wall2

  *The second half of the back wall.*

## 5.15.1 Detailed Description

The Scene contains all the Models used to represent the scene in which the simulation takes place. The floor, the walls, the windows, the roof, the door and a skybox surrounding the room Aside from the roof, all Models are added polymorphicly. The roof get a special treatment because it is not necessarily drawn, depending on the current mode.

## 5.15.2 Member Function Documentation

#### 5.15.2.1 void Scene::AddModel ( Model * *sourceModel* ) [private]

Adds a Model to the Scene

- `sourceModel` The Model to add

#### 5.15.2.2 void Scene::CleanUp ( )

Cleans up all the Models of the Scene

#### 5.15.2.3 void Scene::Init ( float *roomSize* )

Initializes the Scene

- `roomSize` the size of the room in the middle of the Scene

#### 5.15.2.4 void const Scene::Render ( const glm::mat4 & *VP*, bool *drawRoof* )

Drawns all the elements of the Scene and the roof if the current mode is RoomView

- `VP` the Projection-View matrix
- `drawRoof` whether or not the roof should be drawn

### 5.15.3 Member Data Documentation

#### 5.15.3.1 Quad* Scene::d_front_wall1 [private]

The first half of the back wall.

#### 5.15.3.2 Quad* Scene::d_front_wall2 [private]

The second half of the back wall.

#### 5.15.3.3 std::vector<const Model*> Scene::d_models [private]

The models contained in the scene.

#### 5.15.3.4 size_t Scene::d_nModels [private]

The number of models in the scene.

#### 5.15.3.5 Quad* Scene::d_roof [private]

The roof.

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

- RoombotsSimulator/Scene.hh
- RoombotsSimulator/Scene.cc

## 5.16 ShaderLoader Class Reference

`#include <ShaderLoader.hh>`

### Public Member Functions

- GLuint CreateProgram (const char *VertexShaderFilename, const char *FragmentShaderFilename)

### Static Public Member Functions

- static std::string DefaultVertexShader ()
- static std::string DefaultFragmentShader ()

### Private Member Functions

- std::string ReadShader (const char *filename)
- GLuint CreateShader (GLenum shaderType, std::string source, char *shaderName)

## 5.16.1 Member Function Documentation

#### 5.16.1.1 GLuint ShaderLoader::CreateProgram ( const char * *VertexShaderFilename*, const char * *FragmentShaderFilename* )

creates a new program using the two shaders indicated by the names passed in arguments and returns its ID

**Returns**

the `GLuint` representing the new program

#### 5.16.1.2 GLuint ShaderLoader::CreateShader ( GLenum *shaderType*, std::string *source*, char * *shaderName* ) [private]

creates a new shader

- `shaderType` the type of shader to create
- `source` the `string` containing the shader program
- `shaderName` the name of the shader

**Returns**

the `GLuint` representing the new shader

#### 5.16.1.3 std::string ShaderLoader::DefaultFragmentShader ( ) [static]

Returns the following simple fragment shader :

```
#version 330 core
uniform sampler2D tex;
in vec2 uv;
out vec4 color;
void main(){
color = vec4(1.0,1.0,1.0,1.0);
}
```

It draws the whole model in plain opaque white

#### 5.16.1.4 std::string ShaderLoader::DefaultVertexShader ( ) [static]

Returns the following simple vertex shader :

```
#version 330 core
uniform mat4 MVP;
in vec3 vpoint;
in vec2 vtexcoord;
out vec2 uv;
void main(){
gl_Position = MVP * vec4(vpoint,1.0);
uv = vtexcoord;
}
```

It simply applies the MVP matrix to the vertex' position

#### 5.16.1.5 std::string ShaderLoader::ReadShader ( const char * *filename* ) [private]

reads a glsl program from a file

- `filename` the name of the file containing the glsl program

**Returns**

a `string` containing the whole file or "invalidShader" if the file couln't be read

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

- RoombotsSimulator/ShaderLoader.hh
- RoombotsSimulator/ShaderLoader.cc

## 5.17 Simulation Class Reference

`#include <Simulation.hh>`

### Public Member Functions

- void Initialize (const std::vector< Position > roombotsFinalPositions, PathFinder *pathFinder)
- bool NextStep ()
- void Draw (const glm::mat4 &VP)
- bool IsOver ()
- bool IsInitialized ()
- void Run ()

### Private Member Functions

- void Reset ()

### Private Attributes

- std::vector< Path > d_paths

  *The vector of all the Roombots' path.*
- std::vector< HalfModule > d_halfModules

  *The vector of all halfModules that will move during Simulation.*
- unsigned int d_currentStep = 0

  *The index of the current simulation step.*
- bool d_init = false

  *Whether or not the Simulation has been initialized.*
- bool d_over = true

  *Whether or not the Simulation is over. It is considered over when not initialized.*
- std::clock_t d_refClock

  *The reference clock used to time the calls to 'nextStep()'.*

## 5.17.1 Detailed Description

The Simulation encapsulates all that is needed to simulate the movements of the Roombot modules and allows to see how they would move around in the room to take their final place in the scene the User would have previously set up. It runs in parallel with the rest of the software once initialized but does not use any concurrent feature such as "std::thread".

IMPORTANT NOTE : All modules and their path must be retreive outside of the Simulation and must be given to it when initialized.

## 5.17.2 Member Function Documentation

### 5.17.2.1 void Simulation::Draw ( const glm::mat4 & *VP* )

Draws all modules needed to perform the Simulation

### 5.17.2.2 void Simulation::Initialize ( const std::vector< **Position** > *roombotsFinalPositions*, **PathFinder** * *pathFinder* )

Initializes the Simulation.

- `roombotsFinalPositions` A `vector` of all the Positions of all the Roombot
- `pathFinder` The PathFinder to use to compute all the Path

### 5.17.2.3 bool Simulation::IsInitialized ( ) [inline]

Returns whether or not the Simulation has been initialized

### 5.17.2.4 bool Simulation::IsOver ( ) [inline]

Returns whether or not the simulation is over.

#### 5.17.2.5 bool Simulation::NextStep ( )

Executes a step of the Simulation. Returns false if the simulation is over and true otherwise

#### 5.17.2.6 void Simulation::Reset ( ) [private]

Resets the Simulation

#### 5.17.2.7 void Simulation::Run ( )

Executes one steps of the Simulation if it's not over and ensures the execution of the successive steps is well-timed

### 5.17.3 Member Data Documentation

#### 5.17.3.1 unsigned int Simulation::d_currentStep = 0 [private]

The index of the current simulation step.

#### 5.17.3.2 std::vector<HalfModule> Simulation::d_halfModules [private]

The vector of all halfModules that will move during Simulation.

#### 5.17.3.3 bool Simulation::d_init = false [private]

Whether or not the Simulation has been initialized.

#### 5.17.3.4 bool Simulation::d_over = true [private]

Whether or not the Simulation is over. It is considered over when not initialized.

#### 5.17.3.5 std::vector<Path> Simulation::d_paths [private]

The vector of all the Roombots' path.

#### 5.17.3.6 std::clock_t Simulation::d_refClock [private]

The reference clock used to time the calls to 'nextStep()'.

