DropletGUI

Generated by Doxygen 1.8.3.1

Mon May 13 2013 02:36:28

Contents

1 DropletGUI

1.1 Introduction

DropletGUI is an example client for the Droplets simulation library.

1.2 Dependencies

- Visual Studio 2010∗
- Bullet 2.80+
- EasyBMP 1.06
- Qt 4.8.x
- DropletSimLibrary

∗Visual Studio 2012 can also be used, however it has to use the Visual Studio 2010 compatability mode. See the build guide for more information.

1.3 Installation

Please consult the build guide for a more detailed of how to build DropletGUI.

1.3.1 Step 1: Obtaining Source Code

Source code for this project can be attained from the `cu-droplet Google Code page`. To download it you will need to have installed a `git client`.

1.3.2 Step 2: Obtaining Qt

DropletGUI requires Qt 4.8 in order to properly build and compile. Qt 4.8 and the recommended Visual Studio 2010 plug-in for Qt can currently be downloaded from qt-projects.org. It is highly recommended that you install Qt inside the default directory when prompted.

Due to major changes in Qt 5.0, DropletGUI is not compatible with Qt 5.0 and will not build or run correctly with it.

1.3.3 Step 3: Building DropletGUI

Building DropletGUI is fairly straight-forward. To do this, you simply perform the following:

1. Navigate to the project folder found in `DropletSimulator/DropletSimDemos/DropletGUI/vs2010/`
2. Open the Visual Studio solution file `DropletGUI.sln`
3. Select one of the build configurations. See below for an explanation of each.
4. Under the Build menu, select Build Solution. On a clean checkout this will force it to build all dependent libraries and may take several minutes.

1.4 How to contribute

TODO: Add policies on contributing.

1.4.1 Issue Tracker

Current known issues with the Droplets project can be found at the cu-droplet issues tracker on Google Code.

1.4.2 Contacting us.

TODO: Add primary contact information for the project.

2 Build Guide

This will be a full build guide.

2.1 Dependencies

- Visual Studio 2010
- Bullet 2.80+
- EasyBMP 1.06
- Qt 4.8
- DropletSimLibrary

2.1.1 Visual Studio 2012 warning

Due to a lack of Visual Studio 2012 support in Qt 4.8 it is recommended that you use Visual Studio 2010 in order to build DropletGUI. If you do decide to use Visual Studio 2012 it is important that you do not update the project files when given the opportunity and that you have installed Qt in the default directories. Failing to do either will result in a broken build environment that is unable to build the project.

2.2 Installing Qt

DropletGUI requires Qt 4.8 in order to properly build and compile. Qt 4.8 and the recommended Visual Studio 2010 plug-in for Qt can currently be downloaded from `qt-projects.org`. It is highly recommended that you install Qt in the default directory as the build environment depends on being able to find the Qt libraries, and if you are attempting to build DropletGUI under Visual Studio 2012 or without the Visual Studio plug-in it will be unable to locate the Qt libraries if they are not in the default path.

Due to major changes in Qt 5.0, DropletGUI is not compatible with Qt 5.0 and will not run correctly with it.

2.3 Building DropletGUI

Building DropletGUI is fairly straight-forward. To do this, you simply perform the following:

1. Navigate to the project folder found in `DropletSimulator/DropletSimDemos/DropletGUI/vs2010/`
2. Open the Visual Studio solution file `DropletGUI.sln`
3. Select one of the build configurations. See below for an explanation of each.
4. Under the Build menu, select Build Solution. On a clean checkout this will force it to build all dependent libraries and may take several minutes.

2.4 Building a .PDF Manual

In order to facilitate a wide variety of development environments, the solution has several build configurations available to it. They are organized in the following manner:

- Debug / Retail configurations have support for property sheets to allow you to specify a custom Qt directory and are not configured to support running the executable inside the debugger.
- DebugTeam / RetailTeam configurations use the pre-defined $(QTDIR) macro to use the default Qt directory. The Debug configurations build with full debugging symbols and with no optimizations enabled, whereas the Retail configurations build with no debugging symbols and full optimizations enabled. For performance reasons it is highly recommended that you use the Retail configurations if you are not actively developing the simulator.

2.5 Interacting with the debugging configuration

In order to facilitate interactive debugging inside Visual Studio, the DebugTeam and ReleaseTeam configurations also perform an additional step where they consolidate all files necessary to execute DropletGUI into the vs2010//bin// folder so it can execute it from inside the debugger. This offers huge productivity enhancements and guarantees that it always has access to common files that it depends on such as the shared Projector Images, but also can be a source of confusion while developing. If you are executing DropletGUI from within the debugger and wish to modify the assets, it is important that you keep copies outside of the vs2010//bin folder as they will be overwritten the next time you build the project.

3 Doxygen Guide

This contains instructions for how to use Doxygen to build the documentation.

3.1 Dependencies

- Doxygen
- GraphViz
- LaTex (or MiKTeX) to build PDF guides

3.2 Running Doxygen

Building documentation is straight forward. Assuming you have correctly installed Doxygen and GraphViz, you simply do the following:

1. Open DoxyWizard
2. Under the "File" menu, select "Open" and navigate to`DropletSimulator/DropletSimDemos/DropletGUI/docs/Doxyfile`
3. Select the "Run" tab
4. Click "Run Doxygen"

This will create two directories inside `DropletSimulator/DropletSimDemos/DropletGUI/docs/` - one named `html/` that contains a set of HTML documents that can be uploaded to a server and one titled `latex/` that contains a LaTeX-formatted manual.

3.3 Building a .PDF Manual

Once the above is done it is possible to generate a PDF version of the LaTeX manual by navigating to `DropletSimulator/DropletSimDemos/DropletGUI/docs/latex` and running `make` (under the Linux command line) or `make.bat` (under Windows). This will generate a PDF-formatted version of the manual in that folder named `refman.pdf` that can be safely copied elsewhere.

4 Supported Features

4.1 Droplets

- Six directions of linear movement
- In-place rotation
- Communication with customizable range
- Range and bearing from communications
- True color illumination
- RGB color sensing
- Self-righting

4.2 Simulator

- Realistic physics simulation using the Bullet physics library
- Cross-compiling programs between simulator and hardware
- Load projection images onto the arena
- Supports custom arenas
- Add spheres and cubes into arena
- Run heterogeneous programs on Droplets
- Add Droplets during simulation
- Track leg power status of Droplets
- Set of demo programs and blank program templates for the user
- Console version which can be compiled on UNIX systems

4.3 GUI

- Select setup files for parameters
  - Simulation speed
  - Arena dimensions
  - Custom arena file
  - Droplet radius
  - Projection image
  - Droplet program and number (randomly generated positions)
  - Droplet program and position
  - Object type, number, radius, mass, friction (randomly generated positions)
  - Object type, position, radius, mass, friction
- Add Droplets into simulation with desired number, radius, and program
- Select custom arena
- Specify dimensions of default arena rectangle
- Select projection image
- Refresh drop-down menus when new asset files are added
- Log certain statistics
- Re-launch simulator with new setting without re-launching application
- Playback control: pause, resume, reset
- 3D rendering with lighting using OpenGL
- Keyboard and mouse camera control
- Simulation speed control
- Toggled keybinding help menu
- Toggled HUD
- Debugging view mode
- High resolution screenshots

5 File Formats

DropletGUI uses a variety of plain text files to control run time properties. Below is a description of the various kinds of files used.

5.1 Experiments

Below are descriptions of files that can be modified to adjust how experiments run.

5.1.1 Setup Files

Setup files are found in `\DropletSimulator\DropletSimDemos\DropletGUI\assets\Setup\`. A setup file can specify droplet and arena variables. The simulator will just use a variable's default value if it isn't specified in the setup file. Each variable declaration must be on its own line. Please note that if your setup file adds droplets to the simulator, these are in addition to the number of droplets parameter.

Variables are specified by:

```
<variable name> <options>
```

Projections are specified by:

```
projecting <projection image>
```

Custom arena files are specified by:

```
arena <arena file name>
```

A group of droplets is specified by:

```
droplets <droplet program> <number of droplets>
```

To specify an exact starting position of an individual droplet:

```
droplets <droplet program> <x-coordinate> <y-coordinate>
```

Physical objects are similarly specified:

```
<object> <x> <y> <radius> <mass> <friction>
```

Note that radius, mass, and friction are optional. `<object>` can be either `cube` or `sphere`.

A group of objects is specified by:

```
<objects> <number> <radius> <mass> <friction>
```

Again, radius, mass, and friction are optional. `<objects>` can be either `cubes` or `spheres`.

Please refer to testSetup.txt in the Setup folder for an example.

5.1.2 Arena Files

Custom arena files are found in `\DropletSimulator\DropletSimDemos\DropletGUI\assets\Floors\`. Custom arena files specify information about a particular floor tile. Every tile declaration needs to be on its own line.

To specify a tile:

```
<x-coordinate> <y-coordinate> <top wall> <right wall> <bottom wall> <left wall>
```

X and Y coordinates represent the grid position of the tile. You do not have to account for the length of a tile.

Example grid:

```
| 0,2 | 1,2 | 2,2 |
| 0,1 | 1,1 | 2,1 |
| 0,0 | 1,0 | 2,0 |
```

The wall variables are either `yes` or `no`. They are specific to one tile.

Please refer to any of the example custom arena files in the Floors folder.

5.2 Advanced

Below are descriptions of files that control which assets DropletGUI uses while rendering objects. These should not be modified for normal use as doing so may result in undefined behavior.

5.2.1 Manifest file

The main manifest file contains a list that maps specific assets to specific objects used by the renderer and is located at `\DropletSimulator\DropletSimDemos\DropletGUI\assets\manifest.txt`.

The format for the file is:

```
<variable name> <value>
```

For each line `<variable name>` is one of the items indicated below, and `<value>`is the asset name to be assigned to it.

Texture names:

These correspond with a file located in `assets\Textures` and are to be specified *without* the file extension.

- `droplet_texture` is the texture to be applied to droplets
- `floor_texture` is the texture to be applied to floor tiles
- `wall_texture` is the texture to be applied to walls
- `tower_texture` is the texture to be applied to IR towers
- `object_texture` is the texture to be applied to sphere objects
- `object_cube_texture` is the texture to be applied to cube objects

Model names:

These correspond with a file located in `assets\Models` and are to be specified *without* the file extension.

- `droplet_mesh` is the model to be used for rendering droplets
- `floor_mesh` is the model to be used for rendering floor tiles
- `wall_mesh` is the model to be used for rendering walls
- `tower_mesh` is the model to be used for rendering IR towers
- `object_mesh` is the model to be used for rendering sphere objects
- `object_cube_mesh` is the model to be used for rendering cube objects

Shader names:

These correspond with an entry defind inside the shader manifest file (explained below).

- `droplet_shader` is the shader to be used for rendering droplets when the projector is off
- `droplet_projection` is the shader to be used for rendering droplets when the projector is on
- `floor_shader` is the shader to be used for rendering floor tiles when the projector is off
- `floor_projection` is the shader to be used for rendering floor tiles when the projector is on
- `wall_shader` is the shader to be used for rendering walls regardless of the projector status
- `tower_shader` is the shader to be used for rendering IR towers regardless of the projector status
- `object_shader` is the shader to be used for rendering sphere objects when the projector is off
- `object_projection` is the shader to be used for rendering sphere objects when the projector is on
- `object_cube_shader` is the shader to be used for rendering cube objects when the projector is off
- `object_cube_projection` is the shader to be used for rendering cube objects when the projector is on

5.2.2 Shader manifest file

The shader manifest file defines how vertex and fragment shaders are compiled for use by OpenGL and is located at `\DropletSimulator\DropletSimDemos\DropletGUI\assets\Shaders\shaders.txt`.

The format for the file is:

```
<program name> <vertex shader> <fragment shader>
```

For each statement `<program name>` is the name you want to assign to the compiled program and `<vertex shader>` and `<fragment shader>` are complete file names referencing GLSL shader. For example:

```
debug DebugVertex.glsl DebugFragment.glsl
```

Is used to define a shader program named `debug` that is comprised of the vertex shader found in `DebugVertex.glsl` and the fragment shader found in `DebugFragmnet.glsl`.

6 Tutorials

6.1 Using the Renderer

6.1.1 Base Functionality

The GUI allows you to specify simulator settings and then launch a visual rendering of the simulator.

- Load Setup File
  - Setup File Selection - allows you to use settings in a particular file. The GUI reflects these changes.
    - Note that if you select file and then modify some settings in the GUI, to revert back to the file's settings, you will need to select a different file and then the original file.
- Droplet Parameters
  - Number of Droplets
  - Droplet Radius

  - Droplet Programs
- Arena Parameters
  - Arena Selection - loads in a custom arena if not set to default. Otherwise allows you to specify the dimensions of the arena
  - Projection Images - image projected onto the arena.
- Launch and Playback
  - Log Droplet Information - FRANK SHOULD EXPLAIN
  - View Simulation - Opens a window that displays the simulation. Multiple windows can be opened. Always view the current simulation.
  - Update Simulator - Resets the simulator to the settings in the GUI.
    * Note that view simulation does not update the simulator. If you close the renderer window, change settings in the GUI and relaunch the renderer, you will need to also hit the Update Simulator button.
  - Pause
  - Resume
  - Reset - does not consider GUI settings, merely resets the current simulation.

6.1.2 Using the Renderer

The render is a visual representation of the simulator. The top left of this window displays droplet information such as:

- Number of droplets
- Simulation step size
- Simulator time elapsed
- Real time elapsed
- Requested/real time ratio
- Estimated simulator/real time ratio

In addition to being able to control the camera by clicking inside the window, you can control the simulation through the following keys:

- h: toggles the help menu/key bindings in the lower left corner of the window.
- w/s: Rotates up/down
- a/d: Rotates right/left
- q/e and -/+: Zooms out/in
- [/]: Lower/raise speed limit
- l: Disable simulation speed limiting
- p: Pause the simulator
- Spacebar: change camera modes
- Escape: close the window
- Control-h: Toggle HUD on/off
- Control-b: Toggle debug rendering on/off
- Control-l: Force reloading of rendering assets
- Control-r: Reset simulation to starting stat

6.1.3 Custom Parameters

You can extend the functionality of DropletGUI by defining custom arenas and simulation parameters through setup and arena files. To add a new file you simply create a new text file, specify the settings you would like, and add the file to either the `assets\Setup` file for simulator settings or `assets\Floors` for custom arenas.

For more information on what settings are available please see the section on file formats.

6.2 Custom Programs

You can modify the custom program files in `cu-droplet\DropletSimulator\DropletPrograms\Custom-Programs`. Please refer to `cu-droplet\DropletSimulator\DropletPrograms\Default-Programs` for examples on how to write your program. Please keep in mind that the Droplets don't wait for one command to finish before executing the next line/command. What I mean by this is if you have a sequence:

```
move_steps(NORTH, 400);
move_steps(SOUTH, 400);
```

The droplets will not move north for 400 steps. You need to separate such calls with if statements and flags.

7 Module Index

7.1 Modules

Here is a list of all modules:

8 Hierarchical Index

8.1 Class Hierarchy

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

9 Class Index

9.1 Class List

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

10 File Index

10.1 File List

Here is a list of all files with brief descriptions:

11 Module Documentation

11.1 Assets

Items related to managing assets required for rendering.

Collaboration diagram for Assets:

Classes

- class AssetManager

  Items related to managing assets required for rendering.
- class MeshManager

Container class for meshes.

- class TextureManager

  Container class for a texture.

11.1.1 Detailed Description

Items related to managing assets required for rendering.

11.2 Main Window

Items related to the main GUI window.

Classes

- class MainWindow

  View controller for the main window.

11.2.1 Detailed Description

Items related to the main GUI window.

11.3 Renderer

Items related to visualizing the state of the simulation.

Collaboration diagram for Renderer:

Modules

- Assets

  Items related to managing assets required for rendering.

Classes

- class RenderWidget

  View controller for QGLWidget that renders the simulation state.

11.3.1 Detailed Description

Items related to visualizing the state of the simulation.

11.4 Info Logging

Items related to logging information from the simulator.

Classes

- class simInfoLogger

  Simulation information logger.

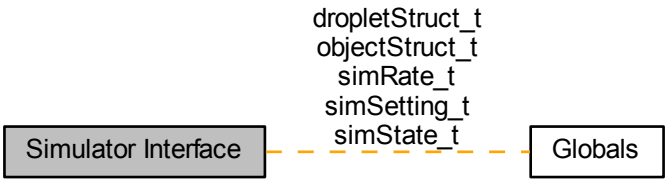
11.4.1 Detailed Description

Items related to logging information from the simulator.

11.5 Simulator Interface

Items related to visualizing the state of the simulation.

Collaboration diagram for Simulator Interface:



Classes

- struct dropletStruct_t

  Defines properties about droplets that are emitted from SimInterface.
- struct objectStruct_t

  Defines properties about physical objects that are emitted from SimInterface.
- class SimInterface

  Model that contains and controls the droplet simulator.
- struct simRate_t

  Defines the current state of rate limiting emitted by the SimInterface.
- struct simSetting_t

  Defines simulator settings that are passed in and out of SimInterface.
- struct simState_t

  Defines current state of the simulator that is emitted by SimInterface.

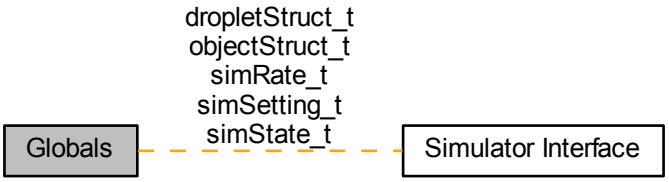
11.5.1 Detailed Description

Items related to visualizing the state of the simulation.

11.6 Globals

Global types, structures, and functions.

Collaboration diagram for Globals:



Classes

- struct collisionShapeStruct_t

  Defines the properties of a collision shape.
- struct dropletStruct_t

  Defines properties about droplets that are emitted from SimInterface.
- struct objectStruct_t

  Defines properties about physical objects that are emitted from SimInterface.
- struct simRate_t

  Defines the current state of rate limiting emitted by the SimInterface.
- struct simSetting_t

  Defines simulator settings that are passed in and out of SimInterface.
- struct simState_t

  Defines current state of the simulator that is emitted by SimInterface.
- union vec2

  Two element vector of floats.
- union vec3

  Three element vector of floats.
- union vec3i

  Three element vector of integers.
- union vec4

  Four element vector of floats.

Enumerations

- enum droplet_t {
  RGBSense, RandomWalk, March, Rainbow,
  StickPullers, TurnTest, CommTest, PowerTest,
  Granola, StickPullersUpdated, Ants, CustomOne,
  CustomTwo, CustomThree, CustomFour, CustomFive,
  CustomSix, CustomSeven, CustomEight, CustomNine,
  CustomTen, NUM_DROPLET_PROGRAMS }

  Enumerated type defining possible droplet programs.
- enum object_t { Sphere, Cube, Wall, Floor }

  Enumerated type defining possible physical objects.

11.6.1 Detailed Description

Global types, structures, and functions. Vector types inspired by Apple's GLKVector types: http://developer.apple.com/library/mac/#documentation/GLkit/Reference/GLKit_Collection/_index.html

11.6.2 Enumeration Type Documentation

11.6.2.1 enum droplet_t

Enumerated type defining possible droplet programs.

Enumerator

RGBSense
RandomWalk
March
Rainbow
StickPullers
TurnTest
CommTest
PowerTest
Granola
StickPullersUpdated
Ants
CustomOne
CustomTwo
CustomThree
CustomFour
CustomFive
CustomSix
CustomSeven
CustomEight
CustomNine
CustomTen
NUM_DROPLET_PROGRAMS

Definition at line 160 of file structs.h.

11.6.2.2 enum object_t

Enumerated type defining possible physical objects.

Enumerator

Sphere
Cube
Wall
Floor

Definition at line 211 of file structs.h.

12 Class Documentation

12.1 AssetManager Class Reference

Items related to managing assets required for rendering.

`#include <assetmanager.h>`

Public Member Functions

- AssetManager ()
  Default constructor.
- ~AssetManager ()
  Destructor.
- void clearAssets ()
  Clears the assets.
- MeshManager * getMesh (QString name)
  Gets a mesh.
- QGLShaderProgram * getShader (QString name)
  Gets a shader.
- TextureManager * getTexture (QString name)
  Gets a texture.
- bool loadAssets ()
  Loads the assets.
- QString lookupAssetName (QString name)
  Looks up a given key to find its associated asset name.
- void reloadAssets ()
  Reload assets.
- void setAssetDir (QString dir)
  Sets asset dir.
- void setManifest (QString file)
  Sets a manifest.
- void setMeshDir (QString dir)
  Sets mesh dir.
- void setShaderDir (QString dir)
  Sets shader dir.
- void setShaderManifest (QString file)
  Sets shader manifest.
- void setTextureDir (QString dir)
  Sets texture dir.

Private Member Functions

- void clearManifest ()
  Clears the manifest.
- void clearMeshes ()
  Clears the meshes.
- void clearShaders ()
  Clears the shaders.
- void clearTextures ()
  Clears the textures.

- void loadManifest ()

  Loads the manifest.
- void loadMeshes ()

  Loads the meshes.
- void loadShaders ()

  Loads the shaders.
- void loadTextures ()

  Loads the textures.

Private Attributes

- QString _assetDir

  The asset dir.
- QString _assetManifest

  A list of assets.
- QHash< QString, QString > _manifest

  The manifest.
- QString _meshDir

  The mesh dir.
- QHash< QString, MeshManager * > _meshes

  The meshes.
- QString _projectionDir

  The projection dir.
- QString _shaderDir

  The shader dir.
- QString _shaderManifest

  A list of shaders.
- QHash< QString, QGLShaderProgram * > _shaders

  The shaders.
- QString _textureDir

  The texture dir.
- QHash< QString, TextureManager * > _textures

  The textures.
- bool _unused

  true if unused.

12.1.1 Detailed Description

Items related to managing assets required for rendering.

Definition at line 35 of file assetmanager.h.

12.1.2 Constructor & Destructor Documentation

12.1.2.1 AssetManager::AssetManager ( )

Default constructor.

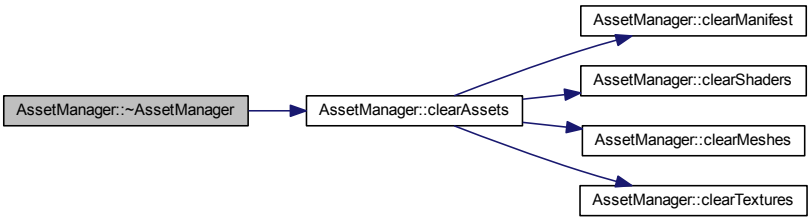
Definition at line 9 of file assetmanager.cpp.

12.1.2.2 AssetManager::~AssetManager ( )

Destructor.

Definition at line 20 of file assetmanager.cpp.

Here is the call graph for this function:



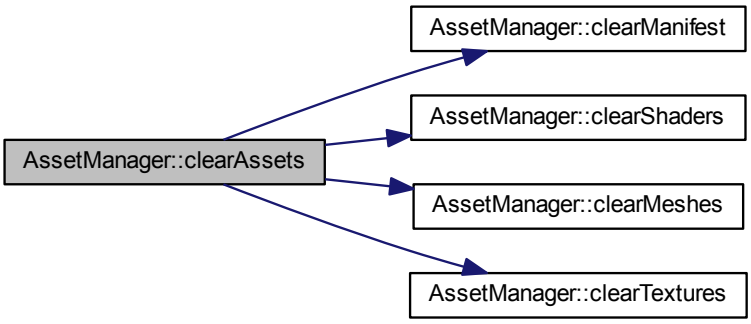
12.1.3 Member Function Documentation

12.1.3.1 void AssetManager::clearAssets ( )

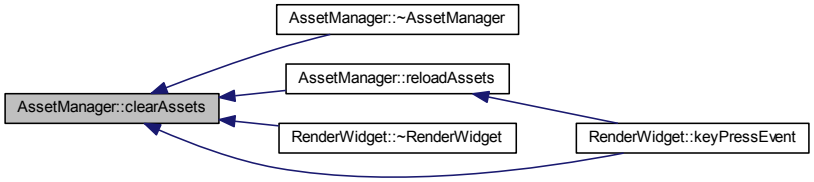
Clears the assets.

Definition at line 291 of file assetmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

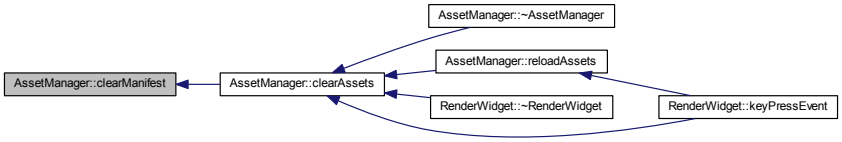


12.1.3.2 void AssetManager::clearManifest ( ) [private]

Clears the manifest.

Definition at line 359 of file assetmanager.cpp.

Here is the caller graph for this function:

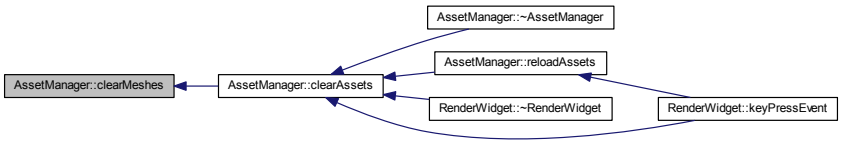


12.1.3.3 void AssetManager::clearMeshes ( ) [private]

Clears the meshes.

Definition at line 275 of file assetmanager.cpp.

Here is the caller graph for this function:

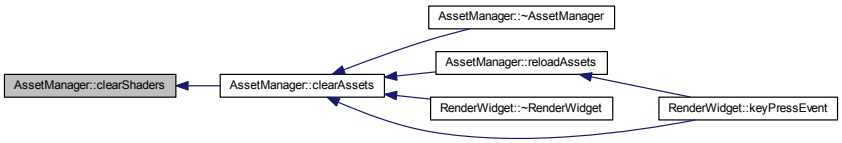


12.1.3.4 void AssetManager::clearShaders ( ) [private]

Clears the shaders.

Definition at line 244 of file assetmanager.cpp.

Here is the caller graph for this function:

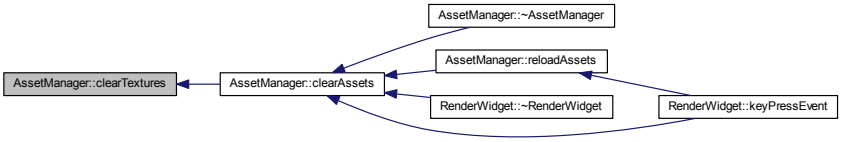


12.1.3.5 void AssetManager::clearTextures ( ) [private]

Clears the textures.

Definition at line 260 of file assetmanager.cpp.

Here is the caller graph for this function:



12.1.3.6 MeshManager * AssetManager::getMesh (QString *name*)

Gets a mesh.

Parameters

name	The name.

Returns

null if it fails, else the mesh.

Definition at line 216 of file assetmanager.cpp.

Here is the caller graph for this function:

12.1.3.7 QGLShaderProgram * AssetManager::getShader (QString *name*)

Gets a shader.

Parameters

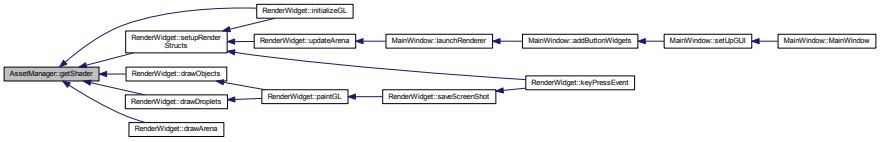
name	The name.

Returns

null if it fails, else the shader.

Definition at line 204 of file assetmanager.cpp.

Here is the caller graph for this function:



12.1.3.8 TextureManager * AssetManager::getTexture (QString *name*)

Gets a texture.

Parameters

name	The name.

Returns

null if it fails, else the texture.

Definition at line 228 of file assetmanager.cpp.

Here is the caller graph for this function:

12.1.3.9 bool AssetManager::loadAssets ( )

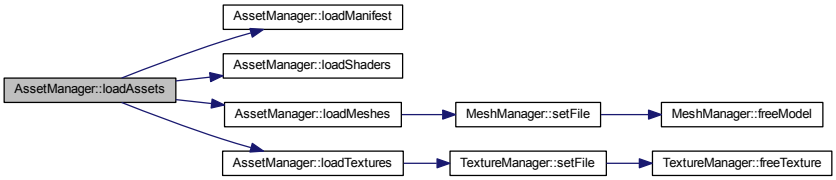
Loads the assets.

Returns

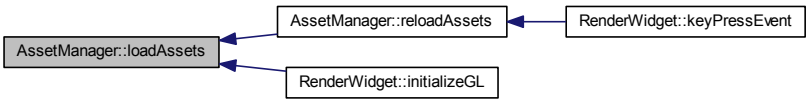
true if it succeeds, false if it fails.

Definition at line 28 of file assetmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

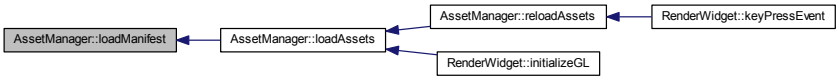


12.1.3.10 void AssetManager::loadManifest ( ) [private]

Loads the manifest.

Definition at line 307 of file assetmanager.cpp.

Here is the caller graph for this function:

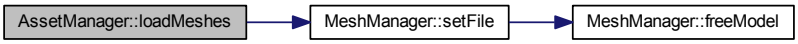


12.1.3.11 void AssetManager::loadMeshes ( ) [private]

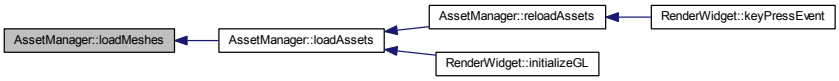
Loads the meshes.

Definition at line 38 of file assetmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

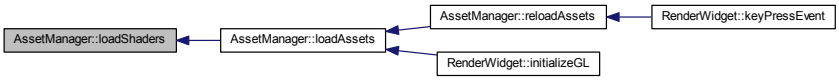


12.1.3.12 void AssetManager::loadShaders ( ) [private]

Loads the shaders.

Definition at line 92 of file assetmanager.cpp.

Here is the caller graph for this function:

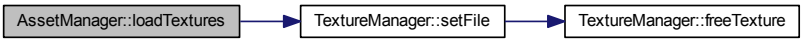


12.1.3.13 void AssetManager::loadTextures ( ) [private]

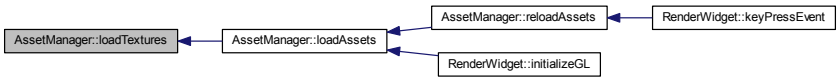
Loads the textures.

Definition at line 160 of file assetmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.1.3.14 QString AssetManager::lookupAssetName (QString *name*)

Looks up a given key to find its associated asset name.

Parameters

name	The name.

Returns

.

Definition at line 364 of file assetmanager.cpp.

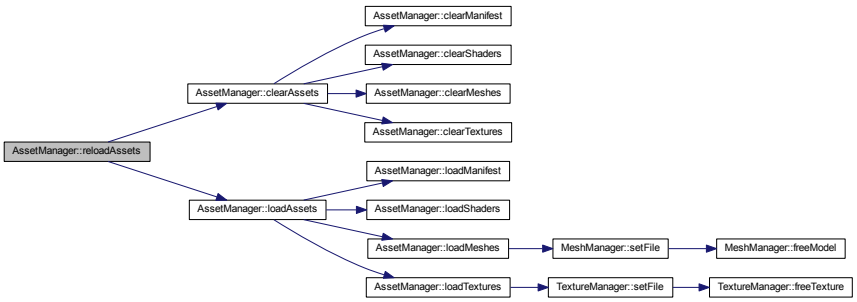
Here is the caller graph for this function:

12.1.3.15 void AssetManager::reloadAssets ( )

Reload assets.

Definition at line 300 of file assetmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.1.3.16 void AssetManager::setAssetDir (QString *dir*)

Sets asset dir.

Parameters

dir	The dir.

Definition at line 80 of file assetmanager.cpp.

Here is the caller graph for this function:

12.1.3.17 void AssetManager::setManifest (QString *file*)

Sets a manifest.

Parameters

file	The file.

Definition at line 84 of file assetmanager.cpp.

12.1.3.18 void AssetManager::setMeshDir (QString *dir*)

Sets mesh dir.

Parameters

dir	The dir.

Definition at line 71 of file assetmanager.cpp.

Here is the caller graph for this function:

12.1.3.19 void AssetManager::setShaderDir (QString *dir*)

Sets shader dir.

Parameters

dir	The dir.

Definition at line 67 of file assetmanager.cpp.

Here is the caller graph for this function:

12.1.3.20 void AssetManager::setShaderManifest (QString *file*)

Sets shader manifest.

Parameters

file	The file.

Definition at line 88 of file assetmanager.cpp.

12.1.3.21 void AssetManager::setTextureDir (QString *dir*)

Sets texture dir.

Parameters

dir	The dir.

Definition at line 75 of file assetmanager.cpp.

Here is the caller graph for this function:

12.1.4 Member Data Documentation

12.1.4.1 QString AssetManager::_assetDir [private]

The asset dir.

Definition at line 243 of file assetmanager.h.

12.1.4.2 QString AssetManager::_assetManifest [private]

A list of assets.

Definition at line 249 of file assetmanager.h.

12.1.4.3 QHash<QString,QString> AssetManager::_manifest [private]

The manifest.

Definition at line 255 of file assetmanager.h.

12.1.4.4 QString AssetManager::_meshDir [private]

The mesh dir.

Definition at line 219 of file assetmanager.h.

12.1.4.5 QHash<QString,MeshManager *> AssetManager::_meshes [private]

The meshes.

Definition at line 207 of file assetmanager.h.

12.1.4.6 QString AssetManager::_projectionDir [private]

The projection dir.

Definition at line 231 of file assetmanager.h.

12.1.4.7 QString AssetManager::_shaderDir [private]

The shader dir.

Definition at line 213 of file assetmanager.h.

12.1.4.8 QString AssetManager::_shaderManifest [private]

A list of shaders.

Definition at line 237 of file assetmanager.h.

12.1.4.9 QHash<QString,QGLShaderProgram *> AssetManager::_shaders [private]

The shaders.

Definition at line 195 of file assetmanager.h.

12.1.4.10 QString AssetManager::_textureDir [private]

The texture dir.

Definition at line 225 of file assetmanager.h.

12.1.4.11 QHash<QString,TextureManager *> AssetManager::_textures [private]

The textures.

Definition at line 201 of file assetmanager.h.

12.1.4.12 bool AssetManager::_unused [private]

true if unused.

Definition at line 261 of file assetmanager.h.

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

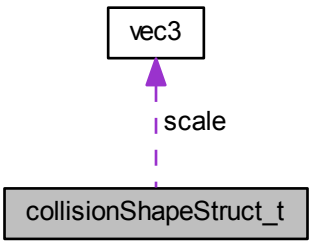
- assetmanager.h
- assetmanager.cpp

12.2 collisionShapeStruct_t Struct Reference

Defines the properties of a collision shape.

`#include <structs.h>`

Collaboration diagram for collisionShapeStruct_t:



Public Attributes

- int collisionID

- float objectRadius
- object_t oType
- vec3 scale

12.2.1 Detailed Description

Defines the properties of a collision shape.

Definition at line 268 of file structs.h.

12.2.2 Member Data Documentation

12.2.2.1 int collisionShapeStruct_t::collisionID

Definition at line 272 of file structs.h.

12.2.2.2 float collisionShapeStruct_t::objectRadius

Definition at line 270 of file structs.h.

12.2.2.3 object_t collisionShapeStruct_t::oType

Definition at line 269 of file structs.h.

12.2.2.4 vec3 collisionShapeStruct_t::scale

Definition at line 271 of file structs.h.

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

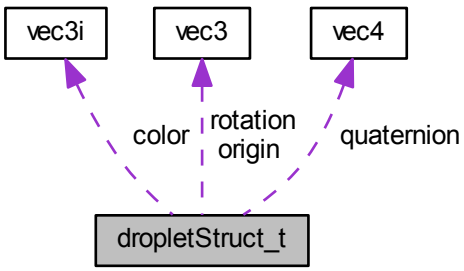
- structs.h

12.3 dropletStruct_t Struct Reference

Defines properties about droplets that are emitted from SimInterface.

`#include <structs.h>`

Collaboration diagram for dropletStruct_t:



Public Attributes

- bool changed
- vec3i color
- DropletCommData commData
- int dropletID
- vec3 origin
- vec4 quaternion
- vec3 rotation

12.3.1 Detailed Description

Defines properties about droplets that are emitted from SimInterface.

Definition at line 194 of file structs.h.

12.3.2 Member Data Documentation

12.3.2.1 bool dropletStruct_t::changed

Definition at line 201 of file structs.h.

12.3.2.2 vec3i dropletStruct_t::color

Definition at line 199 of file structs.h.

12.3.2.3 DropletCommData dropletStruct_t::commData

Definition at line 200 of file structs.h.

12.3.2.4 int dropletStruct_t::dropletID

Definition at line 195 of file structs.h.

12.3.2.5 vec3 dropletStruct_t::origin

Definition at line 196 of file structs.h.

12.3.2.6 vec4 dropletStruct_t::quaternion

Definition at line 198 of file structs.h.

12.3.2.7 vec3 dropletStruct_t::rotation

Definition at line 197 of file structs.h.

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

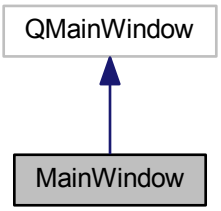
- structs.h

12.4 MainWindow Class Reference

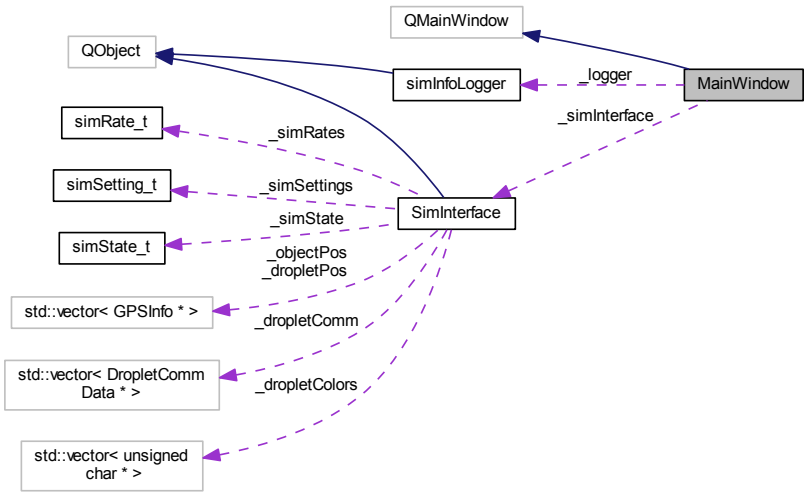
View controller for the main window.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Slots

- void enableDisableDropletTable (const QString &)
- void loadFloorFiles (QString path)

  Populates the list of floor files and adds them to the GUI: _floorFiles, arenaSelectionCombo.
- bool loadFromFile (QString filename, simSetting_t &settings)

  Loads simulator settings from an input file.
- void loadProjectionTextures (QString path)

  Populates the list of projection textures and adds them to the GUI: _projectionTextures, projectorImageCombo.
- void loadSetupFiles (QString path)

  Populates the list of setup files and adds them to the GUI: _setupFiles, loadSetupFileCombo.
- bool saveToFile (QString filename, simSetting_t settings)

  Saves current simulator settings to a file.
- void setUI (simSetting_t settings)

  Sets GUI parameters values stored in simSetting_t settings.

- void showHideLogWidget (int state)

  Toggles the logWidget between show and hide.
- void showHideRowColWidget (const QString &text)

  Toggles the rowColWidget between show and hide.
- void updateParams (const QString &text)

  Parses a specified setup file and updates the params in the GUI.
- void updateSetupFile (const QString &file)

  Checks if the file specified matches the currently selected setup file, and if it does it refreshes the GUI on change.

Signals

- void closing ()

  Signal emitted in MainWindow::closeEvent(QCloseEvent ∗event).
- void disableLimit (void)

  Signal to disable real/simulator time limiting.
- void enableLimit (void)

  Signal to enable real/simulator time limiting.
- void launchSim (simSetting_t)

  Signal emmited when a simulator is launched.
- void setUpdateRate (float)

  Sets update rate.

Public Member Functions

- MainWindow (QWidget ∗parent=0)

  Constructor. Basically initializes the GUI.
- ∼MainWindow ()

Protected Member Functions

- void closeEvent (QCloseEvent ∗event)

  Uses the closing signal to signal all child objects to close/quit.
- bool eventFilter (QObject ∗watched, QEvent ∗event)

  Determines if the application is in focus. If application regains focus, projector image and floor file lists are refreshed.

Private Slots

- void launchRenderer ()

  Creates an instance of the renderer. Connects signals/slots between the simulator interface and the renderer. If no simulator is currently running, also launches the simulator.
- void launchSimulator ()

  Creates an instance of the simulator. If a simulator is already running when this launchSimulator() is called, that simulator is first destroyed so there are not multiple intances of a simulator. Uses GUI parameters. If logCheckBox is checked, also creates an instance of the logger class.
- void removeRenderer (QObject ∗obj=0)

  Removes the renderer.

Private Member Functions

- void addArenaWidgets ()

  Sets up the arena parameter widgets.
- void addButtonWidgets ()

  Sets up the launch button widgets.
- void addDropletWidgets ()

  Sets up the droplet parameter widgets.
- void addLoadSetupFileWidgets ()

  Sets up the setup file widgets.
- void setUpGUI ()

  Sets up the graphical user interface. Calls addDropletWidgets(), addArenaWidgets(), and addButtonWidgets()

Private Attributes

- struct {
  QVector< QStringList > droplets
  QVector< QStringList > objects
  } _arenaObjects

- QStringList _floorFiles

  List of filenames for the projection textures and custom floor files.
- simInfoLogger _logger

  Instance of the logging class, used to log simulation data.
- QStringList _projectionTextures
- QLinkedList< RenderWidget * > _renderWidgets

  List of render widgets.
- QStringList _setupFiles
- SimInterface _simInterface

  Instance of the simulator interface.
- QThread _simThread
- bool _simulatorRunning

  true to simulator running.
- QFrame * arenaParams
- QComboBox * arenaSelectionCombo
- QLabel * arenaSelectionLabel
- QVBoxLayout * arenaSelectionLayout

  Qt widgets for selecting the default or a custom floor.
- QListView * arenaSelectionList
- QFileSystemWatcher arenaWatcher
- QFrame * buttons

  Qt frames to separate parameter types.
- QCheckBox * colCheckBox
- QCheckBox * commSACheckBox
- QFrame * dropletParams
- QTableWidget * dropletTableWidget
- QComboBox * dropProgramsCombo

  Qt widgets for selecting the droplet program.
- QLabel * dropProgramsLabel
- QVBoxLayout * dropProgramsLayout
- QListView * dropProgramsList
- QDoubleSpinBox * dropRadBox
- QLabel * dropRadLabel

- QHBoxLayout * dropRadLayout

  Qt widgets for selecting the droplet radius parameter.
- QString fileName

  Initialized to the assest directory.
- QPushButton * launchRenderWidget

  Qt buttons for launching the simulator or renderer.
- QPushButton * launchSimulation
- QFrame * loadSetupFile
- QComboBox * loadSetupFileCombo
- QLabel * loadSetupFileLabel
- QVBoxLayout * loadSetupFileLayout

  Qt widgets for selecting the default or a custom floor.
- QListView * loadSetupFileList
- QCheckBox * logCheckBox
- QVBoxLayout * logLayout
- QWidget * logWidget

  Qt widgets for combine all of the logging widgets, for show hide capability.
- QHBoxLayout * macroLayout
- QCheckBox * macroRedCheckBox
- QCheckBox * macroSACheckBox

  Qt checkboxes for selecting if the user wants to log data, and what data he/she wants to log.
- QHBoxLayout * microLayout
- QSpinBox * numColBox
- QLabel * numColLabel
- QHBoxLayout * numColLayout

  Qt widgets for selecting the number of arena columns parameter.
- QSpinBox * numDropBox
- QLabel * numDropLabel
- QHBoxLayout * numDropLayout

  Qt widgets for selecting the number of droplets parameter.
- QSpinBox * numRowBox
- QLabel * numRowLabel
- QHBoxLayout * numRowLayout

  Qt widgets for selecting the number of arena rows parameter.
- QPushButton * pause_button
- QCheckBox * posCheckBox
- QFileSystemWatcher projectionWatcher
- QComboBox * projectorImageCombo
- QLabel * projectorImageLabel
- QListView * projectorImageList
- QVBoxLayout * projectorImagesLayout

  Qt widgets for selecting the projected image.
- QPushButton * reset_button

  Qt buttons for playback options.
- QPushButton * resume_button
- QCheckBox * rotCheckBox
- QVBoxLayout * rowColLayout
- QWidget * rowColWidget

  The row widgets and column widgets are shown or hid, so they are within a single rowColWid.
- QFileSystemWatcher setupWatcher

12.4.1 Detailed Description

View controller for the main window.

Definition at line 55 of file mainwindow.h.

12.4.2 Constructor & Destructor Documentation

12.4.2.1 MainWindow::MainWindow (QWidget * *parent* = 0)

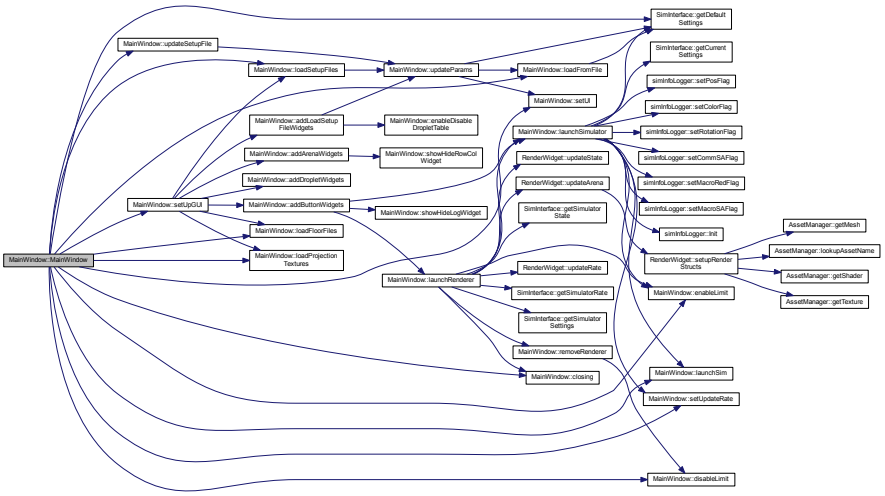
Constructor. Basically initializes the GUI.

Parameters

in,out	*parent*	(Optional) If non-null, (Optional) the parent.

Definition at line 11 of file mainwindow.cpp.

Here is the call graph for this function:



12.4.2.2 MainWindow::~MainWindow ( )

Definition at line 72 of file mainwindow.cpp.

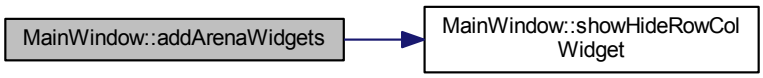
12.4.3 Member Function Documentation

12.4.3.1 void MainWindow::addArenaWidgets ( ) [private]

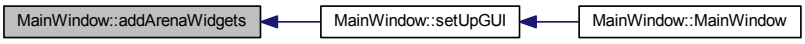
Sets up the arena parameter widgets.

Definition at line 375 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

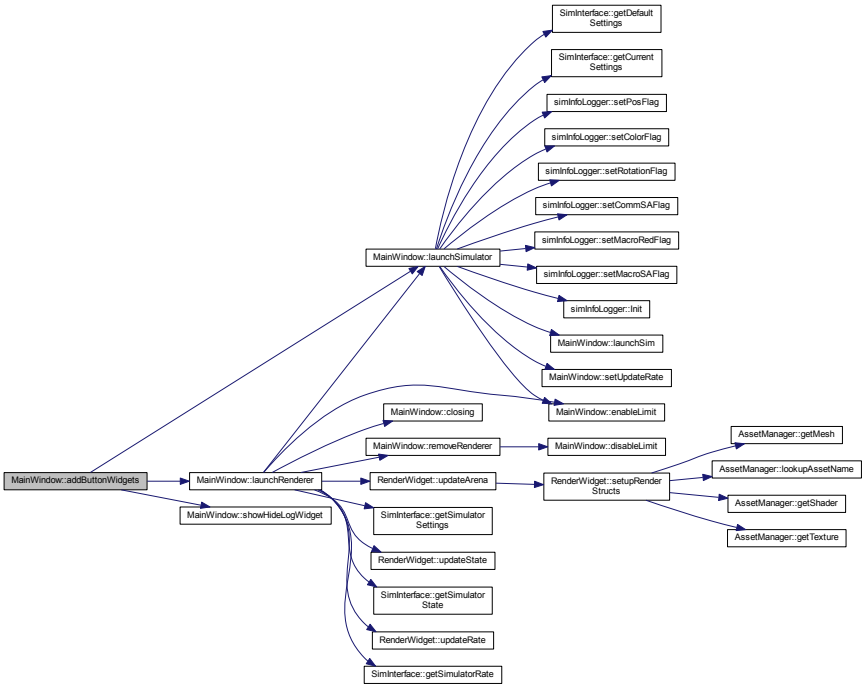


12.4.3.2 void MainWindow::addButtonWidgets ( ) [private]

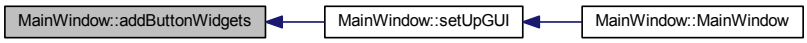
Sets up the launch button widgets.