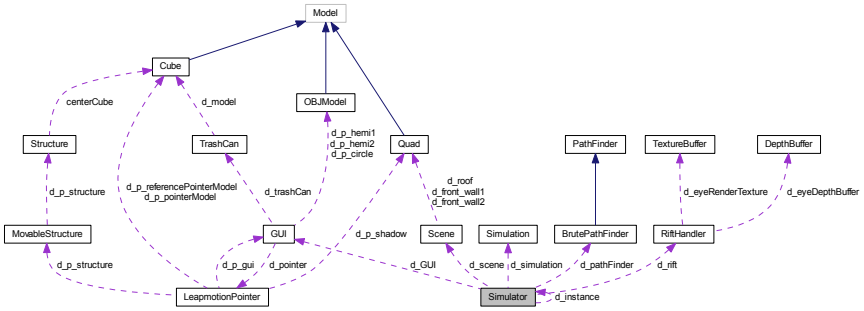
The documentation for this class was generated from the following files:

- RoombotsSimulator/Simulation.hh
- RoombotsSimulator/Simulation.cc

## 5.18 Simulator Class Reference

`#include <Simulator.hh>`

Collaboration diagram for Simulator:




### Public Member Functions

- void Init (int argc, char **argv, DisplayFunction display, DisplayFunction renderScene, void(*keyboard↩ Func)(unsigned char, int, int), void(*resizeFunc)(int, int), void(*closeFunc)())
- void Start ()
- void CleanUp ()
- void Resize (int w, int h)
- void RenderScene ()
- void Display ()
- void HandleKeyboard (unsigned char key, int x, int y)
- void Close ()
- glm::mat4 WorldViewMatrix ()

### Static Public Member Functions

- static Simulator & Instance ()

### Private Member Functions

- Simulator ()
- ~Simulator ()
- void Forward ()
- void Left ()
- void Backwards ()
- void Right ()
- void SwitchViewMode ()
- void InitRift (DisplayFunction function)
- void InitSimulation ()
- void MainLoop ()

### Private Attributes

- BrutePathFinder d_pathFinder

  *The pathFinder used to create all the paths before Simulation.*
- Simulation d_simulation

  *The Simulation that will run once the scene is set up.*
- Scene d_scene

  *The scene containing all the static elements of the environment.*
- RiftHandler d_rift

  *The object allowing easy use of the Oculus Rift.*
- GUI d_GUI

  *The Graphic User Interface allowing interaction with the environment.*
- unsigned int d_width = 0

  *The window's width.*
- unsigned int d_height = 0

  *The window's height.*
- glm::mat4 d_worldMatrix = glm::mat4()

  *The worldMatrix that changes depending on the current mode.*
- bool d_mode = true

  *viewing mode. false for "in-room" view, true for "box" view*
- bool d_running = true

  *Whether or not the Simulator is running.*
- int d_windowID

  *The OpenGL context window's ID.*

### Static Private Attributes

- static Simulator d_instance = Simulator()

  *The static instance of the Simulator, making it a singleton.*

## 5.18.1 Detailed Description

The Simulator class is the main class of the software and binds everything together. It is a singleton, as there's no need for multiple instances of it. It mainly contains the Scene and the Graphic User Interface and is in charge of handling all windowing system-related components, as well as handling the Oculus Rift and initializing the Simulation and run it when ready.

About the mode : The environment can interacted with through two modes, the 'Room-View' mode and the 'Box-↩ View' mode. The first one is just as if the User was sitting in the room and the second is just as if the User was sitting in front of a small box containing the scene. The Room-View mode allows to move freely in the room by using the "WASD" keys and the Box-View mode allows to grab and drop Structures from containers called "Buttons"

## 5.18.2 Constructor & Destructor Documentation

### 5.18.2.1 Simulator::Simulator ( ) [private]

The constructor is private to ensure the singleton properties

#### 5.18.2.2 Simulator::~Simulator ( ) [private]

### 5.18.3 Member Function Documentation

#### 5.18.3.1 void Simulator::Backwards ( ) [private]

#### 5.18.3.2 void Simulator::CleanUp ( )

Cleans up everything

#### 5.18.3.3 void Simulator::Close ( )

#### 5.18.3.4 void Simulator::Display ( )

Displays the rendered scene into the Oculus Rift

#### 5.18.3.5 void Simulator::Forward ( ) [private]

#### 5.18.3.6 void Simulator::HandleKeyboard ( unsigned char *key*, int *x*, int *y* )

Handles the keystrokes. IMPORTANT NOTE : This is based on the value of the pressed key on a QWERTZ keyboard.

#### 5.18.3.7 void Simulator::Init ( int *argc*, char ** *argv*, DisplayFunction *display*, DisplayFunction *renderScene*, void(*)(unsigned char, int, int) *keyboardFunc*, void(*)(int, int) *resizeFunc*, void(*)() *closeFunc* )

Initializes the Simulator by passing the various callback functions as argument.

- `argc` ,`argv` The first two are passed to the OpenGL context creation function.
- `display` The method that will be called at every rendering loop of the OpenGL context.
- `renderScene` The method that will be called everytime the scene has to be drawn
- `keyboardFunc` The method that handles keystrokes
- `resizeFunc` The method called everytime the window is resized

#### 5.18.3.8 void Simulator::InitRift ( DisplayFunction *function* ) [private]

Initializes the Scene Initializes the Oculus Rift

#### 5.18.3.9 void Simulator::InitSimulation ( ) [private]

Initializes the Simulation

#### 5.18.3.10 Simulator & Simulator::Instance ( ) [static]

Returns the unique instance of the singleton Simulator

#### 5.18.3.11 void Simulator::Left ( ) [private]

#### 5.18.3.12 void Simulator::MainLoop ( ) [private]

this method allows us to have control over the main OpenGL context loop. we call one iteration of the loop ourself

#### 5.18.3.13 void Simulator::RenderScene ( )

Renders everything

#### 5.18.3.14 void Simulator::Resize ( int *w*, int *h* )

Gets called when the windows is resized. It forces the window to a certain size

#### 5.18.3.15 void Simulator::Right ( ) [private]

#### 5.18.3.16 void Simulator::Start ( )

Starts the Simulator

#### 5.18.3.17 void Simulator::SwitchViewMode ( ) [private]

Switches between modes

#### 5.18.3.18 glm::mat4 Simulator::WorldViewMatrix ( )

Returns the world matrix

### 5.18.4 Member Data Documentation

#### 5.18.4.1 GUI Simulator::d_GUI [private]

The Graphic User Interface allowing interaction with the environment.

#### 5.18.4.2 unsigned int Simulator::d_height = 0 [private]

The window's height.

#### 5.18.4.3 Simulator Simulator::d_instance = Simulator() [static], [private]

The static instance of the Simulator, making it a singleton.

#### 5.18.4.4 bool Simulator::d_mode = true [private]

viewing mode. false for "in-room" view, true for "box" view

#### 5.18.4.5 BrutePathFinder Simulator::d_pathFinder [private]

The pathFinder used to create all the paths before Simulation.

#### 5.18.4.6 RiftHandler Simulator::d_rift [private]

The object allowing easy use of the Oculus Rift.

#### 5.18.4.7 bool Simulator::d_running = true [private]

Whether or not the Simulator is running.

#### 5.18.4.8 Scene Simulator::d_scene [private]

The scene containing all the static elements of the environment.

#### 5.18.4.9 Simulation Simulator::d_simulation [private]

The Simulation that will run once the scene is set up.

#### 5.18.4.10 unsigned int Simulator::d_width = 0 [private]

The window's width.

#### 5.18.4.11 int Simulator::d_windowID [private]

The OpenGL context window's ID.

#### 5.18.4.12 glm::mat4 Simulator::d_worldMatrix = glm::mat4() [private]

The worldMatrix that changes depending on the current mode.