Definition at line 456 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

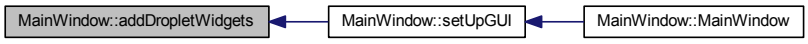


12.4.3.3 void MainWindow::addDropletWidgets ( ) [private]

Sets up the droplet parameter widgets.

Definition at line 266 of file mainwindow.cpp.

Here is the caller graph for this function:

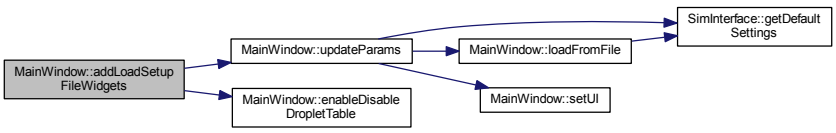


12.4.3.4 void MainWindow::addLoadSetupFileWidgets ( ) [private]

Sets up the setup file widgets.

Definition at line 232 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.4.3.5 void MainWindow::closeEvent (QCloseEvent * *event*) [protected]

Uses the closing signal to signal all child objects to close/quit.

Parameters

in, out	*event*	If non-null, the event.

Definition at line 569 of file mainwindow.cpp.

Here is the call graph for this function:

12.4.3.6 void MainWindow::closing ( ) [signal]

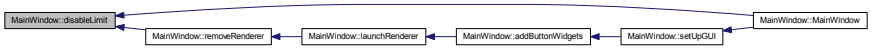
Signal emitted in MainWindow::closeEvent(QCloseEvent *event).

Here is the caller graph for this function:

12.4.3.7 void MainWindow::disableLimit (void) [signal]

Signal to disable real/simulator time limiting.

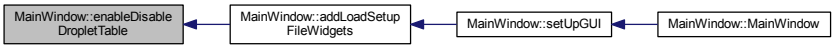
Here is the caller graph for this function:



12.4.3.8 void MainWindow::enableDisableDropletTable (const QString & *text*) [slot]

Definition at line 1051 of file mainwindow.cpp.

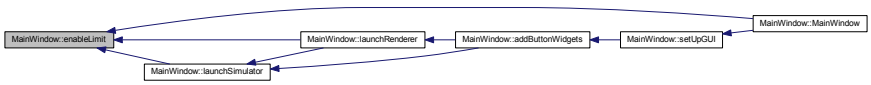
Here is the caller graph for this function:



12.4.3.9 void MainWindow::enableLimit (void) [signal]

Signal to enable real/simulator time limiting.

Here is the caller graph for this function:



12.4.3.10 bool MainWindow::eventFilter (QObject * *watched,* QEvent * *event*) [protected]

Determines if the application is in focus. If application regains focus, projector image and floor file lists are refreshed.

Parameters

in,out	*watched*	If non-null, the watched.
in,out	*event*	If non-null, the event.

Returns

true if it succeeds, false if it fails.

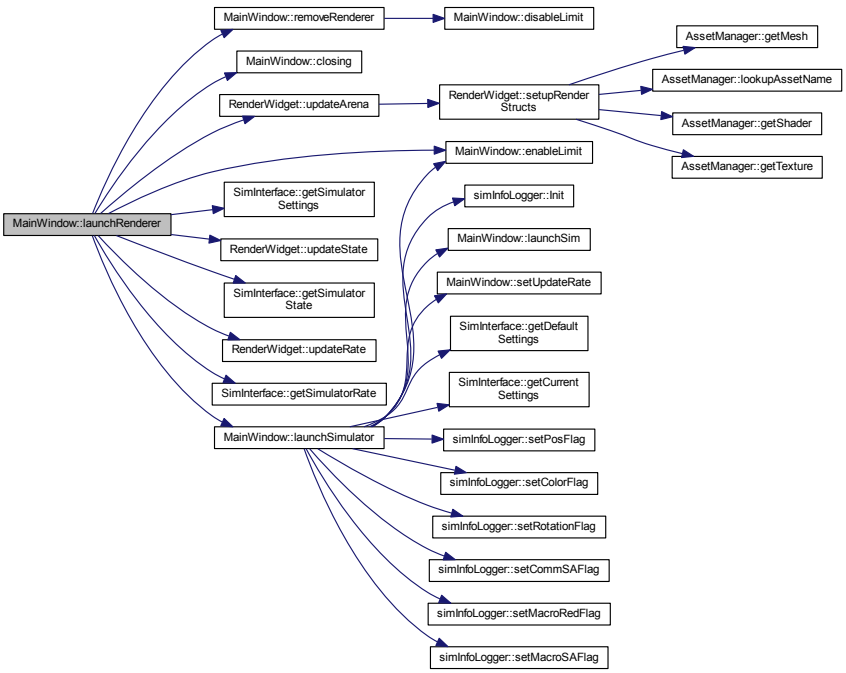
Definition at line 77 of file mainwindow.cpp.

12.4.3.11 void MainWindow::launchRenderer ( ) [private], [slot]

Creates an instance of the renderer. Connects signals/slots between the simulator interface and the renderer. If no simulator is currently running, also launches the simulator.

Definition at line 182 of file mainwindow.cpp.

Here is the call graph for this function:

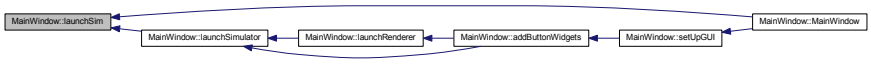


Here is the caller graph for this function:

12.4.3.12 void MainWindow::launchSim (simSetting_t) [signal]

Signal emmited when a simulator is launched.

Here is the caller graph for this function:

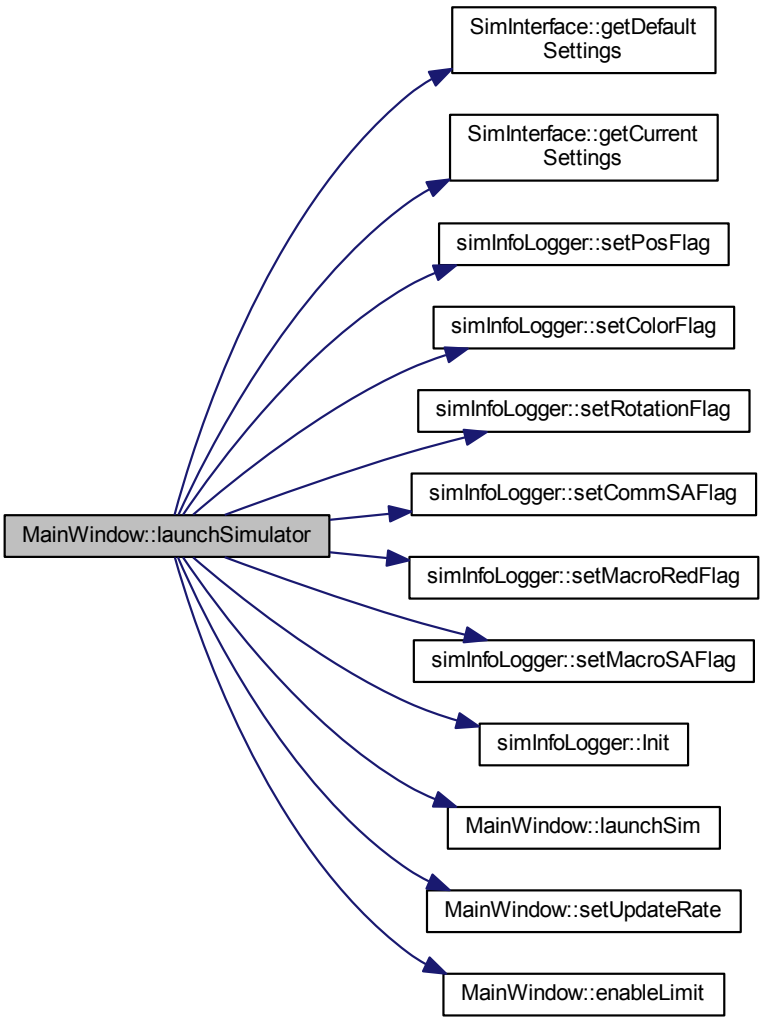


12.4.3.13 void MainWindow::launchSimulator ( ) [private], [slot]

Creates an instance of the simulator. If a simulator is already running when this launchSimulator() is called, that simulator is first destroyed so there are not multiple intances of a simulator. Uses GUI parameters. If logCheckBox is checked, also creates an instance of the logger class.

Definition at line 100 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.4.3.14 void MainWindow::loadFloorFiles (QString *path*) [slot]

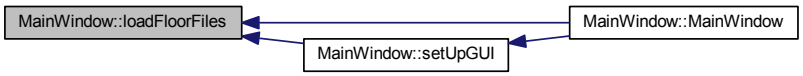
Populates the list of floor files and adds them to the GUI: _floorFiles, arenaSelectionCombo.

Parameters

path	Full pathname of the file.

Definition at line 605 of file mainwindow.cpp.

Here is the caller graph for this function:



12.4.3.15 bool MainWindow::loadFromFile (QString *filename,* simSetting_t & *settings*) [slot]

Loads simulator settings from an input file.

Parameters

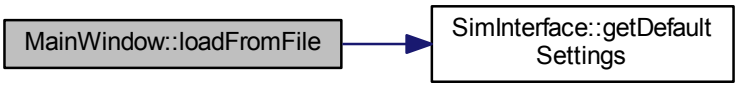
	filename	Filename of the file.
in,out	*settings*	Contains simulator settings.

Returns

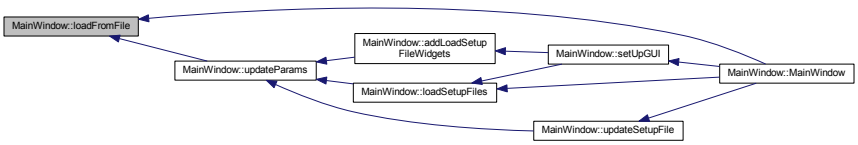
true if it succeeds, false if it fails.

Definition at line 866 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.4.3.16 void MainWindow::loadProjectionTextures (QString *path*) [slot]

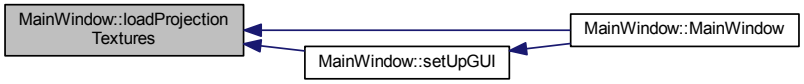
Populates the list of projection textures and adds them to the GUI: _projectionTextures, projectorImageCombo.

Parameters

path	Full pathname of the file.

Definition at line 581 of file mainwindow.cpp.

Here is the caller graph for this function:

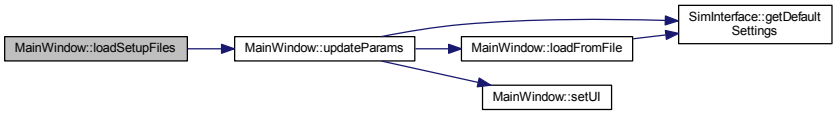


12.4.3.17 void MainWindow::loadSetupFiles (QString *path*) [slot]

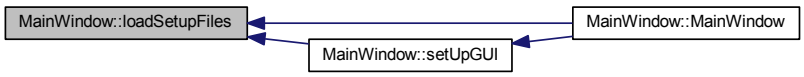
Populates the list of setup files and adds them to the GUI: _setupFiles, loadSetupFileCombo.

Definition at line 630 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.4.3.18 void MainWindow::removeRenderer (QObject * *obj* = 0) [private], [slot]

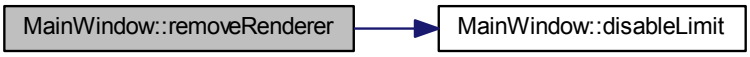
Removes the renderer.

Parameters

in,out	*obj*	(Optional) If non-null, (Optional) the object.

Definition at line 550 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.4.3.19 bool MainWindow::saveToFile (QString *filename,* simSetting_t *settings*) [slot]

Saves current simulator settings to a file.

Parameters

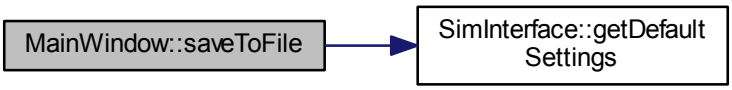
filename	Filename of the file.
settings	Contains simulator settings.

Returns

true if it succeeds, false if it fails.

Definition at line 828 of file mainwindow.cpp.

Here is the call graph for this function:



12.4.3.20 void MainWindow::setUI (simSetting_t *settings*) [slot]

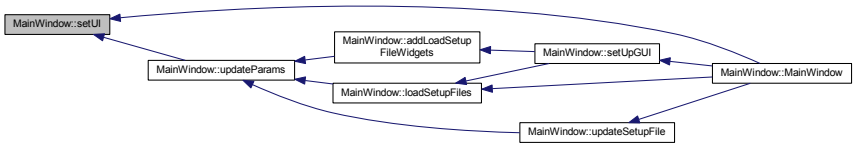
Sets GUI parameters values stored in simSetting_t settings.

Parameters

settings	Contains simulator settings.

Definition at line 659 of file mainwindow.cpp.

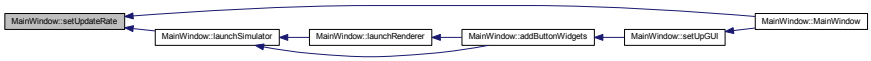
Here is the caller graph for this function:



12.4.3.21 void MainWindow::setUpdateRate (float) [signal]

Sets update rate.

Here is the caller graph for this function:

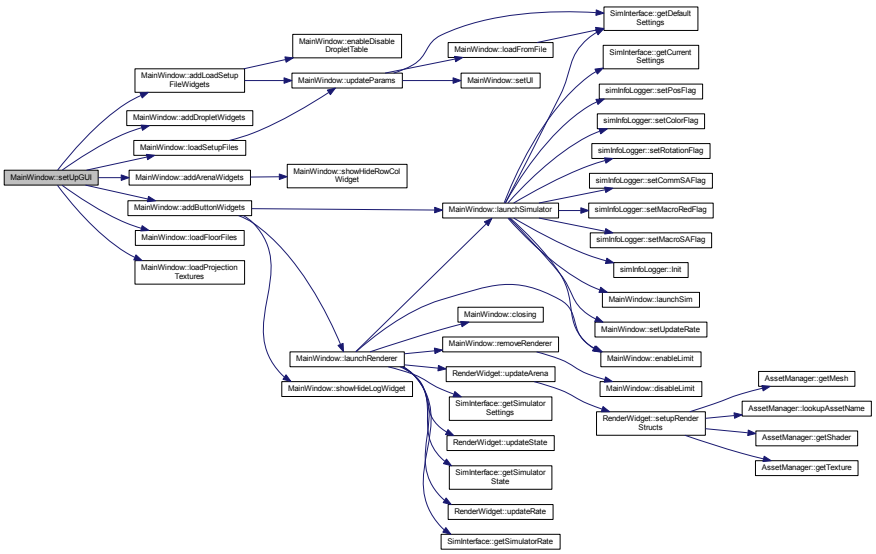


12.4.3.22 void MainWindow::setUpGUI ( ) [private]

Sets up the graphical user interface. Calls addDropletWidgets(), addArenaWidgets(), and addButtonWidgets()

Definition at line 513 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.4.3.23 void MainWindow::showHideLogWidget (int *state*) [slot]

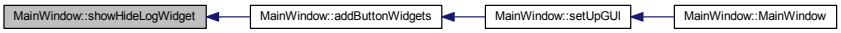
Toggles the logWidget between show and hide.

Parameters

state	The state of the logCheckBox.

Definition at line 997 of file mainwindow.cpp.

Here is the caller graph for this function:



12.4.3.24 void MainWindow::showHideRowColWidget (const QString & *text*) [slot]

Toggles the rowColWidget between show and hide.

Parameters

text	if 'Default (Rectangle)', widget is shown, otherwise hid.

Definition at line 1009 of file mainwindow.cpp.

Here is the caller graph for this function:

12.4.3.25 void MainWindow::updateParams (const QString & *text*) [slot]

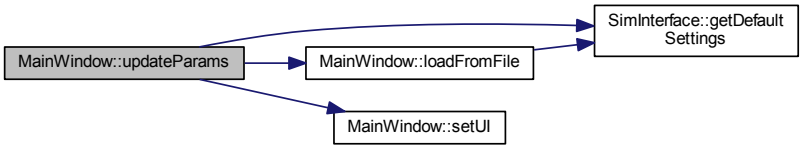
Parses a specified setup file and updates the params in the GUI.

Parameters

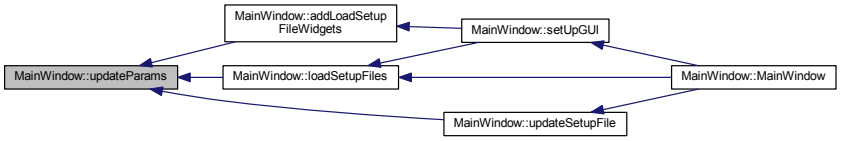
text	specifies a setup file

Definition at line 1033 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.4.3.26 void MainWindow::updateSetupFile (const QString & *file*) [slot]

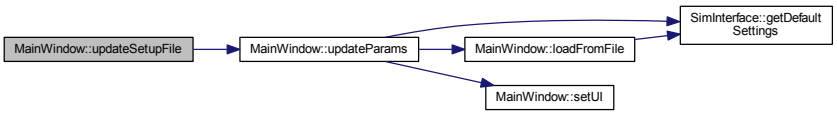
Checks if the file specified matches the currently selected setup file, and if it does it refreshes the GUI on change.

Parameters

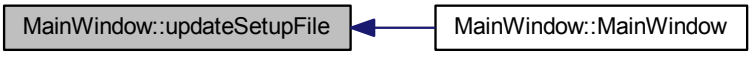
file	Path to the file.

Definition at line 1021 of file mainwindow.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.4.4 Member Data Documentation

12.4.4.1 struct { ... } MainWindow::_arenaObjects [private]

12.4.4.2 QStringList _projectionTextures QStringList MainWindow::_floorFiles [private]

List of filenames for the projection textures and custom floor files.

Definition at line 231 of file mainwindow.h.

12.4.4.3 simInfoLogger MainWindow::_logger [private]

Instance of the logging class, used to log simulation data.

Definition at line 248 of file mainwindow.h.

12.4.4.4 QStringList MainWindow::_projectionTextures [private]

Definition at line 230 of file mainwindow.h.

12.4.4.5 QLinkedList< RenderWidget * > MainWindow::_renderWidgets [private]

List of render widgets.

Definition at line 223 of file mainwindow.h.

12.4.4.6 QStringList MainWindow::_setupFiles [private]

Definition at line 232 of file mainwindow.h.

12.4.4.7 SimInterface MainWindow::_simInterface [private]

Instance of the simulator interface.

Definition at line 217 of file mainwindow.h.

12.4.4.8 QThread MainWindow::_simThread [private]

Definition at line 241 of file mainwindow.h.

12.4.4.9 bool MainWindow::_simulatorRunning [private]

true to simulator running.

Definition at line 240 of file mainwindow.h.

12.4.4.10 QFrame* MainWindow::arenaParams [private]

Definition at line 389 of file mainwindow.h.

12.4.4.11 QComboBox* MainWindow::arenaSelectionCombo [private]

Definition at line 299 of file mainwindow.h.

12.4.4.12 QLabel* MainWindow::arenaSelectionLabel [private]

Definition at line 298 of file mainwindow.h.

12.4.4.13 QLabel *arenaSelectionLabel QComboBox *arenaSelectionCombo QListView *arenaSelectionList QVBoxLayout * MainWindow::arenaSelectionLayout [private]

Qt widgets for selecting the default or a custom floor.

Definition at line 301 of file mainwindow.h.

12.4.4.14 QListView* MainWindow::arenaSelectionList [private]

Definition at line 300 of file mainwindow.h.

12.4.4.15 QFileSystemWatcher MainWindow::arenaWatcher [private]

Definition at line 393 of file mainwindow.h.

12.4.4.16 QFrame *loadSetupFile QFrame *dropletParams QFrame *arenaParams QFrame * MainWindow::buttons [private]

Qt frames to separate parameter types.

Definition at line 390 of file mainwindow.h.

12.4.4.17 QCheckBox* MainWindow::colCheckBox [private]

Definition at line 358 of file mainwindow.h.

12.4.4.18 QCheckBox* MainWindow::commSACheckBox [private]

Definition at line 360 of file mainwindow.h.

12.4.4.19 QFrame* MainWindow::dropletParams [private]

Definition at line 388 of file mainwindow.h.

12.4.4.20 QVector<QStringList> MainWindow::droplets

Definition at line 442 of file mainwindow.h.

12.4.4.21 QTableWidget* MainWindow::dropletTableWidget [private]

Definition at line 331 of file mainwindow.h.

12.4.4.22 QLabel *dropProgramsLabel QComboBox * MainWindow::dropProgramsCombo [private]

Qt widgets for selecting the droplet program.

Definition at line 328 of file mainwindow.h.

12.4.4.23 QLabel* MainWindow::dropProgramsLabel [private]

Definition at line 327 of file mainwindow.h.

12.4.4.24 QVBoxLayout* MainWindow::dropProgramsLayout [private]

Definition at line 330 of file mainwindow.h.

12.4.4.25 QListView* MainWindow::dropProgramsList [private]

Definition at line 329 of file mainwindow.h.

12.4.4.26 QDoubleSpinBox* MainWindow::dropRadBox [private]

Definition at line 318 of file mainwindow.h.

12.4.4.27 QLabel* MainWindow::dropRadLabel [private]

Definition at line 319 of file mainwindow.h.

12.4.4.28 QDoubleSpinBox *dropRadBox QLabel *dropRadLabel QHBoxLayout * MainWindow::dropRadLayout [private]

Qt widgets for selecting the droplet radius parameter.

Definition at line 320 of file mainwindow.h.

12.4.4.29 QString MainWindow::fileName [private]

Initialized to the assest directory.

Definition at line 254 of file mainwindow.h.

12.4.4.30 QPushButton *launchSimulation QPushButton * MainWindow::launchRenderWidget [private]

Qt buttons for launching the simulator or renderer.

Definition at line 349 of file mainwindow.h.

12.4.4.31 QPushButton* MainWindow::launchSimulation [private]

Definition at line 348 of file mainwindow.h.

12.4.4.32 QFrame* MainWindow::loadSetupFile [private]

Definition at line 387 of file mainwindow.h.

12.4.4.33 QComboBox* MainWindow::loadSetupFileCombo [private]

Definition at line 289 of file mainwindow.h.

12.4.4.34 QLabel* MainWindow::loadSetupFileLabel [private]

Definition at line 288 of file mainwindow.h.

12.4.4.35 QLabel *loadSetupFileLabel QComboBox *loadSetupFileCombo QListView *loadSetupFileList QVBoxLayout * MainWindow::loadSetupFileLayout [private]

Qt widgets for selecting the default or a custom floor.

Definition at line 291 of file mainwindow.h.

12.4.4.36 QListView* MainWindow::loadSetupFileList [private]

Definition at line 290 of file mainwindow.h.

12.4.4.37 QCheckBox* MainWindow::logCheckBox [private]

Definition at line 356 of file mainwindow.h.

12.4.4.38 QVBoxLayout* MainWindow::logLayout [private]

Definition at line 369 of file mainwindow.h.

12.4.4.39 QVBoxLayout *logLayout QHBoxLayout *microLayout QHBoxLayout *macroLayout QWidget * MainWindow::logWidget [private]

Qt widgets for combine all of the logging widgets, for show hide capability.

Definition at line 372 of file mainwindow.h.

12.4.4.40 QHBoxLayout* MainWindow::macroLayout [private]

Definition at line 371 of file mainwindow.h.

12.4.4.41 QCheckBox* MainWindow::macroRedCheckBox [private]

Definition at line 361 of file mainwindow.h.

12.4.4.42 QCheckBox *logCheckBox QCheckBox *posCheckBox QCheckBox *colCheckBox QCheckBox *rotCheckBox QCheckBox *commSACheckBox QCheckBox *macroRedCheckBox QCheckBox * MainWindow::macroSACheckBox [private]

Qt checkboxes for selecting if the user wants to log data, and what data he/she wants to log.

Definition at line 362 of file mainwindow.h.

12.4.4.43 QHBoxLayout* MainWindow::microLayout [private]

Definition at line 370 of file mainwindow.h.

12.4.4.44 QSpinBox* MainWindow::numColBox [private]

Definition at line 270 of file mainwindow.h.

12.4.4.45 QLabel* MainWindow::numColLabel [private]

Definition at line 271 of file mainwindow.h.

12.4.4.46 QSpinBox *numColBox QLabel *numColLabel QHBoxLayout * MainWindow::numColLayout [private]

Qt widgets for selecting the number of arena columns parameter.

Definition at line 272 of file mainwindow.h.

12.4.4.47 QSpinBox* MainWindow::numDropBox [private]

Definition at line 261 of file mainwindow.h.

12.4.4.48 QLabel* MainWindow::numDropLabel [private]

Definition at line 262 of file mainwindow.h.

12.4.4.49 QSpinBox *numDropBox QLabel *numDropLabel QHBoxLayout * MainWindow::numDropLayout [private]

Qt widgets for selecting the number of droplets parameter.

Definition at line 263 of file mainwindow.h.

12.4.4.50 QSpinBox* MainWindow::numRowBox [private]

Definition at line 279 of file mainwindow.h.

12.4.4.51 QLabel* MainWindow::numRowLabel [private]

Definition at line 280 of file mainwindow.h.

12.4.4.52 QSpinBox *numRowBox QLabel *numRowLabel QHBoxLayout * MainWindow::numRowLayout [private]

Qt widgets for selecting the number of arena rows parameter.

Definition at line 281 of file mainwindow.h.

12.4.4.53 QVector<QStringList> MainWindow::objects

Definition at line 443 of file mainwindow.h.

12.4.4.54 QPushButton* MainWindow::pause_button [private]

Definition at line 379 of file mainwindow.h.

12.4.4.55 QCheckBox* MainWindow::posCheckBox [private]

Definition at line 357 of file mainwindow.h.

12.4.4.56 QFileSystemWatcher MainWindow::projectionWatcher [private]

Definition at line 392 of file mainwindow.h.

12.4.4.57 QComboBox* MainWindow::projectorImageCombo [private]

Definition at line 339 of file mainwindow.h.

12.4.4.58 QLabel* MainWindow::projectorImageLabel [private]

Definition at line 338 of file mainwindow.h.

12.4.4.59 QListView* MainWindow::projectorImageList [private]

Definition at line 340 of file mainwindow.h.

12.4.4.60 QLabel *projectorImageLabel QComboBox *projectorImageCombo QListView *projectorImageList QVBoxLayout * MainWindow::projectorImagesLayout [private]

Qt widgets for selecting the projected image.

Definition at line 341 of file mainwindow.h.

12.4.4.61 QPushButton *pause_button QPushButton *resume_button QPushButton * MainWindow::reset_button [private]

Qt buttons for playback options.

Definition at line 381 of file mainwindow.h.

12.4.4.62 QPushButton* MainWindow::resume_button [private]

Definition at line 380 of file mainwindow.h.

12.4.4.63 QCheckBox* MainWindow::rotCheckBox [private]

Definition at line 359 of file mainwindow.h.

12.4.4.64 QVBoxLayout* MainWindow::rowColLayout [private]

Definition at line 308 of file mainwindow.h.

12.4.4.65 QVBoxLayout *rowColLayout QWidget * MainWindow::rowColWidget [private]

The row widgets and column widgets are shown or hid, so they are within a single rowColWid.

Definition at line 309 of file mainwindow.h.

12.4.4.66 QFileSystemWatcher MainWindow::setupWatcher [private]

Definition at line 394 of file mainwindow.h.

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

- mainwindow.h
- mainwindow.cpp

12.5 MeshManager Class Reference

Container class for meshes.

`#include <meshmanager.h>`

Classes

- struct vertexData_t

  Structure type to contain packed vertex attributes.

Public Member Functions

- MeshManager ()

  Default constructor.
- ~MeshManager ()

  Destructor.
- void bindBuffer ()

  Bind buffer.
- void disableAttributeArrays ()

  Disables the attribute arrays.
- void draw ()

  Draws this object.
- void enableAttributeArrays ()

  Enables the attribute arrays.
- void freeModel ()

  Free model.
- GLuint getNormalOffset ()

  Gets normal offset.
- GLuint getNormalStride ()

  Gets normal stride.
- GLuint getNumVertices ()

  Gets number vertices.
- GLuint getTexOffset ()

  Gets tex offset.
- GLuint getTexStride ()

  Gets tex stride.
- GLuint getVertexOffset ()

  Gets vertex offset.
- GLuint getVertexStride ()

  Gets vertex stride.
- bool loadFile (QString fileName)

  Loads a file.
- void setAttribLoc (int locVertex, int locNormal, int locTexCoords)

  Sets attribute location.
- bool setFile (QString fileName)

  Assigns a filename, but defers the actual loading of the file until first use.
- void unbindBuffer ()

  Unbind buffer.

Private Attributes

- QString _fileName

  Filename of the file.
- bool _fileSet

  true to file set.
- bool _hasIndex

  true if this object has index.
- bool _hasNormals

  true if this object has normals.
- bool _hasTexCoords

  true if this object has tex coords.
- GLuint _indexBufferID

  Identifier for the index buffer.
- int _indexCount

  Number of indexes.
- int _indexedVertexCount

  Number of vertices.
- bool _modelLoaded

  true if model loaded.
- struct {
  GLint normal
  GLint texCoords
  GLint vertex
  } _shaderAttribLocs
- QGLBuffer _VBO

  The vbo.
- GLuint _VBOID

  The vboid.
- int _vertexCount

  Number of vertices.

12.5.1 Detailed Description

Container class for meshes.

Definition at line 32 of file meshmanager.h.

12.5.2 Constructor & Destructor Documentation

12.5.2.1 MeshManager::MeshManager ( )

Default constructor.

Definition at line 9 of file meshmanager.cpp.

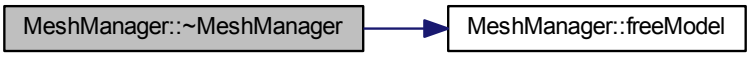
Here is the call graph for this function:

12.5.2.2 MeshManager::~MeshManager ( )

Destructor.

Definition at line 26 of file meshmanager.cpp.

Here is the call graph for this function:



12.5.3 Member Function Documentation

12.5.3.1 void MeshManager::bindBuffer ( )

Bind buffer.

Definition at line 323 of file meshmanager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

12.5.3.2 void MeshManager::disableAttributeArrays ( )

Disables the attribute arrays.

Definition at line 410 of file meshmanager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

12.5.3.3 void MeshManager::draw ( )

Draws this object.

Definition at line 431 of file meshmanager.cpp.

Here is the call graph for this function:

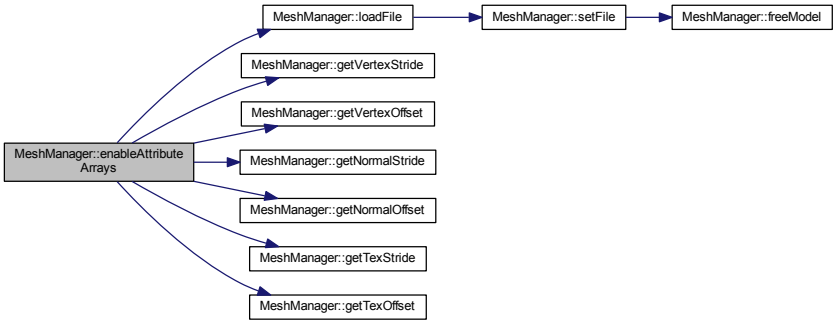
Here is the caller graph for this function:

12.5.3.4 void MeshManager::enableAttributeArrays ( )

Enables the attribute arrays.

Definition at line 383 of file meshmanager.cpp.

Here is the call graph for this function:



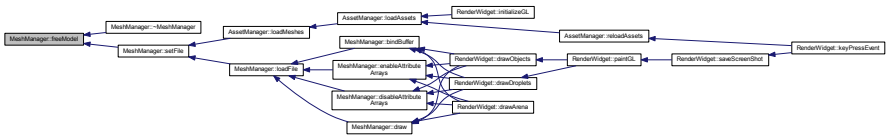
Here is the caller graph for this function:

12.5.3.5 void MeshManager::freeModel ( )

Free model.

Definition at line 458 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.6 GLuint MeshManager::getNormalOffset ( )

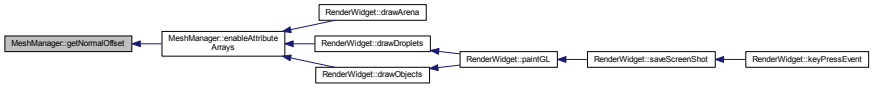
Gets normal offset.

Returns

The normal offset.

Definition at line 373 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.7 GLuint MeshManager::getNormalStride ( )

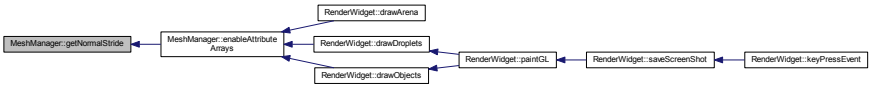
Gets normal stride.

Returns

The normal stride.

Definition at line 361 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.8 GLuint MeshManager::getNumVertices ( )

Gets number vertices.

Returns

The number vertices.

Definition at line 378 of file meshmanager.cpp.

12.5.3.9 GLuint MeshManager::getTexOffset ( )

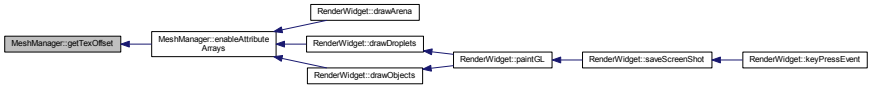
Gets tex offset.

Returns

The tex offset.

Definition at line 369 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.10 GLuint MeshManager::getTexStride ( )

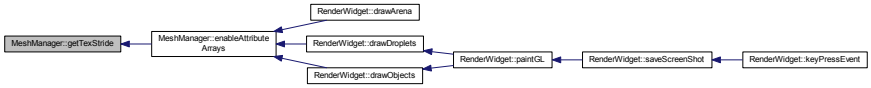
Gets tex stride.

Returns

The tex stride.

Definition at line 356 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.11 GLuint MeshManager::getVertexOffset ( )

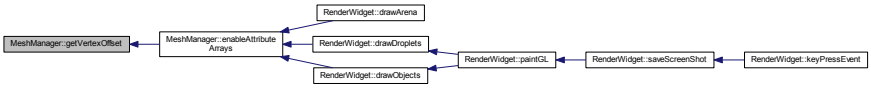
Gets vertex offset.

Returns

The vertex offset.

Definition at line 365 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.12 GLuint MeshManager::getVertexStride ( )

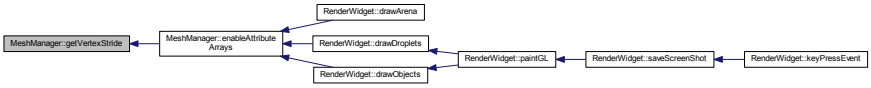
Gets vertex stride.

Returns

The vertex stride.

Definition at line 352 of file meshmanager.cpp.

Here is the caller graph for this function:



12.5.3.13 bool MeshManager::loadFile (QString *fileName*)

Loads a file.

Parameters

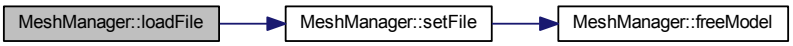
fileName	Filename of the file.

Returns

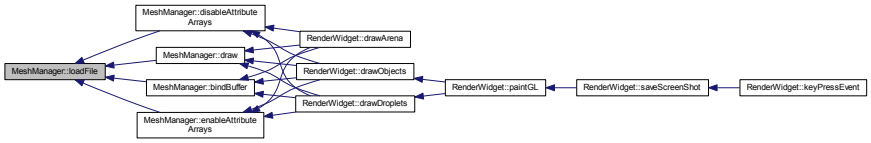
true if it succeeds, false if it fails.

Definition at line 46 of file meshmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.5.3.14 void MeshManager::setAttribLoc (int *locVertex*, int *locNormal*, int *locTexCoords*)

Sets attribute location.

Parameters

locVertex	The location vertex.
locNormal	The location normal.
locTexCoords	The location tex coords.

Definition at line 451 of file meshmanager.cpp.

Here is the caller graph for this function:

12.5.3.15 bool MeshManager::setFile (QString *fileName*)

Assigns a filename, but defers the actual loading of the file until first use.

Parameters

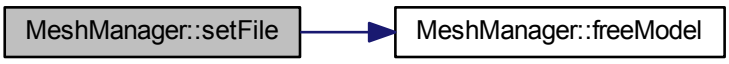
fileName	Filename of the file.

Returns

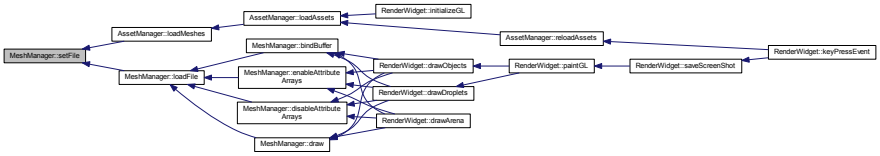
true if it succeeds, false if it fails.

Definition at line 31 of file meshmanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.5.3.16 void MeshManager::unbindBuffer ( )

Unbind buffer.

Definition at line 343 of file meshmanager.cpp.

Here is the caller graph for this function:

12.5.4 Member Data Documentation

12.5.4.1 QString MeshManager::_fileName [private]

Filename of the file.

Definition at line 277 of file meshmanager.h.

12.5.4.2 bool MeshManager::_fileSet [private]

true to file set.

Definition at line 283 of file meshmanager.h.

12.5.4.3 bool MeshManager::_hasIndex [private]

true if this object has index.

Definition at line 253 of file meshmanager.h.

12.5.4.4 bool MeshManager::_hasNormals [private]

true if this object has normals.

Definition at line 265 of file meshmanager.h.

12.5.4.5 bool MeshManager::_hasTexCoords [private]

true if this object has tex coords.

Definition at line 259 of file meshmanager.h.

12.5.4.6 GLuint MeshManager::_indexBufferID [private]

Identifier for the index buffer.

Definition at line 313 of file meshmanager.h.

12.5.4.7 int MeshManager::_indexCount [private]

Number of indexes.

Definition at line 289 of file meshmanager.h.

12.5.4.8 int MeshManager::_indexedVertexCount [private]

Number of vertices.

Definition at line 295 of file meshmanager.h.

12.5.4.9 bool MeshManager::_modelLoaded [private]

true if model loaded.

Definition at line 271 of file meshmanager.h.

12.5.4.10 struct { ... } MeshManager::_shaderAttribLocs [private]

12.5.4.11 QGLBuffer MeshManager::_VBO [private]

The vbo.

Definition at line 307 of file meshmanager.h.

12.5.4.12 GLuint MeshManager::_VBOID [private]

The vboid.

Definition at line 319 of file meshmanager.h.

12.5.4.13 int MeshManager::_vertexCount [private]

Number of vertices.

Definition at line 301 of file meshmanager.h.

12.5.4.14 GLint MeshManager::normal

Definition at line 240 of file meshmanager.h.

12.5.4.15 GLint MeshManager::texCoords

Definition at line 241 of file meshmanager.h.

12.5.4.16 GLint MeshManager::vertex

Definition at line 239 of file meshmanager.h.

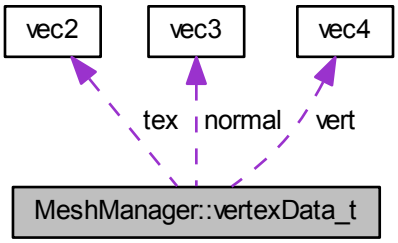
The documentation for this class was generated from the following files:

- meshmanager.h
- meshmanager.cpp

12.6 MeshManager::vertexData_t Struct Reference

Structure type to contain packed vertex attributes.

Collaboration diagram for MeshManager::vertexData_t:



Public Attributes

- vec3 normal

  The normal vector.
- vec2 tex

  The texure coordates.
- vec4 vert

  The vertex coordinates.

12.6.1 Detailed Description

Structure type to contain packed vertex attributes.

Definition at line 216 of file meshmanager.h.

12.6.2 Member Data Documentation

12.6.2.1 vec3 MeshManager::vertexData_t::normal

The normal vector.

Definition at line 228 of file meshmanager.h.

12.6.2.2 vec2 MeshManager::vertexData_t::tex

The texure coordates.

Definition at line 234 of file meshmanager.h.

12.6.2.3 vec4 MeshManager::vertexData_t::vert

The vertex coordinates.

Definition at line 222 of file meshmanager.h.

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

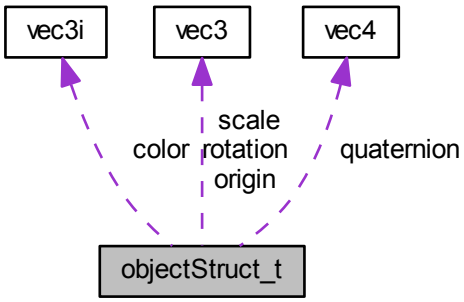
- meshmanager.h

12.7 objectStruct_t Struct Reference

Defines properties about physical objects that are emitted from SimInterface.

`#include <structs.h>`

Collaboration diagram for objectStruct_t:



Public Attributes

- bool changed
- vec3i color
- int objectID
- float objectRadius
- vec3 origin
- object_t oType
- vec4 quaternion
- vec3 rotation
- vec3 scale

12.7.1 Detailed Description

Defines properties about physical objects that are emitted from SimInterface.

Definition at line 225 of file structs.h.

12.7.2 Member Data Documentation

12.7.2.1 bool objectStruct_t::changed

Definition at line 234 of file structs.h.

12.7.2.2 vec3i objectStruct_t::color

Definition at line 233 of file structs.h.

12.7.2.3 int objectStruct_t::objectID

Definition at line 226 of file structs.h.

12.7.2.4 float objectStruct_t::objectRadius

Definition at line 227 of file structs.h.

12.7.2.5 vec3 objectStruct_t::origin

Definition at line 230 of file structs.h.

12.7.2.6 object_t objectStruct_t::oType

Definition at line 228 of file structs.h.

12.7.2.7 vec4 objectStruct_t::quaternion

Definition at line 232 of file structs.h.

12.7.2.8 vec3 objectStruct_t::rotation

Definition at line 231 of file structs.h.

12.7.2.9 vec3 objectStruct_t::scale

Definition at line 229 of file structs.h.

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

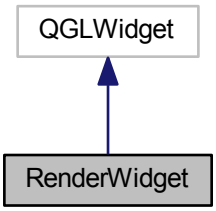
- structs.h

12.8 RenderWidget Class Reference

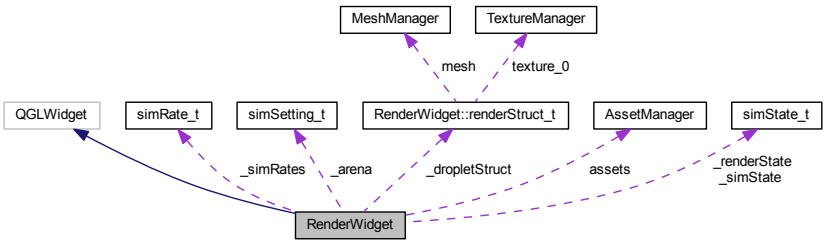
View controller for QGLWidget that renders the simulation state.

```
#include <RenderWidget.h>
```

Inheritance diagram for RenderWidget:



Collaboration diagram for RenderWidget:



Classes

- struct renderStruct_t

  Defines a structure that represents a renderable object.

Public Slots

- void updateArena (simSetting_t settings)

  Updates the arena described by settings.
- void updatePause (bool paused)

  Tells RenderWidget about the pause status of the simulator.
- void updateRate (simRate_t rates)

  Updates the rate described by rates.
- void updateState (simState_t stateInfo)

  Updates the state described by stateInfo.

Signals

- void decreaseRate (void)

  Decrease rate.
- void increaseRate (void)

  Increase rate.

- void requestNewDroplet (float x, float y, droplet_t dType=March, int dropletID=0)

  Request new droplet.
- void restart (void)

  Restarts this object.
- void toggleLimit (void)

  Toggle limit.
- void togglePause (void)

  Toggle pause.

Public Member Functions

- RenderWidget (const QGLFormat &format, QWidget *parent=0)

  Constructor.
- ~RenderWidget ()

  Destructor.
- QSize sizeHint () const

  Gets the size hint.

Protected Member Functions

- void closeEvent (QCloseEvent *event)

  Event invoked when instance is closed.
- void drawArena ()

  Draw arena.
- void drawDroplets ()

  Draw droplets.
- void drawHUD ()

  Draw HUD.
- void drawObjects ()

  Draw objects.
- float getRandomf (float min, float max)

  Gets a random float.
- void initializeGL ()

  Initializes the gl.
- void keyPressEvent (QKeyEvent *event)

  Key press event.
- void keyReleaseEvent (QKeyEvent *event)

  Key release event.
- void mouseMoveEvent (QMouseEvent *event)

  Mouse move event.
- void mousePressEvent (QMouseEvent *event)

  Mouse press event.
- void mouseReleaseEvent (QMouseEvent *event)

  Mouse release event.
- void paintGL ()

  Paints the gl.
- void processInput (float timeSinceLastUpdate)

  Process the input described by timeSinceLastUpdate.
- void resizeGL (int width, int height)

  Resize gl.

- void setFPS (int FPS)
  Sets the FPS.
- void setupRenderStructs ()
  Sets up the render structs.
- void timerEvent (QTimerEvent *event)
  timing control.
- void updateCamera ()
  Updates the camera.
- void wheelEvent (QWheelEvent *event)
  input handling.

Private Member Functions

- glm::mat4 makeModelMatrix (glm::vec3 scale, glm::vec3 rotate, glm::vec3 translation)
  Makes model matrix.
- void saveScreenShot (int width, int height)

Private Attributes

- simSetting_t _arena
  The arena.
- QVector< renderStruct_t > _arenaObjects
  QVector containing local objects to render.
- bool _assetsLoaded
  true if assets loaded.
- struct {
  glm::vec3 lightDir
  unsigned int mode
  GLfloat pan
  QMatrix4x4 projectionMatrix
  GLfloat radius
  GLfloat rotHoriz
  GLfloat rotVert
  GLfloat tilt
  QMatrix4x4 viewMatrix
  GLfloat x
  GLfloat y
  GLfloat z
  } _camera

- bool _drawHelp
  true to draw help.
- renderStruct_t _dropletStruct
  renderStruct_t that contains necessary information about how to render droplets.
- QElapsedTimer _eventTimer
  one off timer, only declared here to keep from needing to create/destroy timers inside functions.
- bool _hud
  Enables or disables the HUD.
- struct {
  int framesSinceLastUpdate
  bool paused
  float simRealTimeRatio
  float simStepSize
  } _hudInfo

- struct {
  bool A
  bool D
  bool E
  bool Minus
  bool Plus
  bool Q
  bool S
  bool W
  } _keysDown

- struct {
  bool leftButtonHeldDown
  GLfloat origHoriz
  GLfloat origPan
  GLfloat origTilt
  GLfloat origVert
  int startX
  int startY
  } _mouseStatus

- double _msElapsedRenderer
- double _msElapsedUpdate

  QTime _runTime;.
- QVector< renderStruct_t > _objectStructs

  QVector containing information on how to render physics objects.
- struct {
  GLuint handle
  GLuint height
  QImage texture
  bool valid
  GLuint width
  } _projectionTexture

- int _renderDebug

  Enables or disables debug rendering.
- bool _renderLock

  Flag to enforce reentrancy of rendering.
- simState_t _renderState
- QElapsedTimer _renderTimer

  timer related objects/variables.
- simRate_t _simRates

  Rate of the simulation.
- simState_t _simState

  State of the simulation.
- QAtomicInt _simStateLock

  true to lock, false to unlock the simulation state.
- int _targetFPS

  FPS variables.
- float _targetFrameTime

  Time of the target frame.
- int _timerID

  Identifier for the timer.
- QElapsedTimer _updateTimer

The update timer.

- AssetManager assets

  The assets.

12.8.1 Detailed Description

View controller for QGLWidget that renders the simulation state.

Definition at line 51 of file RenderWidget.h.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 RenderWidget::RenderWidget (const QGLFormat & *format*, QWidget * *parent* = 0)

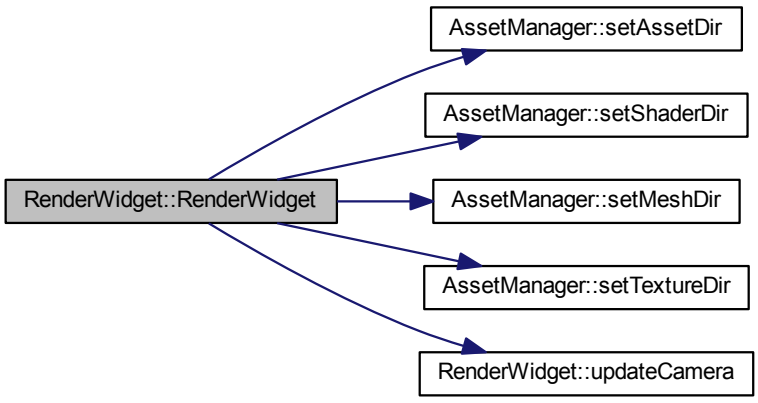
Constructor.

Parameters

	format	Describes the format to use.
in,out	*parent*	(Optional) If non-null, (Optional) the parent.

Definition at line 9 of file RenderWidget.cpp.

Here is the call graph for this function:

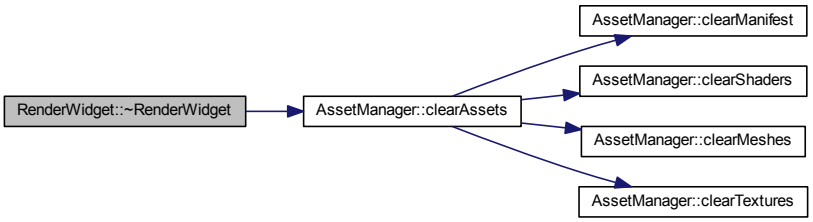


12.8.2.2 RenderWidget::~RenderWidget ( )

Destructor.

Definition at line 60 of file RenderWidget.cpp.

Here is the call graph for this function:



12.8.3 Member Function Documentation

12.8.3.1 void RenderWidget::closeEvent (QCloseEvent * *event*) [protected]

Event invoked when instance is closed.

Parameters

in,out	*event*	If non-null, the event initiating the closing.

Definition at line 1307 of file RenderWidget.cpp.

12.8.3.2 void RenderWidget::decreaseRate (void) [signal]

Decrease rate.

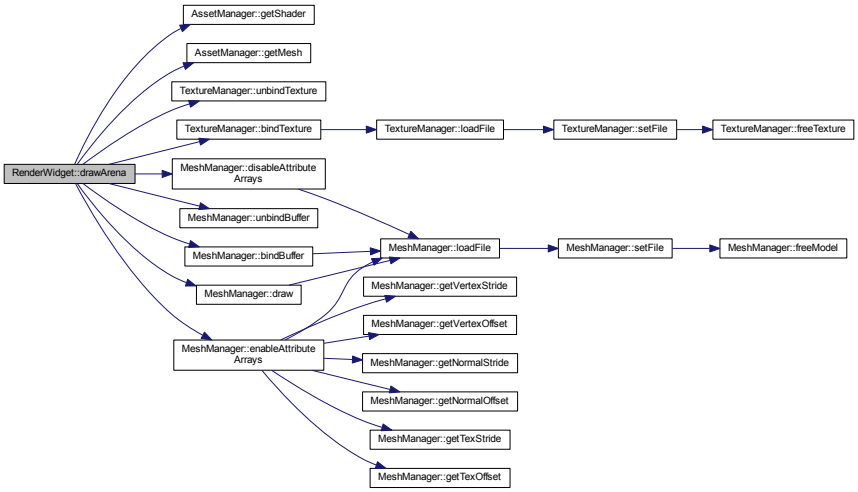
Here is the caller graph for this function:

12.8.3.3 void RenderWidget::drawArena ( ) [protected]

Draw arena.

Definition at line 315 of file RenderWidget.cpp.

Here is the call graph for this function:

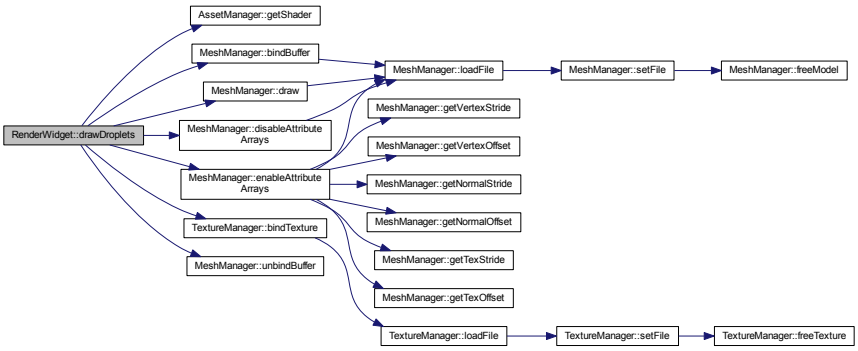


12.8.3.4 void RenderWidget::drawDroplets ( ) [protected]

Draw droplets.

Definition at line 464 of file RenderWidget.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.8.3.5 void RenderWidget::drawHUD ( ) [protected]

Draw HUD.

Definition at line 743 of file RenderWidget.cpp.

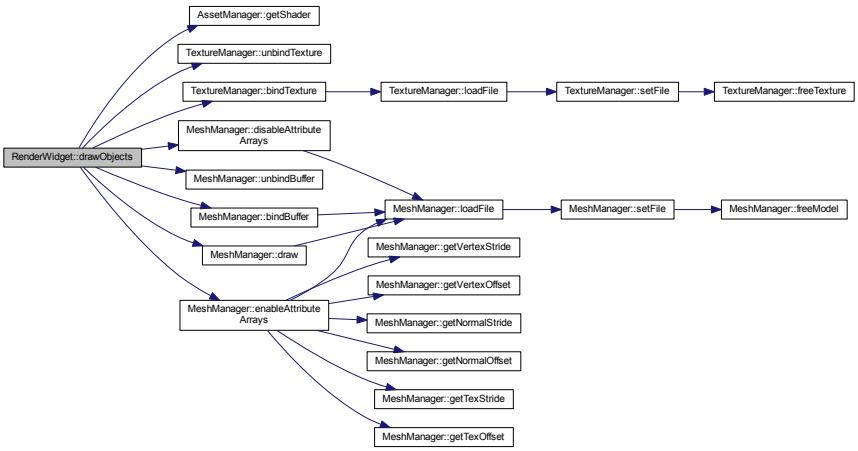
Here is the caller graph for this function:

12.8.3.6 void RenderWidget::drawObjects ( ) [protected]

Draw objects.

Definition at line 571 of file RenderWidget.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.8.3.7 float RenderWidget::getRandomf (float *min*, float *max*) [protected]

Gets a random float.

Parameters

min	The minimum.
max	The maximum.

Returns

The random number.

Definition at line 1319 of file RenderWidget.cpp.

Here is the caller graph for this function:

12.8.3.8 void RenderWidget::increaseRate (void) [signal]

Increase rate.

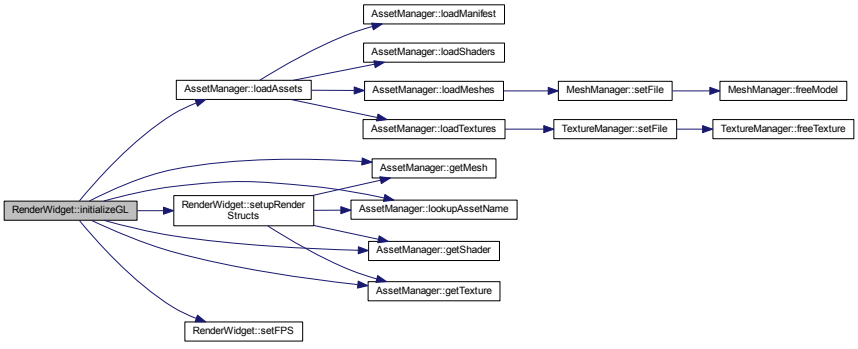
Here is the caller graph for this function:

12.8.3.9 void RenderWidget::initializeGL ( ) [protected]

Initializes the gl.

Definition at line 87 of file RenderWidget.cpp.

Here is the call graph for this function:



12.8.3.10 void RenderWidget::keyPressEvent (QKeyEvent * *event*) [protected]

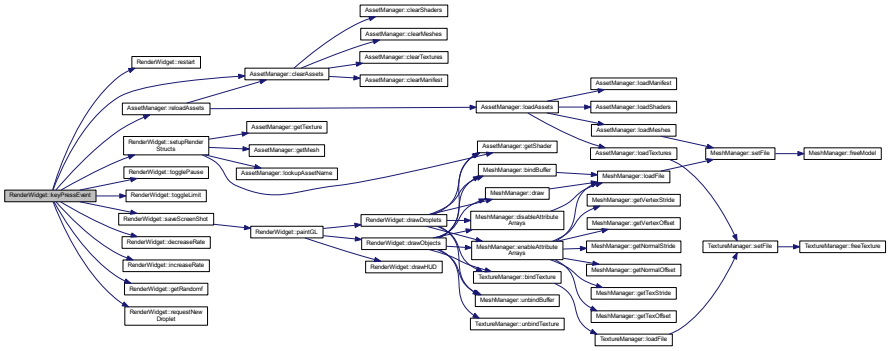
Key press event.

Parameters

in,out	*event*	If non-null, the event.

Definition at line 83 of file RW_input.cpp.

Here is the call graph for this function:



12.8.3.11 void RenderWidget::keyReleaseEvent (QKeyEvent * *event*) [protected]

Key release event.

Parameters

in,out	*event*	If non-null, the event.

Definition at line 287 of file RW_input.cpp.

12.8.3.12 glm::mat4 RenderWidget::makeModelMatrix (glm::vec3 *scale*, glm::vec3 *rotate*, glm::vec3 *translation*) [private]

Makes model matrix.

Parameters

scale	The scale.
rotate	The rotate.
translation	The translation.

Returns

.

Definition at line 1260 of file RenderWidget.cpp.

12.8.3.13 void RenderWidget::mouseMoveEvent (QMouseEvent * *event*) [protected]

Mouse move event.

Parameters

in,out	*event*	If non-null, the event.

Definition at line 10 of file RW_input.cpp.

12.8.3.14 void RenderWidget::mousePressEvent (QMouseEvent * *event*) [protected]

Mouse press event.

Parameters

in,out	*event*	If non-null, the event.

Definition at line 27 of file RW_input.cpp.

12.8.3.15 void RenderWidget::mouseReleaseEvent (QMouseEvent * *event*) [protected]

Mouse release event.

Parameters

in,out	*event*	If non-null, the event.

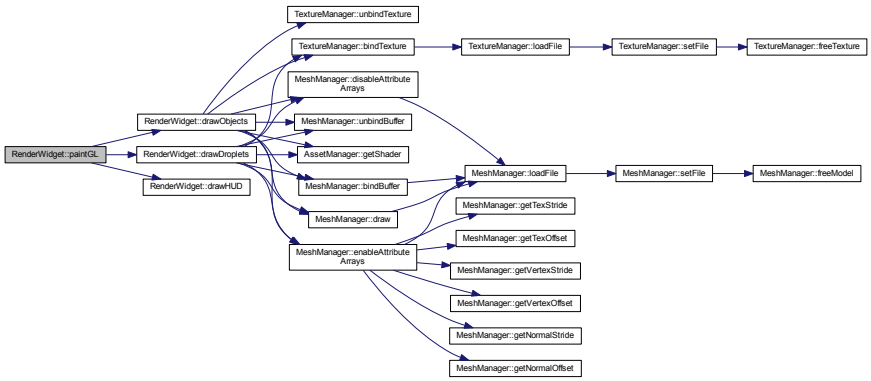
Definition at line 45 of file RW_input.cpp.

12.8.3.16 void RenderWidget::paintGL ( ) [protected]

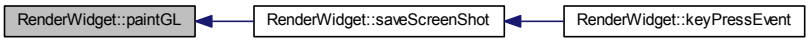
Paints the gl.

Definition at line 222 of file RenderWidget.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.8.3.17 void RenderWidget::processInput (float *timeSinceLastUpdate*) [protected]

Process the input described by timeSinceLastUpdate.

Parameters

timeSinceLast-Update	The time since last update.

Definition at line 330 of file RW_input.cpp.

Here is the caller graph for this function:

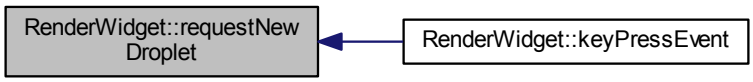
12.8.3.18 void RenderWidget::requestNewDroplet (float *x*, float *y*, droplet_t *dType* = March, int *dropletID* = 0) [signal]

Request new droplet.

Parameters

x	The x coordinate.
y	The y coordinate.
dType	(Optional) the type.
dropletID	(Optional) identifier for the droplet.

Here is the caller graph for this function:



12.8.3.19 void RenderWidget::resizeGL (int *width*, int *height*) [protected]

Resize gl.

Parameters

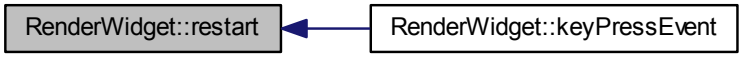
width	The width.
height	The height.

Definition at line 140 of file RenderWidget.cpp.

12.8.3.20 void RenderWidget::restart (void) [signal]

Restarts this object.

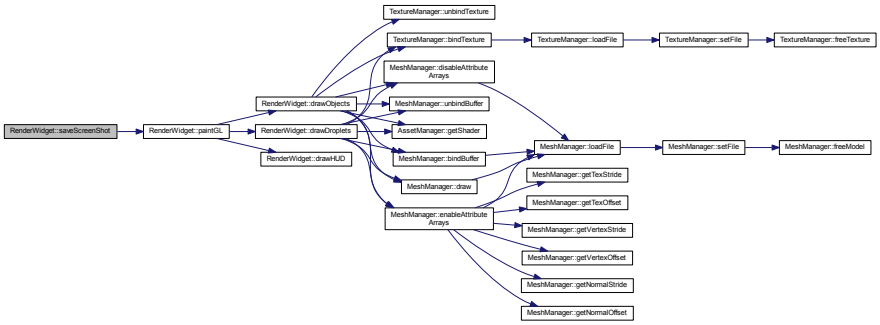
Here is the caller graph for this function:



12.8.3.21 void RenderWidget::saveScreenShot (int *width*, int *height*) [private]

Definition at line 957 of file RenderWidget.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.8.3.22 void RenderWidget::setFPS (int *FPS*) [protected]

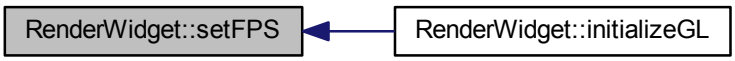
Sets the FPS.

Parameters

FPS	The FPS.

Definition at line 121 of file RenderWidget.cpp.

Here is the caller graph for this function:

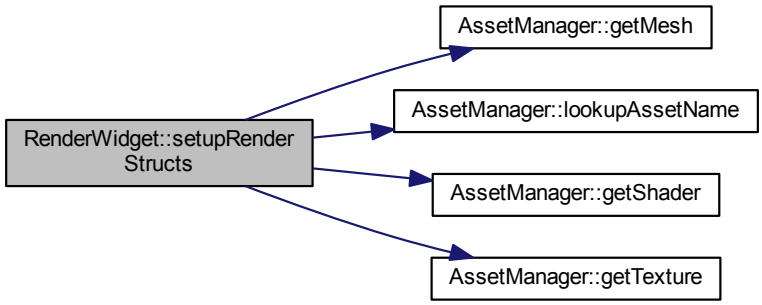


12.8.3.23 void RenderWidget::setupRenderStructs ( ) [protected]

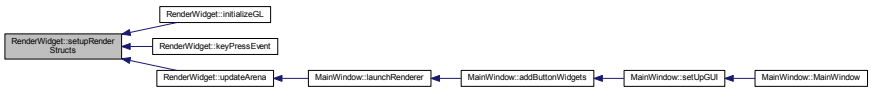
Sets up the render structs.

Definition at line 1035 of file RenderWidget.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.8.3.24 QSize RenderWidget::sizeHint ( ) const

Gets the size hint.

Returns

.

Definition at line 81 of file RenderWidget.cpp.

12.8.3.25 void RenderWidget::timerEvent (QTimerEvent * *event*) [protected]

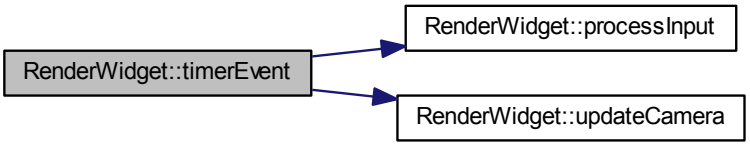
timing control.

Parameters

in,out	*event*	If non-null, the event.

Definition at line 1273 of file RenderWidget.cpp.

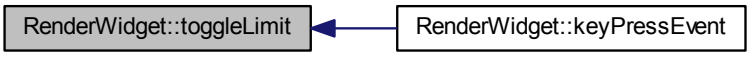
Here is the call graph for this function:



12.8.3.26 void RenderWidget::toggleLimit (void) [signal]

Toggle limit.

Here is the caller graph for this function:



12.8.3.27 signals:void RenderWidget::togglePause (void) [signal]

Toggle pause.

Here is the caller graph for this function:

12.8.3.28 void RenderWidget::updateArena (simSetting_t *settings*) [slot]

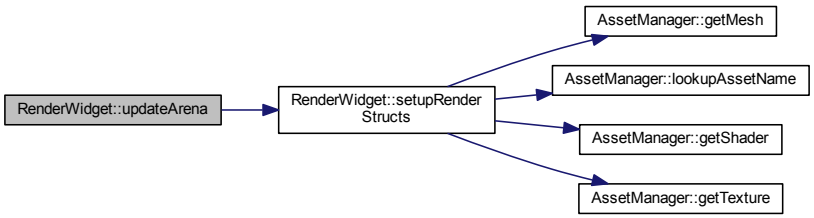
Updates the arena described by settings.

Parameters

settings	Options for controlling the operation.

Definition at line 24 of file RW_slots.cpp.

Here is the call graph for this function:



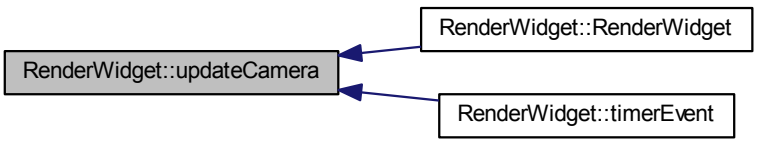
Here is the caller graph for this function:

12.8.3.29 void RenderWidget::updateCamera ( ) [protected]

Updates the camera.

Definition at line 151 of file RenderWidget.cpp.

Here is the caller graph for this function:



12.8.3.30 void RenderWidget::updatePause (bool *paused*) [slot]

Tells RenderWidget about the pause status of the simulator.

Parameters

paused	true if paused.

Definition at line 34 of file RW_slots.cpp.

12.8.3.31 void RenderWidget::updateRate (simRate_t *rates*) [slot]

Updates the rate described by rates.

Parameters

rates	The rates.

Definition at line 43 of file RW_slots.cpp.

Here is the caller graph for this function:

12.8.3.32 void RenderWidget::updateState (simState_t *stateInfo*) [slot]

Updates the state described by stateInfo.

Parameters

stateInfo	Information describing the state.

Definition at line 10 of file RW_slots.cpp.

Here is the caller graph for this function:

12.8.3.33 void RenderWidget::wheelEvent (QWheelEvent * *event*) [protected]

input handling.

Parameters

in,out	*event*	If non-null, the event.

Definition at line 64 of file RW_input.cpp.

12.8.4 Member Data Documentation

12.8.4.1 simSetting_t RenderWidget::_arena [private]

The arena.

Definition at line 589 of file RenderWidget.h.

12.8.4.2 QVector<renderStruct_t> RenderWidget::_arenaObjects [private]

QVector containing local objects to render.

Definition at line 509 of file RenderWidget.h.

12.8.4.3 bool RenderWidget::_assetsLoaded [private]

true if assets loaded.

Definition at line 467 of file RenderWidget.h.

12.8.4.4 struct { ... } RenderWidget::_camera [private]

12.8.4.5 bool RenderWidget::_drawHelp [private]

true to draw help.

Definition at line 491 of file RenderWidget.h.

12.8.4.6 renderStruct_t RenderWidget::_dropletStruct [private]

renderStruct_t that contains necessary information about how to render droplets.

Definition at line 515 of file RenderWidget.h.

12.8.4.7 QElapsedTimer RenderWidget::_eventTimer [private]

one off timer, only declared here to keep from needing to create/destroy timers inside functions.

Definition at line 433 of file RenderWidget.h.

12.8.4.8 bool RenderWidget::_hud [private]

Enables or disables the HUD.

Definition at line 473 of file RenderWidget.h.

12.8.4.9 struct { ... } RenderWidget::_hudInfo [private]

12.8.4.10 struct { ... } RenderWidget::_keysDown [private]

12.8.4.11 struct { ... } RenderWidget::_mouseStatus [private]

12.8.4.12 double RenderWidget::_msElapsedRenderer [private]

Definition at line 443 of file RenderWidget.h.

12.8.4.13 double _msElapsedRenderer RenderWidget::_msElapsedUpdate [private]

QTime _runTime;.

Returns

The milliseconds elapsed update.

Definition at line 443 of file RenderWidget.h.

12.8.4.14 QVector<renderStruct_t> RenderWidget::_objectStructs [private]

QVector containing information on how to render physics objects.

Definition at line 521 of file RenderWidget.h.

12.8.4.15 struct { ... } RenderWidget::_projectionTexture [private]

12.8.4.16 int RenderWidget::_renderDebug [private]

Enables or disables debug rendering.

_renderDebug has three different levels: 0 - Disabled 1 - Display additional information on the HUD 2 - Enable debug rendering (Falls back to debug shaders)

Definition at line 485 of file RenderWidget.h.

12.8.4.17 bool RenderWidget::_renderLock [private]

Flag to enforce reentrancy of rendering.

Definition at line 461 of file RenderWidget.h.

12.8.4.18 simState_t RenderWidget::_renderState [private]

Definition at line 596 of file RenderWidget.h.

12.8.4.19 QElapsedTimer RenderWidget::_renderTimer [private]

timer related objects/variables.

Definition at line 426 of file RenderWidget.h.

12.8.4.20 simRate_t RenderWidget::_simRates [private]

Rate of the simulation.

Definition at line 607 of file RenderWidget.h.

12.8.4.21 simState_t RenderWidget::_simState [private]

State of the simulation.

Definition at line 595 of file RenderWidget.h.

12.8.4.22 QAtomicInt RenderWidget::_simStateLock [private]

true to lock, false to unlock the simulation state.

Definition at line 602 of file RenderWidget.h.

12.8.4.23 int RenderWidget::_targetFPS [private]

FPS variables.

Definition at line 449 of file RenderWidget.h.

12.8.4.24 float RenderWidget::_targetFrameTime [private]

Time of the target frame.

Definition at line 455 of file RenderWidget.h.

12.8.4.25 int RenderWidget::_timerID [private]

Identifier for the timer.

Definition at line 503 of file RenderWidget.h.

12.8.4.26 QElapsedTimer RenderWidget::_updateTimer [private]

The update timer.

Definition at line 497 of file RenderWidget.h.

12.8.4.27 bool RenderWidget::A

Definition at line 567 of file RenderWidget.h.

12.8.4.28 AssetManager RenderWidget::assets [private]

The assets.

Definition at line 420 of file RenderWidget.h.

12.8.4.29 bool RenderWidget::D

Definition at line 569 of file RenderWidget.h.

12.8.4.30 bool RenderWidget::E

Definition at line 566 of file RenderWidget.h.

12.8.4.31 int RenderWidget::framesSinceLastUpdate

Definition at line 527 of file RenderWidget.h.

12.8.4.32 GLuint RenderWidget::handle

Definition at line 576 of file RenderWidget.h.

12.8.4.33 GLuint RenderWidget::height

Definition at line 575 of file RenderWidget.h.

12.8.4.34 bool RenderWidget::leftButtonHeldDown

Definition at line 551 of file RenderWidget.h.

12.8.4.35 glm::vec3 RenderWidget::lightDir

Definition at line 544 of file RenderWidget.h.

12.8.4.36 bool RenderWidget::Minus

Definition at line 570 of file RenderWidget.h.

12.8.4.37 unsigned int RenderWidget::mode

Definition at line 543 of file RenderWidget.h.

12.8.4.38 GLfloat RenderWidget::origHoriz

Definition at line 553 of file RenderWidget.h.

12.8.4.39 GLfloat RenderWidget::origPan

Definition at line 555 of file RenderWidget.h.

12.8.4.40 GLfloat RenderWidget::origTilt

Definition at line 556 of file RenderWidget.h.

12.8.4.41 GLfloat RenderWidget::origVert

Definition at line 554 of file RenderWidget.h.

12.8.4.42 GLfloat RenderWidget::pan

Definition at line 540 of file RenderWidget.h.

12.8.4.43 bool RenderWidget::paused

Definition at line 530 of file RenderWidget.h.

12.8.4.44 bool RenderWidget::Plus

Definition at line 571 of file RenderWidget.h.

12.8.4.45 QMatrix4x4 RenderWidget::projectionMatrix

Definition at line 537 of file RenderWidget.h.

12.8.4.46 bool RenderWidget::Q

Definition at line 564 of file RenderWidget.h.

12.8.4.47 GLfloat RenderWidget::radius

Definition at line 542 of file RenderWidget.h.

12.8.4.48 GLfloat RenderWidget::rotHoriz

Definition at line 541 of file RenderWidget.h.

12.8.4.49 GLfloat RenderWidget::rotVert

Definition at line 541 of file RenderWidget.h.

12.8.4.50 bool RenderWidget::S

Definition at line 568 of file RenderWidget.h.

12.8.4.51 float RenderWidget::simRealTimeRatio

Definition at line 528 of file RenderWidget.h.

12.8.4.52 float RenderWidget::simStepSize

Definition at line 529 of file RenderWidget.h.

12.8.4.53 int RenderWidget::startX

Definition at line 552 of file RenderWidget.h.

12.8.4.54 int RenderWidget::startY

Definition at line 552 of file RenderWidget.h.

12.8.4.55 QImage RenderWidget::texture

Definition at line 578 of file RenderWidget.h.

12.8.4.56 GLfloat RenderWidget::tilt

Definition at line 540 of file RenderWidget.h.

12.8.4.57 bool RenderWidget::valid

Definition at line 577 of file RenderWidget.h.

12.8.4.58 QMatrix4x4 RenderWidget::viewMatrix

Definition at line 538 of file RenderWidget.h.

12.8.4.59 bool RenderWidget::W

Definition at line 565 of file RenderWidget.h.

12.8.4.60 GLuint RenderWidget::width

Definition at line 575 of file RenderWidget.h.

12.8.4.61 GLfloat RenderWidget::x

Definition at line 539 of file RenderWidget.h.

12.8.4.62 GLfloat RenderWidget::y

Definition at line 539 of file RenderWidget.h.

12.8.4.63 GLfloat RenderWidget::z

Definition at line 539 of file RenderWidget.h.

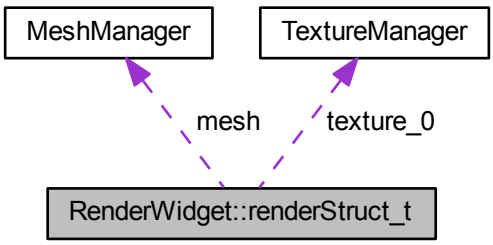
The documentation for this class was generated from the following files:

- RenderWidget.h
- RenderWidget.cpp
- RW_input.cpp
- RW_slots.cpp

12.9 RenderWidget::renderStruct_t Struct Reference

Defines a structure that represents a renderable object.

Collaboration diagram for RenderWidget::renderStruct_t:



Public Attributes

- QGLShaderProgram * baseShader
- glm::vec4 color
- MeshManager * mesh
- glm::mat4 modelMatrix
- glm::vec3 origin
- QGLShaderProgram * projShader
- glm::vec3 rotation

- glm::vec3 scale
- TextureManager * texture_0
- GLuint vaoID
- GLuint vertCount

12.9.1 Detailed Description

Defines a structure that represents a renderable object.

Definition at line 383 of file RenderWidget.h.

12.9.2 Member Data Documentation

12.9.2.1 QGLShaderProgram* RenderWidget::renderStruct_t::baseShader

Definition at line 386 of file RenderWidget.h.

12.9.2.2 glm::vec4 RenderWidget::renderStruct_t::color

Definition at line 392 of file RenderWidget.h.

12.9.2.3 MeshManager* RenderWidget::renderStruct_t::mesh

Definition at line 388 of file RenderWidget.h.

12.9.2.4 glm::mat4 RenderWidget::renderStruct_t::modelMatrix

Definition at line 396 of file RenderWidget.h.

12.9.2.5 glm::vec3 RenderWidget::renderStruct_t::origin

Definition at line 395 of file RenderWidget.h.

12.9.2.6 QGLShaderProgram* RenderWidget::renderStruct_t::projShader

Definition at line 387 of file RenderWidget.h.

12.9.2.7 glm::vec3 RenderWidget::renderStruct_t::rotation

Definition at line 394 of file RenderWidget.h.

12.9.2.8 glm::vec3 RenderWidget::renderStruct_t::scale

Definition at line 393 of file RenderWidget.h.

12.9.2.9 TextureManager* RenderWidget::renderStruct_t::texture_0

Definition at line 389 of file RenderWidget.h.

12.9.2.10 GLuint RenderWidget::renderStruct_t::vaoID

Definition at line 385 of file RenderWidget.h.

12.9.2.11 GLuint RenderWidget::renderStruct_t::vertCount

Definition at line 384 of file RenderWidget.h.

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

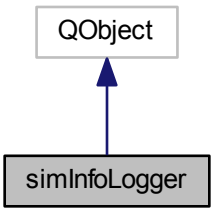
- RenderWidget.h

12.10 simInfoLogger Class Reference

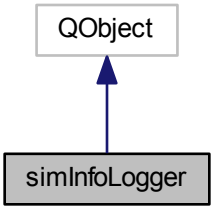
Simulation information logger.

`#include <simInfoLogger.h>`

Inheritance diagram for simInfoLogger:



Collaboration diagram for simInfoLogger:



Public Slots

- void close ()

  Closes output file.
- void timeCheck (simState_t stateInfo)

  Checks to see if enough time has elapsed since last print.

Public Member Functions

- simInfoLogger (QObject *parent=0)
- ~simInfoLogger ()
- void Init ()

  opens the output file and initializes some vars.
- void printDropletData (simState_t stateInfo)

  prints out droplet data to a file
- void setColorFlag (bool flag)

- void setCommSAFlag (bool flag)
- void setMacroRedFlag (bool flag)
- void setMacroSAFlag (bool flag)
- void setPosFlag (bool flag)

  Sets flag.
- void setRotationFlag (bool flag)

Private Attributes

- bool colorFlag
- bool commSAFlag
- FILE * fp

  The output file pointer.
- double lastPrint

  Time of the last print.
- bool macroRedFlag
- bool macroSAFlag

  Flags store what data to print.
- QString newFile

  The output file path.
- bool posFlag
- int redTally
- bool rotationFlag
- int SATally

  Tallies to count macro info.
- double timeInterval

  Time between prints.

12.10.1 Detailed Description

Simulation information logger.

Definition at line 32 of file simInfoLogger.h.

12.10.2 Constructor & Destructor Documentation

12.10.2.1 simInfoLogger::simInfoLogger (QObject * *parent* = 0)

Definition at line 9 of file simInfoLogger.cpp.

12.10.2.2 simInfoLogger::~simInfoLogger ( )

Definition at line 65 of file simInfoLogger.cpp.

12.10.3 Member Function Documentation

12.10.3.1 void simInfoLogger::close ( ) [slot]

Closes output file.

Definition at line 70 of file simInfoLogger.cpp.

12.10.3.2 void simInfoLogger::Init ( )

opens the output file and initializes some vars.

Definition at line 22 of file simInfoLogger.cpp.

Here is the caller graph for this function:

12.10.3.3 void simInfoLogger::printDropletData (simState_t *stateInfo*)

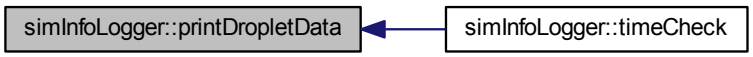
prints out droplet data to a file

Parameters

stateInfo	Information describing the state.

Definition at line 106 of file simInfoLogger.cpp.

Here is the caller graph for this function:



12.10.3.4 void simInfoLogger::setColorFlag (bool *flag*)

Definition at line 85 of file simInfoLogger.cpp.

Here is the caller graph for this function:

12.10.3.5 void simInfoLogger::setCommSAFlag (bool *flag*)

Definition at line 93 of file simInfoLogger.cpp.

Here is the caller graph for this function:

12.10.3.6 void simInfoLogger::setMacroRedFlag (bool *flag*)

Definition at line 97 of file simInfoLogger.cpp.

Here is the caller graph for this function:

12.10.3.7 void simInfoLogger::setMacroSAFlag (bool *flag*)

Definition at line 101 of file simInfoLogger.cpp.

Here is the caller graph for this function:

12.10.3.8 void simInfoLogger::setPosFlag (bool *flag*)

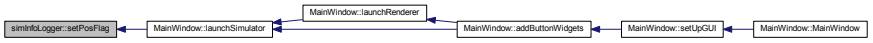
Sets flag.

Parameters

flag	boolean

Definition at line 81 of file simInfoLogger.cpp.

Here is the caller graph for this function:



12.10.3.9 void simInfoLogger::setRotationFlag (bool *flag*)

Definition at line 89 of file simInfoLogger.cpp.

Here is the caller graph for this function:

12.10.3.10 void simInfoLogger::timeCheck (simState_t *stateInfo*) [slot]

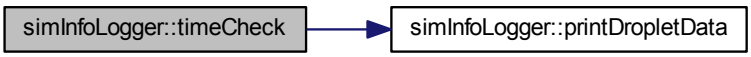
Checks to see if enough time has elapsed since last print.

Parameters

stateInfo	Information describing the state.

Definition at line 168 of file simInfoLogger.cpp.

Here is the call graph for this function:



12.10.4 Member Data Documentation

12.10.4.1 bool simInfoLogger::colorFlag [private]

Definition at line 84 of file simInfoLogger.h.

12.10.4.2 bool simInfoLogger::commSAFlag [private]

Definition at line 84 of file simInfoLogger.h.

12.10.4.3 FILE * simInfoLogger::fp [private]

The output file pointer.

Definition at line 123 of file simInfoLogger.h.

12.10.4.4 int simInfoLogger::lastPrint [private]

Time of the last print.

Definition at line 101 of file simInfoLogger.h.

12.10.4.5 bool simInfoLogger::macroRedFlag [private]

Definition at line 84 of file simInfoLogger.h.

12.10.4.6 bool posFlag colorFlag rotationFlag commSAFlag macroRedFlag simInfoLogger::macroSAFlag [private]

Flags store what data to print.

Definition at line 84 of file simInfoLogger.h.

12.10.4.7 QString simInfoLogger::newFile [private]

The output file path.

Definition at line 116 of file simInfoLogger.h.

12.10.4.8 bool simInfoLogger::posFlag [private]

Definition at line 84 of file simInfoLogger.h.

12.10.4.9 int simInfoLogger::redTally [private]

Definition at line 93 of file simInfoLogger.h.

12.10.4.10 bool simInfoLogger::rotationFlag [private]

Definition at line 84 of file simInfoLogger.h.

12.10.4.11 int redTally simInfoLogger::SATally [private]

Tallies to count macro info.

Definition at line 93 of file simInfoLogger.h.

12.10.4.12 int simInfoLogger::timeInterval [private]

Time between prints.

Definition at line 108 of file simInfoLogger.h.

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

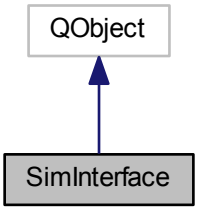
- simInfoLogger.h
- simInfoLogger.cpp

12.11 SimInterface Class Reference

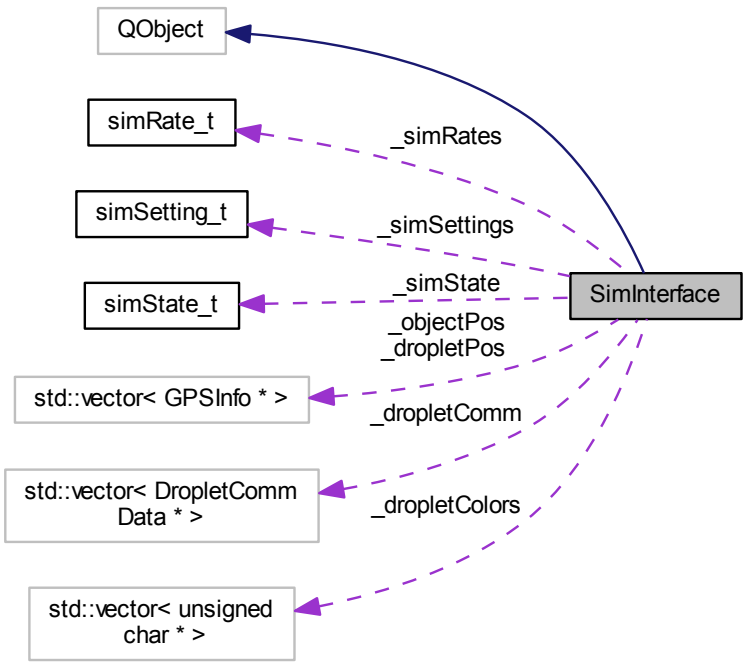
Model that contains and controls the droplet simulator.

`#include <SimInterface.h>`

Inheritance diagram for SimInterface:



Collaboration diagram for SimInterface:



Public Slots

- void addCube (float x, float y, int objectID, vec3 scale, float mass, float friction)

  Adds a cube.
- void addDroplet (float x, float y, droplet_t dType=March, int dropletID=0)

  Adds a droplet.
- void addFloor (float x, float y, int objectID, vec3 scale)

  Adds a floor. This is the same as addCube but the object created is static.
- int addNewShape (object_t objectType, float radius, vec3 scale)

  Adds collision objects.
- void addObject (object_t kind, float x, float y, float radius=1.0f, float mass=DEFAULT_OBJECT_MASS, float friction=DEFAULT_OBJECT_FRICTION)

  Adds an object. Calls either addSphere or addCube.
- void addSphere (float x, float y, int objectID, float radius, float mass, float friction)

  Adds a sphere object.
- void addWall (float x, float y, int objectID, vec3 scale)

  Adds a wall. This is the same as addCube but the object created is static.
- void decreaseUpdateRate (void)

  Decrease update rate. Emits the SimInterface::ratesChanged() signal.
- void disableUpdateLimit (void)

  Disables the update limit. Emits the SimInterface::ratesChanged() signal.
- void enableUpdateLimit (void)

  Enables the update limit. Emits the SimInterface::ratesChanged() signal.
- void increaseUpdateRate (void)

Increase update rate. Emits the SimInterface::ratesChanged() signal.

- void loadArena (simSetting_t arena)

  Updates sim settings to settings stored in simSetting_t arena. Then calls SimInterface::Init()
- void loadTilePositions ()

  Creates the default rectangle arena based on the numRowTiles and numColTiles parameters.
- void loadTilePositions (QString filename)
- void pause (void)

  emits the SimInterface::pauseChanged() signal. Pauses the simulation
- void reset (void)

  Resets this object. Calls SimInterface::Init()
- void resume (void)

  emits the SimInterface::pauseChanged() signal. Resumes the simulation
- void setUpdateRate (float rate)

  Sets the update rate. Emits the SimInterface::ratesChanged() signal.
- void togglePause (void)

  Calls SimInterface::resume() or SimInterface::pause() depending on the current status of the simulation.
- void toggleUpdateLimit (void)

  Toggle update limit. Emits the SimInterface::ratesChanged() signal.
- void Update (float timeSinceLastUpdate)

  Steps the DropletSim object, and updates the DropletSimInfo object. Uses the DropletSimInfo object to update droplet and object positions.

Signals

- void arenaChanged (simSetting_t arena)

  signal emitted in SimInterface::Init(). This signal is connected to the slot RenderWidget::updateArena().
- void pauseChanged (bool paused)

  Signal emitted in SimInterface::pause() and SimInterface::resume(). Signal is connected to the slot RenderWidget::updatePause().
- void ratesChanged (simRate_t rates)

  Signal emitted in SimInterface::setUpdateRate(), SimInterface::increaseUpdateRate(), SimInterface::decreaseUpdateRate(), SimInterface::enableUpdateLimit(), SimInterface::disableUpdateLimit(), and SimInterface::toggleUpdateLimit(). Signal is connected to the slot RenderWidget::updateRate().
- void simulationUpdated (simState_t simInfo)

  Signal emitted in SimInterface::Init(). This signal is connected to the slot RenderWidget::updateState().

Public Member Functions

- SimInterface (QObject *parent=0)
- ~SimInterface ()
- void createArena ()

  Creates the arena.
- simSetting_t getCurrentSettings ()

  Gets current settings.
- simSetting_t getDefaultSettings ()

  Returns a simSetting_t that is set to default settings.
- float getRandomf (float min, float max)

  Generates a random float between min and max.
- simRate_t getSimulatorRate ()

  Gets the simulator rate.
- simSetting_t getSimulatorSettings ()

Returns the current simulator settings.

- simState_t getSimulatorState ()

  Gets the simulator state.

- void Init ()

  Initialises this SimInterface object. Creates vectors that track droplet information. Calls functions that create the arena and adds droplets.

- bool isPaused (void)

  Query if this object is paused.

Protected Member Functions

- void timerEvent (QTimerEvent *event)

  Manages when to call SimInterface::Update().

Private Member Functions

- btCollisionShape * makeCollisionShapeFromFile (QString file)

  Makes collision shape from file.

- void makeDropletCollisionShapeFromFile (QString file)

  Makes droplet collision shape from file.

- IDroplet * newDropletOfType (droplet_t dType, ObjectPhysicsData *dropletPhyDat)

  Creates a new droplet of type.

- void teardownSim ()

  Teardown the simulator.

- void updateTiming ()

  Updates the timing.

Private Attributes

- std::vector< unsigned char * > * _dropletColors
- std::vector< DropletCommData * > * _dropletComm
- std::vector< GPSInfo * > * _dropletPos

  The droplet position.

- struct {
  QStringList multiple
  QStringList single
  } _objectNames

- std::vector< GPSInfo * > * _objectPos

  The object position.

- double _realTime
- DropletSim * _sim
- DropletSimInfo _simInfo
- simRate_t _simRates
- simSetting_t _simSettings

  The simulation settings.

- simState_t _simState

  State of the simulation.

- struct {
  int btDropletShapeID
  int btFloorShapeID
  int btXWallShapeID
  int btYWallShapeID
  float dropletOffset
  btCollisionShape * dropletShape
  bool paused
  double runTime
  } _simStatus

- double _simTimeScale

  The simulation time scale.
- double _targetUpdateTime

  Time of the target update.
- QVector< vec2 > _tilePositions

  The tile positions and corresponding wall booleans for each tile.
- DropletTimeControl * _timer
- int _timerID

  Identifier for the timer.
- double _timeUntilNextUpdate

  The time until next update.
- QElapsedTimer _updateTimer
- QVector< vec4 > _wallBools

12.11.1 Detailed Description

Model that contains and controls the droplet simulator.

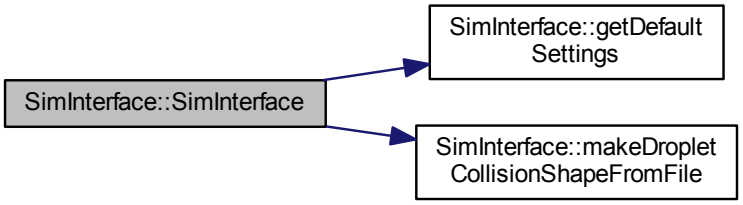
Definition at line 78 of file SimInterface.h.

12.11.2 Constructor & Destructor Documentation

12.11.2.1 SimInterface::SimInterface (QObject * *parent* = 0)

Definition at line 10 of file SimInterface.cpp.

Here is the call graph for this function:



12.11.2.2 SimInterface::~SimInterface ( )

Definition at line 32 of file SimInterface.cpp.

Here is the call graph for this function:

12.11.3 Member Function Documentation

12.11.3.1 void SimInterface::addCube (float *x*, float *y*, int *objectID*, vec3 *scale*, float *mass*, float *friction*) [slot]

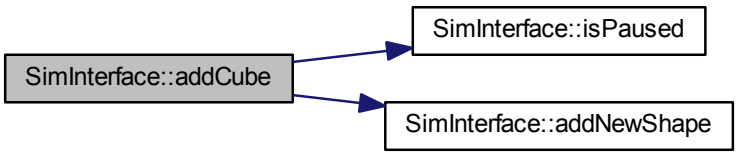
Adds a cube.

Parameters

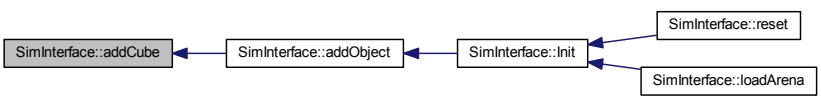
x	The x coordinate.
y	The y coordinate.
objectID	Identifier for the object.
scale	The scale.
mass	The mass.
friction	The friction.

Definition at line 887 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.2 void SimInterface::addDroplet (float *x*, float *y*, droplet_t *dType* = March, int *dropletID* = 0) [slot]

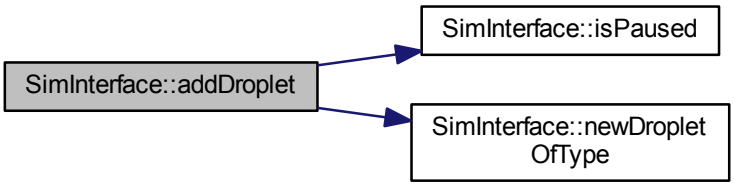
Adds a droplet.

Parameters

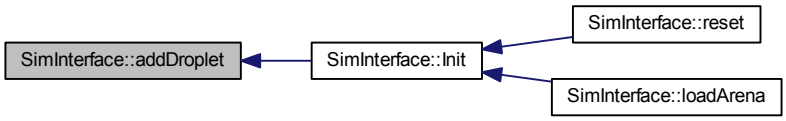
x	The x coordinate.
y	The y coordinate.
dType	(Optional) The droplet program type.
dropletID	(Optional) The Droplet ID. If 0, adds one to the current number of droplets.

Definition at line 692 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.3 void SimInterface::addFloor (float *x*, float *y*, int *objectID*, vec3 *scale*) [slot]

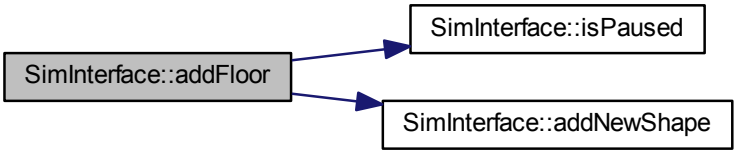
Adds a floor. This is the same as addCube but the object created is static.

Parameters

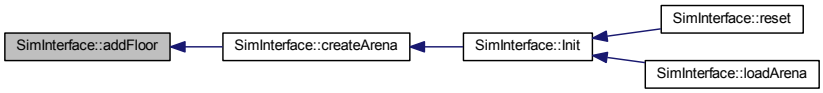
x	The x coordinate.
y	The y coordinate.
objectID	Identifier for the object.
scale	The scale.

Definition at line 995 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.4 int SimInterface::addNewShape (object_t *objectType*, float *radius*, vec3 *scale*) [slot]

Adds collision objects.

Parameters

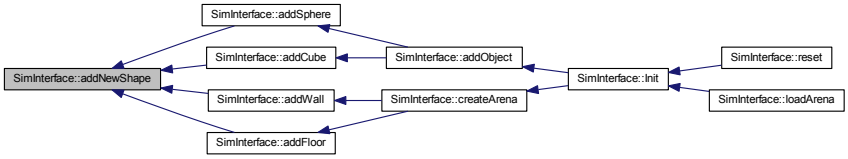
objectType	Type of the object.
radius	The radius.
scale	The scale.