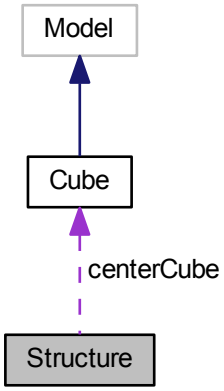
The documentation for this class was generated from the following files:

- RoombotsSimulator/Simulator.hh
- RoombotsSimulator/Simulator.cc

## 5.19 Structure Class Reference

`#include <Structure.hh>`

Collaboration diagram for Structure:




### Public Member Functions

- Structure (std::string sourceFilename, OBJModel *p_h1, OBJModel *p_h2, OBJModel *p_circle)
- void Draw (const glm::mat4 &VP) const
- glm::vec3 CenterOffset () const
- std::vector< Position > RoombotsPositions () const

### Private Attributes

- const std::string d_filename = ""

  *The .rbs file name from which the Structure is loaded.*
- glm::vec3 d_centerOffset

  *The difference between the Structure's position and its center.*
- std::vector< RoomBot > d_roomBots

  *The RoomBot modules of the Structure.*
- Cube * centerCube

### 5.19.1 Detailed Description

This class represents a set of Roombot Modules organized to form a particular structure. It is created from .rbs files, files that contain a list of positions for the Roombot modules. They take the form :

0 0 0
0 1 0
0 0 1

0 1 1
...

Where every pair of triplet is interpreted as a Roombot module.

All coordinates are relative to the Structure and the position (0,0,0) corresponds to the left-most, lowest, closest Roombot module.

IMPORTANT NOTE : There is no verification of the validity of the .rbs file. A wrongly-written file would create a physicaly impossible Structure.

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 Structure::Structure ( std::string *sourceFilename,* OBJModel * *p_h1,* OBJModel * *p_h2,* OBJModel * *p_circle* )

Imports a new OBJModel from a .obj file

- `sourceFilename` The name of the .bj file containing the model to import
- `p_h1,p_h2,p_circle Pointers` to the OBJModels needed by the RoomBot of the Structure

### 5.19.3 Member Function Documentation

#### 5.19.3.1 glm::vec3 Structure::CenterOffset ( ) const

Returns the Structure's center position which is an average of the roombots' positions

#### 5.19.3.2 void Structure::Draw ( const glm::mat4 & *VP* ) const

Draws the Structure

#### 5.19.3.3 std::vector< Position > Structure::RoombotsPositions ( ) const

Returns a vector of all the RoomBots' half-module's positions as AB AB AB...

### 5.19.4 Member Data Documentation

#### 5.19.4.1 Cube* Structure::centerCube [private]

#### 5.19.4.2 glm::vec3 Structure::d_centerOffset [private]

The difference between the Structure's position and its center.

#### 5.19.4.3 const std::string Structure::d_filename = "" [private]

The .rbs file name from which the Structure is loaded.

#### 5.19.4.4 std::vector<RoomBot> Structure::d_roomBots [private]

The RoomBot modules of the Structure.

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

- RoombotsSimulator/Structure.hh
- RoombotsSimulator/Structure.cc

## 5.20 TextureBuffer Struct Reference

`#include <TextureBuffer.hh>`

### Public Member Functions

- TextureBuffer (ovrHmd hmd, bool rendertarget, bool displayableOnHmd, OVR::Sizei size, int mipLevels, unsigned char *data, int sampleCount)
- OVR::Sizei GetSize (void) const
- void SetAndClearRenderSurface (DepthBuffer *dbuffer)
- void UnsetRenderSurface ()
- TextureBuffer (ovrHmd hmd, bool rendertarget, bool displayableOnHmd, OVR::Sizei size, int mipLevels, unsigned char *data, int sampleCount)
- OVR::Sizei GetSize (void) const
- void SetAndClearRenderSurface (DepthBuffer *dbuffer)
- void UnsetRenderSurface ()

### Public Attributes

- ovrSwapTextureSet * TextureSet
- GLuint texId
- GLuint fboId
- OVR::Sizei texSize

### 5.20.1 Constructor & Destructor Documentation

#### 5.20.1.1 TextureBuffer::TextureBuffer ( ovrHmd *hmd,* bool *rendertarget,* bool *displayableOnHmd,* OVR::Sizei *size,* int *mipLevels,* unsigned char * *data,* int *sampleCount* ) [inline]

#### 5.20.1.2 TextureBuffer::TextureBuffer ( ovrHmd *hmd,* bool *rendertarget,* bool *displayableOnHmd,* OVR::Sizei *size,* int *mipLevels,* unsigned char * *data,* int *sampleCount* ) [inline]

### 5.20.2 Member Function Documentation

#### 5.20.2.1 OVR::Sizei TextureBuffer::GetSize ( void ) const [inline]

#### 5.20.2.2 OVR::Sizei TextureBuffer::GetSize ( void ) const [inline]

#### 5.20.2.3 void TextureBuffer::SetAndClearRenderSurface ( DepthBuffer * *dbuffer* ) [inline]

#### 5.20.2.4 void TextureBuffer::SetAndClearRenderSurface ( DepthBuffer * *dbuffer* ) [inline]

#### 5.20.2.5 void TextureBuffer::UnsetRenderSurface ( ) [inline]

#### 5.20.2.6 void TextureBuffer::UnsetRenderSurface ( ) [inline]

### 5.20.3 Member Data Documentation

#### 5.20.3.1 GLuint TextureBuffer::fboId

#### 5.20.3.2 GLuint TextureBuffer::texId

#### 5.20.3.3 OVR::Sizei TextureBuffer::texSize

#### 5.20.3.4 ovrSwapTextureSet * TextureBuffer::TextureSet

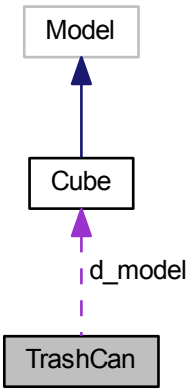
The documentation for this struct was generated from the following files:

- RoombotsSimulator/TextureBuffer.cc
- RoombotsSimulator/TextureBuffer.hh

## 5.21 TrashCan Class Reference

`#include <TrashCan.hh>`

Collaboration diagram for TrashCan:



### Public Member Functions

- TrashCan (glm::vec3 position)
- void Draw (const glm::mat4 &VP) const
- void CleanUp () const
- glm::vec3 Position () const

### Private Attributes

- Cube * d_model
  *the cube model used to represent the trashcan*
- const glm::vec3 d_position
  *its position within the scene*

## 5.21.1 Detailed Description

The TrashCan gives a way to remove MovableStructures from the Scene by dropping inside of it.

## 5.21.2 Constructor & Destructor Documentation

### 5.21.2.1 TrashCan::TrashCan ( glm::vec3 *position* )

Creates a new TrashCan

- `position` The position of the new TrashCan

## 5.21.3 Member Function Documentation

### 5.21.3.1 void TrashCan::CleanUp ( ) const

Cleans up the Model representing the TrashCan

### 5.21.3.2 void TrashCan::Draw ( const glm::mat4 & *VP* ) const

Draws the TrashCan

### 5.21.3.3 glm::vec3 TrashCan::Position ( ) const [inline]

Returns the trashCan's position

## 5.21.4 Member Data Documentation

### 5.21.4.1 Cube* TrashCan::d_model [private]

the cube model used to represent the trashcan

### 5.21.4.2 const glm::vec3 TrashCan::d_position [private]

its position within the scene

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

- RoombotsSimulator/TrashCan.hh
- RoombotsSimulator/TrashCan.cc

# Chapter 6

# File Documentation

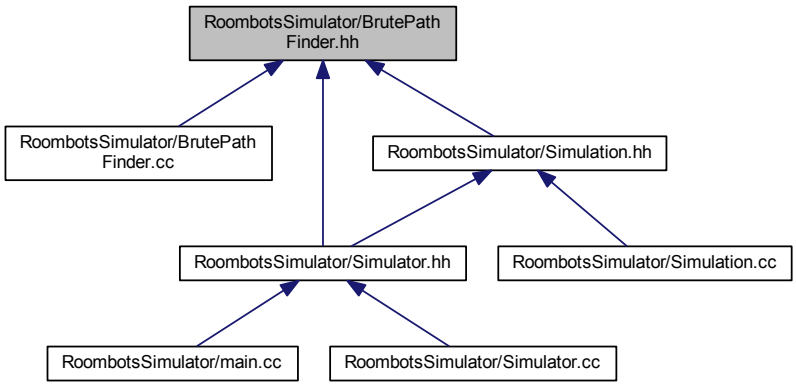
## 6.1 RoombotsSimulator/BrutePathFinder.cc File Reference

```
#include "BrutePathFinder.hh"
```

## 6.2 RoombotsSimulator/BrutePathFinder.hh File Reference

```
#include "PathFinder.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class BrutePathFinder

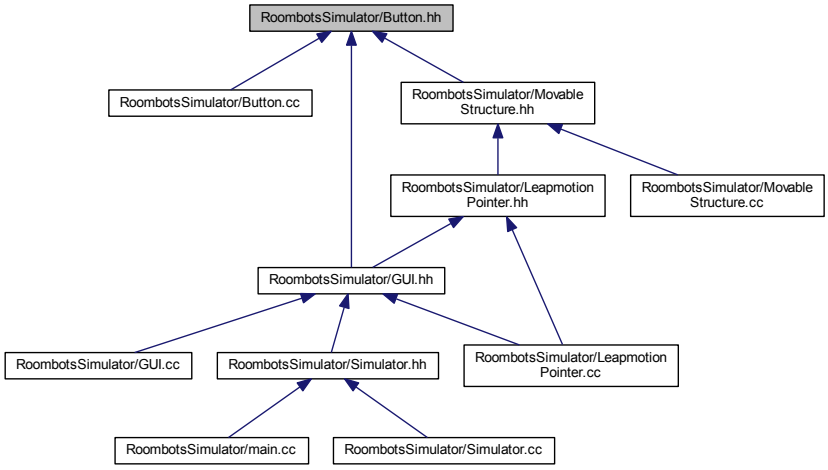
## 6.3 RoombotsSimulator/Button.cc File Reference

```
#include "Button.hh"
#include "Structure.hh"
```

## 6.4 RoombotsSimulator/Button.hh File Reference

```
#include "Cube.hh"
#include "Quad.hh"
#include "Structure.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

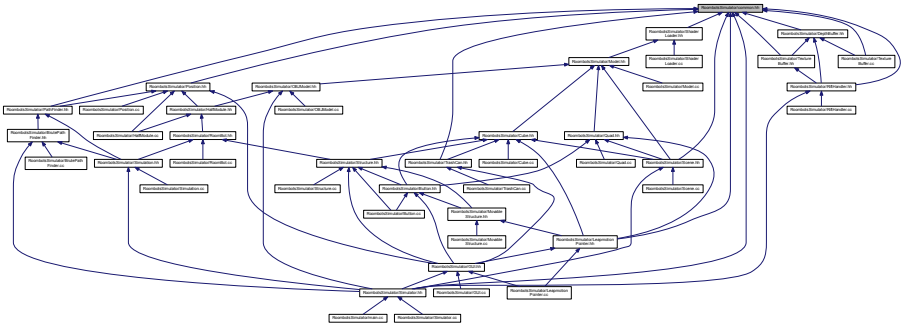
- class Button

## 6.5 RoombotsSimulator/common.hh File Reference

```
#include "Libraries\glew\glew.h"
```

```
#include "Libraries\glew\wglew.h"
#include "Libraries\freeglut\freeglut.h"
#include "glm\glm.hpp"
#include "glm\gtc\matrix_transform.hpp"
#include "glm\gtx\transform.hpp"
#include "glm\gtx\euler_angles.hpp"
#include "Libraries\soil\SOIL.h"
#include "Libraries\wgl\wglext.h"
#include "Libraries\wgl\glext.h"
#include "Libraries\OVR\OVR_CAPI_GL.h"
#include "Libraries\OVR\Extras\OVR_Math.h"
#include <Windows.h>
#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>
#include <vector>
#include <cmath>
#include <math.h>
#include <ctime>
#include "Libraries\Leap\Leap.h"
```

This graph shows which files directly or indirectly include this file:



## Macros

- #define MODULE_SIZE 0.12f

  *This file simply regroups includes of external libraries used in most other files to leave said files clean.*
- #define EYES_POSITION (29 ∗ MODULE_SIZE / 2)

  *The vertical position of the camera. It must be an odd multiple of MODULE_SIZE / 2.*
- #define ROOM_SIZE 5.0f

  *The size of the room.*
- #define BUTTON_SIZE 0.5f

  *The size of a button.*
- #define BUTTON_SEPARATION 1.0f

  *The horizontal distance between two buttons.*
- #define BUTTON_UP_START 1.8f

  *The vertical point where the buttons are positioned.*
- #define BUTTON_DEPTH_OFFSET -4.5f

  *The depth point where the buttons are positioned.*
- #define BUTTON_RIGHT_START 2.0f

  *The horizontal point from where the buttons are drawn.*

- #define LEAP_POINTER_SIZE 1.0f

  *The size of the LeapmotionPointer.*
- #define COORDINATE_SYSTEM_SCALE_CONVERSION 0.005f
- #define BOX_COORDINATE_SYSTEM_SCALE_CONVERSION 0.016f
- #define PINCHING_LIMIT 0.7f

  *The minimal pinching value to consider that the hand is pinching.*
- #define DRAG_RADIUS 0.4f

  *The minimal distance to grab a Structure.*
- #define TRASH_CAN_SIZE 2.0f

  *The size of the TrashCan.*

### 6.5.1 Macro Definition Documentation

#### 6.5.1.1 #define BOX_COORDINATE_SYSTEM_SCALE_CONVERSION 0.016f

A conversion factor to scale the data coming from the Leapmotion device to fit the Box-View mode characteristics

#### 6.5.1.2 #define BUTTON_DEPTH_OFFSET -4.5f

The depth point where the buttons are positioned.

#### 6.5.1.3 #define BUTTON_RIGHT_START 2.0f

The horizontal point from where the buttons are drawn.

#### 6.5.1.4 #define BUTTON_SEPARATION 1.0f

The horizontal distance between two buttons.

#### 6.5.1.5 #define BUTTON_SIZE 0.5f

The size of a button.

#### 6.5.1.6 #define BUTTON_UP_START 1.8f

The vertical point where the buttons are positioned.

#### 6.5.1.7 #define COORDINATE_SYSTEM_SCALE_CONVERSION 0.005f

A conversion factor to scale the data coming from the Leapmotion device to fit the Room-View mode characteristics

#### 6.5.1.8 #define DRAG_RADIUS 0.4f

The minimal distance to grab a Structure.

#### 6.5.1.9 #define EYES_POSITION (29 ∗ MODULE_SIZE / 2)

The vertical position of the camera. It must be an odd multiple of MODULE_SIZE / 2.

#### 6.5.1.10 #define LEAP_POINTER_SIZE 1.0f

The size of the LeapmotionPointer.

#### 6.5.1.11 #define MODULE_SIZE 0.12f

This file simply regroups includes of external libraries used in most other files to leave said files clean.

All the following macros are in meters The size of a a half of a Roombots module

#### 6.5.1.12 #define PINCHING_LIMIT 0.7f

The minimal pinching value to consider that the hand is pinching.

#### 6.5.1.13 #define ROOM_SIZE 5.0f

The size of the room.

#### 6.5.1.14 #define TRASH_CAN_SIZE 2.0f

The size of the TrashCan.