Returns

The index of the corresponding shape in the list of collision objects.

Definition at line 755 of file SimInterface.cpp.

Here is the caller graph for this function:



12.11.3.5 void SimInterface::addObject (object_t *kind*, float *x*, float *y*, float *radius* = 1.0f, float *mass* = DEFAULT_OBJECT_MASS, float *friction* = DEFAULT_OBJECT_FRICTION) [slot]

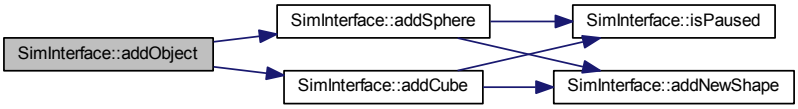
Adds an object. Calls either addSphere or addCube.

Parameters

kind	The kind (Sphere or Cube)
x	The x coordinate.
y	The y coordinate.
radius	(Optional) the radius.
mass	(Optional) the mass.
friction	(Optional) the friction.

Definition at line 1049 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

12.11.3.6 void SimInterface::addSphere (float *x*, float *y*, int *objectID*, float *radius*, float *mass*, float *friction*) [slot]

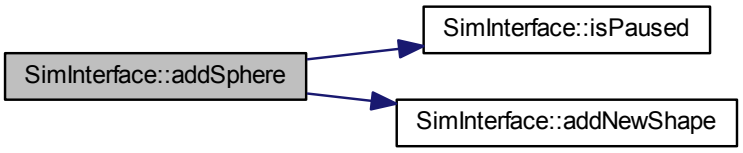
Adds a sphere object.

Parameters

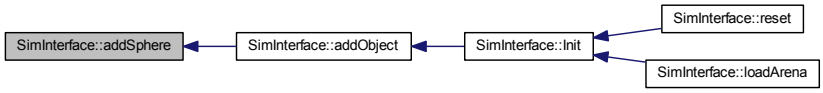
x	The x coordinate.
y	The y coordinate.
objectID	Identifier for the object.
radius	The radius.
mass	The mass.
friction	The friction.

Definition at line 831 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.7 void SimInterface::addWall (float *x*, float *y*, int *objectID*, vec3 *scale*) [slot]

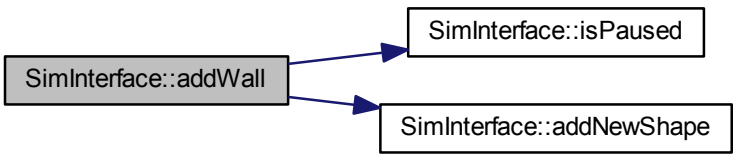
Adds a wall. This is the same as addCube but the object created is static.

Parameters

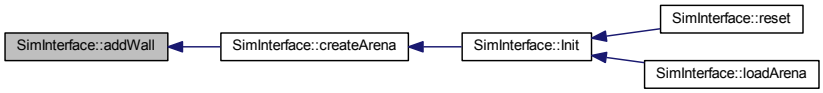
x	The x coordinate.
y	The y coordinate.
objectID	Identifier for the object.
scale	The scale.

Definition at line 942 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.8 void SimInterface::arenaChanged (simSetting_t *arena*) [signal]

signal emitted in SimInterface::Init(). This signal is connected to the slot RenderWidget::updateArena().

Parameters

arena	simulation settings.

Here is the caller graph for this function:

12.11.3.9 void SimInterface::createArena ( )

Creates the arena.

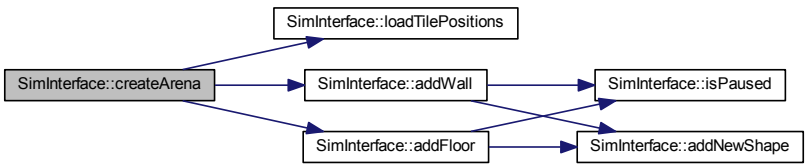
Creates an arena based on an input file. The input file contains lines of the following format: xPos yPos up right down left xPos and yPos are grid coordinates. up right down left are replaced with yes or no, booleans representing if a wall is created for that tile at each position.

Parameters

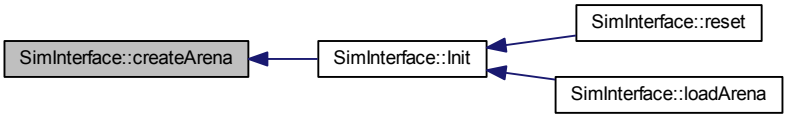
filename	Filename of the floor file.

Definition at line 639 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

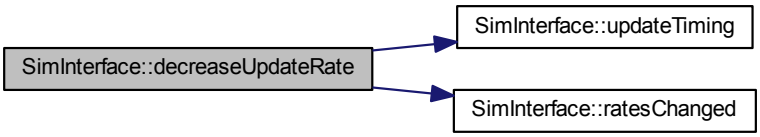


12.11.3.10 void SimInterface::decreaseUpdateRate (void) [slot]

Decrease update rate. Emits the SimInterface::ratesChanged() signal.

Definition at line 1611 of file SimInterface.cpp.

Here is the call graph for this function:

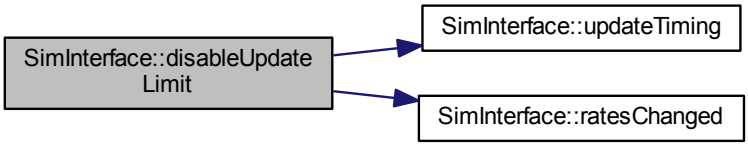


12.11.3.11 void SimInterface::disableUpdateLimit (void) [slot]

Disables the update limit. Emits the SimInterface::ratesChanged() signal.

Definition at line 1630 of file SimInterface.cpp.

Here is the call graph for this function:

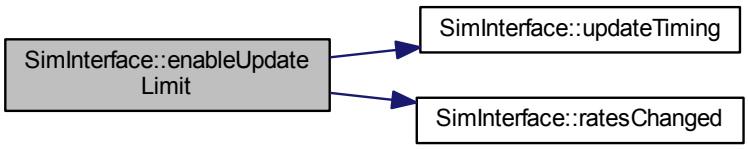


12.11.3.12 void SimInterface::enableUpdateLimit (void) [slot]

Enables the update limit. Emits the SimInterface::ratesChanged() signal.

Definition at line 1620 of file SimInterface.cpp.

Here is the call graph for this function:



12.11.3.13 simSetting_t SimInterface::getCurrentSettings ( )

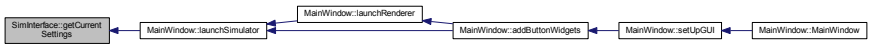
Gets current settings.

Returns

The current settings.

Definition at line 1440 of file SimInterface.cpp.

Here is the caller graph for this function:



12.11.3.14 simSetting_t SimInterface::getDefaultSettings ( )

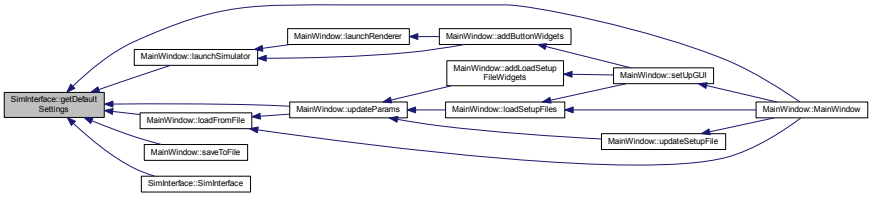
Returns a simSetting_t that is set to default settings.

Returns

The default settings.

Definition at line 1422 of file SimInterface.cpp.

Here is the caller graph for this function:



12.11.3.15 float SimInterface::getRandomf (float *min*, float *max*)

Generates a random float between min and max.

Parameters

min	The minimum.
max	The maximum.

Returns

The random float.

Definition at line 1062 of file SimInterface.cpp.

Here is the caller graph for this function:

12.11.3.16 simRate_t SimInterface::getSimulatorRate ( )

Gets the simulator rate.

Returns

_simRates, The simulator rate.

Definition at line 1647 of file SimInterface.cpp.

Here is the caller graph for this function:

12.11.3.17 simSetting_t SimInterface::getSimulatorSettings ( )

Returns the current simulator settings.

Returns

_simSettings, The current simulator settings.

Definition at line 1366 of file SimInterface.cpp.

Here is the caller graph for this function:

12.11.3.18 simState_t SimInterface::getSimulatorState ( )

Gets the simulator state.

Returns

_simState, The simulator state.

Definition at line 1371 of file SimInterface.cpp.

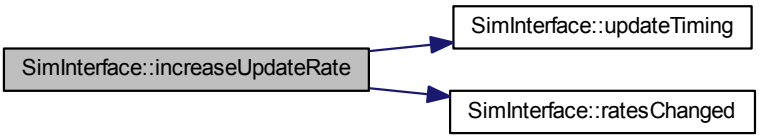
Here is the caller graph for this function:

12.11.3.19 void SimInterface::increaseUpdateRate (void) [slot]

Increase update rate. Emits the SimInterface::ratesChanged() signal.

Definition at line 1601 of file SimInterface.cpp.

Here is the call graph for this function:

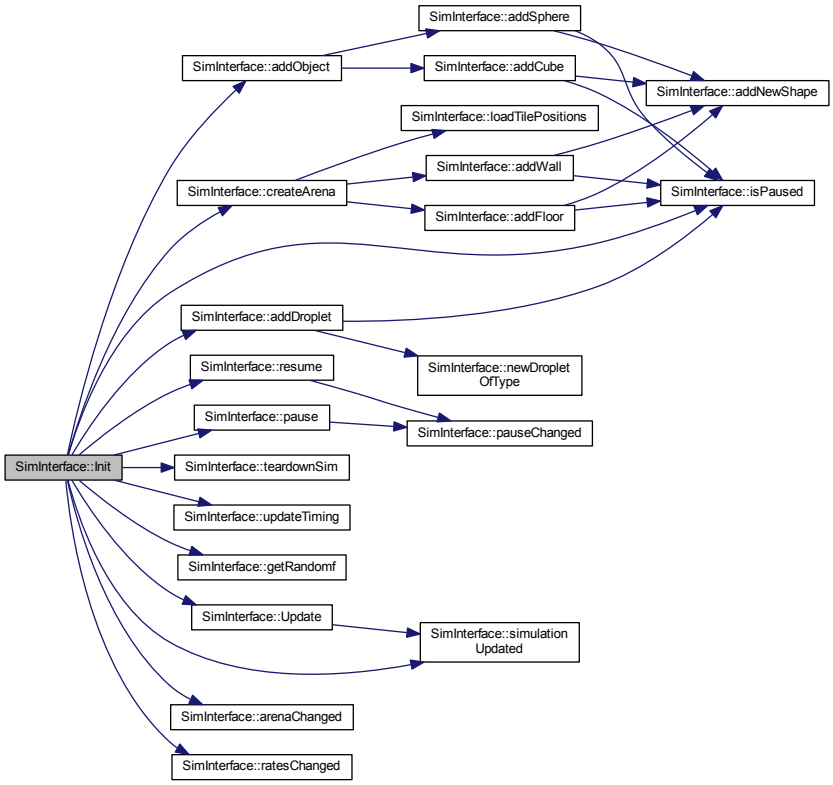


12.11.3.20 void SimInterface::Init ( )

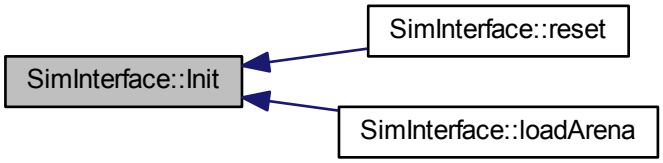
Initialises this SimInterface object. Creates vectors that track droplet information. Calls functions that create the arena and adds droplets.

Definition at line 111 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.21 bool SimInterface::isPaused (void)

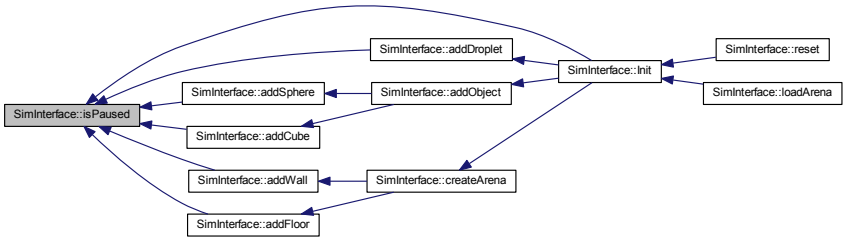
Query if this object is paused.

Returns

true if paused, false if not.

Definition at line 1278 of file SimInterface.cpp.

Here is the caller graph for this function:



12.11.3.22 void SimInterface::loadArena (simSetting_t *arena*) [slot]

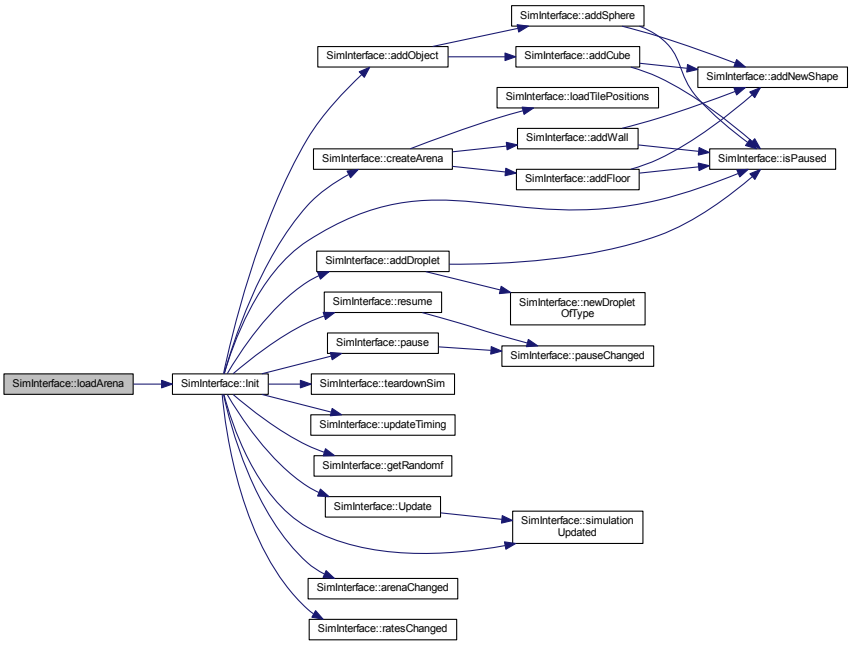
Updates sim settings to settings stored in simSetting_t arena. Then calls SimInterface::Init()

Parameters

arena	The simulation settings.

Definition at line 1416 of file SimInterface.cpp.

Here is the call graph for this function:



12.11.3.23 void SimInterface::loadTilePositions ( ) [slot]

Creates the default rectangle arena based on the numRowTiles and numColTiles parameters.

Definition at line 488 of file SimInterface.cpp.

Here is the caller graph for this function:

12.11.3.24 void SimInterface::loadTilePositions (QString *filename*) [slot]

Definition at line 538 of file SimInterface.cpp.

12.11.3.25 btCollisionShape * SimInterface::makeCollisionShapeFromFile (QString *file*) [private]

Makes collision shape from file.

Parameters

file	The file.

Returns

null if it fails, else.

Definition at line 1533 of file SimInterface.cpp.

12.11.3.26 void SimInterface::makeDropletCollisionShapeFromFile (QString *file*) [private]

Makes droplet collision shape from file.

Parameters

file	The file.

Definition at line 1445 of file SimInterface.cpp.

Here is the caller graph for this function:

12.11.3.27 IDroplet * SimInterface::newDropletOfType (droplet_t *dType*, ObjectPhysicsData * *dropletPhyDat*) [private]

Creates a new droplet of type.

Parameters

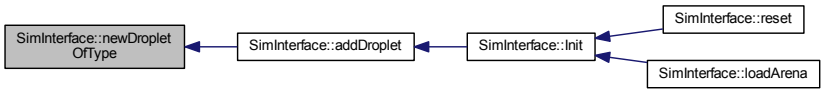
	dType	The type.
in,out	*dropletPhyDat*	If non-null, the droplet phy dat.

Returns

null if it fails, else.

Definition at line 1284 of file SimInterface.cpp.

Here is the caller graph for this function:

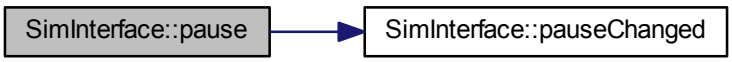


12.11.3.28 void SimInterface::pause (void) [slot]

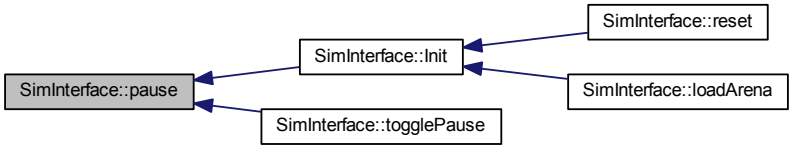
emits the SimInterface::pauseChanged() signal. Pauses the simulation

Definition at line 1249 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



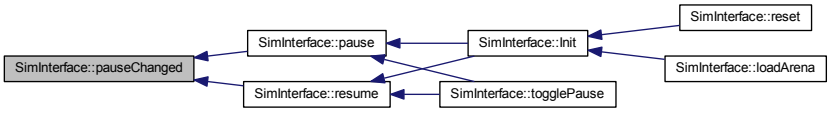
12.11.3.29 void SimInterface::pauseChanged (bool *paused*) [signal]

Signal emitted in SimInterface::pause() and SimInterface::resume(). Signal is connected to the slot RenderWidget-::updatePause().

Parameters

paused	true if paused.

Here is the caller graph for this function:



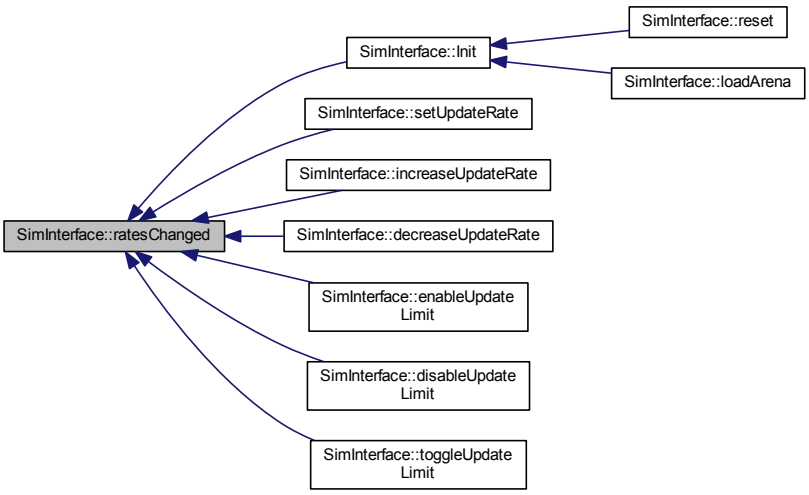
12.11.3.30 void SimInterface::ratesChanged (simRate_t *rates*) [signal]

Signal emitted in SimInterface::setUpdateRate(), SimInterface::increaseUpdateRate(), SimInterface::decreaseUpdateRate(), SimInterface::enableUpdateLimit(), SimInterface::disableUpdateLimit(), and SimInterface::toggleUpdateLimit(). Signal is connected to the slot RenderWidget::updateRate().

Parameters

rates	simulation rate.

Here is the caller graph for this function:

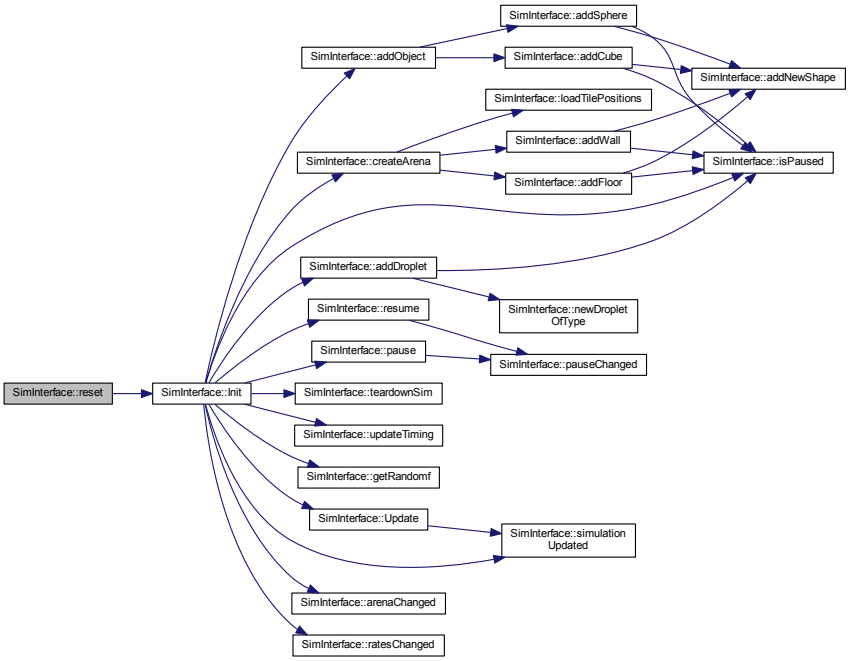


12.11.3.31 void SimInterface::reset (void) [slot]

Resets this object. Calls SimInterface::Init()

Definition at line 1360 of file SimInterface.cpp.

Here is the call graph for this function:

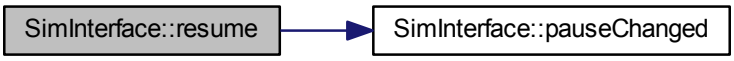


12.11.3.32 void SimInterface::resume (void) [slot]

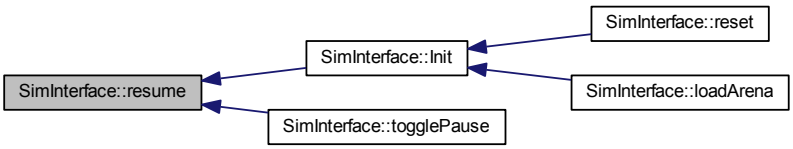
emits the SimInterface::pauseChanged() signal. Resumes the simulation

Definition at line 1259 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.11.3.33 void SimInterface::setUpdateRate (float *rate*) [slot]

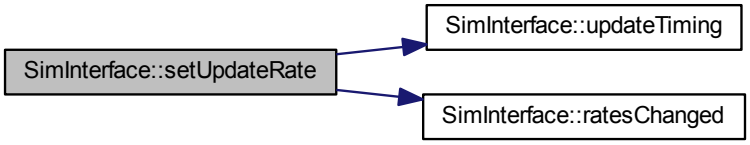
Sets the update rate. Emits the SimInterface::ratesChanged() signal.

Parameters

rate	The update rate.

Definition at line 1591 of file SimInterface.cpp.

Here is the call graph for this function:



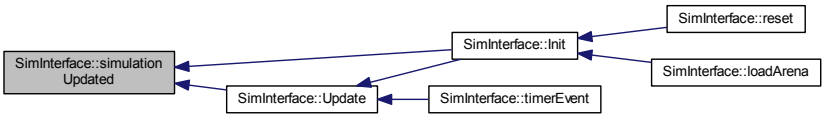
12.11.3.34 void SimInterface::simulationUpdated (simState_t *simInfo*) [signal]

Signal emitted in SimInterface::Init(). This signal is connected to the slot RenderWidget::updateState().

Parameters

simInfo	simulation state.

Here is the caller graph for this function:

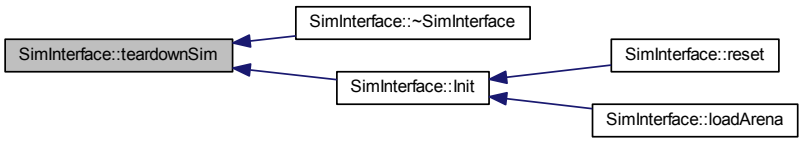


12.11.3.35 void SimInterface::teardownSim ( ) [private]

Teardown the simulator.

Definition at line 46 of file SimInterface.cpp.

Here is the caller graph for this function:



12.11.3.36 void SimInterface::timerEvent (QTimerEvent * *event*) [protected]

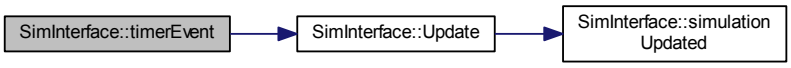
Manages when to call SimInterface::Update().

Parameters

in,out	*event*	If non-null, the event.

Definition at line 1395 of file SimInterface.cpp.

Here is the call graph for this function:

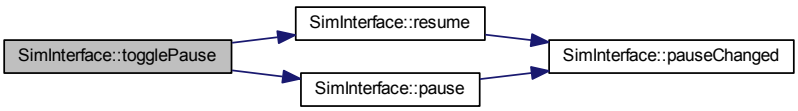


12.11.3.37 void SimInterface::togglePause (void) [slot]

Calls SimInterface::resume() or SimInterface::pause() depending on the current status of the simulation.

Definition at line 1267 of file SimInterface.cpp.

Here is the call graph for this function:

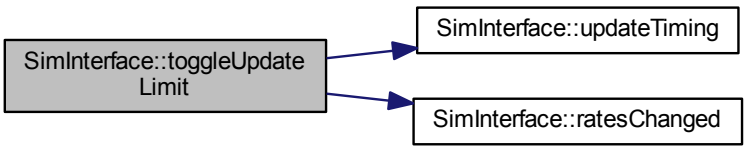


12.11.3.38 void SimInterface::toggleUpdateLimit (void) [slot]

Toggle update limit. Emits the SimInterface::ratesChanged() signal.

Definition at line 1640 of file SimInterface.cpp.

Here is the call graph for this function:



12.11.3.39 void SimInterface::Update (float *timeSinceLastUpdate*) [slot]

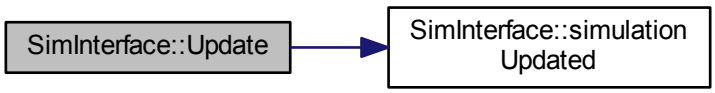
Steps the DropletSim object, and updates the DropletSimInfo object. Uses the DropletSimInfo object to update droplet and object positions.

Parameters

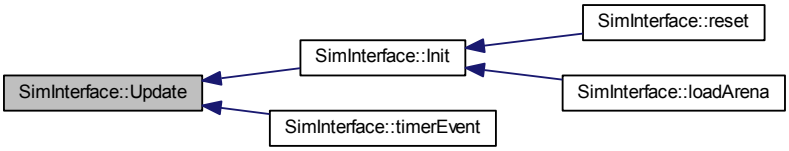
timeSinceLast-Update	The time since last update.

Definition at line 1070 of file SimInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

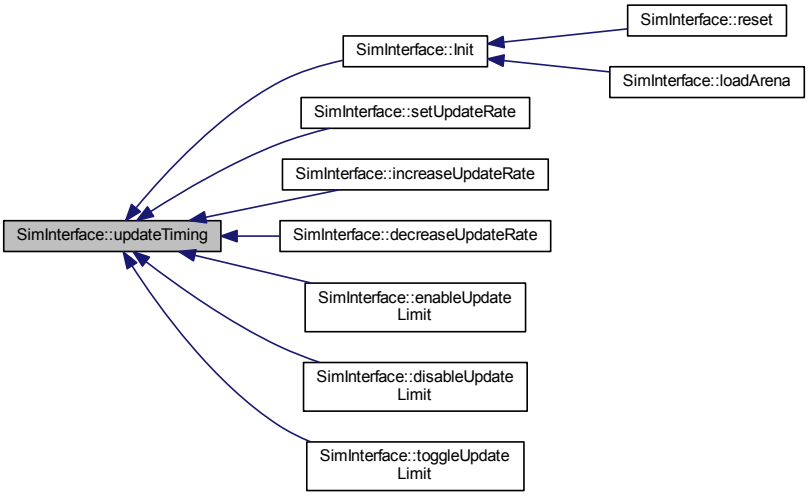


12.11.3.40 void SimInterface::updateTiming ( ) [private]

Updates the timing.

Definition at line 1376 of file SimInterface.cpp.

Here is the caller graph for this function:



12.11.4 Member Data Documentation

12.11.4.1 std::vector<unsigned char *>* SimInterface::_dropletColors [private]

Definition at line 571 of file SimInterface.h.

12.11.4.2 std::vector<DropletCommData *>* SimInterface::_dropletComm [private]

Definition at line 578 of file SimInterface.h.

12.11.4.3 std::vector<GPSInfo *>* SimInterface::_dropletPos [private]

The droplet position.

Definition at line 577 of file SimInterface.h.

12.11.4.4 struct { ... } SimInterface::_objectNames [private]

12.11.4.5 std::vector<GPSInfo *>* SimInterface::_objectPos [private]

The object position.

Definition at line 584 of file SimInterface.h.

12.11.4.6 double SimInterface::_realTime [private]

Definition at line 604 of file SimInterface.h.

12.11.4.7 DropletSim* SimInterface::_sim [private]

Definition at line 527 of file SimInterface.h.

12.11.4.8 DropletSimInfo SimInterface::_simInfo [private]

Definition at line 528 of file SimInterface.h.

12.11.4.9 simRate_t SimInterface::_simRates [private]

Definition at line 535 of file SimInterface.h.

12.11.4.10 simSetting_t SimInterface::_simSettings [private]

The simulation settings.

Definition at line 534 of file SimInterface.h.

12.11.4.11 simState_t SimInterface::_simState [private]

State of the simulation.

Definition at line 561 of file SimInterface.h.

12.11.4.12 struct { ... } SimInterface::_simStatus [private]

12.11.4.13 double SimInterface::_simTimeScale [private]

The simulation time scale.

Definition at line 610 of file SimInterface.h.

12.11.4.14 double SimInterface::_targetUpdateTime [private]

Time of the target update.

Definition at line 616 of file SimInterface.h.

12.11.4.15 QVector<vec2> SimInterface::_tilePositions [private]

The tile positions and corresponding wall booleans for each tile.

Definition at line 591 of file SimInterface.h.

12.11.4.16 DropletTimeControl* SimInterface::_timer [private]

Definition at line 585 of file SimInterface.h.

12.11.4.17 int SimInterface::_timerID [private]

Identifier for the timer.

Definition at line 602 of file SimInterface.h.

12.11.4.18 double SimInterface::_timeUntilNextUpdate [private]

The time until next update.

Definition at line 622 of file SimInterface.h.

12.11.4.19 QElapsedTimer SimInterface::_updateTimer [private]

Definition at line 596 of file SimInterface.h.

12.11.4.20 QVector<vec4> SimInterface::_wallBools [private]

Definition at line 592 of file SimInterface.h.

12.11.4.21 int SimInterface::btDropletShapeID

Definition at line 552 of file SimInterface.h.

12.11.4.22 int SimInterface::btFloorShapeID

Definition at line 551 of file SimInterface.h.

12.11.4.23 int SimInterface::btXWallShapeID

Definition at line 549 of file SimInterface.h.

12.11.4.24 int SimInterface::btYWallShapeID

Definition at line 550 of file SimInterface.h.

12.11.4.25 float SimInterface::dropletOffset

Definition at line 548 of file SimInterface.h.

12.11.4.26 btCollisionShape* SimInterface::dropletShape

Definition at line 547 of file SimInterface.h.

12.11.4.27 QStringList SimInterface::multiple

Definition at line 626 of file SimInterface.h.

12.11.4.28 bool SimInterface::paused

Definition at line 541 of file SimInterface.h.

12.11.4.29 double SimInterface::runTime

Definition at line 546 of file SimInterface.h.

12.11.4.30 QStringList SimInterface::single

Definition at line 625 of file SimInterface.h.

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

- SimInterface.h
- SimInterface.cpp

12.12 simRate_t Struct Reference

Defines the current state of rate limiting emitted by the SimInterface.

`#include <structs.h>`

Public Attributes

- bool limitRate
- float timeScale

12.12.1 Detailed Description

Defines the current state of rate limiting emitted by the SimInterface.

Definition at line 304 of file structs.h.

12.12.2 Member Data Documentation

12.12.2.1 bool simRate_t::limitRate

Definition at line 306 of file structs.h.

12.12.2.2 float simRate_t::timeScale

Definition at line 305 of file structs.h.

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

- structs.h

12.13 simSetting_t Struct Reference

Defines simulator settings that are passed in and out of SimInterface.

`#include <structs.h>`

Public Attributes

- QString arenaName
- float dropletOffset
- float dropletRadius
- QString floorFile
- float fps
- int numColTiles
- int numRowTiles
- bool projecting
- QString projTexture
- QVector< QStringList > startingDroplets
- QVector< QStringList > startingObjects
- float tileLength
- float wallHeight
- float wallWidth

12.13.1 Detailed Description

Defines simulator settings that are passed in and out of SimInterface.

Definition at line 245 of file structs.h.

12.13.2 Member Data Documentation

12.13.2.1 QString simSetting_t::arenaName

Definition at line 246 of file structs.h.

12.13.2.2 float simSetting_t::dropletOffset

Definition at line 254 of file structs.h.

12.13.2.3 float simSetting_t::dropletRadius

Definition at line 250 of file structs.h.

12.13.2.4 QString simSetting_t::floorFile

Definition at line 257 of file structs.h.

12.13.2.5 float simSetting_t::fps

Definition at line 253 of file structs.h.

12.13.2.6 int simSetting_t::numColTiles

Definition at line 248 of file structs.h.

12.13.2.7 int simSetting_t::numRowTiles

Definition at line 247 of file structs.h.

12.13.2.8 bool simSetting_t::projecting

Definition at line 255 of file structs.h.

12.13.2.9 QString simSetting_t::projTexture

Definition at line 256 of file structs.h.

12.13.2.10 QVector<QStringList> simSetting_t::startingDroplets

Definition at line 258 of file structs.h.

12.13.2.11 QVector<QStringList> simSetting_t::startingObjects

Definition at line 259 of file structs.h.

12.13.2.12 float simSetting_t::tileLength

Definition at line 249 of file structs.h.

12.13.2.13 float simSetting_t::wallHeight

Definition at line 251 of file structs.h.

12.13.2.14 float simSetting_t::wallWidth

Definition at line 252 of file structs.h.

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

- structs.h

12.14 simState_t Struct Reference

Defines current state of the simulator that is emitted by SimInterface.

`#include <structs.h>`

Public Attributes

- QList< collisionShapeStruct_t > collisionShapes
- QVector< dropletStruct_t > dropletData
- float dropletOffset
- QVector< objectStruct_t > dynamicObjectData

- QImage projTexture
- bool projTextureChanged
- double realTime
- double simTime
- QVector< objectStruct_t > staticObjectData
- double timeRatio

12.14.1 Detailed Description

Defines current state of the simulator that is emitted by SimInterface.

Definition at line 282 of file structs.h.

12.14.2 Member Data Documentation

12.14.2.1 QList<collisionShapeStruct_t> simState_t::collisionShapes

Definition at line 294 of file structs.h.

12.14.2.2 QVector<dropletStruct_t> simState_t::dropletData

Definition at line 291 of file structs.h.

12.14.2.3 float simState_t::dropletOffset

Definition at line 290 of file structs.h.

12.14.2.4 QVector<objectStruct_t> simState_t::dynamicObjectData

Definition at line 292 of file structs.h.

12.14.2.5 QImage simState_t::projTexture

Definition at line 288 of file structs.h.

12.14.2.6 bool simState_t::projTextureChanged

Definition at line 289 of file structs.h.

12.14.2.7 double simState_t::realTime

Definition at line 286 of file structs.h.

12.14.2.8 double simState_t::simTime

Definition at line 285 of file structs.h.

12.14.2.9 QVector<objectStruct_t> simState_t::staticObjectData

Definition at line 293 of file structs.h.

12.14.2.10 double simState_t::timeRatio

Definition at line 287 of file structs.h.

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

- structs.h

12.15 TextureManager Class Reference

Container class for a texture.

`#include <texturemanager.h>`

Public Member Functions

- TextureManager ()
  Default constructor.
- ~TextureManager ()
- void bindTexture ()
  Bind texture.
- void freeTexture ()
- bool loadFile (QString fileName)
- bool setFile (QString fileName)
  Sets a file.
- void unbindTexture ()
  Unbind texture.

Private Attributes

- QString _fileName
- bool _fileSet
- bool _hasLoaded
- struct {
  GLuint handle
  GLuint height
  bool valid
  GLuint width
  } _textureInfo

12.15.1 Detailed Description

Container class for a texture.

Definition at line 30 of file texturemanager.h.

12.15.2 Constructor & Destructor Documentation

12.15.2.1 TextureManager::TextureManager ( )

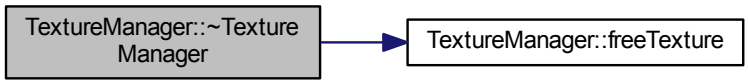
Default constructor.

Definition at line 9 of file texturemanager.cpp.

12.15.2.2 TextureManager::~TextureManager ( )

Definition at line 20 of file texturemanager.cpp.

Here is the call graph for this function:



12.15.3 Member Function Documentation

12.15.3.1 void TextureManager::bindTexture ( )

Bind texture.

Definition at line 73 of file texturemanager.cpp.

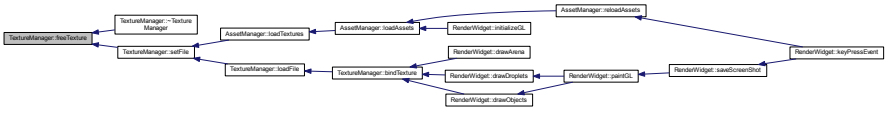
Here is the call graph for this function:

Here is the caller graph for this function:

12.15.3.2 void TextureManager::freeTexture ( )

Definition at line 91 of file texturemanager.cpp.

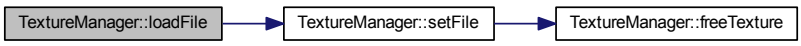
Here is the caller graph for this function:



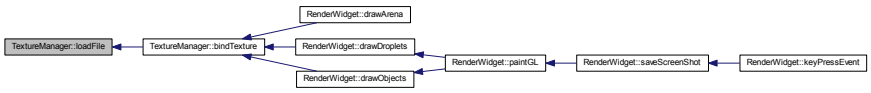
12.15.3.3 bool TextureManager::loadFile (QString *fileName*)

Definition at line 40 of file texturemanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.15.3.4 bool TextureManager::setFile (QString *fileName*)

Sets a file.

Parameters

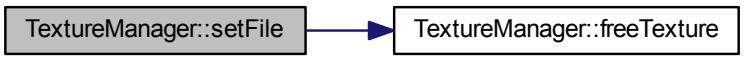
fileName	Filename of the file.

Returns

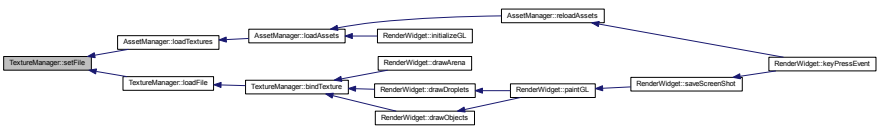
true if it succeeds, false if it fails.

Definition at line 25 of file texturemanager.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



12.15.3.5 void TextureManager::unbindTexture ( )

Unbind texture.

Definition at line 86 of file texturemanager.cpp.

Here is the caller graph for this function:

12.15.4 Member Data Documentation

12.15.4.1 QString TextureManager::_fileName [private]

Definition at line 82 of file texturemanager.h.

12.15.4.2 bool TextureManager::_fileSet [private]

Definition at line 81 of file texturemanager.h.

12.15.4.3 bool TextureManager::_hasLoaded [private]

Definition at line 80 of file texturemanager.h.

12.15.4.4 struct { ... } TextureManager::_textureInfo [private]

12.15.4.5 GLuint TextureManager::handle

Definition at line 77 of file texturemanager.h.

12.15.4.6 GLuint TextureManager::height

Definition at line 76 of file texturemanager.h.

12.15.4.7 bool TextureManager::valid

Definition at line 78 of file texturemanager.h.

12.15.4.8 GLuint TextureManager::width

Definition at line 76 of file texturemanager.h.

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

- texturemanager.h
- texturemanager.cpp

12.16 vec2 Union Reference

Two element vector of floats.

`#include <structs.h>`

Public Attributes

- struct {
  float x
  float y
  };

- struct {
  float s
  float t
  };
- float v [2]

12.16.1 Detailed Description

Two element vector of floats.

Definition at line 73 of file structs.h.

12.16.2 Member Data Documentation

12.16.2.1 struct { ... }

12.16.2.2 struct { ... }

12.16.2.3 float vec2::s

Definition at line 76 of file structs.h.

12.16.2.4 float vec2::t

Definition at line 76 of file structs.h.

12.16.2.5 float vec2::v[2]

Definition at line 77 of file structs.h.

12.16.2.6 float vec2::x

Definition at line 75 of file structs.h.

12.16.2.7 float vec2::y

Definition at line 75 of file structs.h.

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

- structs.h

12.17 vec3 Union Reference

Three element vector of floats.

`#include <structs.h>`

Public Attributes

- struct {
  float x
  float y
  float z
  };

- struct {
  float b
  float g
  float r
  };
- struct {
  float p
  float s
  float t
  };
- float v [3]

12.17.1 Detailed Description

Three element vector of floats.

Definition at line 59 of file structs.h.

12.17.2 Member Data Documentation

12.17.2.1 struct { ... }

12.17.2.2 struct { ... }

12.17.2.3 struct { ... }

12.17.2.4 float vec3::b

Definition at line 62 of file structs.h.

12.17.2.5 float vec3::g

Definition at line 62 of file structs.h.

12.17.2.6 float vec3::p

Definition at line 63 of file structs.h.

12.17.2.7 float vec3::r

Definition at line 62 of file structs.h.

12.17.2.8 float vec3::s

Definition at line 63 of file structs.h.

12.17.2.9 float vec3::t

Definition at line 63 of file structs.h.

12.17.2.10 float vec3::v[3]

Definition at line 64 of file structs.h.

12.17.2.11 float vec3::x

Definition at line 61 of file structs.h.

12.17.2.12 float vec3::y

Definition at line 61 of file structs.h.

12.17.2.13 float vec3::z

Definition at line 61 of file structs.h.

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

- structs.h

12.18 vec3i Union Reference

Three element vector of integers.

`#include <structs.h>`

Public Attributes

- struct {
  unsigned char x
  unsigned char y
  unsigned char z
  };
- struct {
  unsigned char b
  unsigned char g
  unsigned char r
  };
- struct {
  unsigned char p
  unsigned char s
  unsigned char t
  };
- unsigned char v [4]

12.18.1 Detailed Description

Three element vector of integers.

Definition at line 45 of file structs.h.

12.18.2 Member Data Documentation

12.18.2.1 struct { ... }

12.18.2.2 struct { ... }

12.18.2.3 struct { ... }

12.18.2.4 unsigned char vec3i::b

Definition at line 48 of file structs.h.

12.18.2.5 unsigned char vec3i::g

Definition at line 48 of file structs.h.

12.18.2.6 unsigned char vec3i::p

Definition at line 49 of file structs.h.

12.18.2.7 unsigned char vec3i::r

Definition at line 48 of file structs.h.

12.18.2.8 unsigned char vec3i::s

Definition at line 49 of file structs.h.

12.18.2.9 unsigned char vec3i::t

Definition at line 49 of file structs.h.

12.18.2.10 unsigned char vec3i::v[4]

Definition at line 50 of file structs.h.

12.18.2.11 unsigned char vec3i::x

Definition at line 47 of file structs.h.

12.18.2.12 unsigned char vec3i::y

Definition at line 47 of file structs.h.

12.18.2.13 unsigned char vec3i::z

Definition at line 47 of file structs.h.

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

- structs.h

12.19 vec4 Union Reference

Four element vector of floats.

`#include <structs.h>`

Public Attributes

- struct {
  float w
  float x
  float y
  float z
  };

- struct {
  float a
  float b
  float g
  float r

};

- struct {
  float p
  float q
  float s
  float t
  };

- float v [4]

12.19.1 Detailed Description

Four element vector of floats.

Definition at line 31 of file structs.h.

12.19.2 Member Data Documentation

12.19.2.1 struct { ... }

12.19.2.2 struct { ... }

12.19.2.3 struct { ... }

12.19.2.4 float vec4::a

Definition at line 34 of file structs.h.

12.19.2.5 float vec4::b

Definition at line 34 of file structs.h.

12.19.2.6 float vec4::g

Definition at line 34 of file structs.h.

12.19.2.7 float vec4::p

Definition at line 35 of file structs.h.

12.19.2.8 float vec4::q

Definition at line 35 of file structs.h.

12.19.2.9 float vec4::r

Definition at line 34 of file structs.h.

12.19.2.10 float vec4::s

Definition at line 35 of file structs.h.

12.19.2.11 float vec4::t

Definition at line 35 of file structs.h.

12.19.2.12 float vec4::v[4]

Definition at line 36 of file structs.h.

12.19.2.13 float vec4::w

Definition at line 33 of file structs.h.

12.19.2.14 float vec4::x

Definition at line 33 of file structs.h.

12.19.2.15 float vec4::y

Definition at line 33 of file structs.h.

12.19.2.16 float vec4::z

Definition at line 33 of file structs.h.

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

- structs.h

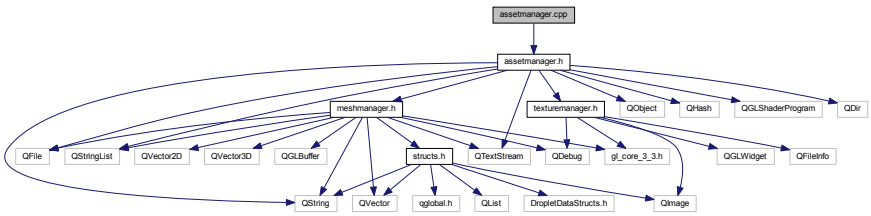
13 File Documentation

13.1 assetmanager.cpp File Reference

Implements the AssetManager class.

```
#include "assetmanager.h"
```

Include dependency graph for assetmanager.cpp:



13.1.1 Detailed Description

Implements the AssetManager class.

Definition in file assetmanager.cpp.