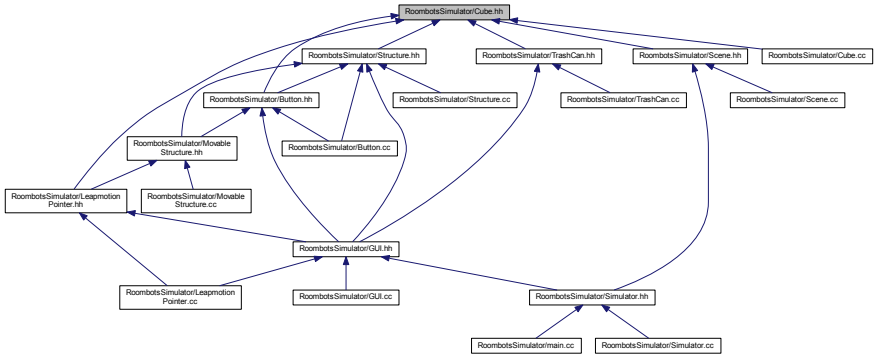
## 6.6 RoombotsSimulator/Cube.cc File Reference

```
#include "Cube.hh"
```

## 6.7 RoombotsSimulator/Cube.hh File Reference

```
#include "Model.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

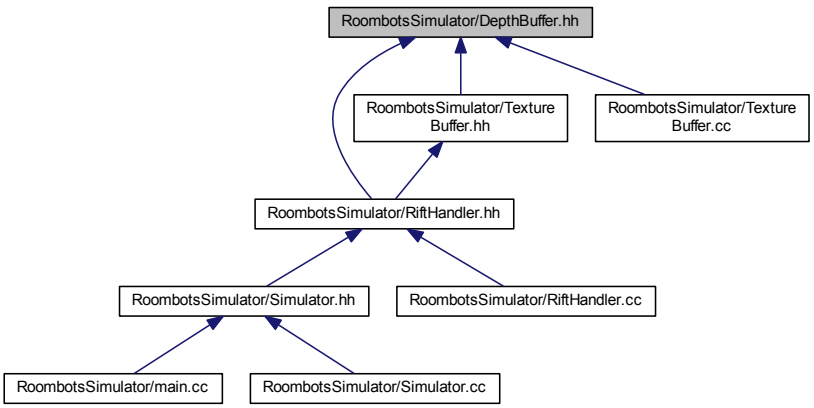
- class Cube

## 6.8 RoombotsSimulator/DepthBuffer.cc File Reference

## 6.9 RoombotsSimulator/DepthBuffer.hh File Reference

```
#include "common.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- struct DepthBuffer

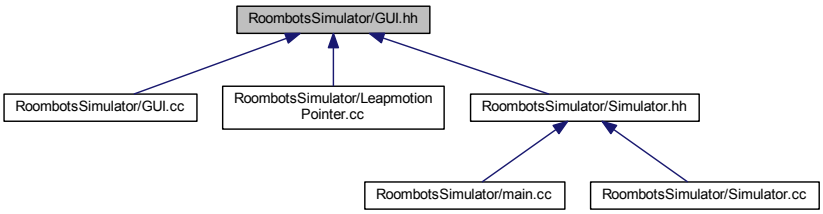
## 6.10 RoombotsSimulator/GUI.cc File Reference

```
#include "GUI.hh"
```

## 6.11 RoombotsSimulator/GUI.hh File Reference

```
#include "Button.hh"
#include "LeapmotionPointer.hh"
#include "Structure.hh"
#include "Position.hh"
#include "TrashCan.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class GUI

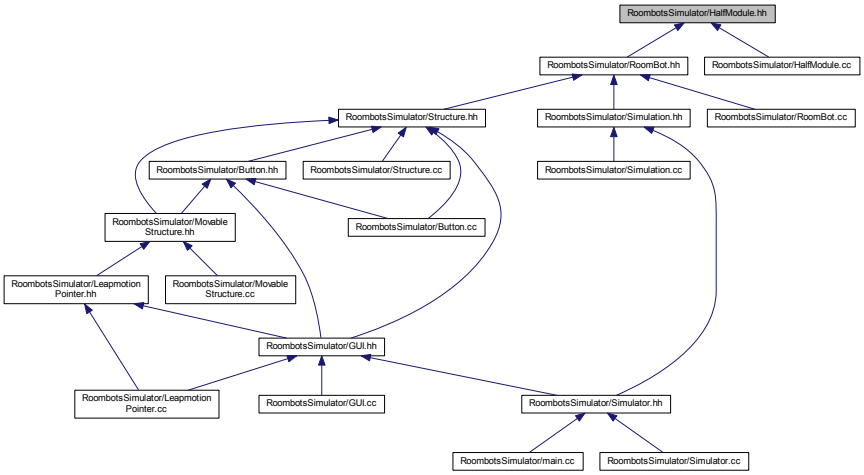
## 6.12 RoombotsSimulator/HalfModule.cc File Reference

```
#include "HalfModule.hh"
#include "Position.hh"
```

## 6.13 RoombotsSimulator/HalfModule.hh File Reference

```
#include "OBJModel.hh"
#include "Position.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class HalfModule

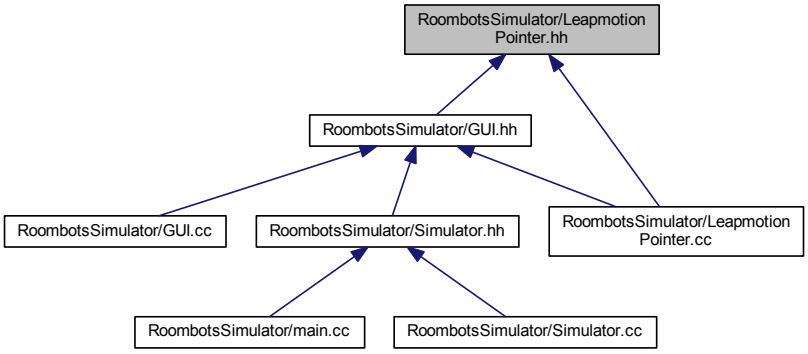
## 6.14 RoombotsSimulator/LeapmotionPointer.cc File Reference

```
#include "LeapmotionPointer.hh"
#include "GUI.hh"
```

## 6.15 RoombotsSimulator/LeapmotionPointer.hh File Reference

```
#include "common.hh"
#include "Cube.hh"
#include "Quad.hh"
#include "MovableStructure.hh"
```

This graph shows which files directly or indirectly include this file:



### Classes

- class LeapmotionPointer

## 6.16 RoombotsSimulator/main.cc File Reference

```
#include "Simulator.hh"
```

### Functions

- void display ()
- void renderScene ()
- void handleKeyboard (unsigned char key, int x, int y)
- void resize (int w, int h)
- void close ()
- int main (int argc, char **argv)

### 6.16.1 Function Documentation

#### 6.16.1.1 void close ( )

#### 6.16.1.2 void display ( )

**Author**

Valentin NIGOLIAN valentin.nigolian@epfl.ch Fall 2015

#### 6.16.1.3 void handleKeyboard ( unsigned char *key*, int *x*, int *y* )

#### 6.16.1.4 int main ( int *argc*, char ** *argv* )

#### 6.16.1.5 void renderScene ( )

#### 6.16.1.6 void resize ( int *w*, int *h* )

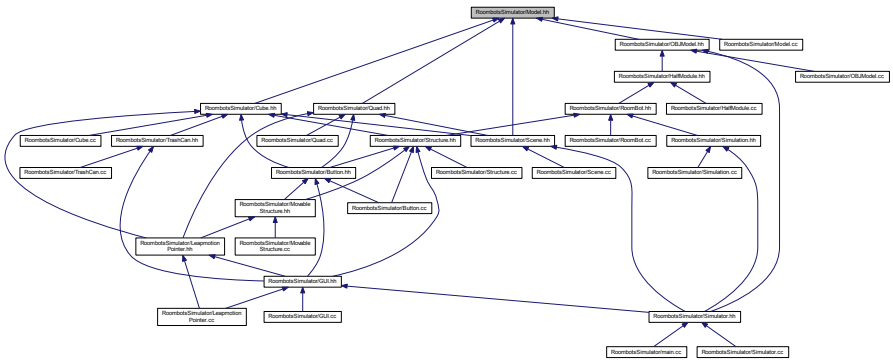
## 6.17 RoombotsSimulator/mainpage.dox File Reference

## 6.18 RoombotsSimulator/Model.cc File Reference

```
#include "Model.hh"
```

## 6.19 RoombotsSimulator/Model.hh File Reference

`#include "ShaderLoader.hh"`
This graph shows which files directly or indirectly include this file:



### Functions

- __declspec (align(16)) class Model

### 6.19.1 Function Documentation

#### 6.19.1.1 __declspec ( align(16) )

This class represents an object that can be drawn in an OpenGL context. All objects in this software that must be drawn must have at least one Model. It uses '__declspec(align(16))' to be aligned on the heap

IMPORTANT NOTE : all derived classes MUST call 'Init()' in their constructor in order to initialize them properly This constructor initializes all the parts of the Model

- `vShaderFileName` The name of the file containing the vertex shader
- `fShaderFileName` The name of the file containing the fragment shader
- `textureFileName` The name of the file containing the texture
- `color` The RGBA-formatted color of the model (if no texture is used)

This constructor is an alias of the first one that uses 'std::string' instead of 'const char*'

This is necessary when using '__declspec'

Same as above

Sets the model matrix

- `M` The model matrix defines the scale, rotation and translation of the model It doesn't change its vertices or definition but how and where it will appear in the scene

Draws the model into the scene.

- `VP` The Projection-View matrix, as required by the shaders

Draws the Model twice. Once normaly and once only with the lines

Cleans up everything that has been set up during initialization :

- the VBO
- the VAO
- the shading program
- the texture

  The two following methods are pure virtual as they are used to define the shape and texture coordinates of the model that has to be rendered and thus have to be defined for every model The vectors passed in argument in both methods are created in the 'Init' method IMPORTANT : both vectors MUST have the same size as OpenGL makes a correspondance between the elements of those vectors

<The model matrix

<Set on 'true' once initialized

< The vertex shader's file name

< The fragment shader's file name

< The texture's file name

<The color of the Model (black by default)

Calls 'SetVertices' and 'SetUVs' and makes sure both have the same size

Defines the vertices of the model. Each series of 3 vertices (each stored as a vec3) will compose a new triangle to be rendered. If the number of vertices isn't a multiple of three, the exceeding vertices will simply be ignored.

Defines the texture coordinates (also called 'UVs') of the model UVs are used to make a correspondance between a vertex and a position in the texture in order to determine the color of every vertex and interpolate between them to set the color of every pixel

Initializes everything. (called from the construtor)

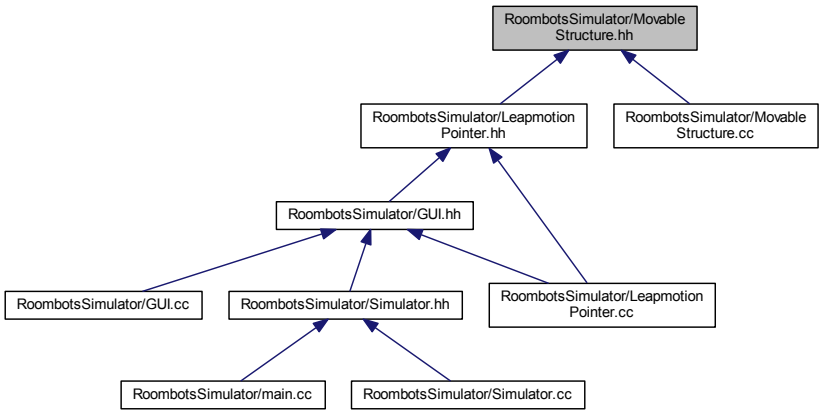
## 6.20 RoombotsSimulator/MovableStructure.cc File Reference

```
#include "MovableStructure.hh"
```

## 6.21 RoombotsSimulator/MovableStructure.hh File Reference

```
#include "Button.hh"
#include "Structure.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class MovableStructure

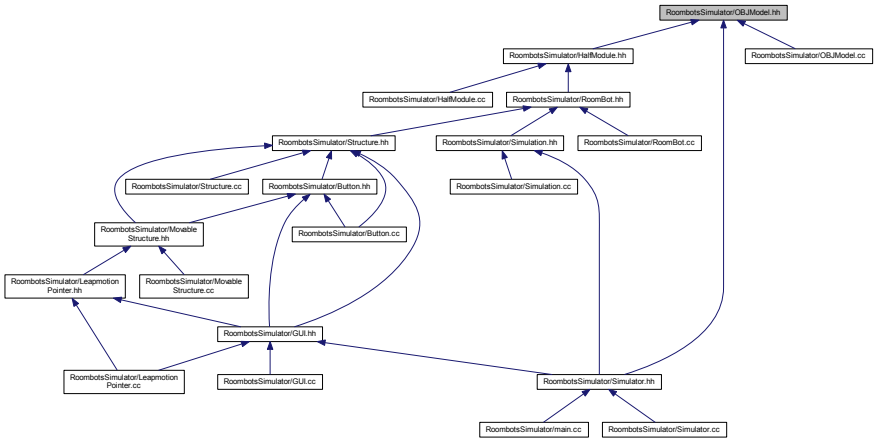
## 6.22 RoombotsSimulator/OBJModel.cc File Reference

```
#include "OBJModel.hh"
```

## 6.23 RoombotsSimulator/OBJModel.hh File Reference

```
#include "Model.hh"
#include <string>
```

This graph shows which files directly or indirectly include this file:



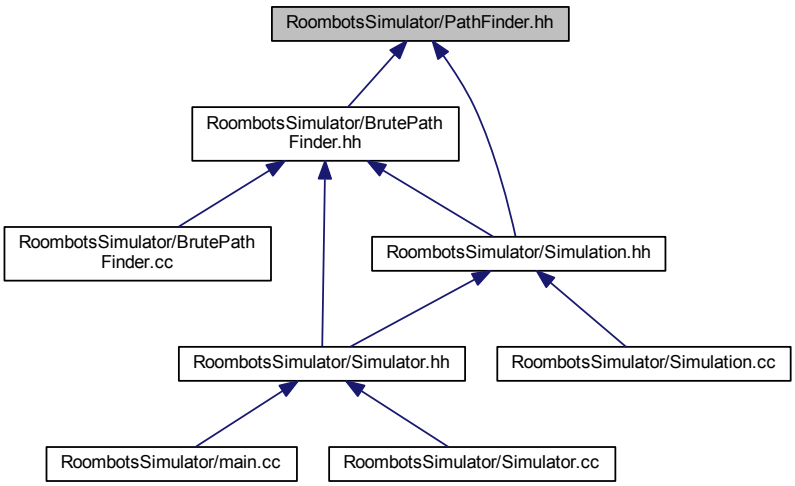

## Classes

- class OBJModel

# 6.24 RoombotsSimulator/PathFinder.hh File Reference

```
#include "common.hh"
#include "Position.hh"
```

This graph shows which files directly or indirectly include this file:




## Classes

- class PathFinder

### Typedefs

- typedef std::vector< Position > Path

### 6.24.1 Typedef Documentation

#### 6.24.1.1 typedef std::vector<Position> Path

**Author**

Valentin NIGOLIAN valentin.nigolian@epfl.ch Fall 2015

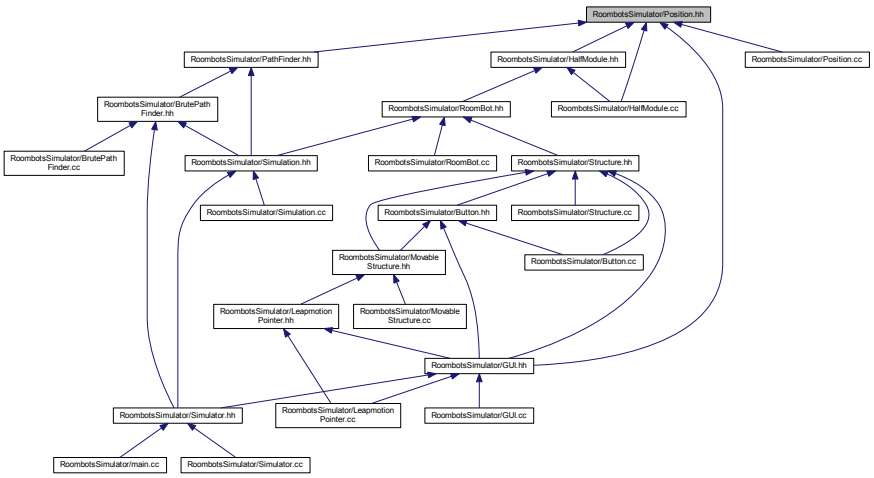
## 6.25 RoombotsSimulator/Position.cc File Reference