13.2 assetmanager.cpp

```
00001
00007 #include "assetmanager.h"
00008
00009 AssetManager::AssetManager()
00010 {
00011  _assetDir = "./";
00012  _assetManifest = "manifest.txt";
00013  _textureDir = "./";
00014  _shaderDir = "./";
00015  _shaderManifest = "shaders.txt";
00016  _meshDir = "./";
00017  _unused = true;
00018 }
00019
```

```
00020 AssetManager::~AssetManager()
00021 {
00022  if (!_unused)
00023  {
00024   clearAssets();
00025  }
00026 }
00027
00028 bool AssetManager::loadAssets()
00029 {
00030  qDebug() << "Loading assets...";
00031  loadManifest();
00032  loadShaders();
00033  loadMeshes();
00034  loadTextures();
00035  return true;
00036 }
00037
00038 void AssetManager::loadMeshes()
00039 {
00040  qDebug() << "Loading Meshes...";
00041  QString meshPath = _assetDir;
00042  meshPath.append(_meshDir);
00043  QDir inDir(meshPath,"*.obj");
00044  if (inDir.exists())
00045  {
00046   QStringList files = inDir.entryList();
00047   QList<QString>::const_iterator i = files.begin();
00048   for (i = files.begin() ; i != files.end() ; i++)
00049   {
00050    qDebug() << *i;
00051    MeshManager *mesh = new MeshManager();
00052    if (mesh->setFile(QString(meshPath).append(*i)))
00053    {
00054     QString name(*i);
00055     name.remove(".obj",Qt::CaseInsensitive);
00056     _meshes[name.toLower()] = mesh;
00057    } else {
00058     delete mesh;
00059    }
00060   }
00061
00062  } else {
00063   qDebug() << "Error: directory not found " << _meshDir;
00064  }
00065 }
00066
00067 void AssetManager::setShaderDir(QString dir)
00068 {
00069  _shaderDir = dir;
00070 }
00071 void AssetManager::setMeshDir(QString dir)
00072 {
00073  _meshDir = dir;
00074 }
00075 void AssetManager::setTextureDir(QString dir)
00076 {
00077  _textureDir = dir;
00078 }
00079
00080 void AssetManager::setAssetDir(QString dir)
00081 {
00082  _assetDir = dir;
00083 }
00084 void AssetManager::setManifest(QString file)
00085 {
00086  _assetManifest = file;
00087 }
00088 void AssetManager::setShaderManifest(QString file)
00089 {
00090  _shaderManifest = file;
00091 }
00092 void AssetManager::loadShaders()
00093 {
00094  qDebug() << "Loading Shaders...";
00095  QString shaderDir(_assetDir);
00096
00097  shaderDir.append(_shaderDir);
00098  QString manifestPath(shaderDir);
00099  manifestPath.append(_shaderManifest);
00100
00101  qDebug() << "Reading in shader manifest " << manifestPath;
00102
00103  QFile file(manifestPath);
00104
00105  if (file.exists())
00106  {
```

```
00107   if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
00108   {
00109    QTextStream in(&file);
00110    while (!in.atEnd())
00111    {
00112     QString line = in.readLine(0);
00113     if (line.count() > 0)
00114     {
00115      if (!line.startsWith("#"))
00116      {
00117       QStringList list = line.split(" ",QString::SkipEmptyParts);
00118       if (list.size() == 3)
00119       {
00120        QGLShaderProgram *prog = new QGLShaderProgram();
00121
00122        if (prog->addShaderFromSourceFile(QGLShader::Vertex,QString(shaderDir).append(list[1])))
00123        {
00124         if (prog->addShaderFromSourceFile(QGLShader::Fragment,QString(shaderDir).append(list[2])))
00125         {
00126          if(prog->link())
00127          {
00128           qDebug() << "Linked program " << list[0];
00129           _shaders[list[0].toLower()] = prog;
00130           _unused = false;
00131          } else {
00132           qDebug() << "Error:" << prog->log();
00133          }
00134         } else {
00135          qDebug() << "Error:" << prog->log();
00136         }
00137        } else {
00138         qDebug() << "Error:" << prog->log();
00139        }
00140       }  else {
00141        qDebug() << "Error: malformed line" << line;
00142       }
00143      } else {
00144       // push the comment to the debug console
00145       qDebug() << line;
00146      }
00147     }
00148    }
00149   } else
00150   {
00151    qDebug() << "Error: " << file.error();
00152
00153   }
00154  } else
00155  {
00156   qDebug() << "Error: file does not exist";
00157  }
00158 }
00159
00160 void AssetManager::loadTextures()
00161 {
00162  qDebug() << "Loading Textures...";
00163  QString texDir(_assetDir);
00164  texDir.append(_textureDir);
00165  QDir inDir(texDir);
00166  QStringList filters;
00167  filters << "*.jpg" << "*.bmp" << "*.png";
00168  inDir.setNameFilters(filters);
00169  if (inDir.exists())
00170  {
00171   QStringList files = inDir.entryList();
00172   QList<QString>::const_iterator i = files.begin();
00173   for (i = files.begin() ; i != files.end() ; i++)
00174   {
00175    qDebug() << *i;
00176
00177    TextureManager *texture = new TextureManager();
00178
00179    if (texture->setFile(QString(texDir).append(*i)))
00180    {
00181
00182     QString name(*i);
00183     name = name.toLower();
00184     name.remove(".jpg");
00185     name.remove(".bmp");
00186     name.remove(".png");
00187     if (!_textures.contains(name))
00188     {
00189      _unused = false;
00190      _textures[name] = texture;
00191     } else {
00192      qDebug() << "Error: namespace collision when loading texture" << *i;
00193      delete texture;
```

```
00194     }
00195    } else {
00196     delete texture;
00197    }
00198   }
00199
00200  } else {
00201   qDebug() << "Error: directory not found " << _textureDir;
00202  }
00203 }
00204 QGLShaderProgram* AssetManager::getShader(QString name)
00205 {
00206  if (_shaders.contains(name.toLower()))
00207  {
00208   return _shaders[name.toLower()];
00209  } else
00210  {
00211   return NULL;
00212  }
00213
00214 }
00215
00216 MeshManager* AssetManager::getMesh(QString name)
00217 {
00218  if (_meshes.contains(name.toLower()))
00219  {
00220   return _meshes[name.toLower()];
00221  } else
00222  {
00223   return NULL;
00224  }
00225
00226 }
00227
00228 TextureManager* AssetManager::getTexture(QString name)
00229 {
00230  name = name.toLower();
00231  name.remove(".jpg");
00232  name.remove(".bmp");
00233  name.remove(".png");
00234  if (_textures.contains(name))
00235  {
00236   return _textures[name];
00237  } else
00238  {
00239   return NULL;
00240  }
00241
00242 }
00243
00244 void AssetManager::clearShaders()
00245 {
00246  qDebug() << "Tearing down shaders...";
00247  QHash<QString,QGLShaderProgram *>::iterator i = _shaders.begin();
00248  while (i != _shaders.end())
00249  {
00250   QGLShaderProgram *prog = i.value();
00251   qDebug() << "Deleting shader" << i.key();
00252   QHash<QString,QGLShaderProgram *>::iterator prev = i;
00253   ++i;
00254   _shaders.erase(prev);
00255   delete prog;
00256
00257  }
00258 }
00259
00260 void AssetManager::clearTextures()
00261 {
00262  qDebug() << "Tearing down textures...";
00263  QHash<QString,TextureManager *>::iterator i = _textures.begin();
00264  while (i != _textures.end())
00265  {
00266   TextureManager *prog = i.value();
00267   QHash<QString,TextureManager *>::iterator prev = i;
00268   ++i;
00269   qDebug() << "Deleting texture" << prev.key();
00270   delete prog;
00271   _textures.erase(prev);
00272  }
00273 }
00274
00275 void AssetManager::clearMeshes()
00276 {
00277  qDebug() << "Tearing down meshes...";
00278
00279  QHash<QString,MeshManager *>::iterator i = _meshes.begin();
00280  while (i != _meshes.end())
```

```
00281  {
00282   MeshManager *prog = i.value();
00283   QHash<QString,MeshManager *>::iterator prev = i;
00284   ++i;
00285   qDebug() << "Deleting mesh" << prev.key();
00286   delete prog;
00287   _meshes.erase(prev);
00288  }
00289 }
00290
00291 void AssetManager::clearAssets()
00292 {
00293  clearManifest();
00294  clearShaders();
00295  clearMeshes();
00296  clearTextures();
00297  _unused = true;
00298 }
00299
00300 void AssetManager::reloadAssets()
00301 {
00302  clearAssets();
00303  loadAssets();
00304 }
00305
00306
00307 void AssetManager::loadManifest()
00308 {
00309  qDebug() << "Loading asset names...";
00310  QString manifestPath(_assetDir) ;
00311  manifestPath.append(_assetManifest);
00312
00313  qDebug() << "Reading in asset manifest " << manifestPath;
00314
00315  QFile file(manifestPath);
00316
00317  if (file.exists())
00318  {
00319   if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
00320   {
00321    QTextStream in(&file);
00322    while (!in.atEnd())
00323    {
00324     QString line = in.readLine(0);
00325     if (line.count() > 0)
00326     {
00327      if (!line.startsWith("#"))
00328      {
00329       QStringList list = line.split(" ",QString::SkipEmptyParts);
00330       if (list.size() == 2)
00331       {
00332        if (!_manifest.contains(list[0]))
00333        {
00334         _manifest[list[0]] = list[1];
00335        } else {
00336         qDebug() << "Error: namespace collision on manifest item" << list[0];
00337        }
00338
00339       }  else {
00340        qDebug() << "Error: malformed line" << line;
00341       }
00342      } else {
00343       // push the comment to the debug console
00344       qDebug() << line;
00345      }
00346     }
00347    }
00348   } else
00349   {
00350    qDebug() << "Error: " << file.error();
00351
00352   }
00353  } else
00354  {
00355   qDebug() << "Error: file does not exist";
00356  }
00357 }
00358
00359 void AssetManager::clearManifest()
00360 {
00361  _manifest.clear();
00362 }
00363
00364 QString AssetManager::lookupAssetName(QString name)
00365 {
00366  if (_manifest.contains(name))
00367  {
```

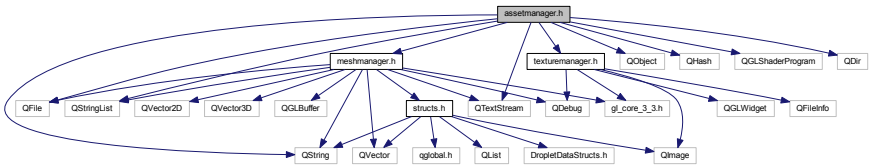
```
00368    return _manifest[name];
00369   } else {
00370    return QString("null");
00371   }
00372 }
```

13.3 assetmanager.h File Reference

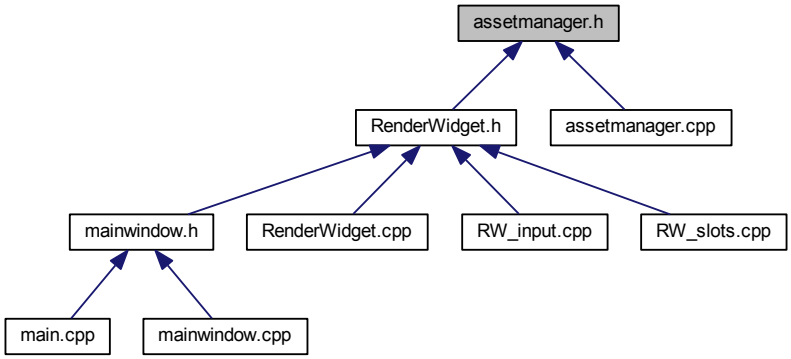
Declares the AssetManager class.

```
#include "texturemanager.h"
#include "meshmanager.h"
#include <QObject>
#include <QHash>
#include <QString>
#include <QFile>
#include <QTextStream>
#include <QGLShaderProgram>
#include <QStringList>
#include <QDir>
```

Include dependency graph for assetmanager.h:



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



Classes

- class AssetManager

  Items related to managing assets required for rendering.

13.3.1 Detailed Description

Declares the AssetManager class.

Definition in file assetmanager.h.

13.4 assetmanager.h

```
00001
00007 #ifndef ASSETMANAGER_H
00008 #define ASSETMANAGER_H
00009
00010
00011 #include "texturemanager.h"
00012 #include "meshmanager.h"
00013
00014 #include <QObject>
00015 #include <QHash>
00016 #include <QString>
00017 #include <QFile>
00018 #include <QTextStream>
00019 #include <QGLShaderProgram>
00020 #include <QStringList>
00021 #include <QDir>
00022
00035 class AssetManager
00036 {
00037
00038 public:
00039
00046  AssetManager();
00047
00054  ~AssetManager();
00055
00064  bool loadAssets();
00065
00072  void clearAssets();
00073
00080  void reloadAssets();
00081
00090  void setShaderDir(QString dir);
00091
00100  void setMeshDir(QString dir);
00101
00110  void setTextureDir(QString dir);
00111
00120  void setAssetDir(QString dir);
00121
00130  void setManifest(QString file);
00131
00140  void setShaderManifest(QString file);
00141
00152  QString lookupAssetName(QString name);
00153
00164  QGLShaderProgram* getShader(QString name);
00165
00176  MeshManager* getMesh(QString name);
00177
00188  TextureManager* getTexture(QString name);
00189 private:
00190
00195  QHash<QString,QGLShaderProgram *> _shaders;
00196
00201  QHash<QString,TextureManager *> _textures;
00202
00207  QHash<QString,MeshManager *> _meshes;
00208
00213  QString _shaderDir;
00214
00219  QString _meshDir;
00220
00225  QString _textureDir;
00226
00231  QString _projectionDir;
00232
00237  QString _shaderManifest;
00238
00243  QString _assetDir;
00244
00249  QString _assetManifest;
00250
00255  QHash<QString,QString> _manifest;
00256
```

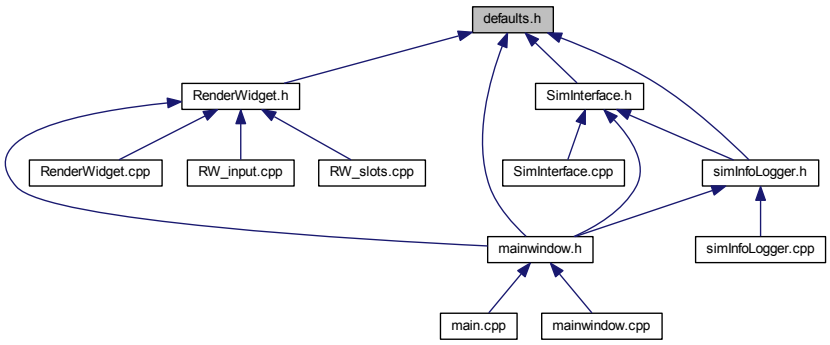
```
00261  bool _unused;
00262
00269  void loadShaders();
00270
00277  void clearShaders();
00278
00285  void loadTextures();
00286
00293  void clearTextures();
00294
00301  void loadMeshes();
00302
00309  void clearMeshes();
00310
00317  void loadManifest();
00318
00325  void clearManifest();
00326 };
00327
00328 #endif // ASSETMANAGER_H
```

13.5 build.dox File Reference

13.6 defaults.h File Reference

Declares the global defaults.

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



Macros

- #define DEBUG_DROPLET_MESH "cylinder"
- #define DEBUG_MESH "cube"
- #define DEBUG_SHADER "debug"
- #define DEFAULT_ASSETDIR "./assets/"
- #define DEFAULT_COL_TILES 4
- #define DEFAULT_DROPLET_CENTER_OF_MASS 0.15f
- #define DEFAULT_DROPLET_FRICTION .5f
- #define DEFAULT_DROPLET_MASS 24.2f
- #define DEFAULT_DROPLET_RADIUS 2.2f
- #define DEFAULT_FLOOR_FRICTION .5f
- #define DEFAULT_FLOOR_MASS 0.0f
- #define DEFAULT_FLOORDIR "Floors/"
- #define DEFAULT_FPS 60.0f
- #define DEFAULT_MESHDIR "Models/"
- #define DEFAULT_OBJECT_FRICTION .25f

- #define DEFAULT_OBJECT_MASS 100.f
- #define DEFAULT_OBJECT_RADIUS 4.0f
- #define DEFAULT_PROJECTDIR "Projector/"
- #define DEFAULT_ROW_TILES 4
- #define DEFAULT_SETTINGS "defaults.txt"
- #define DEFAULT_SETUPDIR "Setup/"
- #define DEFAULT_SHADERDIR "Shaders/"
- #define DEFAULT_TEXTUREDIR "Textures/"
- #define DEFAULT_TILE_LENGTH 25.0f
- #define DEFAULT_WALL_FRICTION .5f
- #define DEFAULT_WALL_HEIGHT 5.0f
- #define DEFAULT_WALL_MASS 0.0f
- #define DEFAULT_WALL_WIDTH 1.0f
- #define DEFAULT_WINDOW_HEIGHT 720
- #define DEFAULT_WINDOW_NAME "Droplets Renderer"
- #define DEFAULT_WINDOW_WIDTH 1280
- #define DROPLET_MESH_NAME "droplet_mesh"
- #define DROPLET_PROJECTION_SHADER_NAME "droplet_projection"
- #define DROPLET_SHADER_NAME "droplet_shader"
- #define DROPLET_TEXTURE_NAME "droplet_texture"
- #define FLOOR_MESH_NAME "floor_mesh"
- #define FLOOR_PROJECTION_SHADER_NAME "floor_projection"
- #define FLOOR_SHADER_NAME "floor_shader"
- #define FLOOR_TEXTURE_NAME "floor_texture"
- #define OBJECT_CUBE_MESH_NAME "object_cube_mesh"
- #define OBJECT_CUBE_PROJECTION_SHADER_NAME "object_cube_projection"
- #define OBJECT_CUBE_SHADER_NAME "object_cube_shader"
- #define OBJECT_CUBE_TEXTURE_NAME "object_cube_texture"
- #define OBJECT_MESH_NAME "object_mesh"
- #define OBJECT_PROJECTION_SHADER_NAME "object_projection"
- #define OBJECT_SHADER_NAME "object_shader"
- #define OBJECT_TEXTURE_NAME "object_texture"
- #define PROJECTOR_TEXTURE_NAME "projector_texture"
- #define SCREENSHOT_HEIGHT 3072
- #define SCREENSHOT_WIDTH 4096
- #define TOWER_MESH_NAME "tower_mesh"
- #define TOWER_SHADER_NAME "tower_shader"
- #define TOWER_TEXTURE_NAME "tower_texture"
- #define WALL_MESH_NAME "wall_mesh"
- #define WALL_SHADER_NAME "wall_shader"
- #define WALL_TEXTURE_NAME "wall_texture"

13.6.1 Detailed Description

Declares the global defaults.

Definition in file defaults.h.

13.6.2 Macro Definition Documentation

13.6.2.1 #define DEBUG_DROPLET_MESH "cylinder"

Definition at line 55 of file defaults.h.

13.6.2.2 #define DEBUG_MESH "cube"

Definition at line 54 of file defaults.h.

13.6.2.3 #define DEBUG_SHADER "debug"

Definition at line 56 of file defaults.h.

13.6.2.4 #define DEFAULT_ASSETDIR "./assets/"

Definition at line 7 of file defaults.h.

13.6.2.5 #define DEFAULT_COL_TILES 4

Definition at line 25 of file defaults.h.

13.6.2.6 #define DEFAULT_DROPLET_CENTER_OF_MASS 0.15f

Definition at line 40 of file defaults.h.

13.6.2.7 #define DEFAULT_DROPLET_FRICTION .5f

Definition at line 37 of file defaults.h.

13.6.2.8 #define DEFAULT_DROPLET_MASS 24.2f

Definition at line 36 of file defaults.h.

13.6.2.9 #define DEFAULT_DROPLET_RADIUS 2.2f

Definition at line 30 of file defaults.h.

13.6.2.10 #define DEFAULT_FLOOR_FRICTION .5f

Definition at line 51 of file defaults.h.

13.6.2.11 #define DEFAULT_FLOOR_MASS 0.0f

Definition at line 50 of file defaults.h.

13.6.2.12 #define DEFAULT_FLOORDIR "Floors/"

Definition at line 13 of file defaults.h.

13.6.2.13 #define DEFAULT_FPS 60.0f

Definition at line 34 of file defaults.h.

13.6.2.14 #define DEFAULT_MESHDIR "Models/"

Definition at line 10 of file defaults.h.

13.6.2.15 #define DEFAULT_OBJECT_FRICTION .25f

Definition at line 44 of file defaults.h.

13.6.2.16 #define DEFAULT_OBJECT_MASS 100.f

Definition at line 43 of file defaults.h.

13.6.2.17 #define DEFAULT_OBJECT_RADIUS 4.0f

Definition at line 45 of file defaults.h.

13.6.2.18 #define DEFAULT_PROJECTDIR "Projector/"

Definition at line 12 of file defaults.h.

13.6.2.19 #define DEFAULT_ROW_TILES 4

Definition at line 24 of file defaults.h.

13.6.2.20 #define DEFAULT_SETTINGS "defaults.txt"

Definition at line 11 of file defaults.h.

13.6.2.21 #define DEFAULT_SETUPDIR "Setup/"

Definition at line 14 of file defaults.h.

13.6.2.22 #define DEFAULT_SHADERDIR "Shaders/"

Definition at line 8 of file defaults.h.

13.6.2.23 #define DEFAULT_TEXTUREDIR "Textures/"

Definition at line 9 of file defaults.h.

13.6.2.24 #define DEFAULT_TILE_LENGTH 25.0f

Definition at line 29 of file defaults.h.

13.6.2.25 #define DEFAULT_WALL_FRICTION .5f

Definition at line 48 of file defaults.h.

13.6.2.26 #define DEFAULT_WALL_HEIGHT 5.0f

Definition at line 31 of file defaults.h.

13.6.2.27 #define DEFAULT_WALL_MASS 0.0f

Definition at line 47 of file defaults.h.

13.6.2.28 #define DEFAULT_WALL_WIDTH 1.0f

Definition at line 33 of file defaults.h.

13.6.2.29 #define DEFAULT_WINDOW_HEIGHT 720

Definition at line 18 of file defaults.h.

13.6.2.30 #define DEFAULT_WINDOW_NAME "Droplets Renderer"

Definition at line 16 of file defaults.h.

13.6.2.31 #define DEFAULT_WINDOW_WIDTH 1280

Definition at line 17 of file defaults.h.

13.6.2.32 #define DROPLET_MESH_NAME "droplet_mesh"

Definition at line 73 of file defaults.h.

13.6.2.33 #define DROPLET_PROJECTION_SHADER_NAME "droplet_projection"

Definition at line 76 of file defaults.h.

13.6.2.34 #define DROPLET_SHADER_NAME "droplet_shader"

Definition at line 75 of file defaults.h.

13.6.2.35 #define DROPLET_TEXTURE_NAME "droplet_texture"

Definition at line 74 of file defaults.h.

13.6.2.36 #define FLOOR_MESH_NAME "floor_mesh"

Definition at line 64 of file defaults.h.

13.6.2.37 #define FLOOR_PROJECTION_SHADER_NAME "floor_projection"

Definition at line 67 of file defaults.h.

13.6.2.38 #define FLOOR_SHADER_NAME "floor_shader"

Definition at line 66 of file defaults.h.

13.6.2.39 #define FLOOR_TEXTURE_NAME "floor_texture"

Definition at line 65 of file defaults.h.

13.6.2.40 #define OBJECT_CUBE_MESH_NAME "object_cube_mesh"

Definition at line 83 of file defaults.h.

13.6.2.41 #define OBJECT_CUBE_PROJECTION_SHADER_NAME "object_cube_projection"

Definition at line 86 of file defaults.h.

13.6.2.42 #define OBJECT_CUBE_SHADER_NAME "object_cube_shader"

Definition at line 85 of file defaults.h.

13.6.2.43 #define OBJECT_CUBE_TEXTURE_NAME "object_cube_texture"

Definition at line 84 of file defaults.h.

13.6.2.44 #define OBJECT_MESH_NAME "object_mesh"

Definition at line 78 of file defaults.h.

13.6.2.45 #define OBJECT_PROJECTION_SHADER_NAME "object_projection"

Definition at line 81 of file defaults.h.

13.6.2.46 #define OBJECT_SHADER_NAME "object_shader"

Definition at line 80 of file defaults.h.

13.6.2.47 #define OBJECT_TEXTURE_NAME "object_texture"

Definition at line 79 of file defaults.h.

13.6.2.48 #define PROJECTOR_TEXTURE_NAME "projector_texture"

Definition at line 58 of file defaults.h.

13.6.2.49 #define SCREENSHOT_HEIGHT 3072

Definition at line 21 of file defaults.h.

13.6.2.50 #define SCREENSHOT_WIDTH 4096

Definition at line 20 of file defaults.h.

13.6.2.51 #define TOWER_MESH_NAME "tower_mesh"

Definition at line 60 of file defaults.h.

13.6.2.52 #define TOWER_SHADER_NAME "tower_shader"

Definition at line 62 of file defaults.h.

13.6.2.53 #define TOWER_TEXTURE_NAME "tower_texture"

Definition at line 61 of file defaults.h.

13.6.2.54 #define WALL_MESH_NAME "wall_mesh"

Definition at line 69 of file defaults.h.

13.6.2.55 #define WALL_SHADER_NAME "wall_shader"

Definition at line 71 of file defaults.h.

13.6.2.56 #define WALL_TEXTURE_NAME "wall_texture"

Definition at line 70 of file defaults.h.

13.7 defaults.h

```
00001
00007 #define DEFAULT_ASSETDIR "./assets/"
00008 #define DEFAULT_SHADERDIR "Shaders/"
00009 #define DEFAULT_TEXTUREDIR "Textures/"
00010 #define DEFAULT_MESHDIR "Models/"
00011 #define DEFAULT_SETTINGS "defaults.txt"
00012 #define DEFAULT_PROJECTDIR "Projector/"
00013 #define DEFAULT_FLOORDIR "Floors/"
00014 #define DEFAULT_SETUPDIR "Setup/"
00015
00016 #define DEFAULT_WINDOW_NAME "Droplets Renderer"
00017 #define DEFAULT_WINDOW_WIDTH 1280
00018 #define DEFAULT_WINDOW_HEIGHT 720
00019
00020 #define SCREENSHOT_WIDTH 4096
00021 #define SCREENSHOT_HEIGHT 3072
00022
00023 // Droplet Simulator specific constants
00024 #define DEFAULT_ROW_TILES 4
00025 #define DEFAULT_COL_TILES 4
00026
00027 //#define DEFAULT_NUM_DROPLETS 128
00028
00029 #define DEFAULT_TILE_LENGTH 25.0f
00030 #define DEFAULT_DROPLET_RADIUS 2.2f
00031 #define DEFAULT_WALL_HEIGHT 5.0f // in cm
00032
```

```
00033 #define DEFAULT_WALL_WIDTH 1.0f
00034 #define DEFAULT_FPS 60.0f // 60 frames per second
00035
00036 #define DEFAULT_DROPLET_MASS  24.2f // in g
00037 #define DEFAULT_DROPLET_FRICTION .5f
00038
00039 // 0.0 is the bottom of the mesh, 1.0 is the top of the mesh
00040 #define DEFAULT_DROPLET_CENTER_OF_MASS 0.15f
00041
00042
00043 #define DEFAULT_OBJECT_MASS  100.f // in g
00044 #define DEFAULT_OBJECT_FRICTION .25f
00045 #define DEFAULT_OBJECT_RADIUS 4.0f
00046
00047 #define DEFAULT_WALL_MASS 0.0f // in kg
00048 #define DEFAULT_WALL_FRICTION .5f
00049
00050 #define DEFAULT_FLOOR_MASS 0.0f // in kg
00051 #define DEFAULT_FLOOR_FRICTION .5f
00052
00053
00054 #define DEBUG_MESH "cube"
00055 #define DEBUG_DROPLET_MESH "cylinder"
00056 #define DEBUG_SHADER "debug"
00057
00058 #define PROJECTOR_TEXTURE_NAME "projector_texture"
00059
00060 #define TOWER_MESH_NAME "tower_mesh"
00061 #define TOWER_TEXTURE_NAME "tower_texture"
00062 #define TOWER_SHADER_NAME "tower_shader"
00063
00064 #define FLOOR_MESH_NAME "floor_mesh"
00065 #define FLOOR_TEXTURE_NAME "floor_texture"
00066 #define FLOOR_SHADER_NAME "floor_shader"
00067 #define FLOOR_PROJECTION_SHADER_NAME "floor_projection"
00068
00069 #define WALL_MESH_NAME "wall_mesh"
00070 #define WALL_TEXTURE_NAME "wall_texture"
00071 #define WALL_SHADER_NAME "wall_shader"
00072
00073 #define DROPLET_MESH_NAME "droplet_mesh"
00074 #define DROPLET_TEXTURE_NAME "droplet_texture"
00075 #define DROPLET_SHADER_NAME "droplet_shader"
00076 #define DROPLET_PROJECTION_SHADER_NAME "droplet_projection"
00077
00078 #define OBJECT_MESH_NAME "object_mesh"
00079 #define OBJECT_TEXTURE_NAME "object_texture"
00080 #define OBJECT_SHADER_NAME "object_shader"
00081 #define OBJECT_PROJECTION_SHADER_NAME "object_projection"
00082
00083 #define OBJECT_CUBE_MESH_NAME "object_cube_mesh"
00084 #define OBJECT_CUBE_TEXTURE_NAME "object_cube_texture"
00085 #define OBJECT_CUBE_SHADER_NAME "object_cube_shader"
00086 #define OBJECT_CUBE_PROJECTION_SHADER_NAME "object_cube_projection"
```

13.8 doxgyen.dox File Reference

13.9 features.dox File Reference

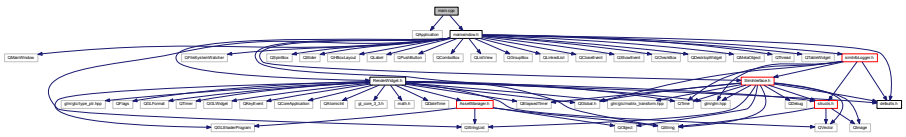
13.10 files.dox File Reference

13.11 main.cpp File Reference

Implements the main function.

```
#include <QApplication>
#include "mainwindow.h"
```

Include dependency graph for main.cpp:



Functions

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

  [0]

13.11.1 Detailed Description

Implements the main function.

Definition in file main.cpp.

13.11.2 Function Documentation

13.11.2.1 int main (int *argc*, char ** *argv*)

[0]

Definition at line 11 of file main.cpp.

13.12 main.cpp

```
00001
00007 #include <QApplication>
00008 #include "mainwindow.h"
00009
00011 int main(int argc, char **argv)
00012 {
00013  // initialize the QApplication object
00014  QApplication a(argc, argv);
00015
00016  // create the main window
00017  MainWindow mw;
00018  QDesktopWidget * desktop = QApplication::desktop();
00019  QSize size = mw.sizeHint();
00020  int w = size.width();
00021  int h = size.height();
00022  int screen_width = desktop->width();
00023  int screen_height = desktop->height();
00024  //mw.move(screen_width - w, 0);
00025  //mw.setGeometry(screen_width - (w+10), 30, w, h);
00026  mw.setGeometry(screen_width/2 - w/2, screen_height/2 - h/2, w, h);
00027  // show the main window
00028  mw.show();
00029
00030  // start the main program execution loop
00031  int result = a.exec();
00032
00033  // cleanup
00034  return result;
00035 }
```

13.13 main.dox File Reference

13.14 mainwindow.cpp File Reference

Implements the MainWindow class.

```
#include "mainwindow.h"
#include <QtGui>
```

Include dependency graph for mainwindow.cpp:

13.14.1 Detailed Description

Implements the MainWindow class.

Definition in file mainwindow.cpp.