```
#include "Position.hh"
```

## 6.26 RoombotsSimulator/Position.hh File Reference

```
#include "common.hh"
#include "glm\glm.hpp"
#include "glm\gtc\matrix_transform.hpp"
#include "glm\gtx\transform.hpp"
#include "glm\gtx\euler_angles.hpp"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class Position

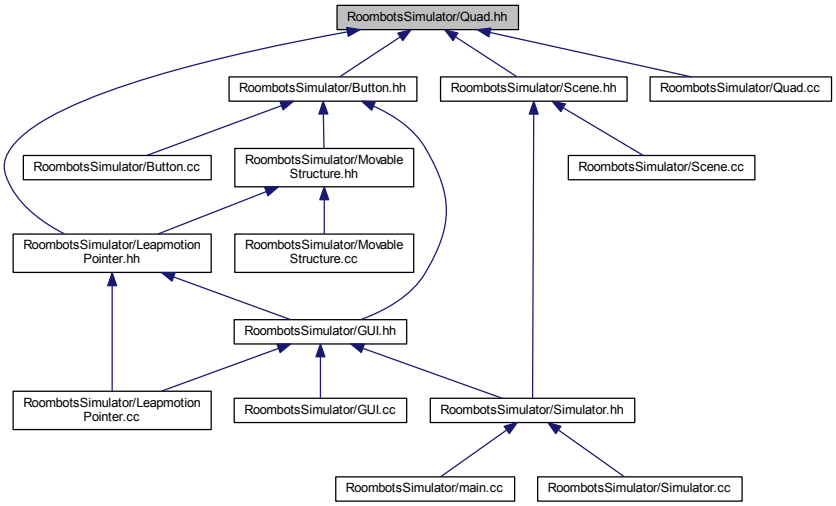
## 6.27 RoombotsSimulator/Quad.cc File Reference

```
#include "Quad.hh"
```

## 6.28 RoombotsSimulator/Quad.hh File Reference

```
#include "Model.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class Quad

## 6.29 RoombotsSimulator/RiftHandler.cc File Reference

```
#include "RiftHandler.hh"
```

### Functions

- glm::mat4 OVR_Mat4_to_GLM_mat4 (OVR::Matrix4f sourceMatrix)

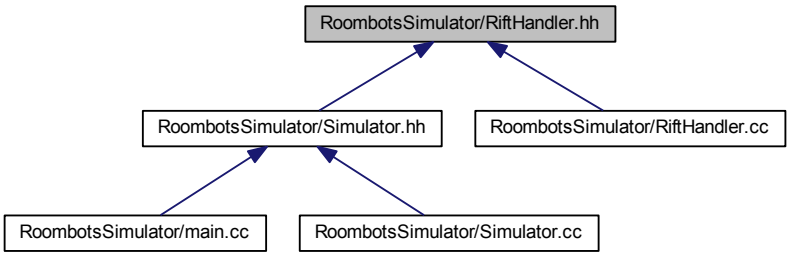
### 6.29.1 Function Documentation

#### 6.29.1.1 glm::mat4 OVR_Mat4_to_GLM_mat4 ( OVR::Matrix4f *sourceMatrix* )

## 6.30 RoombotsSimulator/RiftHandler.hh File Reference

```
#include "common.hh"
#include "DepthBuffer.hh"
#include "TextureBuffer.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class RiftHandler

### Typedefs

- typedef void(∗ DisplayFunction) ()

### 6.30.1 Typedef Documentation

#### 6.30.1.1 typedef void(∗ DisplayFunction) ()

This class makes the Rift much simpler to use. All that must be done is first initialise the Rift with 'Init(Display↩ Function)', where 'DisplayFunction' is the function that draws what we want to display on the Rift and then call 'DisplayOnRift' when we want to display something on the Rift. To take its orientation into account when drawing the scene (in the DisplayFunction passed in argument in 'Init()'), one must can use the 'ovrViewProjMatrix()' or 'glm↩ ViewProjMatrix()' methods to get the view-projection matrix in the form of a OVR::Matrix4f or a glm::mat4 matrix. The OVR::Matrix4f is how it is created by the Rift's SDK but the glm::mat4 one is easier to use with OpenGL

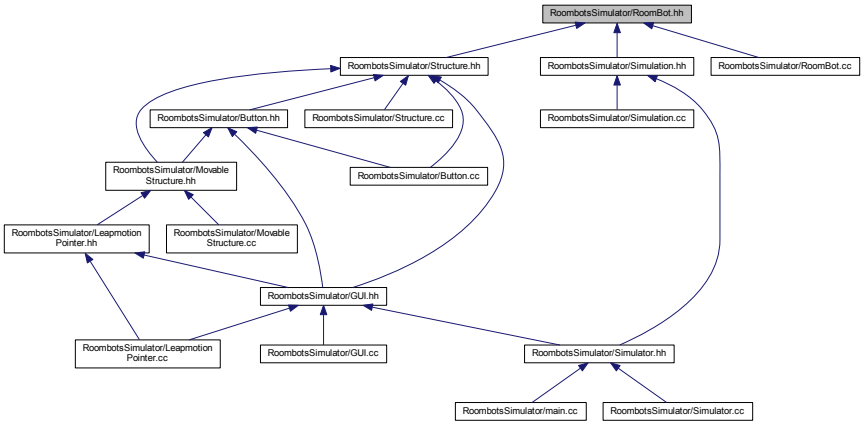
## 6.31 RoombotsSimulator/RoomBot.cc File Reference

```
#include "RoomBot.hh"
```

## 6.32 RoombotsSimulator/RoomBot.hh File Reference

```
#include "HalfModule.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class RoomBot

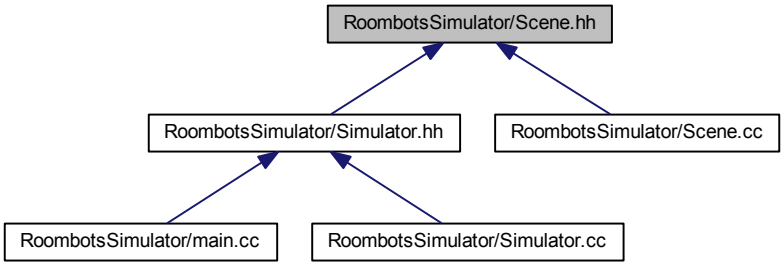
## 6.33 RoombotsSimulator/Scene.cc File Reference

```
#include "Scene.hh"
```

## 6.34 RoombotsSimulator/Scene.hh File Reference

```
#include "common.hh"
#include "Model.hh"
#include "Quad.hh"
#include "Cube.hh"
```

This graph shows which files directly or indirectly include this file:



### Classes

- class Scene

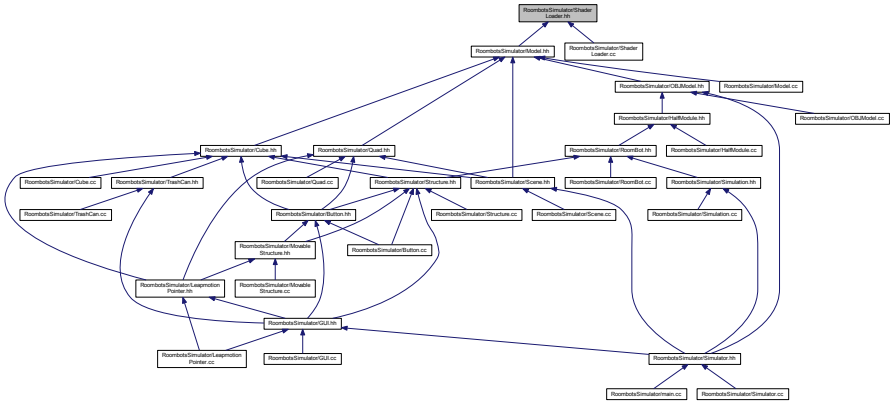
## 6.35 RoombotsSimulator/ShaderLoader.cc File Reference

```
#include "ShaderLoader.hh"
```

## 6.36 RoombotsSimulator/ShaderLoader.hh File Reference

```
#include "common.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class ShaderLoader

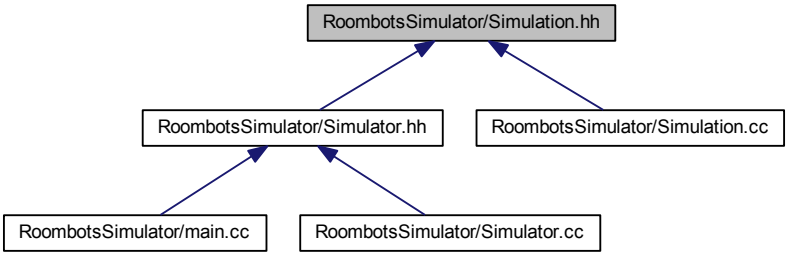
## 6.37 RoombotsSimulator/Simulation.cc File Reference

```
#include "Simulation.hh"
```

## 6.38 RoombotsSimulator/Simulation.hh File Reference

```
#include "RoomBot.hh"
#include "BrutePathFinder.hh"
#include "PathFinder.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class Simulation

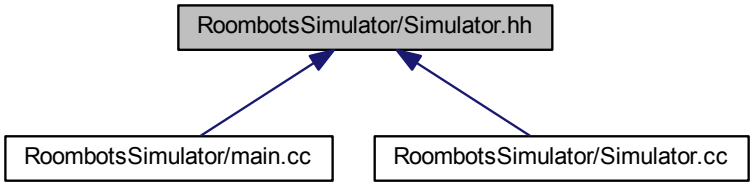
## 6.39 RoombotsSimulator/Simulator.cc File Reference

```
#include "Simulator.hh"
```

## 6.40 RoombotsSimulator/Simulator.hh File Reference

```
#include "common.hh"
#include "RiftHandler.hh"
#include "Scene.hh"
#include "GUI.hh"
#include "OBJModel.hh"
#include "BrutePathFinder.hh"
#include "Simulation.hh"
```

This graph shows which files directly or indirectly include this file:



### Classes

- class Simulator

## 6.41 RoombotsSimulator/Structure.cc File Reference

```
#include "Structure.hh"
```

### Functions

- int min (int a, int b)
- int min (int a, int b, int c)
- int max (int a, int b)
- int max (int a, int b, int c)

### 6.41.1 Function Documentation

#### 6.41.1.1 int max ( int *a,* int *b* )

#### 6.41.1.2 int max ( int *a,* int *b,* int *c* )

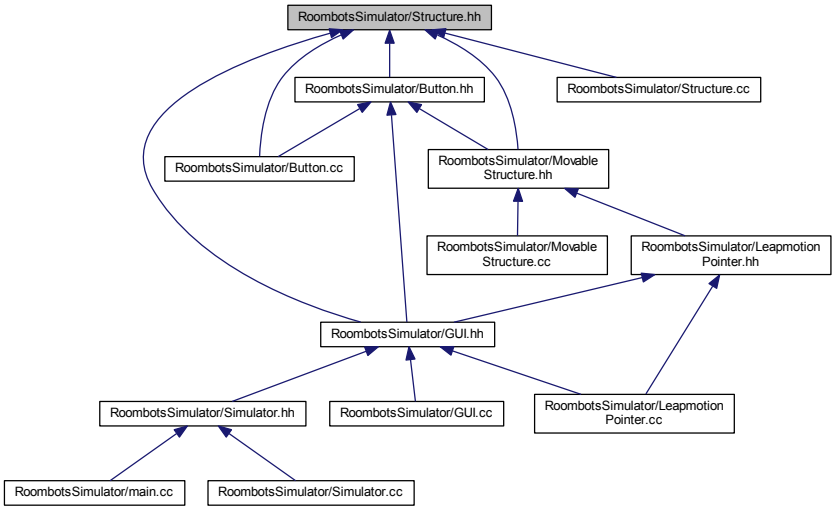
#### 6.41.1.3 int min ( int *a,* int *b* )

#### 6.41.1.4 int min ( int *a,* int *b,* int *c* )

## 6.42 RoombotsSimulator/Structure.hh File Reference

```
#include "RoomBot.hh"
#include "Cube.hh"
```

This graph shows which files directly or indirectly include this file:



### Classes

- class Structure

## 6.43 RoombotsSimulator/TextureBuffer.cc File Reference

```
#include "common.hh"
#include "DepthBuffer.hh"
```

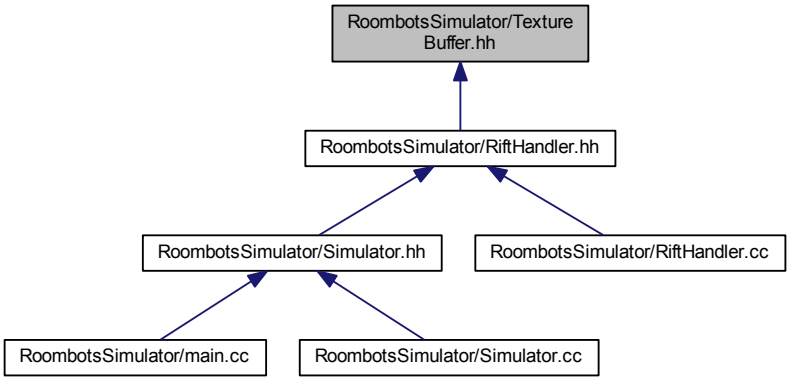
### Classes

- struct TextureBuffer

## 6.44 RoombotsSimulator/TextureBuffer.hh File Reference

```
#include "common.hh"
#include "DepthBuffer.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- struct TextureBuffer

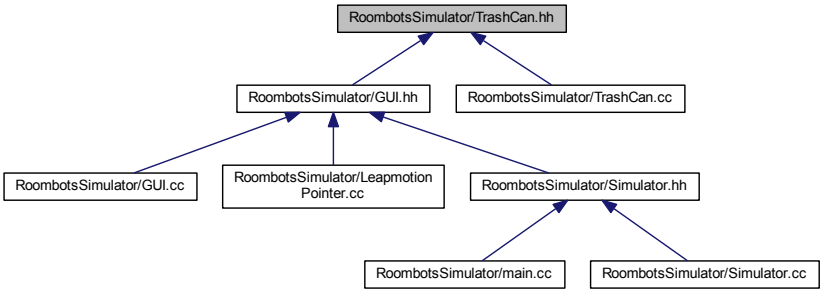
## 6.45 RoombotsSimulator/TrashCan.cc File Reference

```
#include "TrashCan.hh"
```

## 6.46 RoombotsSimulator/TrashCan.hh File Reference

```
#include "common.hh"
#include "Cube.hh"
```

This graph shows which files directly or indirectly include this file:




### Classes

- class TrashCan

# Index