13.15 mainwindow.cpp

```
00001
00007 #include "mainwindow.h"
00008 #include <QtGui>
00009
00010
00011 MainWindow::MainWindow(QWidget *parent)
00012  : QMainWindow(parent)
00013 {
00014  //ui.setupUi(this);
00015  QStringList argsList = qApp->arguments();
00016  fileName = QString(DEFAULT_ASSETDIR).append(
      DEFAULT_SETUPDIR).append(DEFAULT_SETTINGS);
00017  if (argsList.count() > 1)
00018  {
00019   // read the first item as the arena file, otherwise use the hard coded default
00020   fileName = argsList[1];
00021  }
00022  if (qApp)
00023   qApp->installEventFilter(this);
00024
00025  setUpGUI();
00026
00027  simSetting_t settings = _simInterface.
      getDefaultSettings();
00028  if (loadFromFile(fileName,settings))
00029  {
00030   qDebug() << "Loaded settings out of file" << fileName;
00031  }
00032
00033  // TODO: Update UI so these are propagated without a hack
00034  _arenaObjects.droplets.clear();
00035  _arenaObjects.objects.clear();
00036  _arenaObjects.droplets = settings.startingDroplets;
00037  _arenaObjects.objects = settings.startingObjects;
00038
00039  setUI(settings);
00040  _simulatorRunning = false;
00041
00042
00043  // connect the file system monitors to the appropriate functions
00044  connect(&projectionWatcher,SIGNAL(directoryChanged(QString)),this,SLOT(
      loadProjectionTextures(QString)));
00045  connect(&arenaWatcher,SIGNAL(directoryChanged(QString)),this,SLOT(
      loadFloorFiles(QString)));
00046  connect(&setupWatcher,SIGNAL(directoryChanged(QString)),this,SLOT(
      loadSetupFiles(QString)));
00047  connect(&setupWatcher,SIGNAL(fileChanged(QString)),this,SLOT(
      updateSetupFile(QString)));
00048
00049  projectionWatcher.addPath((QString(DEFAULT_ASSETDIR).append(
      DEFAULT_PROJECTDIR)));
00050  arenaWatcher.addPath(QString(DEFAULT_ASSETDIR).append(
      DEFAULT_FLOORDIR));
00051  setupWatcher.addPath(QString(DEFAULT_ASSETDIR).append(
      DEFAULT_SETUPDIR));
00052
00053  qRegisterMetaType<simState_t>("simState_t");
00054  qRegisterMetaType<simRate_t>("simRate_t");
00055  qRegisterMetaType<simSetting_t>("simSetting_t");
00056  qRegisterMetaType<droplet_t>("droplet_t");
```

```
00057
00058  _simInterface.moveToThread(&_simThread);
00059  _simThread.start();
00060  connect(this,SIGNAL(launchSim(simSetting_t)),&
       _simInterface,SLOT(loadArena(simSetting_t)));
00061  connect(this,SIGNAL(closing(void)),&_simThread,SLOT(quit(void)));
00062  connect(this,SIGNAL(closing(void)),&_logger,SLOT(close(void)));
00063
00064
00065  connect(this,SIGNAL(enableLimit(void)),&_simInterface,SLOT(enableUpdateLimit(void)
       ));
00066  connect(this,SIGNAL(disableLimit(void)),&_simInterface,SLOT(disableUpdateLimit(
       void)));
00067  connect(this,SIGNAL(setUpdateRate(float)),&_simInterface,SLOT(
       setUpdateRate(float)));
00068
00069
00070 }
00071
00072 MainWindow::~MainWindow()
00073 {
00074
00075 }
00076
00077 bool MainWindow::eventFilter(QObject *watched, QEvent *event)
00078 {
00079  if (watched == qApp)
00080  {
00081   if (event->type() == QEvent::ApplicationActivate)
00082   {
00083    qDebug() << "Application gained focus";
00084
00085
00086
00087  //  loadProjectionTextures();
00088  //  loadFloorFiles();
00089  //  loadSetupFiles();
00090
00091   } else if (event->type() == QEvent::ApplicationDeactivate)
00092   {
00093    qDebug() << "Application lost focus";
00094
00095   }
00096  }
00097  return QMainWindow::eventFilter(watched,event);
00098 }
00099
00100 void MainWindow::launchSimulator()
00101 {
00102  simSetting_t newSettings = _simInterface.
       getDefaultSettings();
00103  if(arenaSelectionCombo->currentText() != QString("Default (Rectangle)"))
00104  {
00105   newSettings.floorFile = arenaSelectionCombo->currentText();
00106   newSettings.floorFile.append(".txt");
00107   simSetting_t currentSimSettings = _simInterface.
       getCurrentSettings();
00108   newSettings.numColTiles = currentSimSettings.numColTiles;
00109   newSettings.numRowTiles = currentSimSettings.numRowTiles;
00110  }
00111  else
00112  {
00113   newSettings.numColTiles = numColBox->value();
00114   newSettings.numRowTiles = numRowBox->value();
00115  }
00116  //newSettings.dropletRadius = dropRadBox->value();
00117  if (projectorImageCombo->currentText() != QString("None"))
00118  {
00119   newSettings.projecting = true;
00120   newSettings.projTexture = projectorImageCombo->currentText();
00121   newSettings.projTexture.append(".bmp");
00122  }
00123
00124  newSettings.startingDroplets = _arenaObjects.droplets;
00125  newSettings.startingObjects = _arenaObjects.objects;
00126
00127
00128  //droplets.append(dropProgramsCombo->currentText());
00129  //droplets.append(QString("%1").arg(numDropBox->value()));
00130  //newSettings.startingDroplets.append(droplets);
00131
00132  if(loadSetupFileCombo->currentText() == QString("None"))
00133  {
00134
00135   newSettings.startingDroplets.clear();
00136   for(int i = 0; i < dropletTableWidget->rowCount(); i++)
00137   {
```

```
00138    QStringList droplets;
00139    //QTableWidgetItem * header;
00140    //header = dropletTableWidget->verticalHeaderItem(i);
00141    qDebug() << "header" << dropletTableWidget->verticalHeaderItem(i)->text();
00142    qDebug() << "header row" << dropletTableWidget->verticalHeaderItem(i)->row();
00143    qDebug() << "item to text" << dropletTableWidget->item(i,0)->text();
00144    qDebug() << "item to text to int" << dropletTableWidget->item(i,0)->text().toInt();
00145    int entry = dropletTableWidget->item(i,0)->text().toInt();
00146    qDebug() << "entry" << entry;
00147    if(entry != 0)
00148    {
00149     droplets.append(QString(dropletTableWidget->verticalHeaderItem(i)->text()));
00150     droplets.append(QString("%1").arg(entry));
00151     newSettings.startingDroplets.append(droplets);
00152    }
00153   }
00154  }
00155  //saveToFile(QString("LastRun.txt"),newSettings);
00156
00157  qDebug() << "checkbox: " << logCheckBox->isChecked();
00158  if(logCheckBox->isChecked())
00159  {
00160   qDebug() << "logbox checked";
00161   _logger.setPosFlag(posCheckBox->isChecked());
00162   _logger.setColorFlag(colCheckBox->isChecked());
00163   _logger.setRotationFlag(rotCheckBox->isChecked());
00164   _logger.setCommSAFlag(commSACheckBox->isChecked());
00165   _logger.setMacroRedFlag(macroRedCheckBox->isChecked());
00166   _logger.setMacroSAFlag(macroSACheckBox->isChecked());
00167   _logger.Init();
00168   connect(&_simInterface,SIGNAL(simulationUpdated(simState_t)),&
     _logger,SLOT(timeCheck(simState_t)));
00169  }
00170
00171  emit launchSim(newSettings);
00172
00173  if (!_simulatorRunning)
00174  {
00175   emit setUpdateRate(1.0f);
00176   emit enableLimit();
00177
00178   _simulatorRunning = true;
00179  }
00180 }
00181
00182 void MainWindow::launchRenderer()
00183 {
00184
00185  // set the format for the OpenGL window
00186  QGLFormat glFormat = QGLFormat(QGL::DoubleBuffer | QGL::DirectRendering);
00187  glFormat.setVersion( 3, 3 );
00188  glFormat.setDepthBufferSize(24);
00189  glFormat.setProfile( QGLFormat::CompatibilityProfile ); // Requires >=Qt-4.8.0
00190
00191  // create the window and pass it the simulator as a parameter
00192  RenderWidget *renderer =  new RenderWidget(glFormat);
00193  if (_renderWidgets.count() == 0)
00194  {
00195   emit enableLimit();
00196  }
00197  _renderWidgets.append(renderer);
00198
00199  connect(this,SIGNAL(closing(void)),renderer,SLOT(close(void)));
00200
00201  connect(&_simInterface,SIGNAL(simulationUpdated(simState_t)),renderer,SLOT(
     updateState(simState_t)),Qt::DirectConnection);
00202  connect(&_simInterface,SIGNAL(arenaChanged(simSetting_t)),renderer,SLOT(
     updateArena(simSetting_t)));
00203  connect(&_simInterface,SIGNAL(pauseChanged(bool)),renderer,SLOT(updatePause(bool)));
00204  connect(&_simInterface,SIGNAL(ratesChanged(simRate_t)),renderer,SLOT(updateRate(
     simRate_t)));
00205
00206  connect(renderer,SIGNAL(togglePause(void)),&_simInterface,SLOT(togglePause(void)));
00207  connect(renderer,SIGNAL(restart(void)),&_simInterface,SLOT(reset(void)));
00208  connect(renderer,SIGNAL(requestNewDroplet(float,float,droplet_t)),&
     _simInterface,SLOT(addDroplet(float,float,droplet_t)));
00209  connect(renderer,SIGNAL(toggleLimit(void)),&_simInterface,SLOT(toggleUpdateLimit(void)));
00210  connect(renderer,SIGNAL(increaseRate(void)),&_simInterface,SLOT(increaseUpdateRate(void)));
00211  connect(renderer,SIGNAL(decreaseRate(void)),&_simInterface,SLOT(decreaseUpdateRate(void)));
00212
00213  connect(renderer,SIGNAL(destroyed(QObject*)),this,SLOT(removeRenderer(QObject*)));
00214
00215  renderer->updateArena(_simInterface.
     getSimulatorSettings());
00216  renderer->updateState(_simInterface.getSimulatorState());
00217  renderer->updateRate(_simInterface.getSimulatorRate());
00218  // position the window
```

```
00219  renderer->setWindowTitle(QString(DEFAULT_WINDOW_NAME));
00220  renderer->move(0,0);
00221  renderer->resize(DEFAULT_WINDOW_WIDTH,DEFAULT_WINDOW_HEIGHT);
00222
00223
00224  // show the window
00225  renderer->show();
00226
00227  if (!_simulatorRunning)
00228   launchSimulator();
00229
00230 }
00231
00232 void MainWindow::addLoadSetupFileWidgets()
00233 {
00234  loadSetupFileCombo = new QComboBox;
00235  loadSetupFileList = new QListView(loadSetupFileCombo);
00236  //loadSetupFileButton = new QPushButton("Refresh Setup File Selection List");
00237  //QObject::connect(loadSetupFileButton,SIGNAL(clicked()),this,SLOT(loadSetupFiles()));
00238
00239
00240  loadSetupFileList->setStyleSheet("QListView::item {  \
00241            border-bottom: 5px solid white; margin:3px; } \
00242            QListView::item:selected { \
00243            border-bottom: 5px solid black; margin:3px; \
00244            color:black; \
00245            } \
00246            ");
00247  loadSetupFileCombo->setView(loadSetupFileList);
00248  loadSetupFileLabel = new QLabel(tr("Setup File Selection"));
00249  loadSetupFileLabel->setBuddy(loadSetupFileCombo);
00250  loadSetupFileLayout = new QVBoxLayout;
00251  loadSetupFileLayout->addWidget(loadSetupFileLabel);
00252  loadSetupFileLayout->addWidget(loadSetupFileCombo);
00253
00254  QObject::connect(loadSetupFileCombo, SIGNAL(currentIndexChanged(const QString &)),this,
      SLOT(updateParams(const QString &)));
00255  QObject::connect(loadSetupFileCombo, SIGNAL(currentIndexChanged(const QString &)),this,
      SLOT(enableDisableDropletTable(const QString &)));
00256
00257  loadSetupFile = new QFrame();
00258  loadSetupFile->setWindowTitle(tr("Load Parameters from Setup File"));
00259  QVBoxLayout *setupFileLayout = new QVBoxLayout();
00260  setupFileLayout->addLayout(loadSetupFileLayout);
00261  loadSetupFile->setLayout(setupFileLayout);
00262  loadSetupFile->setFrameStyle(QFrame::Panel);
00263  loadSetupFile->setLineWidth(1);
00264 }
00265
00266 void MainWindow::addDropletWidgets()
00267 {
00268  /*
00269  // Number of Droplets
00270  numDropBox = new QSpinBox;
00271  numDropBox->setRange(0,1024);
00272  numDropBox->setValue(0);
00273  numDropLabel = new QLabel(tr("Number of Droplets: "));
00274  numDropLabel->setBuddy(numDropBox);
00275  numDropLayout = new QHBoxLayout;
00276  numDropLayout->addWidget(numDropLabel);
00277  numDropLayout->addWidget(numDropBox);
00278
00279  // Droplet Radius
00280  dropRadBox = new QDoubleSpinBox;
00281  dropRadBox->setRange(0.5,4);
00282  dropRadBox->setValue(DEFAULT_DROPLET_RADIUS);
00283  dropRadLabel = new QLabel(tr("Droplet radius: "));
00284  dropRadLabel->setBuddy(dropRadBox);
00285  dropRadLayout = new QHBoxLayout;
00286  dropRadLayout->addWidget(dropRadLabel);
00287  dropRadLayout->addWidget(dropRadBox);
00288  */
00289
00290  dropletTableWidget = new QTableWidget(21,1);
00291  QStringList vertHeaders;
00292  vertHeaders.append("March");
00293  vertHeaders.append("Rainbow");
00294  vertHeaders.append("RandomWalk");
00295  vertHeaders.append("RGBSense");
00296  vertHeaders.append("StickPullers");
00297  vertHeaders.append("TurnTest");
00298  vertHeaders.append("CommTest");
00299  vertHeaders.append("PowerTest");
00300  vertHeaders.append("Granola");
00301  vertHeaders.append("StickPullersUpdated");
00302  vertHeaders.append("Ants");
00303  vertHeaders.append("CustomOne");
```

```
00304  vertHeaders.append("CustomTwo");
00305  vertHeaders.append("CustomThree");
00306  vertHeaders.append("CustomFour");
00307  vertHeaders.append("CustomFive");
00308  vertHeaders.append("CustomSix");
00309  vertHeaders.append("CustomSeven");
00310  vertHeaders.append("CustomEight");
00311  vertHeaders.append("CustomNine");
00312  vertHeaders.append("CustomTen");
00313  dropletTableWidget->setVerticalHeaderLabels(vertHeaders);
00314  QStringList horHeaders;
00315  horHeaders.append("Number");
00316  dropletTableWidget->setHorizontalHeaderLabels(horHeaders);
00317  for(int i = 0; i < dropletTableWidget->rowCount(); i++)
00318  {
00319   dropletTableWidget->setItem(i, 0, new QTableWidgetItem("0"));
00320
00321  }
00322
00323  // Droplet Programs (would like to not hardcode)
00324  /*dropProgramsCombo = new QComboBox;
00325  dropProgramsList = new QListView(dropProgramsCombo);
00326  dropProgramsCombo->addItem("March");
00327  dropProgramsCombo->addItem("Rainbow");
00328  dropProgramsCombo->addItem("RandomWalk");
00329  dropProgramsCombo->addItem("RGBSense");
00330  dropProgramsCombo->addItem("StickPullers");
00331  dropProgramsCombo->addItem("TurnTest");
00332  dropProgramsCombo->addItem("CommTest");
00333  dropProgramsCombo->addItem("PowerTest");
00334  dropProgramsCombo->addItem("Granola");
00335  dropProgramsCombo->addItem("StickPullersUpdated");
00336  dropProgramsCombo->addItem("Ants");
00337  dropProgramsCombo->addItem("CustomOne");
00338  dropProgramsCombo->addItem("CustomTwo");
00339  dropProgramsCombo->addItem("CustomThree");
00340  dropProgramsCombo->addItem("CustomFour");
00341  dropProgramsCombo->addItem("CustomFive");
00342  dropProgramsCombo->addItem("CustomSix");
00343  dropProgramsCombo->addItem("CustomSeven");
00344  dropProgramsCombo->addItem("CustomEight");
00345  dropProgramsCombo->addItem("CustomNine");
00346  dropProgramsCombo->addItem("CustomTen");
00347  dropProgramsList->setStyleSheet("QListView::item {  \
00348          border-bottom: 5px solid white; margin:3px; } \
00349          QListView::item:selected { \
00350          border-bottom: 5px solid black; margin:3px; \
00351          color:black; \
00352          } \
00353          ");
00354  dropProgramsCombo->setCurrentIndex(0);
00355  dropProgramsCombo->setView(dropProgramsList);
00356  dropProgramsLabel = new QLabel(tr("Droplet Programs"));
00357  dropProgramsLabel->setBuddy(dropProgramsCombo);
00358  dropProgramsLayout = new QVBoxLayout;
00359  dropProgramsLayout->addWidget(dropProgramsLabel);
00360  dropProgramsLayout->addWidget(dropProgramsCombo);
00361  */
00362
00363  dropletParams = new QFrame;
00364  dropletParams->setWindowTitle(tr("Droplet Parameters"));
00365  QVBoxLayout *dropletParamsLayout = new QVBoxLayout;
00366  //dropletParamsLayout->addLayout(numDropLayout);
00367  //dropletParamsLayout->addLayout(dropRadLayout);
00368  //dropletParamsLayout->addLayout(dropProgramsLayout);
00369  dropletParamsLayout->addWidget(dropletTableWidget);
00370  dropletParams->setLayout(dropletParamsLayout);
00371  dropletParams->setFrameStyle(QFrame::Panel);
00372  dropletParams->setLineWidth(1);
00373 }
00374
00375 void MainWindow::addArenaWidgets()
00376 {
00377  // Floor Files
00378  arenaSelectionCombo = new QComboBox;
00379  arenaSelectionList = new QListView(arenaSelectionCombo);
00380  arenaSelectionCombo->addItem("Default (Rectangle)");
00381
00382  arenaSelectionList->setStyleSheet("QListView::item {  \
00383            border-bottom: 5px solid white; margin:3px; } \
00384            QListView::item:selected { \
00385            border-bottom: 5px solid black; margin:3px; \
00386            color:black; \
00387            } \
00388            ");
00389  arenaSelectionCombo->setView(arenaSelectionList);
00390  arenaSelectionLabel = new QLabel(tr("Arena Selection"));
```

```
00391  arenaSelectionLabel->setBuddy(arenaSelectionCombo);
00392  arenaSelectionLayout = new QVBoxLayout;
00393  arenaSelectionLayout->addWidget(arenaSelectionLabel);
00394  arenaSelectionLayout->addWidget(arenaSelectionCombo);
00395
00396  QObject::connect(arenaSelectionCombo, SIGNAL(currentIndexChanged(const QString &)),this
      ,SLOT(showHideRowColWidget(const QString &)));
00397  QObject::connect(arenaSelectionCombo, SIGNAL(currentIndexChanged(const QString &)),&
      _simInterface,SLOT(loadTilePositions(const QString &)));
00398
00399  rowColLayout = new QVBoxLayout;
00400  rowColWidget = new QWidget;
00401
00402  // Number of Columns
00403  numColBox = new QSpinBox;
00404  numColBox->setRange(1,10);
00405  numColBox->setValue(DEFAULT_COL_TILES);
00406  numColLabel = new QLabel(tr("Number of columns in arena: "));
00407  numColLabel->setBuddy(numColBox);
00408  numColLayout = new QHBoxLayout;
00409  numColLayout->addWidget(numColLabel);
00410  numColLayout->addWidget(numColBox);
00411
00412  // Number of Rows
00413  numRowBox = new QSpinBox;
00414  numRowBox->setRange(1,10);
00415  numRowBox->setValue(DEFAULT_ROW_TILES);
00416  numRowLabel = new QLabel(tr("Number of rows in arena: "));
00417  numRowLabel->setBuddy(numRowBox);
00418  numRowLayout = new QHBoxLayout;
00419  numRowLayout->addWidget(numRowLabel);
00420  numRowLayout->addWidget(numRowBox);
00421
00422  rowColLayout->addLayout(numColLayout);
00423  rowColLayout->addLayout(numRowLayout);
00424  rowColWidget->setLayout(rowColLayout);
00425
00426  // Projector Images
00427  projectorImageCombo = new QComboBox;
00428  projectorImageList = new QListView(projectorImageCombo);
00429  projectorImageCombo->addItem("None");
00430
00431  projectorImageList->setStyleSheet("QListView::item {  \
00432            border-bottom: 5px solid white; margin:3px; } \
00433            QListView::item:selected { \
00434            border-bottom: 5px solid black; margin:3px; \
00435            color:black; \
00436            } \
00437            ");
00438  projectorImageCombo->setView(projectorImageList);
00439  projectorImageLabel = new QLabel(tr("Projection Images"));
00440  projectorImageLabel->setBuddy(projectorImageCombo);
00441  projectorImagesLayout = new QVBoxLayout;
00442  projectorImagesLayout->addWidget(projectorImageLabel);
00443  projectorImagesLayout->addWidget(projectorImageCombo);
00444
00445  arenaParams = new QFrame();
00446  arenaParams->setWindowTitle(tr("Arena Parameters"));
00447  QVBoxLayout *arenaParamsLayout = new QVBoxLayout();
00448  arenaParamsLayout->addLayout(arenaSelectionLayout);
00449  arenaParamsLayout->addWidget(rowColWidget);
00450  arenaParamsLayout->addLayout(projectorImagesLayout);
00451  arenaParams->setLayout(arenaParamsLayout);
00452  arenaParams->setFrameStyle(QFrame::Panel);
00453  arenaParams->setLineWidth(1);
00454 }
00455
00456 void MainWindow::addButtonWidgets()
00457 {
00458  // Launch Buttons
00459  launchSimulation = new QPushButton(tr("Update Simulator"));
00460  connect(launchSimulation,SIGNAL(clicked()),this,SLOT(
      launchSimulator()));
00461  launchRenderWidget = new QPushButton(tr("View Simulation"));
00462  QObject::connect(launchRenderWidget,SIGNAL(clicked()),this,SLOT(
      launchRenderer()));
00463  QHBoxLayout *launchButtons = new QHBoxLayout;
00464  launchButtons->addWidget(launchSimulation);
00465  launchButtons->addWidget(launchRenderWidget);
00466
00467  // Simulation Interation Buttons
00468  pause_button = new QPushButton("Pause");
00469  resume_button = new QPushButton("Resume");
00470  reset_button = new QPushButton("Reset");
00471  QHBoxLayout *playbackButtons = new QHBoxLayout;
00472  playbackButtons->addWidget(pause_button);
00473  playbackButtons->addWidget(resume_button);
```

```
00474  playbackButtons->addWidget(reset_button);
00475
00476  logCheckBox = new QCheckBox("Log Droplet Infomation");
00477  posCheckBox = new QCheckBox("Position");
00478  colCheckBox = new QCheckBox("Color");
00479  rotCheckBox = new QCheckBox("Rotation");
00480  commSACheckBox = new QCheckBox("CommSA");
00481  macroRedCheckBox = new QCheckBox("MacroRed");
00482  macroSACheckBox = new QCheckBox("macroSA");
00483  logLayout = new QVBoxLayout;
00484  microLayout = new QHBoxLayout;
00485  macroLayout = new QHBoxLayout;
00486  microLayout->addWidget(posCheckBox);
00487  microLayout->addWidget(colCheckBox);
00488  microLayout->addWidget(rotCheckBox);
00489  microLayout->addWidget(commSACheckBox);
00490  macroLayout->addWidget(macroRedCheckBox);
00491  macroLayout->addWidget(macroSACheckBox);
00492  logLayout->addLayout(microLayout);
00493  logLayout->addLayout(macroLayout);
00494  logWidget = new QWidget;
00495  logWidget->setLayout(logLayout);
00496  connect(logCheckBox, SIGNAL(stateChanged(int)), this, SLOT(
     showHideLogWidget(int)));
00497
00498  buttons = new QFrame();
00499  buttons->setWindowTitle(tr("Launch/Playback Buttons"));
00500  QVBoxLayout *buttonsLayout = new QVBoxLayout();
00501
00502
00503  buttonsLayout->addWidget(logCheckBox);
00504  buttonsLayout->addWidget(logWidget);
00505  logWidget->hide();
00506  buttonsLayout->addLayout(launchButtons);
00507  buttonsLayout->addLayout(playbackButtons);
00508  buttons->setLayout(buttonsLayout);
00509  buttons->setFrameStyle(QFrame::Panel);
00510  buttons->setLineWidth(1);
00511 }
00512
00513 void MainWindow::setUpGUI()
00514 {
00515  addLoadSetupFileWidgets();
00516  addDropletWidgets();
00517  addArenaWidgets();
00518  addButtonWidgets();
00519
00520  // Combine to main window
00521  QWidget *window = new QWidget;
00522  QVBoxLayout *mainLayout = new QVBoxLayout(window);
00523  QLabel *loadSetupFileLabel = new QLabel("Load Setup File");
00524  mainLayout->addWidget(loadSetupFileLabel);
00525  mainLayout->addWidget(loadSetupFile);
00526  QLabel *dropletParamsLabel = new QLabel("Droplet Parameters");
00527  mainLayout->addWidget(dropletParamsLabel);
00528  mainLayout->addWidget(dropletParams);
00529  QLabel *arenaParamsLabel = new QLabel("Arena Parameters");
00530  mainLayout->addWidget(arenaParamsLabel);
00531  mainLayout->addWidget(arenaParams);
00532  QLabel *buttonsLabel = new QLabel("Launch and Playback");
00533  mainLayout->addWidget(buttonsLabel);
00534  mainLayout->addWidget(buttons);
00535  window->setLayout(mainLayout);
00536  setCentralWidget(window);
00537  setWindowTitle(tr("Droplet GUI"));
00538
00539  connect(pause_button,SIGNAL(released()),&_simInterface,SLOT(pause()));
00540  connect(resume_button,SIGNAL(released()),&_simInterface,SLOT(resume()));
00541  connect(reset_button,SIGNAL(released()),&_simInterface,SLOT(reset()));
00542
00543  // only load these after the GUI is set up
00544  loadSetupFiles(QString(DEFAULT_ASSETDIR).append(
     DEFAULT_SETUPDIR));
00545  loadFloorFiles(QString(DEFAULT_ASSETDIR).append(
     DEFAULT_FLOORDIR));
00546  loadProjectionTextures(QString(DEFAULT_ASSETDIR).append(
     DEFAULT_PROJECTDIR));
00547 }
00548
00549
00550 void MainWindow::removeRenderer(QObject *obj)
00551 {
00552  if (obj != NULL)
00553  {
00554   RenderWidget *renderer = (RenderWidget *) obj;
00555
00556   if (_renderWidgets.contains(renderer))
```

```
00557   {
00558    _renderWidgets.removeOne(renderer);
00559    qDebug() << QString("Removed RenderWidget");
00560   }
00561   //delete renderer;
00562  }
00563
00564  if (_renderWidgets.count() == 0)
00565   emit disableLimit();
00566 }
00567
00568
00569 void MainWindow::closeEvent(QCloseEvent *event)
00570 {
00571  // use the closing signal to signal all child objects to close/quit
00572  emit closing();
00573  qApp->processEvents();
00574
00575  // spinlock until child thread is done
00576  while(_simThread.isRunning()) {};
00577  event->accept();
00578 }
00579
00580
00581 void MainWindow::loadProjectionTextures(QString path)
00582 {
00583  qDebug() << "Refreshing texture list";
00584  QString currentTexture = projectorImageCombo->currentText();
00585
00586  QDir inDir(path);
00587  QStringList filters;
00588  filters << "*.bmp";
00589  inDir.setNameFilters(filters);
00590
00591  _projectionTextures = inDir.entryList();
00592  for(QStringList::iterator i = _projectionTextures.begin() ; i !=
     _projectionTextures.end();i++)
00593  {
00594   i->remove(".bmp");
00595  }
00596  projectorImageCombo->clear();
00597  projectorImageCombo->addItem("None");
00598  projectorImageCombo->addItems(_projectionTextures);
00599  int index = projectorImageCombo->findText(currentTexture,Qt::MatchFixedString);
00600  if (index == -1)
00601   index = 0;
00602  projectorImageCombo->setCurrentIndex(index);
00603 }
00604
00605 void MainWindow::loadFloorFiles(QString path)
00606 {
00607  qDebug() << "Refreshing file list";
00608  QString currentArena = arenaSelectionCombo->currentText();
00609
00610  QDir inDir(path);
00611  QStringList filters;
00612  filters << "*.txt";
00613  inDir.setNameFilters(filters);
00614
00615  _floorFiles = inDir.entryList();
00616  for(QStringList::iterator i = _floorFiles.begin() ; i != _floorFiles.end();i++)
00617  {
00618   i->remove(".txt");
00619  }
00620
00621  arenaSelectionCombo->clear();
00622  arenaSelectionCombo->addItem("Default (Rectangle)");
00623  arenaSelectionCombo->addItems(_floorFiles);
00624  int index = arenaSelectionCombo->findText(currentArena,Qt::MatchFixedString);
00625  if (index < 0)
00626   index = 0;
00627  arenaSelectionCombo->setCurrentIndex(index);
00628 }
00629
00630 void MainWindow::loadSetupFiles(QString path)
00631 {
00632  QString currentSetup = loadSetupFileCombo->currentText();
00633
00634  QDir inDir(path);
00635  QStringList filters;
00636  filters << "*.txt";
00637  inDir.setNameFilters(filters);
00638
00639  _setupFiles = inDir.entryList();
00640  for(QStringList::iterator i = _setupFiles.begin() ; i != _setupFiles.end();i++)
00641  {
00642   QString temp = path;
```

```
00643   temp.append(*i);
00644   qDebug() << "Added file" << temp << "to watch list";
00645   setupWatcher.addPath(temp);
00646   i->remove(".txt");
00647  }
00648
00649  loadSetupFileCombo->clear();
00650  loadSetupFileCombo->addItem(QString("None"));
00651  loadSetupFileCombo->addItems(_setupFiles);
00652  int index = loadSetupFileCombo->findText(currentSetup,Qt::MatchFixedString);
00653  if (index < 0)
00654   index = 0;
00655  loadSetupFileCombo->setCurrentIndex(index);
00656  updateParams(currentSetup);
00657 }
00658
00659 void MainWindow::setUI(simSetting_t settings)
00660 {
00661  //dropRadBox->setValue(settings.dropletRadius);
00662  numColBox->setValue(settings.numColTiles);
00663  numRowBox->setValue(settings.numRowTiles);
00664
00665  int index = 0;
00666  if (settings.projecting == true)
00667  {
00668   QString temp = settings.projTexture;
00669   temp.remove(".bmp");
00670   index = projectorImageCombo->findText(temp,Qt::MatchFixedString);
00671   if (index == -1)
00672    index = 0;
00673  }
00674
00675
00676  projectorImageCombo->setCurrentIndex(index);
00677
00678  // if an arena was specified set the UI for it
00679  QString arenaName = settings.arenaName.remove(".txt");
00680  index = arenaSelectionCombo->findText(arenaName,Qt::MatchFixedString);
00681  if (index == -1)
00682   index = 0;
00683  arenaSelectionCombo->setCurrentIndex(index);
00684
00685  index = 0;
00686  int numDroplets = 0;
00687
00688
00689  // since there is no place to store this info, it simply get stored elsewhere for later
00690  /*
00691  if (settings.startingDroplets.count() > 0)
00692  {
00693   QStringList droplets = settings.startingDroplets[0];
00694   if (droplets.count() == 2)
00695   {
00696    numDroplets = droplets[1].toInt();
00697    index = dropProgramsCombo->findText(droplets[0],Qt::MatchFixedString);
00698    if (index == -1)
00699     index = 0;
00700   }
00701  }
00702  */
00703  //numDropBox->setValue(numDroplets);
00704  //dropProgramsCombo->setCurrentIndex(index);
00705  foreach(QStringList list, settings.startingDroplets)
00706  {
00707   int addition;
00708   if(list.count() == 2)
00709   {
00710    addition = list[1].toInt();
00711   }
00712   if(list.count() == 3)
00713   {
00714    addition = 1;
00715   }
00716   QString type = list[0].toLower();
00717   if (type == QString("march"))
00718   {
00719    int temp = dropletTableWidget->item(0,0)->text().toInt()  + addition;
00720    QString string = QString::number(temp);
00721    dropletTableWidget->setItem(0,0,new QTableWidgetItem(string));
00722   } else if (type == QString("rainbow"))
00723   {
00724    int temp = dropletTableWidget->item(1,0)->text().toInt()  + addition;
00725    QString string = QString::number(temp);
00726    dropletTableWidget->setItem(1,0,new QTableWidgetItem(string));
00727   } else if (type == QString("randomwalk"))
00728   {
00729    int temp = dropletTableWidget->item(2,0)->text().toInt()  + addition;
```

```
00730    QString string = QString::number(temp);
00731    dropletTableWidget->setItem(2,0,new QTableWidgetItem(string));
00732   } else if (type == QString("rgbsense"))
00733   {
00734    int temp = dropletTableWidget->item(3,0)->text().toInt()  + addition;
00735    QString string = QString::number(temp);
00736    dropletTableWidget->setItem(3,0,new QTableWidgetItem(string));
00737   } else if (type == QString("stickpullers"))
00738   {
00739    int temp = dropletTableWidget->item(4,0)->text().toInt()  + addition;
00740    QString string = QString::number(temp);
00741    dropletTableWidget->setItem(4,0,new QTableWidgetItem(string));
00742   } else if (type == QString("turntest"))
00743   {
00744    int temp = dropletTableWidget->item(5,0)->text().toInt()  + addition;
00745    QString string = QString::number(temp);
00746    dropletTableWidget->setItem(5,0,new QTableWidgetItem(string));
00747   } else if (type == QString("commtest"))
00748   {
00749    int temp = dropletTableWidget->item(6,0)->text().toInt()  + addition;
00750    QString string = QString::number(temp);
00751    dropletTableWidget->setItem(6,0,new QTableWidgetItem(string));
00752   } else if (type == QString("powertest"))
00753   {
00754    int temp = dropletTableWidget->item(7,0)->text().toInt()  + addition;
00755    QString string = QString::number(temp);
00756    dropletTableWidget->setItem(7,0,new QTableWidgetItem(string));
00757   } else if (type == QString("granola"))
00758   {
00759    int temp = dropletTableWidget->item(8,0)->text().toInt()  + addition;
00760    QString string = QString::number(temp);
00761    dropletTableWidget->setItem(8,0,new QTableWidgetItem(string));
00762   } else if (type == QString("stickpullersupdated"))
00763   {
00764    int temp = dropletTableWidget->item(9,0)->text().toInt()  + addition;
00765    QString string = QString::number(temp);
00766    dropletTableWidget->setItem(9,0,new QTableWidgetItem(string));
00767   } else if (type == QString("ants"))
00768   {
00769    int temp = dropletTableWidget->item(10,0)->text().toInt()  + addition;
00770    QString string = QString::number(temp);
00771    dropletTableWidget->setItem(10,0,new QTableWidgetItem(string));
00772   } else if (type == QString("customone"))
00773   {
00774    int temp = dropletTableWidget->item(11,0)->text().toInt()  + addition;
00775    QString string = QString::number(temp);
00776    dropletTableWidget->setItem(11,0,new QTableWidgetItem(string));
00777   } else if (type == QString("customtwo"))
00778   {
00779    int temp = dropletTableWidget->item(12,0)->text().toInt()  + addition;
00780    QString string = QString::number(temp);
00781    dropletTableWidget->setItem(12,0,new QTableWidgetItem(string));
00782   } else if (type == QString("customthree"))
00783   {
00784    int temp = dropletTableWidget->item(13,0)->text().toInt()  + addition;
00785    QString string = QString::number(temp);
00786    dropletTableWidget->setItem(13,0,new QTableWidgetItem(string));
00787   } else if (type == QString("customfour"))
00788   {
00789    int temp = dropletTableWidget->item(14,0)->text().toInt()  + addition;
00790    QString string = QString::number(temp);
00791    dropletTableWidget->setItem(14,0,new QTableWidgetItem(string));
00792   } else if (type == QString("customfive"))
00793   {
00794    int temp = dropletTableWidget->item(15,0)->text().toInt()  + addition;
00795    QString string = QString::number(temp);
00796    dropletTableWidget->setItem(15,0,new QTableWidgetItem(string));
00797   } else if (type == QString("customsix"))
00798   {
00799    int temp = dropletTableWidget->item(16,0)->text().toInt()  + addition;
00800    QString string = QString::number(temp);
00801    dropletTableWidget->setItem(16,0,new QTableWidgetItem(string));
00802   } else if (type == QString("customseven"))
00803   {
00804    int temp = dropletTableWidget->item(17,0)->text().toInt()  + addition;
00805    QString string = QString::number(temp);
00806    dropletTableWidget->setItem(17,0,new QTableWidgetItem(string));
00807   } else if (type == QString("customeight"))
00808   {
00809    int temp = dropletTableWidget->item(18,0)->text().toInt()  + addition;
00810    QString string = QString::number(temp);
00811    dropletTableWidget->setItem(18,0,new QTableWidgetItem(string));
00812   } else if (type == QString("customnine"))
00813   {
00814    int temp = dropletTableWidget->item(19,0)->text().toInt()  + addition;
00815    QString string = QString::number(temp);
00816    dropletTableWidget->setItem(19,0,new QTableWidgetItem(string));
```

```
00817   } else if (type == QString("customten"))
00818   {
00819    int temp = dropletTableWidget->item(20,0)->text().toInt()  + addition;
00820    QString string = QString::number(temp);
00821    dropletTableWidget->setItem(20,0,new QTableWidgetItem(string));
00822   }
00823
00824  }
00825
00826 }
00827
00828 bool MainWindow::saveToFile(QString filename, simSetting_t settings)
00829 {
00830
00831  QFile file(filename);
00832  if (file.open(QIODevice ::WriteOnly | QIODevice ::Text | QIODevice::Truncate))
00833  {
00834   simSetting_t defaults = _simInterface.
      getDefaultSettings();
00835   QTextStream out(&file);
00836   out << "col_tiles " << settings.numColTiles << "\n";
00837   out << "row_tiles " << settings.numRowTiles << "\n";
00838   out << "droplet_radius " << settings.dropletRadius << "\n";
00839   if (settings.projecting == true && settings.projTexture != QString("None"))
00840   {
00841    out << "projecting " << settings.projTexture << "\n";
00842
00843   }
00844   foreach(QStringList droplet,settings.startingDroplets)
00845   {
00846    out << "droplets";
00847    foreach(QString string,droplet)
00848    {
00849     out << " " << string;
00850    }
00851    out << "\n";
00852   }
00853   foreach(QStringList object,settings.startingObjects)
00854   {
00855    foreach(QString string,object)
00856    {
00857     out << string << " ";
00858    }
00859   }
00860   file.close();
00861   return true;
00862  }
00863  return false;
00864 }
00865
00866 bool MainWindow::loadFromFile(QString filename,
      simSetting_t &settings)
00867 {
00868
00869  for(int i = 0; i < dropletTableWidget->rowCount(); i++)
00870  {
00871   dropletTableWidget->setItem(i, 0, new QTableWidgetItem("0"));
00872
00873  }
00874
00875  QFile file(filename);
00876
00877  simSetting_t newSettings;
00878
00879  newSettings = _simInterface.getDefaultSettings();
00880
00881  if (file.exists())
00882  {
00883   if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
00884   {
00885    QStringList objects;
00886    objects << "cube" << "cubes" << "sphere" << "spheres";
00887    QTextStream in(&file);
00888    while (!in.atEnd())
00889    {
00890     QString line = in.readLine(0);
00891     if (line.count() > 0)
00892     {
00893      if (!line.startsWith("#"))
00894      {
00895       QStringList list = line.split(" ",QString::SkipEmptyParts);
00896       if (list.count() > 0)
00897       {
00898        list[0] = list[0].toLower();
00899        if (objects.contains(list[0]))
00900        {
00901         newSettings.startingObjects.append(list);
```

```
00902         }  else if (list.size() == 2)
00903         {
00904
00905          if (list[0] == QString("droplets") || list[0] == QString("droplet"))
00906          {
00907           QStringList dropletsToAdd;
00908           dropletsToAdd.append(list[1]);
00909           dropletsToAdd.append(QString("1"));
00910           newSettings.startingDroplets.append(dropletsToAdd);
00911          } else if (list[0] == QString("col_tiles"))
00912          {
00913           int numColTiles = list[1].toInt();
00914           newSettings.numColTiles = numColTiles;
00915          } else if (list[0] == QString("row_tiles"))
00916          {
00917           int numRowTiles = list[1].toInt();
00918           newSettings.numRowTiles = numRowTiles;
00919          } else if (list[0] == QString("fps"))
00920          {
00921           int newFPS = list[1].toInt();
00922           newSettings.fps = newFPS;
00923          } else if (list[0] == QString("droplet_radius"))
00924          {
00925           float newRadius = list[1].toFloat();
00926           newSettings.dropletRadius = newRadius;
00927          } else if (list[0] == QString("wall_height"))
00928          {
00929           float newHeight = list[1].toFloat();
00930           newSettings.wallHeight = newHeight;
00931          } else if (list[0] == QString("wall_width"))
00932          {
00933           float newWidth = list[1].toFloat();
00934           newSettings.wallWidth = newWidth;
00935          } else if (list[0] == QString("projecting"))
00936          {
00937           QString filePath = QString(DEFAULT_ASSETDIR).append(
      DEFAULT_PROJECTDIR).append(list[1]);
00938           QFileInfo file(filePath);
00939           if (file.exists())
00940           {
00941            newSettings.projecting = true;
00942            newSettings.projTexture = QString(list[1]);
00943           } else {
00944            newSettings.projecting = false;
00945            newSettings.projTexture = QString("none");
00946           }
00947          } else if (list[0] == QString("arena"))
00948          {
00949           newSettings.arenaName = list[1];
00950          } else {
00951           qDebug() << "Error: unrecognized line" << line;
00952          }
00953         }  else if (list.count() >= 3)
00954         {
00955          if (list[0] == QString("droplets"))
00956          {
00957           QStringList dropletsToAdd;
00958           for(int i = 1; i < list.count(); i++)
00959           {
00960            dropletsToAdd.append(list[i]);
00961
00962           }
00963           qDebug() << dropletsToAdd;
00964           newSettings.startingDroplets.append(dropletsToAdd);
00965          }
00966         } else
00967         {
00968          qDebug() << "Error: malformed line" << line;
00969         }
00970
00971
00972        } else {
00973         // push the comment to the debug console
00974         qDebug() << line;
00975        }
00976       }
00977      }
00978     }
00979
00980
00981    } else
00982    {
00983     qDebug() << "Error: " << file.error();
00984     file.close();
00985     return false;
00986    }
00987    file.close();
```

```
00988   settings = newSettings;
00989   return true;
00990  } else
00991  {
00992   qDebug() << "Error: file does not exist - using compiled defaults";
00993  }
00994  return false;
00995 }
00996
00997 void MainWindow::showHideLogWidget(int state)
00998 {
00999  if (state == 2) {
01000   logWidget->show();
01001  }
01002  else {
01003   logWidget->hide();
01004  }
01005 }
01006
01007
01008
01009 void MainWindow::showHideRowColWidget(const QString & text)
01010 {
01011  if(text.operator==("Default (Rectangle)"))
01012  {
01013   rowColWidget->show();
01014  }
01015  else
01016  {
01017   rowColWidget->hide();
01018  }
01019 }
01020
01021 void MainWindow::updateSetupFile(const QString &file)
01022 {
01023
01024  QString fileName = file.section('\\', -1);
01025  if (fileName == file)
01026   fileName = file.section('/', -1);
01027
01028  fileName = fileName.remove(".txt");
01029  qDebug() << "File" << fileName << "changed";
01030  if (loadSetupFileCombo->currentText() == fileName)
01031   updateParams(fileName);
01032 }
01033 void MainWindow::updateParams(const QString & text)
01034 {
01035  fileName = QString(DEFAULT_ASSETDIR).append(
      DEFAULT_SETUPDIR).append(text).append(".txt");
01036  simSetting_t settings = _simInterface.
      getDefaultSettings();
01037  if (loadFromFile(fileName,settings))
01038  {
01039   qDebug() << "Loaded settings out of file" << fileName;
01040  }
01041
01042  // TODO: update UI so these are propagated without need for hack
01043  _arenaObjects.droplets.clear();
01044  _arenaObjects.objects.clear();
01045  _arenaObjects.droplets = settings.startingDroplets;
01046  _arenaObjects.objects = settings.startingObjects;
01047
01048  setUI(settings);
01049 }
01050
01051 void MainWindow::enableDisableDropletTable(const QString & text)
01052 {
01053  if(text.operator!=("None"))
01054  {
01055   dropletTableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);
01056  }
01057  else
01058  {
01059   dropletTableWidget->setEditTriggers(QAbstractItemView::AllEditTriggers);
01060  }
01061
01062 }
```

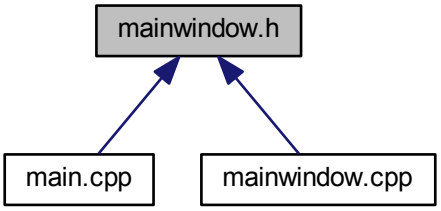
13.16 mainwindow.h File Reference

Declares the MainWindow class.

```
#include <QMainWindow>
#include "RenderWidget.h"
#include "SimInterface.h"
#include <QFileSystemWatcher>
#include <QStringList>
#include <QString>
#include <QSpinBox>
#include <QSlider>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>
#include <QComboBox>
#include <QListView>
#include <QGroupBox>
#include <QLinkedList>
#include <QCloseEvent>
#include <QShowEvent>
#include <QCheckBox>
#include <QDesktopWidget>
#include <QMetaObject>
#include <QThread>
#include <QTableWidget>
#include "simInfoLogger.h"
#include "defaults.h"
```

Include dependency graph for mainwindow.h:

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



Classes

- class MainWindow

  View controller for the main window.

Macros

- #define MAINWINDOW_H

A macro that defines mainwindow h.

13.16.1 Detailed Description

Declares the MainWindow class.

Definition in file mainwindow.h.

13.16.2 Macro Definition Documentation

13.16.2.1 #define MAINWINDOW_H

A macro that defines mainwindow h.

Definition at line 16 of file mainwindow.h.

13.17 mainwindow.h

```
00001
00007 #ifndef MAINWINDOW_H
00008
00016 #define MAINWINDOW_H
00017
00018 #include <QMainWindow>
00019 //#include "ui_mainwindow.h"
00020 #include "RenderWidget.h"
00021 #include "SimInterface.h"
00022 #include <QFileSystemWatcher>
00023 #include <QStringList>
00024 #include <QString>
00025 #include <QSpinBox>
00026 #include <QSlider>
00027 #include <QHBoxLayout>
00028 #include <QLabel>
00029 #include <QPushButton>
00030 #include <QComboBox>
00031 #include <QListView>
00032 #include <QGroupBox>
00033 #include <QLinkedList>
00034 #include <QCloseEvent>
00035 #include <QShowEvent>
00036 #include <QCheckBox>
00037 #include <QDesktopWidget>
00038 #include <QMetaObject>
00039 #include <QThread>
00040 #include <QTableWidget>
00041 #include "simInfoLogger.h"
00042 #include "defaults.h"
00055 class MainWindow : public QMainWindow
00056 {
00057  Q_OBJECT
00058
00059 public:
00060
00069  MainWindow(QWidget *parent = 0);
00070  ~MainWindow();
00071
00072 public slots:
00073
00083  void loadProjectionTextures(QString path);
00084
00094  void loadFloorFiles(QString path);
00095
00103  void loadSetupFiles(QString path);
00104
00112  void updateParams(const QString & text);
00113
00123  void updateSetupFile(const QString &file);
00124
00136  bool loadFromFile(QString filename, simSetting_t &settings);
00137
00149  bool saveToFile(QString filename, simSetting_t settings);
00150
00159  void setUI(simSetting_t settings);
00160
00169  void showHideRowColWidget(const QString & text);
00170
```

```
00179  void showHideLogWidget(int state);
00180
00181  void enableDisableDropletTable(const QString &);
00182
00183 protected:
00184
00193  void closeEvent(QCloseEvent *event);
00194
00208  bool eventFilter(QObject *watched, QEvent *event);
00209
00210 private:
00211  //Ui::MainWindow ui;
00212
00217  SimInterface _simInterface;
00218
00223  QLinkedList<RenderWidget *> _renderWidgets;
00224
00230  QStringList _projectionTextures;
00231  QStringList _floorFiles;
00232  QStringList _setupFiles;
00233
00234
00240  bool _simulatorRunning;
00241  QThread _simThread;
00242
00248  simInfoLogger _logger;
00249
00254  QString fileName;
00255
00261  QSpinBox *numDropBox;
00262  QLabel *numDropLabel;
00263  QHBoxLayout *numDropLayout;
00264
00270  QSpinBox *numColBox;
00271  QLabel *numColLabel;
00272  QHBoxLayout *numColLayout;
00273
00279  QSpinBox *numRowBox;
00280  QLabel *numRowLabel;
00281  QHBoxLayout *numRowLayout;
00282
00288  QLabel *loadSetupFileLabel;
00289  QComboBox *loadSetupFileCombo;
00290  QListView *loadSetupFileList;
00291  QVBoxLayout *loadSetupFileLayout;
00292
00298  QLabel *arenaSelectionLabel;
00299  QComboBox *arenaSelectionCombo;
00300  QListView *arenaSelectionList;
00301  QVBoxLayout *arenaSelectionLayout;
00302
00308  QVBoxLayout *rowColLayout;
00309  QWidget *rowColWidget;
00310
00311
00312
00318  QDoubleSpinBox *dropRadBox;
00319  QLabel *dropRadLabel;
00320  QHBoxLayout *dropRadLayout;
00321
00327  QLabel *dropProgramsLabel;
00328  QComboBox *dropProgramsCombo;
00329  QListView *dropProgramsList;
00330  QVBoxLayout *dropProgramsLayout;
00331  QTableWidget *dropletTableWidget;
00332
00338  QLabel *projectorImageLabel;
00339  QComboBox *projectorImageCombo;
00340  QListView *projectorImageList;
00341  QVBoxLayout *projectorImagesLayout;
00342
00348  QPushButton *launchSimulation;
00349  QPushButton *launchRenderWidget;
00350
00356  QCheckBox *logCheckBox;
00357  QCheckBox *posCheckBox;
00358  QCheckBox *colCheckBox;
00359  QCheckBox *rotCheckBox;
00360  QCheckBox *commSACheckBox;
00361  QCheckBox *macroRedCheckBox;
00362  QCheckBox *macroSACheckBox;
00363
00369  QVBoxLayout *logLayout;
00370  QHBoxLayout *microLayout;
00371  QHBoxLayout *macroLayout;
00372  QWidget *logWidget;
00373
```

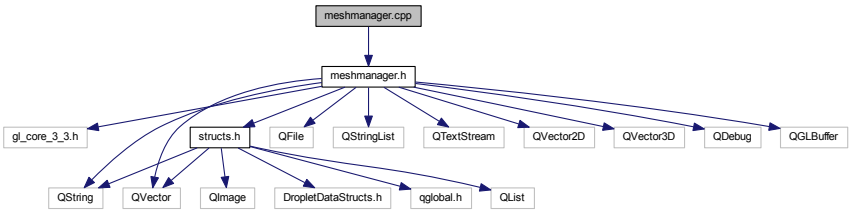
```
00379  QPushButton *pause_button;
00380  QPushButton *resume_button;
00381  QPushButton *reset_button;
00382
00387  QFrame *loadSetupFile;
00388  QFrame *dropletParams;
00389  QFrame *arenaParams;
00390  QFrame *buttons;
00391
00392  QFileSystemWatcher projectionWatcher;
00393  QFileSystemWatcher arenaWatcher;
00394  QFileSystemWatcher setupWatcher;
00395
00402  void addLoadSetupFileWidgets();
00410  void addDropletWidgets();
00411
00419  void addArenaWidgets();
00420
00427  void addButtonWidgets();
00428
00436  void setUpGUI();
00437
00438
00439  // temporary fix
00440  // TODO: make thsi better
00441  struct {
00442   QVector<QStringList> droplets;
00443   QVector<QStringList> objects;
00444  } _arenaObjects;
00445 private slots:
00446
00455  void launchRenderer();
00456
00466  void launchSimulator();
00467
00476  void removeRenderer(QObject *obj = 0);
00477
00478 signals:
00479
00487  void launchSim(simSetting_t);
00488
00495  void closing();
00496
00503  void enableLimit(void);
00504
00511  void disableLimit(void);
00512
00520  void setUpdateRate(float);
00521
00522
00523 };
00524
00525 #endif // MAINWINDOW_H
```

13.18 meshmanager.cpp File Reference

Implements the MeshManager class.

`#include "meshmanager.h"`

Include dependency graph for meshmanager.cpp:



13.18.1 Detailed Description

Implements the MeshManager class.

Definition in file meshmanager.cpp.

13.19 meshmanager.cpp

```
00001
00007 #include "meshmanager.h"
00008
00009 MeshManager::MeshManager()
00010 {
00011  _hasIndex = false;
00012  _hasNormals = false;
00013  _hasTexCoords = false;
00014  _modelLoaded = false;
00015  _fileSet = false;
00016  setAttribLoc(0,1,2);
00017  _indexCount = 0;
00018  _indexedVertexCount = 0;
00019  _vertexCount = 0;
00020
00021  _VBOID = 0;
00022  _indexBufferID = 0;
00023  ogl_LoadFunctions();
00024 }
00025
00026 MeshManager::~MeshManager()
00027 {
00028  freeModel();
00029 }
00030
00031 bool MeshManager::setFile(QString fileName)
00032 {
00033
00034  QFile file(fileName);
00035  if (file.exists())
00036  {
00037   freeModel();
00038   _fileName = fileName;
00039   _fileSet = true;
00040   return true;
00041  } else
00042  {
00043   return false;
00044  }
00045 }
00046 bool MeshManager::loadFile(QString fileName)
00047 {
00048  if (setFile(fileName))
00049  {
00050   QString inLine;
00051   int n;
00052   QVector<int> vertexIndices, uvIndices, normalIndices;
00053   QVector<QVector3D> temp_vertices;
00054   QVector<QVector2D> temp_uvs;
00055   QVector<QVector3D> temp_normals;
00056   vertexData_t *vertexData = NULL;
00057   vertexData_t *indexedVertexData = NULL;
00058   GLuint *arrayIndex = NULL;
00059
00060   QFile file(fileName);
00061
00062   if (file.exists())
00063   {
00064    if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
00065    {
00066     QTextStream in(&file);
00067     while (!in.atEnd())
00068     {
00069      inLine = in.readLine(0);
00070      if(inLine.length() == 0)
00071       continue;
00072      // qDebug() << inLine;
00073
00074      QStringList list = inLine.split(" ",QString::SkipEmptyParts);
00075
00076      //read in vertex texture coordinates
00077      if (list[0] == QString("vt") && list.size() == 3)
00078      {
00079       QVector2D uv;
00080       uv.setX(list[1].toFloat());
```

```
00081       uv.setY(list[2].toFloat());
00082       // qDebug() << uv;
00083       temp_uvs.push_back(uv);
00084      }
00085      //read in vertex normals
00086      else if(list[0] == QString("vn") && list.size() == 4)
00087      {
00088       QVector3D normal;
00089       normal.setX(list[1].toFloat());
00090       normal.setY(list[2].toFloat());
00091       normal.setZ(list[3].toFloat());
00092       // qDebug() << normal;
00093       temp_normals.push_back(normal);
00094      }
00095      //read in vertices
00096      else if(list[0] == QString("v") && list.size() == 4)
00097      {
00098       QVector3D vertex;
00099       vertex.setX(list[1].toFloat());
00100       vertex.setY(list[2].toFloat());
00101       vertex.setZ(list[3].toFloat());
00102       // qDebug() << vertex;
00103       temp_vertices.push_back(vertex);
00104      }
00105      //obj files include face information, the following parses through
00106      //three vertices at a time (for each triangle) storing indices
00107      else if(list[0] == QString("f"))
00108      {
00109       for (int i = 1 ; i < list.size() ; i++)
00110       {
00111        QStringList face = list[i].split("/");
00112        int vertexIndex, uvIndex, normalIndex;
00113
00114        if (face.size() >= 1)
00115        {
00116
00117         vertexIndex = face[0].toInt();
00118         vertexIndices.push_back(vertexIndex);
00119
00120        }
00121        if (face.size() >= 2)
00122        {
00123
00124         uvIndex = face[1].toInt();
00125         uvIndices.push_back(uvIndex);
00126
00127        }
00128        if (face.size() == 3)
00129        {
00130
00131         normalIndex = face[2].toInt();
00132         normalIndices.push_back(normalIndex);
00133        }
00134       }
00135      }
00136     }
00137
00138     file.close();
00139
00140     if (uvIndices.size() == vertexIndices.size())
00141     {
00142      _hasTexCoords = true;
00143      //qDebug() << "Mesh has texture coordinates";
00144     }
00145
00146     if (normalIndices.size() == vertexIndices.size())
00147     {
00148      _hasNormals = true;
00149      //qDebug() << "Mesh has normals";
00150     }
00151
00152
00153     _vertexCount = vertexIndices.size();
00154
00155     vertexData = (vertexData_t *)malloc(_vertexCount * sizeof(
      vertexData_t));
00156     if (vertexData != NULL)
00157     {
00158      for(int i=0; i < _vertexCount; i++)
00159      {
00160       unsigned int vertexIndex = vertexIndices[i];
00161
00162       //OBJ indexing starts at 1
00163       QVector3D vertex = temp_vertices[vertexIndex-1];
00164
00165       vertexData[i].vert.x = vertex.x();
00166       vertexData[i].vert.y = vertex.z();
```

```
00167       vertexData[i].vert.z = vertex.y();
00168       vertexData[i].vert.w = 1;
00169
00170
00171       if (_hasNormals)
00172       {
00173        unsigned int normalIndex = normalIndices[i];
00174        QVector3D normal = temp_normals[normalIndex-1];
00175        vertexData[i].normal.x=normal.x();
00176        vertexData[i].normal.y=normal.z();
00177        vertexData[i].normal.z=normal.y();
00178       } else
00179       {
00180        vertexData[i].normal.x = 0;
00181        vertexData[i].normal.y = 0;
00182        vertexData[i].normal.z = 0;
00183       }
00184
00185       if (_hasTexCoords)
00186       {
00187        unsigned int uvIndex = uvIndices[i];
00188        QVector2D uv = temp_uvs[uvIndex-1];
00189        vertexData[i].tex.s=uv.x();
00190        vertexData[i].tex.t=uv.y();
00191       } else
00192       {
00193        vertexData[i].tex.s = 0;
00194        vertexData[i].tex.t = 0;
00195       }
00196
00197      }
00198      qDebug() << "Successfully parsed" << _vertexCount << "vertices";
00199
00200      // build list of unique verticies
00201      QVector<vertexData_t> indexedVertices;
00202      QVector<GLuint> indices;
00203      for (int i = 0; i < _vertexCount ; i++)
00204      {
00205       int loc = 0;
00206       bool found = false;
00207       while (loc < indexedVertices.count() && found == false)
00208       {
00209        vertexData_t test = indexedVertices[loc];
00210        if (test.vert.x == vertexData[i].vert.x && test.vert.y == vertexData[i].
      vert.y &&
00211         test.vert.z == vertexData[i].vert.z && test.vert.w == vertexData[i].
      vert.w)
00212        {
00213         if (test.normal.x == vertexData[i].normal.x && test.normal.
      y == vertexData[i].normal.y &&
00214          test.normal.z == vertexData[i].normal.z)
00215         {
00216          if (test.tex.s == vertexData[i].tex.s && test.tex.t == vertexData[i].
      tex.t)
00217          {
00218           //   qDebug() << "Found duplicate vertex at" << loc;
00219           found = true;
00220          }
00221         }
00222        }
00223
00224        if (!found)
00225         loc++;
00226       }
00227
00228       if (!found)
00229        indexedVertices.append(vertexData[i]);
00230
00231       indices.append(loc);
00232
00233      }
00234
00235      qDebug() << "Found" << indexedVertices.count() << "unique vertices";
00236
00237      if (indices.count() > 0)
00238      {
00239       //qDebug() << "Building indexed mesh...";
00240
00241
00242       arrayIndex = (GLuint *) malloc(indices.count() * sizeof(GLuint));
00243       if (arrayIndex != NULL)
00244        memcpy(arrayIndex,indices.constData(),indices.count() * sizeof(GLuint));
00245
00246       indexedVertexData = (vertexData_t *) malloc(indexedVertices.count() * sizeof(
      vertexData_t));
00247       if (indexedVertexData != NULL)
00248        memcpy(indexedVertexData,indexedVertices.constData(),indexedVertices.count() * sizeof(
```

```
      vertexData_t));
00249
00250       if (arrayIndex != NULL && indexedVertexData != NULL)
00251       {
00252        _hasIndex = true;
00253        _indexCount = indices.count();
00254        _indexedVertexCount = indexedVertices.count();
00255        qDebug() << "Indexed mesh generated with" << indices.count() << "indices and" << indexedVertices.
      count() << "vertices";
00256       } else
00257       {
00258
00259        if (indexedVertexData != NULL)
00260         free(indexedVertexData);
00261        indexedVertexData = NULL;
00262        if (arrayIndex != NULL)
00263         free(arrayIndex);
00264        arrayIndex = NULL;
00265        _indexCount = 0;
00266        _indexedVertexCount = 0;
00267       }
00268      }
00269     }
00270    }
00271   } else
00272   {
00273    qDebug() << "Error: file " << fileName << " does not exist";
00274    return false;
00275   }
00276
00277
00278   if (vertexData != NULL)
00279   {
00280    glGenBuffers(1,&_VBOID);
00281
00282    if (_hasIndex)
00283    {
00284     glBindBuffer(GL_ARRAY_BUFFER,_VBOID);
00285     glBufferData(GL_ARRAY_BUFFER,_indexedVertexCount*sizeof(
      vertexData_t),indexedVertexData,GL_STATIC_DRAW);
00286     glGenBuffers(1,&_indexBufferID);
00287     qDebug() << "Created VBO ID " <<_VBOID << "and IB ID" <<
      _indexBufferID;
00288
00289
00290     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,_indexBufferID);
00291     glBufferData(GL_ELEMENT_ARRAY_BUFFER, _indexCount * sizeof(GLuint), arrayIndex,
      GL_STATIC_DRAW);
00292     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,0);
00293     glBindBuffer(GL_ARRAY_BUFFER,0);
00294
00295    } else {
00296     glBindBuffer(GL_ARRAY_BUFFER,_VBOID);
00297     glBufferData(GL_ARRAY_BUFFER,_vertexCount*sizeof(vertexData_t),vertexData,
      GL_STATIC_DRAW);
00298     qDebug() << "Created VBO ID " <<_VBOID;
00299     glBindBuffer(GL_ARRAY_BUFFER,0);
00300
00301    }
00302
00303    // free(vertexData);
00304    _modelLoaded = true;
00305
00306   }
00307   if (indexedVertexData != NULL)
00308    free(indexedVertexData);
00309   indexedVertexData = NULL;
00310   if (arrayIndex != NULL)
00311    free(arrayIndex);
00312   arrayIndex = NULL;
00313   if (vertexData != NULL)
00314    free(vertexData);
00315   vertexData = NULL;
00316
00317   return true;
00318  } else {
00319   return false;
00320  }
00321 }
00322
00323 void MeshManager::bindBuffer()
00324 {
00325  if (!_modelLoaded)
00326  {
00327   loadFile(_fileName);
00328  }
00329  if (_modelLoaded)
```

```
00330  {
00331   glBindBuffer(GL_ARRAY_BUFFER,_VBOID);
00332
00333   if (_hasIndex)
00334   {
00335    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,_indexBufferID);
00336   }
00337  }
00338
00339 }
00340
00341
00342
00343 void MeshManager::unbindBuffer()
00344 {
00345  glBindBuffer(GL_ARRAY_BUFFER,0);
00346  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,0);
00347 }
00348
00349 // these are, for the time being, basically hard coded values
00350 // the intent is that this class could be modified to use structs that are more memory efficient and these
       values might change
00351
00352 GLuint MeshManager::getVertexStride()
00353 {
00354  return (sizeof(vertexData_t));
00355 }
00356 GLuint MeshManager::getTexStride()
00357 {
00358  return (sizeof(vertexData_t));
00359
00360 }
00361 GLuint MeshManager::getNormalStride()
00362 {
00363  return (sizeof(vertexData_t));
00364 }
00365 GLuint MeshManager::getVertexOffset()
00366 {
00367  return (0);
00368 }
00369 GLuint MeshManager::getTexOffset()
00370 {
00371  return (7 * sizeof(float));
00372 }
00373 GLuint MeshManager::getNormalOffset()
00374 {
00375  return (4 * sizeof(float));
00376 }
00377
00378 GLuint MeshManager::getNumVertices()
00379 {
00380  return _vertexCount;
00381 }
00382
00383 void MeshManager::enableAttributeArrays()
00384 {
00385  if (!_modelLoaded && _fileSet)
00386  {
00387   loadFile(_fileName);
00388  }
00389  if (_modelLoaded)
00390  {
00391   if (_shaderAttribLocs.vertex > -1)
00392   {
00393    glVertexAttribPointer(_shaderAttribLocs.vertex,4,GL_FLOAT,GL_FALSE,
      getVertexStride(),BUFFER_OFFSET(getVertexOffset()));
00394    glEnableVertexAttribArray(_shaderAttribLocs.vertex);
00395   }
00396   if (_hasNormals && _shaderAttribLocs.normal > -1)
00397   {
00398    glVertexAttribPointer(_shaderAttribLocs.normal,3,GL_FLOAT,GL_FALSE,
      getNormalStride(),BUFFER_OFFSET(getNormalOffset()));
00399    glEnableVertexAttribArray(_shaderAttribLocs.normal);
00400   }
00401   if (_hasTexCoords && _shaderAttribLocs.texCoords > -1)
00402   {
00403    glVertexAttribPointer(_shaderAttribLocs.texCoords,2,GL_FLOAT,GL_FALSE,
      getTexStride(),BUFFER_OFFSET(getTexOffset()));
00404    glEnableVertexAttribArray(_shaderAttribLocs.texCoords);
00405   }
00406
00407  }
00408 }
00409
00410 void MeshManager::disableAttributeArrays()
00411 {
00412  if (!_modelLoaded && _fileSet)
```

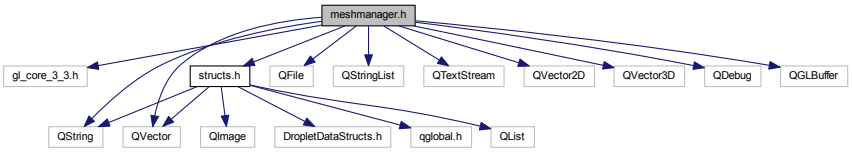
```
00413  {
00414   loadFile(_fileName);
00415  }
00416  if (_modelLoaded)
00417  {
00418   if (_shaderAttribLocs.vertex > -1)
00419    glDisableVertexAttribArray(_shaderAttribLocs.vertex);
00420   if (_hasNormals && _shaderAttribLocs.normal > -1)
00421   {
00422    glDisableVertexAttribArray(_shaderAttribLocs.normal);
00423   }
00424   if (_hasTexCoords && _shaderAttribLocs.texCoords > -1)
00425   {
00426    glDisableVertexAttribArray(_shaderAttribLocs.texCoords);
00427   }
00428
00429  }
00430 }
00431 void MeshManager::draw()
00432 {
00433  if (!_modelLoaded && _fileSet)
00434  {
00435   loadFile(_fileName);
00436  }
00437  if (_modelLoaded)
00438  {
00439
00440   if (_hasIndex)
00441   {
00442
00443    glDrawElements(GL_TRIANGLES,_indexCount,GL_UNSIGNED_INT,0);
00444
00445   } else {
00446    glDrawArrays(GL_TRIANGLES,0,_vertexCount);
00447   }
00448  }
00449 }
00450
00451 void MeshManager::setAttribLoc(int locVertex, int locNormal, int locTexCoords)
00452 {
00453  _shaderAttribLocs.vertex = locVertex;
00454  _shaderAttribLocs.normal = locNormal;
00455  _shaderAttribLocs.texCoords = locTexCoords;
00456 }
00457
00458 void MeshManager::freeModel()
00459 {
00460  if (_modelLoaded)
00461  {
00462   glDeleteBuffers(1,&_VBOID);
00463   _VBOID = 0;
00464   _modelLoaded = false;
00465   _hasIndex = false;
00466   _hasTexCoords = false;
00467   _hasNormals = false;
00468   _indexCount = 0;
00469   _indexedVertexCount = 0;
00470   _vertexCount = 0;
00471   _indexBufferID = 0;
00472  }
00473 }
```

13.20 meshmanager.h File Reference

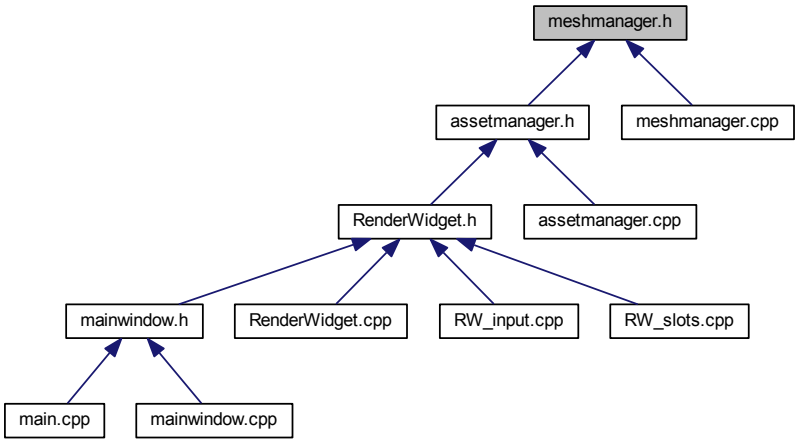
Declares the MeshManager class.

```
#include "gl_core_3_3.h"
#include <QString>
#include <QFile>
#include <QStringList>
#include <QTextStream>
#include <QVector>
#include <QVector2D>
#include <QVector3D>
#include <QDebug>
#include <QGLBuffer>
#include "structs.h"
```

Include dependency graph for meshmanager.h:



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



Classes

- class MeshManager

  Container class for meshes.
- struct MeshManager::vertexData_t

  Structure type to contain packed vertex attributes.

Macros

- #define BUFFER_OFFSET(bytes) ((GLubyte∗) NULL + (bytes))

13.20.1 Detailed Description

Declares the MeshManager class.

Definition in file meshmanager.h.

13.20.2 Macro Definition Documentation

13.20.2.1 #define BUFFER_OFFSET(*bytes*) ((GLubyte∗) NULL + (bytes))

Definition at line 23 of file meshmanager.h.

13.21 meshmanager.h

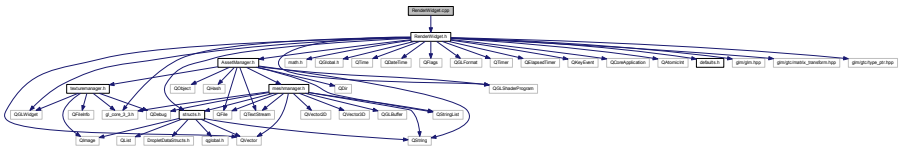
```
00001
00007 #ifndef MESHMANAGER_H
00008 #define MESHMANAGER_H
00009
00010 #include "gl_core_3_3.h"
00011
00012 #include <QString>
00013 #include <QFile>
00014 #include <QStringList>
00015 #include <QTextStream>
00016 #include <QVector>
00017 #include <QVector2D>
00018 #include <QVector3D>
00019 #include <QDebug>
00020 #include <QGLBuffer>
00021 #include "structs.h"
00022
00023 #define BUFFER_OFFSET(bytes) ((GLubyte*) NULL + (bytes))
00024
00032 class MeshManager
00033 {
00034
00035
00036 public:
00037
00044  MeshManager();
00045
00052  ~MeshManager();
00053
00064  bool loadFile(QString fileName);
00065
00076  bool setFile(QString fileName);
00077
00084  void freeModel();
00085
00092  void bindBuffer();
00093
00100  void unbindBuffer();
00101
00110  GLuint getVertexStride();
00111
00120  GLuint getTexStride();
00121
00130  GLuint getNormalStride();
00131
00140  GLuint getVertexOffset();
00141
00150  GLuint getTexOffset();
00151
00160  GLuint getNormalOffset();
00161
00170  GLuint getNumVertices();
00171
00182  void setAttribLoc(int locVertex, int locNormal, int locTexCoords);
00183
00190  void enableAttributeArrays();
00191
00198  void disableAttributeArrays();
00199
00206  void draw();
00207
00208 private:
00209
00216  struct vertexData_t
00217  {
00222   vec4 vert;
00223
00228   vec3 normal;
00229
00234   vec2 tex;
00235  };
00236
00237
00238  struct {
00239   GLint vertex;
00240   GLint normal;
00241   GLint texCoords;
00242
00247  }  _shaderAttribLocs;
00248
00253  bool _hasIndex;
00254
00259  bool _hasTexCoords;
00260
00265  bool _hasNormals;
```

```
00266
00271  bool _modelLoaded;
00272
00277  QString _fileName;
00278
00283  bool _fileSet;
00284
00289  int _indexCount;
00290
00295  int _indexedVertexCount;
00296
00301  int _vertexCount;
00302
00307  QGLBuffer _VBO;
00308
00313  GLuint _indexBufferID;
00314
00319  GLuint _VBOID;
00320
00321 };
00322
00323 #endif // MeshManager_H
```

13.22 RenderWidget.cpp File Reference

Implements the main portion of the RenderWidget class.

`#include "RenderWidget.h"`
Include dependency graph for RenderWidget.cpp:



13.22.1 Detailed Description

Implements the main portion of the RenderWidget class.

Definition in file RenderWidget.cpp.

13.23 RenderWidget.cpp

```
00001
00007 #include "RenderWidget.h"
00008
00009 RenderWidget::RenderWidget(const QGLFormat& format, QWidget *parent)
00010  : QGLWidget(format, parent)
00011 {
00012  assets.setAssetDir(DEFAULT_ASSETDIR);
00013  assets.setShaderDir(DEFAULT_SHADERDIR);
00014  assets.setMeshDir(DEFAULT_MESHDIR);
00015  assets.setTextureDir(DEFAULT_TEXTUREDIR);
00016
00017  _assetsLoaded = false;
00018  _renderLock = false;
00019  _simStateLock = false;
00020  _renderDebug = 0;
00021  _hud = true;
00022  _drawHelp = false;
00023  _hudInfo.paused = false;
00024  // initialize the camera
00025  _camera.x = 0;
00026  _camera.y = 0;
00027  _camera.z = 0;
00028  _camera.tilt = 0;
00029  _camera.pan = 0;
00030  _camera.radius = 100.0f;
00031  _camera.rotHoriz = 180;
00032  _camera.rotVert = 0;
```

```
00033  _camera.mode = 0;
00034  _camera.viewMatrix.setToIdentity();
00035  _camera.lightDir = glm::vec3(1,1,3);
00036  updateCamera();
00037
00038  _timerID = 0;
00039  // start out not unpaused
00040
00041  // intialize key status
00042  _keysDown.Q = false;
00043  _keysDown.W = false;
00044  _keysDown.E = false;
00045  _keysDown.A = false;
00046  _keysDown.S = false;
00047  _keysDown.D = false;
00048  _keysDown.Plus = false;
00049  _keysDown.Minus = false;
00050
00051  _projectionTexture.width = 0;
00052  _projectionTexture.height = 0;
00053  _projectionTexture.handle = 0;
00054  _projectionTexture.valid = false;
00055  QTime time = QTime::currentTime();
00056  qsrand((uint)time.msec());
00057  _hudInfo.framesSinceLastUpdate = 99;
00058 }
00059
00060 RenderWidget::~RenderWidget()
00061 {
00062  //delete _sim;
00063  if (_timerID != 0)
00064  {
00065   killTimer(_timerID);
00066   _timerID = 0;
00067  }
00068  makeCurrent();
00069
00070  if (_projectionTexture.valid)
00071  {
00072   glDeleteTextures(1,&_projectionTexture.handle);
00073   _projectionTexture.valid = false;
00074   _projectionTexture.handle = 0;
00075  }
00076  assets.clearAssets();
00077  _assetsLoaded = false;
00078  doneCurrent();
00079 }
00080
00081 QSize RenderWidget::sizeHint() const
00082 {
00083  return QSize(1280, 720);
00084 }
00085
00086
00087 void RenderWidget::initializeGL()
00088 {
00089
00090  ogl_LoadFunctions();
00091
00092  // depth testing and culling
00093  glEnable(GL_DEPTH_TEST);
00094  glEnable(GL_CULL_FACE);
00095  glFrontFace(GL_CW);
00096  glEnable(GL_TEXTURE_2D);
00097  // set the clear color to black
00098  glClearColor(0.0f,0.0f,0.0f,0.0f);
00099  qDebug() << "Initializing OpenGL";
00100  // load all assets
00101  _assetsLoaded = assets.loadAssets();
00102
00103  _dropletStruct.color = glm::vec4(0,0,0,1);
00104  _dropletStruct.origin = glm::vec3(0,0,0);
00105  _dropletStruct.rotation = glm::vec3(0,0,0);
00106  _dropletStruct.scale = glm::vec3(1,1,1);
00107  _dropletStruct.mesh = assets.getMesh(assets.
      lookupAssetName(DROPLET_MESH_NAME));
00108  _dropletStruct.baseShader = assets.getShader(
      assets.lookupAssetName(DROPLET_SHADER_NAME));
00109  _dropletStruct.projShader = assets.getShader(
      assets.lookupAssetName(DROPLET_PROJECTION_SHADER_NAME));
00110  _dropletStruct.texture_0 = assets.getTexture(
      assets.lookupAssetName(DROPLET_TEXTURE_NAME));
00111
00112  setupRenderStructs();
00113  _arenaObjects.clear();
00114  //setupArena();
00115  setFPS(60);
```

```
00116  //_runTime.start();
00117  _updateTimer.start();
00118  _timerID = startTimer(_targetFrameTime);
00119 }
00120
00121 void RenderWidget::setFPS(int FPS)
00122 {
00123
00124  if (FPS > 0)
00125  {
00126   _targetFPS = FPS;
00127   _targetFrameTime = 1000.0/FPS;
00128
00129   qDebug() << "Set render goal for" << (int) floor(1000.0/FPS) << "ms";
00130  } else
00131  {
00132   _targetFPS = 0;
00133   _targetFrameTime = 0;
00134  }
00135  _renderTimer.start();
00136 }
00137
00138
00139
00140 void RenderWidget::resizeGL(int width, int height)
00141 {
00142  if (height == 0) {
00143   height = 1;
00144  }
00145
00146  _camera.projectionMatrix.setToIdentity();
00147  _camera.projectionMatrix.perspective(60.0, (float) width / (float) height, 1, 1000);
00148  glViewport(0, 0, width, height);
00149 }
00150
00151 void RenderWidget::updateCamera()
00152 {
00153  // clamp camera values
00154
00155
00156  if (_camera.pan > 360)
00157  {
00158   _camera.pan -= 360;
00159  } else if (_camera.pan < 0)
00160  {
00161   _camera.pan += 360;
00162  }
00163
00164  if (_camera.tilt > 90)
00165  {
00166   _camera.tilt = 90;
00167  } else if (_camera.tilt < -90)
00168  {
00169   _camera.tilt = -90;
00170  }
00171
00172  /* Clamp camera rotations */
00173  if (_camera.rotHoriz > 360)
00174  {
00175  _camera.rotHoriz -= 360;
00176  } else if (_camera.rotHoriz < 0)
00177  {
00178  _camera.rotHoriz += 360;
00179  }
00180
00181  if (_camera.rotVert > 90)
00182  {
00183  _camera.rotVert = 90;
00184  } else if (_camera.rotVert < 0)
00185  {
00186  _camera.rotVert = 0;
00187  }
00188
00189  /* Clamp zooming in/out */
00190  if (_camera.radius < 5)
00191  {
00192   _camera.radius = 5;
00193  } else if (_camera.radius > 300)
00194  {
00195   _camera.radius = 300;
00196  }
00197
00198
00199  // If in the default camera mode
00200  // Convert the rotations around a sphere into xyz coordinate
00201  if (_camera.mode == 0)
00202  {
```

```
00203   // generate x, y, z and pan/tilt angles off of rotation around a sphere
00204
00205   _camera.y = _camera.radius * sin(_camera.rotVert * M_PI / 180.0) * cos(
      _camera.rotHoriz * M_PI / 180.0);
00206   _camera.x = _camera.radius * sin(_camera.rotVert * M_PI / 180.0) * sin(
      _camera.rotHoriz * M_PI / 180.0);
00207   _camera.z = _camera.radius * cos(_camera.rotVert * M_PI / 180.0);
00208
00209   _camera.pan = 180 - _camera.rotHoriz;
00210   _camera.tilt = -90 +  _camera.rotVert;
00211  }
00212
00213
00214  _camera.viewMatrix.setToIdentity();
00215  _camera.viewMatrix.rotate(-90,1,0,0);
00216  _camera.viewMatrix.rotate(_camera.tilt,-1,0,0);
00217  _camera.viewMatrix.rotate(_camera.pan,0,0,-1);
00218  _camera.viewMatrix.translate(-_camera.x,-_camera.y,-
      _camera.z);
00219  // _camera.viewMatrix.optimize();
00220 }
00221
00222 void RenderWidget::paintGL()
00223 {
00224  if (!_renderLock && _assetsLoaded)
00225  {
00226   _renderLock = true;
00227   _eventTimer.start();
00228
00229   /* get new state */
00230
00231   // block while resource is in use, then retain resource
00232   while(!_simStateLock.testAndSetOrdered(0,1)) {}
00233
00234   // update the local state with a more recent one
00235   _renderState = _simState;
00236
00237   // release resource
00238   _simStateLock.fetchAndStoreOrdered(0);
00239
00240
00241   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00242
00243   if (_renderDebug > 1)
00244   {
00245    glDisable(GL_CULL_FACE);
00246   }
00247
00248   if (!_projectionTexture.valid && _arena.projecting)
00249   {
00250    if (_projectionTexture.handle != 0)
00251    {
00252     glDeleteTextures(1,&_projectionTexture.handle);
00253    }
00254    glGenTextures(1,&_projectionTexture.handle);
00255    _projectionTexture.valid = true;
00256   }
00257
00258   if ( _arena.projecting && _projectionTexture.valid)
00259   {
00260    if (_projectionTexture.texture != _renderState.
      projTexture)
00261    {
00262     _projectionTexture.texture = _renderState.
      projTexture;
00263     glActiveTexture(GL_TEXTURE1);
00264     QImage image = QGLWidget::convertToGLFormat(_projectionTexture.texture);
00265     _projectionTexture.width = image.width();
00266     _projectionTexture.height = image.height();
00267
00268     glBindTexture(GL_TEXTURE_2D, _projectionTexture.handle);
00269     glTexImage2D(GL_TEXTURE_2D, 0, 4, _projectionTexture.width,
      _projectionTexture.height, 0,  GL_RGBA, GL_UNSIGNED_BYTE, image.bits());
00270     glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_LINEAR); // Enable Mipmapping
00271     glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // Linear Filtering
00272     glGenerateMipmap(GL_TEXTURE_2D); // generate mipmaps
00273     glActiveTexture(GL_TEXTURE0);
00274
00275    } else {
00276     glActiveTexture(GL_TEXTURE1);
00277     glBindTexture(GL_TEXTURE_2D, _projectionTexture.handle);
00278     glActiveTexture(GL_TEXTURE0);
00279    }
00280   }
00281
00282   //drawArena();
00283   drawDroplets();
```

```
00284   drawObjects();
00285
00286   if (_arena.projecting && _projectionTexture.valid)
00287   {
00288    glActiveTexture(GL_TEXTURE1);
00289    glBindTexture(GL_TEXTURE_2D,0);
00290    glActiveTexture(GL_TEXTURE0);
00291   }
00292   if (_renderDebug > 1)
00293   {
00294    glEnable(GL_CULL_FACE);
00295    glFrontFace(GL_CW);
00296   }
00297
00298   glFinish();
00299
00300   qint64 nsElapsed = _eventTimer.nsecsElapsed();
00301   _msElapsedRenderer = nsElapsed / 1000000.0;
00302
00303   _hudInfo.framesSinceLastUpdate++;
00304
00305
00306   if (_hud)
00307   {
00308    drawHUD();
00309   }
00310   _renderLock = false;
00311  }
00312 }
00313
00314
00315 void RenderWidget::drawArena()
00316 {
00317  QGLShaderProgram *currentShader = NULL;
00318  MeshManager *currentMesh = NULL;
00319  TextureManager *currentTex0 = NULL;
00320
00321  GLint lLoc, cLoc, mLoc;
00322  GLint t0Loc,t1Loc;
00323  GLint pLoc;
00324
00325  float floorWidth = _arena.tileLength * _arena.numColTiles;
00326  float floorLength = _arena.tileLength * _arena.
     numRowTiles;
00327  glm::vec2 lengths = glm::vec2(floorWidth,floorLength);
00328
00329  // for every object in the arena
00330  foreach(renderStruct_t object,_arenaObjects)
00331  {
00332   // new variables
00333   QGLShaderProgram *newShader;
00334   MeshManager *newMesh;
00335   TextureManager *newTex0;
00336
00337   /* set current render object shader, mesh, and textures */
00338   // Check if visual debugging is on
00339   if (_renderDebug > 1)
00340   {
00341    // if debugging is on, use the debug assets instead of checking the object properties
00342    newShader = assets.getShader(DEBUG_SHADER);
00343    newMesh = assets.getMesh(DEBUG_MESH);
00344    newTex0 = NULL;
00345   } else
00346   {
00347    // if not debugging set new variables to object properties
00348    if (_arena.projecting)
00349    {
00350     newShader = object.projShader;
00351
00352    } else {
00353     newShader = object.baseShader;
00354    }
00355    newMesh = object.mesh;
00356    newTex0 = object.texture_0;
00357
00358   }
00359
00360
00361   /* compare existing state to new state */
00362
00363   // rebind shaders
00364   if (currentShader != newShader)
00365   {
00366    // release the current shader if bound
00367    if (currentShader != NULL)
00368    {
00369     currentShader->release();
```

```
00370    }
00371    currentShader = newShader;
00372    // bind uniform attributes
00373    if (currentShader != NULL)
00374    {
00375     currentShader->bind();
00376     currentShader->setUniformValue("in_Projection",_camera.projectionMatrix);
00377     currentShader->setUniformValue("in_View",_camera.viewMatrix);
00378     lLoc = currentShader->uniformLocation("in_lightDir");
00379     glUniform3fv(lLoc,1,glm::value_ptr(_camera.lightDir));
00380     cLoc = currentShader->uniformLocation("in_Color");
00381     mLoc = currentShader->uniformLocation("in_Model");
00382     t0Loc = currentShader->uniformLocation("objectTexture");
00383     t1Loc = currentShader->uniformLocation("projectionTexture");
00384     pLoc = currentShader->uniformLocation("in_ProjOffsets");
00385     glUniform2fv(pLoc,1,glm::value_ptr(lengths));
00386     glUniform1i(t0Loc,0);
00387     glUniform1i(t1Loc,1);
00388    }
00389   }
00390
00391   // test and rebind texture 0
00392   if (currentTex0 != newTex0)
00393   {
00394    glActiveTexture(GL_TEXTURE0);
00395
00396    if (currentTex0 != NULL)
00397     currentTex0->unbindTexture();
00398    currentTex0 = newTex0;
00399
00400    if (currentTex0 != NULL)
00401    {
00402     currentTex0->bindTexture();
00403    }
00404   }
00405
00406
00407
00408
00409   // test and rebind mesh
00410   if (currentMesh != newMesh)
00411   {
00412    if (currentMesh != NULL)
00413    {
00414     currentMesh->disableAttributeArrays();
00415     currentMesh->unbindBuffer();
00416    }
00417
00418    currentMesh = newMesh;
00419
00420    if (currentMesh != NULL)
00421    {
00422     currentMesh->bindBuffer();
00423     currentMesh->enableAttributeArrays();
00424    }
00425   }
00426
00427   /* now that all properties are bound, draw */
00428
00429   // error testing
00430   if (currentShader != NULL && currentMesh != NULL)
00431   {
00432    // bind color and model matrix uniforms
00433    glUniform4fv(cLoc,1,glm::value_ptr(object.color));
00434    glUniformMatrix4fv(mLoc,1,GL_FALSE,glm::value_ptr(object.modelMatrix));
00435
00436    // draw
00437    currentMesh->draw();
00438   }
00439
00440  } // end foreach
00441
00442  /* all objects are rendered, so unbind all assets */
00443
00444  if (currentShader != NULL)
00445  {
00446   currentShader->release();
00447  }
00448
00449  if (currentMesh != NULL)
00450  {
00451   currentMesh->disableAttributeArrays();
00452   currentMesh->unbindBuffer();
00453  }
00454
00455
00456  if (currentTex0 != NULL)
```

```
00457  {
00458   glActiveTexture(GL_TEXTURE0);
00459   currentTex0->unbindTexture();
00460  }
00461
00462 }
00463
00464 void RenderWidget::drawDroplets()
00465 {
00466  if (_renderState.dropletData.count() > 0)
00467  {
00468   QGLShaderProgram *currentShader = NULL;
00469   MeshManager *currentMesh = NULL;
00470   TextureManager *currentTex0 = NULL;
00471
00472   GLint lLoc, cLoc, mLoc;
00473   GLint t0Loc,t1Loc;
00474   GLint pLoc;
00475   glActiveTexture(GL_TEXTURE0);
00476
00477   // check if rendering debugging is on
00478   if (_renderDebug > 1)
00479   {
00480    currentShader = assets.getShader(DEBUG_SHADER);
00481    //currentMesh = assets.getMesh(DEBUG_DROPLET_MESH);
00482
00483    currentTex0 = NULL;
00484
00485   } else
00486   {
00487    if (_arena.projecting)
00488    {
00489     currentShader = _dropletStruct.projShader;
00490    } else {
00491     currentShader = _dropletStruct.baseShader;
00492
00493    }
00494
00495    currentTex0 = _dropletStruct.texture_0;
00496   }
00497
00498   currentMesh = _dropletStruct.mesh;
00499   if (currentShader != NULL && currentMesh != NULL)
00500   {
00501
00502    // bind and set up droplet shader and mesh
00503    currentShader->bind();
00504    currentShader->setUniformValue("in_Projection",_camera.projectionMatrix);
00505    currentShader->setUniformValue("in_View",_camera.viewMatrix);
00506    //_droplet->setUniformValue("in_lightDir",0.5,1,1);
00507    lLoc = currentShader->uniformLocation("in_lightDir");
00508    glUniform3fv(lLoc,1,glm::value_ptr(_camera.lightDir));
00509    t0Loc = currentShader->uniformLocation("objectTexture");
00510    t1Loc = currentShader->uniformLocation("projectionTexture");
00511    glUniform1i(t0Loc,0);
00512    glUniform1i(t1Loc,1);
00513    currentMesh->bindBuffer();
00514    currentMesh->enableAttributeArrays();
00515
00516    cLoc = currentShader->uniformLocation("in_Color");
00517    mLoc = currentShader->uniformLocation("in_Model");
00518    pLoc = currentShader->uniformLocation("in_ProjOffsets");
00519    float floorWidth = _arena.tileLength * _arena.
      numColTiles;
00520    float floorLength = _arena.tileLength * _arena.
      numRowTiles;
00521    glm::vec2 lengths = glm::vec2(floorWidth,floorLength);
00522    // draw each droplet
00523    glUniform2fv(pLoc,1,glm::value_ptr(lengths));
00524
00525    if (currentTex0 != NULL)
00526    {
00527     glActiveTexture(GL_TEXTURE0);
00528     currentTex0->bindTexture();
00529
00530    }
00531
00532
00533    foreach(dropletStruct_t droplet,_renderState.
      dropletData)
00534    {
00535     // make the model matrix
00536     glm::vec3 origin = glm::vec3( droplet.origin.x, droplet.origin.
      y, droplet.origin.z);
00537
00538     glm::quat quaternion = glm::quat(droplet.quaternion.w,droplet.
      quaternion.x,
```

```
00539       droplet.quaternion.y,droplet.quaternion.z);
00540
00541      glm::mat4 model =  glm::translate(glm::mat4(1.0f),origin);
00542      model = model * glm::mat4_cast(quaternion);
00543      model = glm::scale(model,glm::vec3(_arena.dropletRadius));
00544      model = glm::translate(model,glm::vec3(0,0,_arena.dropletOffset));
00545
00546
00547      vec4 color = {
00548       droplet.color.r / 255.0f,
00549       droplet.color.g / 255.0f,
00550       droplet.color.b / 255.0f,
00551       1.0f
00552      };
00553
00554      // bind droplet uniforms
00555      glUniform4fv(cLoc,1,color.v);
00556      glUniformMatrix4fv(mLoc,1,GL_FALSE,glm::value_ptr(model));
00557      // glDrawArrays(GL_TRIANGLES,0,vertCount);
00558      currentMesh->draw();
00559
00560     }
00561     currentMesh->disableAttributeArrays();
00562     currentMesh->unbindBuffer();
00563     currentShader->release();
00564     glActiveTexture(GL_TEXTURE0);
00565     glBindTexture(GL_TEXTURE_2D,0);
00566
00567    }
00568    glActiveTexture(GL_TEXTURE0);
00569   }
00570 }
00571 void RenderWidget::drawObjects()
00572 {
00573  QVector<objectStruct_t> objects = _renderState.dynamicObjectData +
     _renderState.staticObjectData;
00574  if (objects.count() > 0){
00575   QGLShaderProgram *currentShader = NULL;
00576   MeshManager *currentMesh = NULL;
00577   TextureManager *currentTex0 = NULL;
00578
00579   GLint lLoc, cLoc, mLoc;
00580   GLint t0Loc,t1Loc;
00581   GLint pLoc;
00582
00583
00584   glActiveTexture(GL_TEXTURE0);
00585
00586   float floorWidth = _arena.tileLength * _arena.
     numColTiles;
00587   float floorLength = _arena.tileLength * _arena.
     numRowTiles;
00588   glm::vec2 lengths = glm::vec2(floorWidth,floorLength);
00589
00590   // draw each object
00591
00592   // for every object in the arena
00593   foreach(objectStruct_t object,objects)
00594   {
00595    // new variables
00596    QGLShaderProgram *newShader;
00597    MeshManager *newMesh;
00598    TextureManager *newTex0;
00599
00600    /* set current render object shader, mesh, and textures */
00601    // check if debuggin is on
00602    if (_renderDebug > 1)
00603    {
00604     // if debugging is on, use the debug assets instead of checking the object properties
00605     newShader = assets.getShader(DEBUG_SHADER);
00606     newTex0 = NULL;
00607
00608    } else
00609    {
00610     // if not debugging set new variables to object properties
00611     if (_arena.projecting)
00612     {
00613      newShader = _objectStructs[object.oType].projShader;
00614     } else {
00615      newShader = _objectStructs[object.oType].baseShader;
00616
00617     }
00618     newTex0 = _objectStructs[object.oType].texture_0;
00619    }
00620
00621    // render the proper mesh regardless of debug mode
00622    newMesh = _objectStructs[object.oType].mesh;
```

```
00623
00624     /* compare existing state to new state */
00625
00626     // rebind shaders
00627     if (currentShader != newShader)
00628     {
00629      // release the current shader if bound
00630      if (currentShader != NULL)
00631      {
00632       currentShader->release();
00633      }
00634      currentShader = newShader;
00635      // bind uniform attributes
00636      if (currentShader != NULL)
00637      {
00638       currentShader->bind();
00639       currentShader->setUniformValue("in_Projection",_camera.projectionMatrix);
00640       currentShader->setUniformValue("in_View",_camera.viewMatrix);
00641       lLoc = currentShader->uniformLocation("in_lightDir");
00642       glUniform3fv(lLoc,1,glm::value_ptr(_camera.lightDir));
00643       cLoc = currentShader->uniformLocation("in_Color");
00644       mLoc = currentShader->uniformLocation("in_Model");
00645       t0Loc = currentShader->uniformLocation("objectTexture");
00646       t1Loc = currentShader->uniformLocation("projectionTexture");
00647       pLoc = currentShader->uniformLocation("in_ProjOffsets");
00648       glUniform2fv(pLoc,1,glm::value_ptr(lengths));
00649       glUniform1i(t0Loc,0);
00650       glUniform1i(t1Loc,1);
00651      }
00652     }
00653
00654     // test and rebind texture 0
00655     if (currentTex0 != newTex0)
00656     {
00657      glActiveTexture(GL_TEXTURE0);
00658
00659      if (currentTex0 != NULL)
00660       currentTex0->unbindTexture();
00661      currentTex0 = newTex0;
00662
00663      if (currentTex0 != NULL)
00664      {
00665       currentTex0->bindTexture();
00666      }
00667     }
00668
00669     // test and rebind mesh
00670     if (currentMesh != newMesh)
00671     {
00672      if (currentMesh != NULL)
00673      {
00674       currentMesh->disableAttributeArrays();
00675       currentMesh->unbindBuffer();
00676      }
00677
00678      currentMesh = newMesh;
00679
00680      if (currentMesh != NULL)
00681      {
00682       currentMesh->bindBuffer();
00683       currentMesh->enableAttributeArrays();
00684      }
00685     }
00686
00687     /* now that all properties are bound, draw */
00688
00689     // error testing
00690     if (currentShader != NULL && currentMesh != NULL)
00691     {
00692      glm::vec3 origin = glm::vec3( object.origin.x, object.origin.y, object.origin.z);
00693
00694      glm::quat quaternion = glm::quat(object.quaternion.w,object.quaternion.x,
00695       object.quaternion.y,object.quaternion.z);
00696
00697      glm::mat4 model =  glm::translate(glm::mat4(1.0f),origin);
00698
00699      model = model * glm::mat4_cast(quaternion);
00700      model = glm::scale(model,glm::vec3(object.objectRadius*object.scale.x,object.objectRadius*object.scale.
      y,object.objectRadius*object.scale.z));
00701
00702      vec4 color = {
00703       object.color.r / 255.0f,
00704       object.color.g / 255.0f,
00705       object.color.b / 255.0f,
00706       1.0f
00707      };
00708
```

```
00709     // bind color and model matrix uniforms
00710     glUniform4fv(cLoc,1,color.v);
00711     glUniformMatrix4fv(mLoc,1,GL_FALSE,glm::value_ptr(model));
00712
00713     // draw
00714     currentMesh->draw();
00715    }
00716
00717
00718   } // end foreach
00719
00720   /* all objects are rendered, so unbind all assets */
00721
00722   if (currentShader != NULL)
00723   {
00724    currentShader->release();
00725   }
00726
00727   if (currentMesh != NULL)
00728   {
00729    currentMesh->disableAttributeArrays();
00730    currentMesh->unbindBuffer();
00731   }
00732
00733   if (currentTex0 != NULL)
00734   {
00735    glActiveTexture(GL_TEXTURE0);
00736    currentTex0->unbindTexture();
00737   }
00738
00739  }
00740 }
00741
00742 // TIMER
00743 void RenderWidget::drawHUD()
00744 {
00745
00746  //QTime time = _renderState.realTime;
00747  double time = _renderState.realTime;
00748  if (_hudInfo.framesSinceLastUpdate > 20)
00749  {
00750   _hudInfo.framesSinceLastUpdate = 0;
00751
00752   // TIMER
00753   /*
00754   int deltaRT = time.msecsTo(_hudInfo.lastRunTime);
00755   int deltaSim = _renderState.simTime.msecsTo(_hudInfo.lastSimTime);
00756
00757
00758   _hudInfo.lastRunTime = time;
00759   _hudInfo.lastSimTime = _renderState.simTime;
00760   float timeRatio = (float) deltaSim / (float) deltaRT;
00761   */
00762   double timeRatio = _renderState.timeRatio;
00763   _hudInfo.simRealTimeRatio = floor(timeRatio*10000.0 + 0.5) / 10000.0;
00764
00765
00766  }
00767  // render the current screen size into the window
00768
00769
00770  int baseOffset = 20;
00771  QString text = QString("Number of droplets: %1").arg(_renderState.
     dropletData.count(),0);
00772  renderText(10,baseOffset,text,this->font());
00773  baseOffset += 20;
00774
00775  text = QString("Simulator step size: %1 ms").arg(_hudInfo.simStepSize,0);
00776  renderText(10,baseOffset,text,this->font());
00777  baseOffset += 20;
00778
00779  int secs = _renderState.simTime;
00780  int mSecs = (_renderState.simTime - secs) * 1000.0;
00781  QTime iTime = QTime(0,0,0,0).addSecs(secs).addMSecs(mSecs);
00782
00783  text = QString("Simulator time elapsed: %1").arg(iTime.toString(QString("H : mm : ss.zzz")));
00784  //text = QString("Simulator time elapsed: %1").arg(floor(_renderState.simTime*100.0 + 0.5) / 100.0,0);
00785  renderText(10,baseOffset,text,this->font());
00786  baseOffset += 20;
00787
00788
00789  secs = floor(_renderState.realTime);
00790  mSecs = (_renderState.realTime - secs) * 1000.0;
00791  iTime = QTime(0,0,0,0).addSecs(secs).addMSecs(mSecs);
00792  text = QString("Real time elapsed: %1").arg(iTime.toString(QString("H : mm : ss.zzz")));
00793  //text = QString("Real time elapsed: %1").arg(floor(_renderState.realTime*100.0 + 0.5) / 100.0,0);
00794  renderText(10,baseOffset,text,this->font());
```

```
00795  baseOffset += 20;
00796
00797
00798  if (_simRates.limitRate)
00799  {
00800   text = QString("Requested simulator/real time ratio: %1x").arg(_simRates.
     timeScale,2);
00801   renderText(10,baseOffset,text,this->font());
00802   baseOffset += 20;
00803  }
00804
00805  if (_hudInfo.paused)
00806  {
00807   text = QString("PAUSED");
00808   renderText(10,baseOffset,text,this->font());
00809   baseOffset += 20;
00810  } else {
00811
00812   // If you display this while the display is paused it will be invalid data
00813   text = QString("Estimated simulator/real time ratio: %1x").arg(_hudInfo.simRealTimeRatio,2);
00814   renderText(10,baseOffset,text,this->font());
00815   baseOffset += 20;
00816  }
00817
00818  // if *any* level of debugging is enabled
00819  if (_renderDebug > 0)
00820  {
00821
00822   // print off a whole slew of debug information related to rendering
00823   QString width;
00824   QString height;
00825   width.setNum(this->width());
00826   height.setNum(this->height());
00827
00828   QString text = QString("Res: ").append(width).append(" x ").append(height);
00829   renderText(10,baseOffset,text,this->font());
00830   baseOffset += 20;
00831
00832
00833   int numObjects = _arenaObjects.count() + _renderState.
     dropletData.count() + _renderState.dynamicObjectData.count() +
     _renderState.staticObjectData.count();
00834   text = QString("Number of render objects: %1").arg(numObjects,0);
00835   renderText(10,baseOffset,text,this->font());
00836   baseOffset += 20;
00837
00838   // draw the frame time
00839   QString frameTime = QString("%1").arg(_msElapsedRenderer,0,'g',4);
00840   text = QString("Frame Time: ").append(frameTime).append(" ms");
00841   renderText(10,baseOffset,text,this->font());
00842   baseOffset += 20;
00843
00844   if (_camera.mode == 0)
00845   {
00846    text = QString("Fixed camera - Elevation: %1  Rotation: %2 Radius: %3").arg(90 -
     _camera.rotVert,0,'g',3).arg(_camera.rotHoriz,0,'g',3).arg(_camera.radius,0,'g',3);
00847    renderText(10,baseOffset,text,this->font());
00848   } else if (_camera.mode == 1)
00849   {
00850    text = QString("Free camera - X: %1  Y: %2 Z: %3 Pan: %4 Tilt: %5").arg(
     _camera.x,0,'g',4).arg(_camera.y,0,'g',4).arg(_camera.z,0,'g',2).arg(
     _camera.pan,0,'g',3).arg(_camera.tilt,0,'g',3);
00851    renderText(10,baseOffset,text,this->font());
00852   }
00853   baseOffset += 20;
00854
00855   // if debug rendering is enabled
00856   if (_renderDebug > 1)
00857   {
00858    text = QString("DEBUG RENDERING");
00859    renderText(10,baseOffset,text,this->font());
00860    baseOffset += 20;
00861   }
00862  }
00863  if (_camera.mode)
00864  {
00865   text = QString("Free camera");
00866   renderText(10,baseOffset,text,this->font());
00867   baseOffset += 20;
00868
00869  }
00870
00871
00872  if (_drawHelp)
00873  {
00874   baseOffset = height() - 10;
00875   text = QString("Control-R: Reset simulator to starting state");
```

```
00876   renderText(10,baseOffset,text,this->font());
00877   baseOffset -= 20;
00878   text = QString("Control-L: Force reloading of rendering assets");
00879   renderText(10,baseOffset,text,this->font());
00880   baseOffset -= 20;
00881   text = QString("Control-B: Toggle debug rendering on/off");
00882   renderText(10,baseOffset,text,this->font());
00883   baseOffset -= 20;
00884   text = QString("Control-H: Toggle HUD on/off");
00885   renderText(10,baseOffset,text,this->font());
00886   baseOffset -= 20;
00887   text = QString("Escape: Close this window");
00888   renderText(10,baseOffset,text,this->font());
00889   baseOffset -= 20;
00890   text = QString("Spacebar: Change camera modes");
00891   renderText(10,baseOffset,text,this->font());
00892   baseOffset -= 20;
00893   text = QString("H: Toggle this help on/off");
00894   renderText(10,baseOffset,text,this->font());
00895   baseOffset -= 20;
00896
00897   if (_hudInfo.paused)
00898   {
00899    text = QString("P: Unpause the simulator");
00900    renderText(10,baseOffset,text,this->font());
00901    baseOffset -= 20;
00902   } else {
00903    text = QString("P: Pause the simulator");
00904    renderText(10,baseOffset,text,this->font());
00905    baseOffset -= 20;
00906   }
00907   if (_simRates.limitRate)
00908   {
00909    text = QString("L: Disable simulation speed limiting");
00910    renderText(10,baseOffset,text,this->font());
00911    baseOffset -= 20;
00912    text = QString("[/]: Lower/Raise speed limit");
00913    renderText(10,baseOffset,text,this->font());
00914    baseOffset -= 20;
00915   } else {
00916    text = QString("L: Enable simulation speed limiting");
00917    renderText(10,baseOffset,text,this->font());
00918    baseOffset -= 20;
00919   }
00920   /*
00921   text = QString("Z: Create StickPullers droplet");
00922   renderText(10,baseOffset,text,this->font());
00923   baseOffset -= 20;
00924   text = QString("X: Create RGBSense droplet");
00925   renderText(10,baseOffset,text,this->font());
00926   baseOffset -= 20;
00927   */
00928
00929   if (_camera.mode == 0)
00930   {
00931    text = QString("Q/E and -/+: Zoom out/in");
00932    renderText(10,baseOffset,text,this->font());
00933    baseOffset -= 20;
00934    text = QString("A/D: Rotate right/left");
00935    renderText(10,baseOffset,text,this->font());
00936    baseOffset -= 20;
00937    text = QString("W/S: Rotate up/down");
00938    renderText(10,baseOffset,text,this->font());
00939    baseOffset -= 20;
00940
00941   } else if (_camera.mode == 1)
00942   {
00943    text = QString("Q/E: Lower/raise camera height");
00944    renderText(10,baseOffset,text,this->font());
00945    baseOffset -= 20;
00946    text = QString("A/D: Strafe camera left/right");
00947    renderText(10,baseOffset,text,this->font());
00948    baseOffset -= 20;
00949    text = QString("W/S: Move forward/back");
00950    renderText(10,baseOffset,text,this->font());
00951    baseOffset -= 20;
00952
00953   }
00954  }
00955
00956 }
00957 void RenderWidget::saveScreenShot(int width, int height)
00958 {
00959  GLint currentFBO = 0;
00960  glGetIntegerv(GL_FRAMEBUFFER_BINDING,&currentFBO);
00961
00962  GLuint offscreenFBO = 0;
```

```
00963  GLuint renderedTexture = 0;
00964  GLuint renderBuffer = 0;
00965  glGenFramebuffers(1,&offscreenFBO);
00966  glGenTextures(1,&renderedTexture);
00967  glGenRenderbuffers(1,&renderBuffer);
00968  qDebug() << "Created texture" << renderedTexture;
00969  // glActiveTexture(GL_TEXTURE3);
00970  glBindTexture(GL_TEXTURE_2D,renderedTexture);
00971  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
00972  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
00973
00974  glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,width,height,0,GL_RGBA,GL_UNSIGNED_BYTE,0);
00975  glBindTexture(GL_TEXTURE_2D,0);
00976
00977  glBindFramebuffer(GL_FRAMEBUFFER,offscreenFBO);
00978  glFramebufferTexture2D(GL_FRAMEBUFFER,GL_COLOR_ATTACHMENT0,GL_TEXTURE_2D,renderedTexture,0);
00979
00980  glBindRenderbuffer(GL_RENDERBUFFER,renderBuffer);
00981  glRenderbufferStorage(GL_RENDERBUFFER,GL_DEPTH_COMPONENT,width,height);
00982  glFramebufferRenderbuffer(GL_FRAMEBUFFER,GL_DEPTH_ATTACHMENT,GL_RENDERBUFFER,renderBuffer);
00983
00984  if (glCheckFramebufferStatus(GL_FRAMEBUFFER) == GL_FRAMEBUFFER_COMPLETE)
00985  {
00986   QMatrix4x4 oldCamera = _camera.projectionMatrix;
00987
00988   _camera.projectionMatrix.setToIdentity();
00989   _camera.projectionMatrix.perspective(60., (float) width / (float) height, 1, 1000);
00990   glViewport(0, 0, width, height);
00991
00992   bool wasHudEnabled = _hud;
00993   _hud = false;
00994   paintGL();
00995   _hud = wasHudEnabled;
00996
00997   _camera.projectionMatrix = oldCamera;
00998   glViewport(0,0,this->width(),this->height());
00999
01000   glBindFramebuffer(GL_FRAMEBUFFER,currentFBO);
01001
01002   glBindTexture(GL_TEXTURE_2D,renderedTexture);
01003   uchar* image = (uchar *) malloc(width * height * 4 * sizeof(uchar));
01004   glGetTexImage(GL_TEXTURE_2D,0,GL_RGB,GL_UNSIGNED_BYTE,image);
01005
01006   glBindTexture(GL_TEXTURE_2D,0);
01007   /*
01008   uchar* newImage = (uchar *) malloc(width * height * 3 * sizeof(uchar));
01009
01010   for (int i = 0; i < width * height ; i++)
01011   {
01012   int j = i * 3;
01013   int k = i * 4;
01014   newImage[j] = image[k];
01015   newImage[j+1] = image[k+1];
01016   newImage[j+2] = image[k+2];
01017   }
01018   */
01019   QImage screenShot(image,width,height,QImage::Format_RGB888);
01020   screenShot = screenShot.mirrored(false,true);
01021   QString fileName = QDateTime::currentDateTime().toString("yyyy-MM-dd_hh-mm-ss");
01022   qDebug() << fileName;
01023   screenShot.save(QString("%1.png").arg(fileName));
01024   free(image);
01025   //  free(newImage);
01026  }
01027
01028  glBindFramebuffer(GL_FRAMEBUFFER,currentFBO);
01029
01030  glDeleteBuffers(1, &renderBuffer);
01031  glDeleteTextures(1, &renderedTexture);
01032  glDeleteFramebuffers(1, &offscreenFBO);
01033
01034 }
01035 void RenderWidget::setupRenderStructs()
01036 {
01037  // setUpdateRate(_arena.fps);
01038  _dropletStruct.mesh = assets.getMesh(assets.
      lookupAssetName(DROPLET_MESH_NAME));
01039  _dropletStruct.baseShader = assets.getShader(
      assets.lookupAssetName(DROPLET_SHADER_NAME));
01040  _dropletStruct.projShader = assets.getShader(
      assets.lookupAssetName(DROPLET_PROJECTION_SHADER_NAME));
01041  _dropletStruct.texture_0 = assets.getTexture(
      assets.lookupAssetName(DROPLET_TEXTURE_NAME));
01042
01043  _objectStructs.clear();
01044  renderStruct_t sphereStruct;
01045  _objectStructs.append(sphereStruct);
```

```
01046
01047  _objectStructs[Sphere].mesh = assets.getMesh(
       assets.lookupAssetName(OBJECT_MESH_NAME));
01048  _objectStructs[Sphere].baseShader = assets.getShader(
       assets.lookupAssetName(OBJECT_SHADER_NAME));
01049  _objectStructs[Sphere].projShader = assets.getShader(
       assets.lookupAssetName(OBJECT_PROJECTION_SHADER_NAME));
01050  _objectStructs[Sphere].texture_0 = assets.getTexture(
       assets.lookupAssetName(OBJECT_TEXTURE_NAME));
01051
01052  renderStruct_t cubeStruct;
01053  _objectStructs.append(cubeStruct);
01054
01055  _objectStructs[Cube].mesh = assets.getMesh(
       assets.lookupAssetName(OBJECT_CUBE_MESH_NAME));
01056  _objectStructs[Cube].baseShader = assets.getShader(
       assets.lookupAssetName(OBJECT_CUBE_SHADER_NAME));
01057  _objectStructs[Cube].projShader = assets.getShader(
       assets.lookupAssetName(OBJECT_CUBE_PROJECTION_SHADER_NAME
       ));
01058  _objectStructs[Cube].texture_0 = assets.getTexture(
       assets.lookupAssetName(OBJECT_CUBE_TEXTURE_NAME));
01059
01060  renderStruct_t wallStruct;
01061  _objectStructs.append(wallStruct);
01062
01063  _objectStructs[Wall].mesh = assets.getMesh(
       assets.lookupAssetName(OBJECT_CUBE_MESH_NAME));
01064  _objectStructs[Wall].baseShader = assets.getShader(
       assets.lookupAssetName(WALL_SHADER_NAME));
01065  _objectStructs[Wall].projShader = assets.getShader(
       assets.lookupAssetName(OBJECT_CUBE_PROJECTION_SHADER_NAME
       ));
01066  _objectStructs[Wall].texture_0 = assets.getTexture(
       assets.lookupAssetName(WALL_TEXTURE_NAME));
01067
01068  renderStruct_t floorStruct;
01069  _objectStructs.append(floorStruct);
01070
01071  _objectStructs[Floor].mesh = assets.getMesh(
       assets.lookupAssetName(OBJECT_CUBE_MESH_NAME));
01072  _objectStructs[Floor].baseShader = assets.getShader(
       assets.lookupAssetName(FLOOR_SHADER_NAME));
01073  _objectStructs[Floor].projShader = assets.getShader(
       assets.lookupAssetName(OBJECT_CUBE_PROJECTION_SHADER_NAME
       ));
01074  _objectStructs[Floor].texture_0 = assets.getTexture(
       assets.lookupAssetName(FLOOR_TEXTURE_NAME));
01075
01076 }
01077
01078 /*
01079 void RenderWidget::setupArena()
01080 {
01081
01082 _arenaObjects.clear();
01083
01084 glm::vec4 grey = glm::vec4(0.5,0.5,0.5,1);
01085 float xTilePos, yTilePos;
01086
01087 // make floor tiles first
01088 xTilePos = (-_arena.tileLength * _arena.numColTiles / 2.0f) + (_arena.tileLength/2);
01089 yTilePos = (_arena.tileLength * _arena.numRowTiles / 2.0f) - (_arena.tileLength/2);
01090
01091 float xTexWidth,yTexWidth;
01092 xTexWidth = (1.0 / _arena.numColTiles);
01093 yTexWidth = (1.0 / _arena.numRowTiles);
01094
01095 // CREATE FLOOR TILES
01096 for(int i = 0; i < _arena.numRowTiles * _arena.numColTiles; i++)
01097 {
01098 float xTexOffset, yTexOffset;
01099 if(xTilePos >= (_arena.tileLength * _arena.numColTiles / 2.0f))
01100 {
01101 xTilePos = (-_arena.tileLength * _arena.numColTiles / 2.0f) + (_arena.tileLength/2);
01102 yTilePos -= _arena.tileLength;
01103 }
01104
01105 xTexOffset = (xTilePos - (_arena.tileLength / 2.0) + (_arena.tileLength * _arena.numColTiles / 2.0f)) /
       (_arena.tileLength * _arena.numColTiles);
01106 yTexOffset = (yTilePos - (_arena.tileLength / 2.0) + (_arena.tileLength * _arena.numRowTiles / 2.0f)) /
       (_arena.tileLength * _arena.numRowTiles);
01107 // qDebug() << xTexOffset << yTexOffset;
01108 renderStruct_t floor;
01109 floor.color = grey;
01110 //floor.color = glm::vec4(qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,1);
01111 floor.origin = glm::vec3(xTilePos, yTilePos, -0.05f);
```

```
01112 floor.scale = glm::vec3(_arena.tileLength, _arena.tileLength,  0.1f);
01113 floor.rotation = glm::vec3(0,0,0);
01114 floor.modelMatrix = makeModelMatrix(floor.scale,floor.rotation,floor.origin);
01115 floor.mesh = assets.getMesh(assets.lookupAssetName(FLOOR_MESH_NAME));
01116 floor.baseShader = assets.getShader(assets.lookupAssetName(FLOOR_SHADER_NAME));
01117 floor.projShader = assets.getShader(assets.lookupAssetName(FLOOR_PROJECTION_SHADER_NAME));
01118 floor.texture_0 = assets.getTexture(assets.lookupAssetName(FLOOR_TEXTURE_NAME));
01119 floor.texture_1 = assets.getProjection(_arena.projTexture);
01120 _arenaObjects.push_back(floor);
01121 xTilePos += _arena.tileLength;
01122
01123 }
01124
01125 //walls
01126 for(int j = 0; j < 2; j++)
01127 {
01128 if(j < 1)
01129 {
01130 yTilePos = -_arena.tileLength * _arena.numRowTiles / 2.0f + _arena.tileLength/2.0f;
01131 xTilePos = -_arena.tileLength * _arena.numColTiles / 2.0f + _arena.tileLength/2.0f;
01132 }
01133 else
01134 {
01135 yTilePos = _arena.tileLength * _arena.numRowTiles / 2.0f - _arena.tileLength/2.0f;
01136 xTilePos = _arena.tileLength * _arena.numColTiles / 2.0f - _arena.tileLength/2.0f;
01137 }
01138
01139 for(int i = 0; i < _arena.numRowTiles; i++)
01140 {
01141 float xPos;
01142 if (j < 1)
01143 {
01144 xPos = -_arena.tileLength * _arena.numColTiles/2.0f - _arena.wallWidth/2.0f;
01145 }
01146 else{
01147 xPos = _arena.tileLength * _arena.numColTiles/2.0f + _arena.wallWidth/2.0f;
01148 }
01149 float zpos = (.5 * _arena.wallHeight) - (.5 * 1) - .5;
01150
01151 renderStruct_t wall;
01152 // wall.color = glm::vec4(qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,1);
01153 wall.color = grey;
01154 wall.origin = glm::vec3(xPos,yTilePos,zpos);
01155 wall.scale = glm::vec3(_arena.wallWidth,_arena.tileLength,_arena.wallHeight);
01156 wall.rotation = glm::vec3(0,0,0);
01157 wall.modelMatrix = makeModelMatrix(wall.scale,wall.rotation,wall.origin);
01158
01159 wall.mesh = assets.getMesh(assets.lookupAssetName(WALL_MESH_NAME));
01160 wall.baseShader = assets.getShader(assets.lookupAssetName(WALL_SHADER_NAME));
01161 wall.projShader = assets.getShader(assets.lookupAssetName(WALL_SHADER_NAME));
01162 wall.texture_0 = assets.getTexture(assets.lookupAssetName(WALL_TEXTURE_NAME));
01163 wall.texture_1 = NULL;
01164
01165
01166
01167 _arenaObjects.push_back(wall);
01168
01169
01170 if(j < 1) yTilePos += _arena.tileLength; else yTilePos -= _arena.tileLength;
01171 }
01172 for(int i = 0; i < _arena.numColTiles; i++)
01173 {
01174 float yPos;
01175 if (j < 1)
01176 {
01177 yPos = _arena.tileLength * _arena.numRowTiles/2.0f + _arena.wallWidth/2.0f;
01178 }
01179 else{
01180 yPos = -_arena.tileLength * _arena.numRowTiles/2.0f - _arena.wallWidth/2.0f;
01181 }
01182 float zpos = (.5 * _arena.wallHeight) - (.5 * 1) -.5;
01183
01184 renderStruct_t wall;
01185 //wall.color = glm::vec4(qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,1);
01186 wall.color = grey;
01187 wall.origin = glm::vec3(xTilePos,yPos,zpos);
01188 wall.scale = glm::vec3(_arena.tileLength,_arena.wallWidth,_arena.wallHeight);
01189 wall.rotation = glm::vec3(0,0,0);
01190 wall.modelMatrix = makeModelMatrix(wall.scale,wall.rotation,wall.origin);
01191 wall.mesh = assets.getMesh(assets.lookupAssetName(WALL_MESH_NAME));
01192 wall.baseShader = assets.getShader(assets.lookupAssetName(WALL_SHADER_NAME));
01193 wall.projShader = assets.getShader(assets.lookupAssetName(WALL_SHADER_NAME));
01194 wall.texture_0 = assets.getTexture(assets.lookupAssetName(WALL_TEXTURE_NAME));
01195 wall.texture_1 = NULL;
01196
01197 _arenaObjects.push_back(wall);
01198
```

```
01199
01200 if(j < 1) xTilePos += _arena.tileLength; else xTilePos -= _arena.tileLength;
01201 }
01202 }
01203
01204
01205
01206
01207
01208
01209 float xPos;
01210 float yPos;
01211 float yRot;
01212 float mult;
01213 // Create four IR Towers
01214 for(int i = 0; i < 4; i++)
01215 {
01216 yRot = 0;
01217 mult = -1;
01218 if(i % 2 == 0)
01219 {
01220 xPos = -_arena.tileLength * _arena.numColTiles/2.0f - 6;
01221 yRot += 180;
01222 }
01223 else
01224 {
01225 xPos = _arena.tileLength * _arena.numColTiles/2.0f + 6;
01226 mult = 1;
01227 }
01228 if(i / 2 == 0)
01229 {
01230 yPos = -_arena.tileLength * _arena.numRowTiles/2.0f - 6;
01231 yRot -= mult * 45;
01232 }
01233 else
01234 {
01235 yPos = _arena.tileLength * _arena.numRowTiles/2.0f + 6;
01236 yRot += mult * 45;
01237 }
01238
01239 renderStruct_t tower;
01240 tower.color = grey;
01241 //   tower.color = glm::vec4(qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,qrand()/(float) RAND_MAX,1);
01242 tower.origin = glm::vec3(xPos, yPos, 0.0);
01243 tower.scale = glm::vec3(2.0f,2.0f,2.0f);
01244 tower.rotation = glm::vec3(0,0,yRot);
01245 tower.modelMatrix = makeModelMatrix(tower.scale,tower.rotation,tower.origin);
01246 tower.mesh = assets.getMesh(assets.lookupAssetName(TOWER_MESH_NAME));
01247 tower.baseShader = assets.getShader(assets.lookupAssetName(TOWER_SHADER_NAME));
01248 tower.projShader = assets.getShader(assets.lookupAssetName(TOWER_SHADER_NAME));
01249 tower.texture_0 = assets.getTexture(assets.lookupAssetName(TOWER_TEXTURE_NAME));
01250 tower.texture_1 = NULL;
01251 _arenaObjects.push_back(tower);
01252
01253 }
01254
01255 }
01256
01257 */
01258
01259 // generate a model matrix
01260 glm::mat4 RenderWidget::makeModelMatrix(glm::vec3 scale, glm::vec3 rotate,
      glm::vec3 translation)
01261 {
01262  glm::mat4 model =  glm::translate(glm::mat4(1.0f),translation);
01263  model = glm::rotate(model,rotate.x,glm::vec3(1,0,0));
01264  model = glm::rotate(model,rotate.y,glm::vec3(0,1,0));
01265  model = glm::rotate(model,rotate.z,glm::vec3(0,0,1));
01266  model = glm::scale(model,scale);
01267  return model;
01268 }
01269
01270
01271
01272
01273 void RenderWidget::timerEvent ( QTimerEvent * event )
01274 {
01275  qint64 nsElapsed = _renderTimer.nsecsElapsed();
01276  float msElapsedSinceRender = nsElapsed / 1000000.0;
01277
01278  if (isVisible())
01279  {
01280   // restart the timer
01281   _renderTimer.start();
01282
01283   // qApp->processEvents();
01284   _eventTimer.start();
```

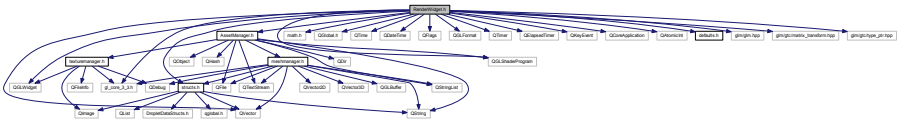
```
01285
01286   processInput(msElapsedSinceRender);
01287
01288   // event updating goes here
01289
01290   // update the simulator at a fixed step
01291   updateCamera();
01292
01293   nsElapsed = _eventTimer.nsecsElapsed();
01294   _msElapsedUpdate = nsElapsed / 1000000.0;
01295
01296   // qApp->processEvents();
01297   if (!_renderLock )
01298   {
01299    updateGL();
01300   }
01301   // qApp->processEvents();
01302
01303  }
01304
01305 }
01306
01307 void RenderWidget::closeEvent(QCloseEvent *event)
01308 {
01309  if (_timerID != 0)
01310  {
01311   killTimer(_timerID);
01312   _timerID = 0;
01313  }
01314  _renderLock = true;
01315  deleteLater();
01316  event->accept();
01317 }
01318
01319 float RenderWidget::getRandomf(float min, float max)
01320 {
01321  uint randNum = rand();
01322  float range = max - min;
01323  return (min + ((range * randNum) / RAND_MAX));
01324 }
```

13.24 RenderWidget.h File Reference

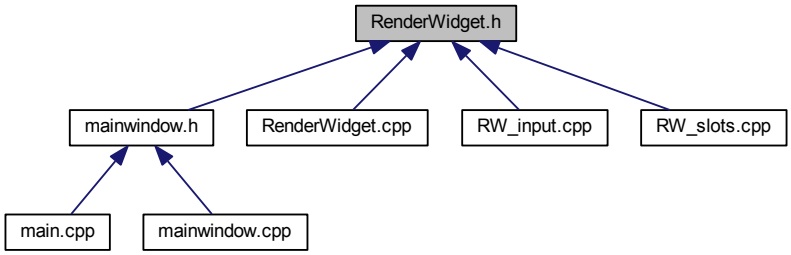
Declares the RenderWidget class.

```
#include "gl_core_3_3.h"
#include <math.h>
#include <QGlobal.h>
#include <QTime>
#include <QDateTime>
#include <QVector>
#include <QFlags>
#include <QGLFormat>
#include <QTimer>
#include <QGLWidget>
#include <QGLShaderProgram>
#include <QElapsedTimer>
#include <QKeyEvent>
#include <QCoreApplication>
#include <QAtomicInt>
#include "defaults.h"
#include "structs.h"
#include "AssetManager.h"
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

Include dependency graph for RenderWidget.h:



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



Classes

- class RenderWidget

  View controller for QGLWidget that renders the simulation state.
- struct RenderWidget::renderStruct_t

  Defines a structure that represents a renderable object.

Macros

- #define _USE_MATH_DEFINES

13.24.1 Detailed Description

Declares the RenderWidget class.

Definition in file RenderWidget.h.

13.24.2 Macro Definition Documentation

13.24.2.1 #define _USE_MATH_DEFINES

Definition at line 10 of file RenderWidget.h.

13.25 RenderWidget.h

```
00001
00007 #ifndef RENDERWIDGET_H
00008 #define RENDERWIDGET_H
00009
00010 #define _USE_MATH_DEFINES
00011
```

```
00012 #include "gl_core_3_3.h"
00013
00014 #include <math.h>
00015 #include <QGlobal.h>
00016 #include <QTime>
00017 #include <QDateTime>
00018 #include <QVector>
00019 #include <QFlags>
00020 #include <QGLFormat>
00021 #include <QTimer>
00022 #include <QGLWidget>
00023 #include <QGLShaderProgram>
00024 //#include <QGLFunctions>
00025 #include <QElapsedTimer>
00026 #include <QKeyEvent>
00027 #include <QCoreApplication>
00028 #include <QAtomicInt>
00029
00030 #include "defaults.h"
00031 #include "structs.h"
00032 //#include "SimInterface.h"
00033 #include "AssetManager.h"
00034
00035 #include <glm/glm.hpp>
00036 #include <glm/gtc/matrix_transform.hpp>
00037 #include <glm/gtc/type_ptr.hpp>
00038
00051 class RenderWidget : public QGLWidget
00052 {
00053  Q_OBJECT
00054
00055 public:
00056
00066  RenderWidget(const QGLFormat& format, QWidget *parent = 0);
00067
00074  ~RenderWidget();
00075
00084  QSize sizeHint() const;
00085
00086  public slots:
00096   void updateState(simState_t stateInfo);
00097
00106   void updatePause(bool paused);
00107
00116   void updateArena(simSetting_t settings);
00117
00126   void updateRate(simRate_t rates);
00127
00128
00129 signals:
00136   void togglePause(void);
00137
00144   void restart(void);
00145
00158   void requestNewDroplet(float x, float y, droplet_t dType =
      March, int dropletID = 0);
00159
00166   void toggleLimit(void);
00167
00174   void increaseRate(void);
00175
00182   void decreaseRate(void);
00183
00184 protected:
00185
00192  void initializeGL();
00193
00203  void resizeGL(int width, int height);
00204
00211  void paintGL();
00212
00219  void updateCamera();
00220
00227  void drawHUD();
00228
00235  void drawArena();
00236
00243  void drawDroplets();
00244
00251  void drawObjects();
00252
00261  void timerEvent(QTimerEvent *event);
00262
00271  void setFPS(int FPS);
00272
00279  void setupRenderStructs();
00280  //void setupArena();
```

```
00281
00290  void wheelEvent(QWheelEvent *event);
00291
00300  void keyPressEvent(QKeyEvent *event);
00301
00310  void keyReleaseEvent(QKeyEvent *event);
00311
00320  void processInput(float timeSinceLastUpdate);
00321
00330  void mouseMoveEvent ( QMouseEvent * event );
00331
00340  void mousePressEvent ( QMouseEvent * event );
00341
00350  void mouseReleaseEvent ( QMouseEvent * event );
00351
00360  void closeEvent(QCloseEvent *event);
00361
00373  float getRandomf(float min, float max);
00374
00375
00376 private:
00383   struct renderStruct_t {
00384    GLuint vertCount;
00385    GLuint vaoID;
00386    QGLShaderProgram *baseShader;
00387    QGLShaderProgram *projShader;
00388    MeshManager *mesh;
00389    TextureManager *texture_0;
00390    // TextureManager *texture_1;
00391
00392    glm::vec4 color;
00393    glm::vec3 scale;
00394    glm::vec3 rotation;
00395    glm::vec3 origin;
00396    glm::mat4 modelMatrix;
00397   };
00398
00412  glm::mat4 makeModelMatrix(glm::vec3 scale, glm::vec3 rotate, glm::vec3 translation);
00413
00414
00415  void saveScreenShot(int width, int height);
00420  AssetManager assets;
00421
00426  QElapsedTimer _renderTimer;
00427
00433  QElapsedTimer _eventTimer;
00434
00443  double _msElapsedRenderer, _msElapsedUpdate;
00444
00449  int _targetFPS;
00450
00455  float _targetFrameTime;
00456
00461  bool _renderLock;
00462
00467  bool _assetsLoaded;
00468
00473  bool _hud;
00474
00485  int _renderDebug;
00486
00491  bool _drawHelp;
00492
00497  QElapsedTimer _updateTimer;
00498
00503  int _timerID;
00504
00509  QVector<renderStruct_t> _arenaObjects;
00510
00515  renderStruct_t _dropletStruct;
00516
00521  QVector<renderStruct_t> _objectStructs;
00522
00526  struct {
00527   int framesSinceLastUpdate;
00528   float simRealTimeRatio;
00529   float simStepSize;
00530   bool paused;
00531  } _hudInfo;
00532
00536  struct {
00537   QMatrix4x4 projectionMatrix;
00538   QMatrix4x4 viewMatrix;
00539   GLfloat x,y,z;
00540   GLfloat pan, tilt;
00541   GLfloat rotHoriz,rotVert;
00542   GLfloat radius;
```

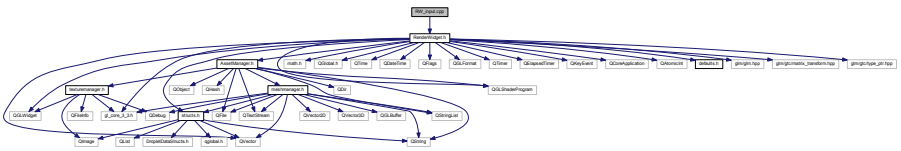
```
00543   unsigned int mode;
00544   glm::vec3 lightDir;
00545  } _camera;
00546
00550  struct {
00551   bool leftButtonHeldDown;
00552   int startX, startY;
00553   GLfloat origHoriz;
00554   GLfloat origVert;
00555   GLfloat origPan;
00556   GLfloat origTilt;
00557  } _mouseStatus;
00558
00559
00563  struct {
00564   bool Q;
00565   bool W;
00566   bool E;
00567   bool A;
00568   bool S;
00569   bool D;
00570   bool Minus;
00571   bool Plus;
00572  } _keysDown;
00573
00574  struct {
00575   GLuint width, height;
00576   GLuint handle;
00577   bool valid;
00578   QImage texture;
00583  } _projectionTexture;
00584
00589  simSetting_t _arena;
00590
00595  simState_t _simState;
00596   simState_t _renderState;
00597
00602  QAtomicInt _simStateLock;
00607  simRate_t _simRates;
00608 };
00609
00610 #endif // RENDERWIDGET_H
```

13.26 RW_input.cpp File Reference

Implements the input handling for the RenderWidget class.

```
#include "RenderWidget.h"
```

Include dependency graph for RW_input.cpp:



13.26.1 Detailed Description

Implements the input handling for the RenderWidget class.

Definition in file RW_input.cpp.

13.27 RW_input.cpp

```
00001
00007 #include "RenderWidget.h"
00008
00009 // handle mouse movement
00010 void RenderWidget::mouseMoveEvent ( QMouseEvent * event )
00011 {
00012  if (event->buttons().testFlag(Qt::LeftButton) && _mouseStatus.leftButtonHeldDown)
```

```
00013  {
00014   if (_camera.mode == 0)
00015   {
00016    _camera.rotHoriz = _mouseStatus.origHoriz + (event->x() -
      _mouseStatus.startX) / 2.0;
00017    _camera.rotVert = _mouseStatus.origVert - (event->y() -
      _mouseStatus.startY) / 2.0;
00018   } else if (_camera.mode == 1) {
00019    _camera.pan = _mouseStatus.origPan - (event->x() -
      _mouseStatus.startX) / 2.0;
00020    _camera.tilt =_mouseStatus.origTilt - (event->y() -
      _mouseStatus.startY) / 2.0;
00021   }
00022  }
00023  event->accept();
00024 }
00025
00026 // handle mouse button pressed
00027 void RenderWidget::mousePressEvent ( QMouseEvent * event )
00028 {
00029  if (event->buttons().testFlag(Qt::LeftButton))
00030  {
00031   _mouseStatus.leftButtonHeldDown = true;
00032   _mouseStatus.startX = event->x();
00033   _mouseStatus.startY = event->y();
00034   _mouseStatus.origHoriz = _camera.rotHoriz;
00035   _mouseStatus.origVert = _camera.rotVert;
00036   _mouseStatus.origPan = _camera.pan;
00037   _mouseStatus.origTilt = _camera.tilt;
00038
00039   setCursor(QCursor(Qt::BlankCursor));
00040  }
00041  event->accept();
00042 }
00043
00044 // handle releasing the mouse button
00045 void RenderWidget::mouseReleaseEvent ( QMouseEvent * event )
00046 {
00047  if (!event->buttons().testFlag(Qt::LeftButton))
00048  {
00049   if (_camera.mode == 0)
00050   {
00051    _camera.rotHoriz = _mouseStatus.origHoriz + (event->x() -
      _mouseStatus.startX) / 2.0;
00052    _camera.rotVert = _mouseStatus.origVert - (event->y() -
      _mouseStatus.startY) / 2.0;
00053   } else if (_camera.mode == 1) {
00054    _camera.pan = _mouseStatus.origPan - (event->x() -
      _mouseStatus.startX) / 2.0;
00055    _camera.tilt =_mouseStatus.origTilt - (event->y() -
      _mouseStatus.startY) / 2.0;
00056   }
00057   _mouseStatus.leftButtonHeldDown = false;
00058   setCursor(QCursor(Qt::ArrowCursor));
00059  }
00060  event->accept();
00061 }
00062
00063 // event to handle scrolling with scroll wheel
00064 void RenderWidget::wheelEvent(QWheelEvent *event)
00065 {
00066
00067  float numDegrees = event->delta() / 8;
00068  float numSteps = numDegrees / 15;
00069
00070  if (event->orientation() == Qt::Vertical && _camera.mode == 0) {
00071 #ifdef __APPLE__
00072   // compensate for apple's inverted scrolling
00073   _camera.radius += numSteps * 2.0;
00074 #else
00075   // everyone else behaves correctly
00076   _camera.radius -= numSteps * 2.0;
00077 #endif
00078  }
00079  event->accept();
00080 }
00081
00082
00083 void RenderWidget::keyPressEvent(QKeyEvent *event)
00084 {
00085  if (!event->isAutoRepeat())
00086  {
00087   if(event->modifiers().testFlag(Qt::ControlModifier))
00088   {
00089    if(event->key() == Qt::Key_H)
00090    {
00091     _hud = !_hud;
```

```
00092
00093     } else if(event->key() == Qt::Key_R)
00094     {
00095      emit restart();
00096      // TIMER
00097      //_runTime.start();
00098
00099     } else if(event->key() == Qt::Key_L)
00100     {
00101      _renderLock = true;
00102      assets.reloadAssets();
00103      setupRenderStructs();
00104      //    setupArena();
00105
00106      _renderLock = false;
00107     } else if(event->key() == Qt::Key_B)
00108     {
00109      if (_renderDebug == 2)
00110      {
00111       _renderDebug = 0;
00112      } else {
00113       _renderDebug++;
00114      }
00115     }
00116    } else {
00117     if(event->key() == Qt::Key_Escape)
00118     {
00119      qApp->processEvents();
00120      close();
00121      makeCurrent();
00122      assets.clearAssets();
00123      doneCurrent();
00124     } else if(event->key() == Qt::Key_P)
00125     {
00126      emit togglePause();
00127     } else if(event->key() == Qt::Key_L)
00128     {
00129      emit toggleLimit();
00130     } else if(event->key() == Qt::Key_H)
00131     {
00132      _drawHelp = !_drawHelp;
00133     } else if(event->key() == Qt::Key_Q)
00134     {
00135      _keysDown.Q = true;
00136     } else if(event->key() == Qt::Key_W)
00137     {
00138      _keysDown.W = true;
00139     } else if(event->key() == Qt::Key_E)
00140     {
00141      _keysDown.E = true;
00142     } else if(event->key() == Qt::Key_A)
00143     {
00144      _keysDown.A = true;
00145     } else if(event->key() == Qt::Key_S)
00146     {
00147      _keysDown.S = true;
00148     } else if(event->key() == Qt::Key_D)
00149     {
00150      _keysDown.D = true;
00151     } else if(event->key() == Qt::Key_Plus)
00152     {
00153      _keysDown.Plus = true;
00154     } else if(event->key() == Qt::Key_Minus)
00155     {
00156      _keysDown.Minus = true;
00157     } else if(event->key() == Qt::Key_Space)
00158     {
00159      if (_camera.mode == 1)
00160      {
00161       _camera.mode = 0;
00162       _mouseStatus.origHoriz = _camera.rotHoriz;
00163       _mouseStatus.origVert = _camera.rotVert;
00164
00165      } else
00166      {
00167       _camera.mode++;
00168       _mouseStatus.origPan = _camera.pan;
00169       _mouseStatus.origTilt = _camera.tilt;
00170      }
00171     } else if(event->key() == Qt::Key_Backslash)
00172     {
00173      makeCurrent();
00174      saveScreenShot(SCREENSHOT_WIDTH,
      SCREENSHOT_HEIGHT);
00175      assets.reloadAssets();
00176      setupRenderStructs();
00177      doneCurrent();
```

```
00178    }
00179
00180   }
00181
00182  }
00183
00184  if(event->key() == Qt::Key_BracketRight)
00185  {
00186   if (_simRates.limitRate)
00187    emit decreaseRate();
00188  }
00189  if(event->key() == Qt::Key_BracketLeft)
00190  {
00191   if (_simRates.limitRate)
00192    emit increaseRate();
00193  }
00194
00195  /* PENDING DELETION */
00196  if(event->key() == Qt::Key_Z)
00197  {
00198   float floorWidth = _arena.tileLength * _arena.
     numColTiles;
00199   float floorLength = _arena.tileLength * _arena.
     numRowTiles;
00200   float posRangeWidth = floorWidth / 2.0f;
00201   float posRangeLength = floorLength / 2.0f;
00202   float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth
      - _arena.dropletRadius);
00203   float yPos = getRandomf(-posRangeLength + _arena.dropletRadius,
     posRangeLength - _arena.dropletRadius);
00204   emit requestNewDroplet(xPos,yPos,StickPullers);
00205
00206  }
00207
00208  /* PENDING DELETION */
00209  if(event->key() == Qt::Key_X)
00210  {
00211   float floorWidth = _arena.tileLength * _arena.
     numColTiles;
00212   float floorLength = _arena.tileLength * _arena.
     numRowTiles;
00213   float posRangeWidth = floorWidth / 2.0f;
00214   float posRangeLength = floorLength / 2.0f;
00215   float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth
      - _arena.dropletRadius);
00216   float yPos = getRandomf(-posRangeLength + _arena.dropletRadius,
     posRangeLength - _arena.dropletRadius);
00217   emit requestNewDroplet(xPos,yPos,RGBSense);
00218
00219  }
00220
00221
00222  /* PENDING DELETION */
00223
00224  //if(event->key() == Qt::Key_C)
00225  //{
00226  // float floorWidth = _arena.tileLength * _arena.numColTiles;
00227  // float floorLength = _arena.tileLength * _arena.numRowTiles;
00228  // float posRangeWidth = floorWidth / 2.0f;
00229  // float posRangeLength = floorLength / 2.0f;
00230  // float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth - _arena.dropletRadius);
00231  // float yPos = getRandomf(-posRangeLength + _arena.dropletRadius, posRangeLength - _arena.dropletRadius);
00232  // emit requestNewDroplet(xPos,yPos,March);
00233
00234  //}
00235
00236  /* PENDING DELETION */
00237  //if(event->key() == Qt::Key_V)
00238  //{
00239  // float floorWidth = _arena.tileLength * _arena.numColTiles;
00240  // float floorLength = _arena.tileLength * _arena.numRowTiles;
00241  // float posRangeWidth = floorWidth / 2.0f;
00242  // float posRangeLength = floorLength / 2.0f;
00243  // float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth - _arena.dropletRadius);
00244  // float yPos = getRandomf(-posRangeLength + _arena.dropletRadius, posRangeLength - _arena.dropletRadius);
00245  // emit requestNewDroplet(xPos,yPos,RandomWalk);
00246
00247  //}
00248  /* PENDING DELETION */
00249  //if(event->key() == Qt::Key_B)
00250  //{
00251  // float floorWidth = _arena.tileLength * _arena.numColTiles;
00252  // float floorLength = _arena.tileLength * _arena.numRowTiles;
00253  // float posRangeWidth = floorWidth / 2.0f;
00254  // float posRangeLength = floorLength / 2.0f;
00255  // float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth - _arena.dropletRadius);
00256  // float yPos = getRandomf(-posRangeLength + _arena.dropletRadius, posRangeLength - _arena.dropletRadius);
```

```
00257  // emit requestNewDroplet(xPos,yPos,Rainbow);
00258
00259  //}
00260  /* PENDING DELETION */
00261  //if(event->key() == Qt::Key_N)
00262  //{
00263  // float floorWidth = _arena.tileLength * _arena.numColTiles;
00264  // float floorLength = _arena.tileLength * _arena.numRowTiles;
00265  // float posRangeWidth = floorWidth / 2.0f;
00266  // float posRangeLength = floorLength / 2.0f;
00267  // float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth - _arena.dropletRadius);
00268  // float yPos = getRandomf(-posRangeLength + _arena.dropletRadius, posRangeLength - _arena.dropletRadius);
00269  // emit requestNewDroplet(xPos,yPos,TurnTest);
00270
00271  //}
00272  /* PENDING DELETION */
00273  //if(event->key() == Qt::Key_M)
00274  //{
00275  // float floorWidth = _arena.tileLength * _arena.numColTiles;
00276  // float floorLength = _arena.tileLength * _arena.numRowTiles;
00277  // float posRangeWidth = floorWidth / 2.0f;
00278  // float posRangeLength = floorLength / 2.0f;
00279  // float xPos = getRandomf(-posRangeWidth + _arena.dropletRadius, posRangeWidth - _arena.dropletRadius);
00280  // float yPos = getRandomf(-posRangeLength + _arena.dropletRadius, posRangeLength - _arena.dropletRadius);
00281  // emit requestNewDroplet(xPos,yPos,CommTest);
00282
00283  //}
00284
00285 }
00286
00287 void RenderWidget::keyReleaseEvent(QKeyEvent *event)
00288 {
00289
00290  if (!event->isAutoRepeat())
00291  {
00292   if(!event->modifiers().testFlag(Qt::ControlModifier))
00293   {
00294    if(event->key() == Qt::Key_Q)
00295    {
00296     _keysDown.Q = false;
00297    }
00298    if(event->key() == Qt::Key_W)
00299    {
00300     _keysDown.W = false;
00301    }
00302    if(event->key() == Qt::Key_E)
00303    {
00304     _keysDown.E = false;
00305    }
00306    if(event->key() == Qt::Key_A)
00307    {
00308     _keysDown.A = false;
00309    }
00310    if(event->key() == Qt::Key_S)
00311    {
00312     _keysDown.S = false;
00313    }
00314    if(event->key() == Qt::Key_D)
00315    {
00316     _keysDown.D = false;
00317    }
00318    if(event->key() == Qt::Key_Plus)
00319    {
00320     _keysDown.Plus = false;
00321    }
00322    if(event->key() == Qt::Key_Minus)
00323    {
00324     _keysDown.Minus = false;
00325    }
00326   }
00327  }
00328 }
00329
00330 void RenderWidget::processInput(float timeSinceLastUpdate)
00331 {
00332
00333  float perFrame = timeSinceLastUpdate / 1000.0;
00334  if (_camera.mode == 0)
00335  {
00336   if ((_keysDown.Q || _keysDown.Minus) && (!_keysDown.E && !
      _keysDown.Plus))
00337   {
00338    _camera.radius += 100 * perFrame;
00339   } else if ((_keysDown.E || _keysDown.Plus) && (!_keysDown.Q && !
      _keysDown.Minus))
00340   {
00341    _camera.radius -= 100 * perFrame;
```

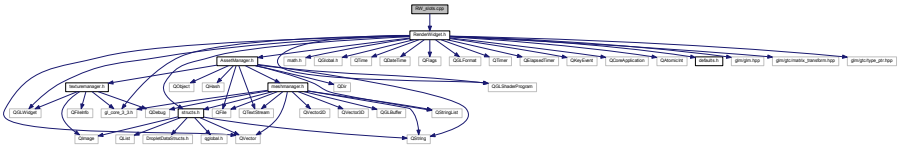
```
00342   }
00343   if (_keysDown.W && !_keysDown.S)
00344   {
00345    _camera.rotVert -= 45  * perFrame;
00346   } else if (_keysDown.S && !_keysDown.W)
00347   {
00348    _camera.rotVert += 45 * perFrame;
00349   }
00350   if (_keysDown.A && !_keysDown.D)
00351   {
00352    _camera.rotHoriz += 90 * perFrame;
00353   } else if (_keysDown.D && !_keysDown.A)
00354   {
00355    _camera.rotHoriz -= 90 * perFrame;
00356   }
00357  } else  if (_camera.mode == 1)
00358  {
00359   if (_keysDown.Q && !_keysDown.E)
00360   {
00361            _camera.z -= 50 * perFrame;
00362   } else if (_keysDown.E && !_keysDown.Q)
00363   {
00364            _camera.z += 50 * perFrame;
00365   }
00366   if (_keysDown.W && !_keysDown.S)
00367   {
00368    _camera.x -= sin(_camera.pan * M_PI / 180) * 50 * perFrame;
00369    _camera.y -= -cos(_camera.pan * M_PI / 180) * 50 * perFrame;
00370
00371
00372   } else if (_keysDown.S && !_keysDown.W)
00373   {
00374    _camera.x += sin(_camera.pan * M_PI / 180) * 50 * perFrame;
00375    _camera.y += -cos(_camera.pan * M_PI / 180) * 50 * perFrame;
00376   }
00377         if (_keysDown.A && !_keysDown.D)
00378         {
00379             //   _camera.pan += 90  * perFrame;
00380
00381             _camera.x -= cos(_camera.pan * M_PI / 180) * 50 * perFrame;
00382             _camera.y -= sinf(_camera.pan * M_PI / 180) * 50 * perFrame;
00383
00384
00385         } else if (_keysDown.D && !_keysDown.A)
00386         {
00387             //  _camera.pan -= 90 * perFrame;
00388
00389             _camera.x += cos(_camera.pan * M_PI / 180) * 50 * perFrame;
00390             _camera.y += sinf(_camera.pan * M_PI / 180) * 50 * perFrame;
00391
00392         }
00393  }
00394
00395 }
```

13.28 RW_slots.cpp File Reference

Implements the slots belonging to the RenderWidget class.

```
#include "RenderWidget.h"
```

Include dependency graph for RW_slots.cpp:



13.28.1 Detailed Description

Implements the slots belonging to the RenderWidget class.

Definition in file RW_slots.cpp.

13.29 RW_slots.cpp

```
00001
00007 #include "RenderWidget.h"
00008
00009
00010 void RenderWidget::updateState(simState_t stateInfo)
00011 {
00012   // block while resource is in use, then retain resource
00013  while(!_simStateLock.testAndSetOrdered(0,1)) {}
00014
00015  // update shared variable
00016  _simState = stateInfo;
00017
00018  // release resource
00019  _simStateLock.fetchAndStoreOrdered(0);
00020
00021 }
00022
00023 // slot that updates the arena configuration
00024 void RenderWidget::updateArena(simSetting_t settings)
00025 {
00026  _arena = settings;
00027  _hudInfo.simStepSize = 1000.0 / _arena.fps;
00028  _hudInfo.simStepSize = floor(_hudInfo.simStepSize *100.0 + 0.5) / 100.0;
00029  setupRenderStructs();
00030  // setupArena();
00031 }
00032
00033 // slot that updates the paused status
00034 void RenderWidget::updatePause(bool paused)
00035 {
00036  _hudInfo.paused = paused;
00037 }
00038
00039 // slot that updates time ratio information
00040 // in theory, this could be passed along as part of updateState and all would work well
00041 // however, this allows the timing information to change asychronously from the actual simulator state
00042 // meaning it can be adjusted while the simulator is paused and not issuing updates
00043 void RenderWidget::updateRate(simRate_t rates)
00044 {
00045  _simRates = rates;
00046
00047 }
```

13.30 simInfoLogger.cpp File Reference

Implements the simInfoLogger class.

`#include "simInfoLogger.h"`
Include dependency graph for simInfoLogger.cpp:

13.30.1 Detailed Description

Implements the simInfoLogger class.

Definition in file simInfoLogger.cpp.

13.31 simInfoLogger.cpp

```
00001
00007 #include "simInfoLogger.h"
00008
00009 simInfoLogger::simInfoLogger(QObject *parent)
00010  : QObject(parent)
00011 {
00012  timeInterval = NULL;
00013  lastPrint = NULL;
00014  posFlag = NULL;
00015  colorFlag = NULL;
```

```
00016  rotationFlag = NULL;
00017  commSAFlag = NULL;
00018  macroRedFlag = NULL;
00019  macroSAFlag = NULL;
00020  QString newFile = NULL;
00021 }
00022 void simInfoLogger::Init()
00023 {
00024  newFile = QString(DEFAULT_ASSETDIR).append("output.txt");
00025     fp = fopen (newFile.toStdString().c_str(),"w");
00026  if (fp==NULL)
00027  {
00028   printf("Cannot create output file\n");
00029  }
00030  timeInterval = .5;
00031  //lastPrint = -timeInterval - 1;
00032     lastPrint = 0;
00033
00034  if(macroRedFlag)
00035  {
00036   redTally = 0;
00037  }
00038  if(macroSAFlag)
00039  {
00040   SATally = 0;
00041  }
00042
00043  // Messages printed one at the top of the file
00044  fprintf(fp, "Format:\n");
00045  fprintf(fp, "Droplet#: ");
00046  if(posFlag)
00047  {
00048   fprintf(fp,"(xPos,yPos,zPos)  |  ");
00049  }
00050  if(colorFlag)
00051  {
00052   fprintf(fp,"(rColor,gColor,bColor)  |  ");
00053  }
00054  if(rotationFlag)
00055  {
00056   fprintf(fp,"(xRot,yRot,zRot)  |  ");
00057  }
00058  if(commSAFlag)
00059  {
00060   fprintf(fp,"SAbool  |  ");
00061  }
00062  fprintf(fp, "\n\n");
00063 }
00064
00065 simInfoLogger::~simInfoLogger()
00066 {
00067
00068 }
00069
00070 void simInfoLogger::close()
00071 {
00072  // If newFile == NULL, then Init has not been called.
00073  // and thus a file has not been opened.
00074  if(newFile != NULL)
00075  {
00076   printf("File had been opened");
00077   fclose(fp);
00078  }
00079 }
00080
00081 void simInfoLogger::setPosFlag(bool flag)
00082 {
00083  posFlag = flag;
00084 }
00085 void simInfoLogger::setColorFlag(bool flag)
00086 {
00087  colorFlag = flag;
00088 }
00089 void simInfoLogger::setRotationFlag(bool flag)
00090 {
00091  rotationFlag = flag;
00092 }
00093 void simInfoLogger::setCommSAFlag(bool flag)
00094 {
00095  commSAFlag = flag;
00096 }
00097 void simInfoLogger::setMacroRedFlag(bool flag)
00098 {
00099  macroRedFlag = flag;
00100 }
00101 void simInfoLogger::setMacroSAFlag(bool flag)
00102 {
```

```
00103  macroSAFlag = flag;
00104 }
00105
00106 void simInfoLogger::printDropletData(simState_t stateInfo)
00107 {
00108  if(macroRedFlag){
00109   redTally = 0;
00110  }
00111  if(macroSAFlag){
00112   SATally = 0;
00113  }
00114
00115  // Messages printed at the start of each step
00116  fprintf(fp, "Simiulation time: %.3f | Real time: %.3f\n", stateInfo.simTime, stateInfo.
      realTime);
00117  fprintf(fp, "-----------------------------------------------\n");
00118
00119  // Messages printed once per droplet per step
00120  foreach(dropletStruct_t droplet, stateInfo.dropletData)
00121  {
00122   fprintf(fp,"%i: ", droplet.dropletID);
00123   if(posFlag)
00124   {
00125    fprintf(fp,"(%3.4f,%3.4f,%3.4f)  |  ",droplet.origin.x,droplet.
      origin.y,droplet.origin.z);
00126   }
00127   if(colorFlag)
00128   {
00129    fprintf(fp,"(%03i,%03i,%03i)  |  ",droplet.color.r,droplet.color.
      g,droplet.color.b);
00130    if(macroRedFlag && droplet.color.r > 240 && droplet.color.
      g < 15 && droplet.color.b < 15){
00131     redTally++;
00132    }
00133   }
00134   if(rotationFlag)
00135   {
00136    fprintf(fp,"(%3.4f,%3.4f,%3.4f)  |  ",droplet.rotation.x,droplet.
      rotation.y,droplet.rotation.z);
00137   }
00138   if(commSAFlag)
00139   {
00140    if(droplet.commData.sendActive)
00141    {
00142     fprintf(fp,"TRUE  |  ");
00143     if(macroSAFlag){
00144      SATally++;
00145     }
00146    }
00147    else
00148    {
00149     fprintf(fp,"FALSE  |  ");
00150    }
00151   }
00152   fprintf(fp,"\n");
00153  }
00154  fprintf(fp, "\n");
00155  if (macroRedFlag)
00156  {
00157   fprintf(fp, "# of red Droplets: %d  |  ", redTally);
00158  }
00159  if (macroSAFlag)
00160  {
00161   fprintf(fp, "# of Droplets communicating: %d  |  ", SATally);
00162  }
00163  fprintf(fp, "\n");
00164  fprintf(fp, "-----------------------------------------------\n\n");
00165  fflush(fp);
00166 }
00167
00168 void simInfoLogger::timeCheck(simState_t stateInfo)
00169 {
00170  double currentTime;
00171  currentTime = stateInfo.simTime;
00172  if((currentTime - lastPrint) > timeInterval)
00173  {
00174   lastPrint = currentTime;
00175   printDropletData(stateInfo);
00176  }
00177 }
```

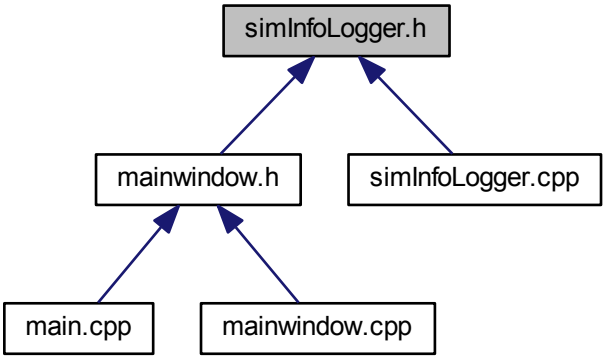
13.32 simInfoLogger.h File Reference

Declares the simulation information logger class. This class logs info while the simulator is running and prints the info into a specified output file.

```
#include <DropletSimInfo.h>
#include <SimInterface.h>
#include <structs.h>
#include <DropletDataStructs.h>
#include <QGlobal.h>
#include <QTime>
#include <stdio.h>
#include "defaults.h"
#include <glm/glm.hpp>
```

Include dependency graph for simInfoLogger.h:

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



Classes

- class simInfoLogger

  Simulation information logger.

13.32.1 Detailed Description

Declares the simulation information logger class. This class logs info while the simulator is running and prints the info into a specified output file.

Definition in file simInfoLogger.h.

13.33 simInfoLogger.h

```
00001
00009 #include <DropletSimInfo.h>
```

```
00010 #include <SimInterface.h>
00011 #include <structs.h>
00012 #include <DropletDataStructs.h>
00013 #include <QGlobal.h>
00014 #include <QTime>
00015 #include <stdio.h>
00016 #include "defaults.h"
00017 #include <glm/glm.hpp>
00018
00032 class simInfoLogger : public QObject
00033 {
00034
00035  Q_OBJECT
00036
00037 public:
00038  simInfoLogger(QObject *parent = 0);
00039
00046  void Init();
00047
00057  void printDropletData(simState_t stateInfo);
00058
00067  void setPosFlag(bool flag);
00068  void setColorFlag(bool flag);
00069  void setRotationFlag(bool flag);
00070  void setCommSAFlag(bool flag);
00071  void setMacroRedFlag(bool flag);
00072  void setMacroSAFlag(bool flag);
00073  ~simInfoLogger();
00074
00075 private:
00076
00084  bool posFlag,colorFlag,rotationFlag,commSAFlag,
      macroRedFlag, macroSAFlag;
00085
00093  int redTally, SATally;
00094
00101  double lastPrint;
00102
00108  double timeInterval;
00109
00116  QString newFile;
00117
00123  FILE * fp;
00124
00125 public slots:
00131  void close();
00132
00140  void timeCheck(simState_t stateInfo);
00141
00142 };
```

13.34 SimInterface.cpp File Reference

Implements the SimInterface class.

`#include "SimInterface.h"`
Include dependency graph for SimInterface.cpp:

Macros

- #define SIMPLIFY_COLLISION_SHAPES

13.34.1 Detailed Description

Implements the SimInterface class.

Definition in file SimInterface.cpp.

13.34.2 Macro Definition Documentation

13.34.2.1 #define SIMPLIFY_COLLISION_SHAPES

Definition at line 7 of file SimInterface.cpp.

13.35 SimInterface.cpp

```
00001
00007 #define SIMPLIFY_COLLISION_SHAPES
00008 #include "SimInterface.h"
00009
00010 SimInterface::SimInterface(QObject *parent)
00011  : QObject(parent)
00012 {
00013  _sim = NULL;
00014  _dropletColors = NULL;
00015  _dropletPos = NULL;
00016  _dropletComm = NULL;
00017  _objectPos = NULL;
00018  _timer = NULL;
00019  _simSettings = getDefaultSettings();
00020  _simStatus.paused = false;
00021  _simRates.timeScale = 1.0;
00022  _simRates.limitRate = false;
00023  _timerID = 0;
00024  _timeUntilNextUpdate = 0;
00025  _simStatus.dropletShape = NULL;
00026  // TODO - abstract this better
00027  makeDropletCollisionShapeFromFile(QString("assets/Models/HQDroplet.obj"))
     ;
00028  _objectNames.single << "cube" << "sphere";
00029  _objectNames.multiple << "cubes" << "spheres";
00030 }
00031
00032 SimInterface::~SimInterface()
00033 {
00034
00035  if (_timerID != 0)
00036  {
00037   killTimer(_timerID);
00038   _timerID = 0;
00039  }
00040
00041
00042  teardownSim();
00043
00044 }
00045
00046 void SimInterface::teardownSim()
00047 {
00048  if (_sim != NULL)
00049  {
00050   _sim->Cleanup();
00051   delete _sim;
00052  }
00053
00054  if (_dropletColors != NULL)
00055  {
00056   std::vector<uint8_t *>::iterator it;
00057   for (it = _dropletColors->begin(); it != _dropletColors->end() ; it++)
00058   {
00059    uint8_t *bob = *it;
00060    free(bob);
00061   }
00062   delete _dropletColors;
00063   _dropletColors = NULL;
00064
00065  }
00066  if (_dropletPos != NULL)
00067  {
00068   std::vector<GPSInfo *>::iterator it;
00069   for (it = _dropletPos->begin(); it != _dropletPos->end() ; it++)
00070   {
00071    GPSInfo *bob = *it;
00072    free(bob);
00073   }
00074   delete _dropletPos;
00075   _dropletPos = NULL;
00076  }
00077
00078  if (_dropletComm != NULL)
00079  {
00080   std::vector<DropletCommData *>::iterator commIt;
00081   for (commIt = _dropletComm->begin(); commIt != _dropletComm->end(); commIt++)
```

```
00082   {
00083    DropletCommData *bob = *commIt;
00084    free(bob);
00085   }
00086   delete _dropletComm;
00087
00088   _dropletComm = NULL;
00089  }
00090
00091  if (_objectPos != NULL)
00092
00093  {
00094   std::vector<GPSInfo *>::iterator it;
00095   for (it = _objectPos->begin(); it != _objectPos->end() ; it++)
00096   {
00097    GPSInfo *bob = *it;
00098    free(bob);
00099
00100   }
00101   delete _objectPos;
00102   _objectPos = NULL;
00103  }
00104
00105  if (_timer != NULL){
00106   free(_timer);
00107   _timer = NULL;
00108  }
00109 }
00110
00111 void SimInterface::Init()
00112 {
00113  bool isPaused = _simStatus.paused;
00114  if (!isPaused)
00115   pause();
00116
00117  teardownSim();
00118  // TIMER
00119
00120  updateTiming();
00121  _timeUntilNextUpdate = 0;
00122
00123  QTime time = QTime::currentTime();
00124  srand((uint)time.msec());
00125
00126
00127
00128  // TIMER
00129  //_simStatus.runTimeSubSec = 0;
00130  //_simStatus.runTimeSec = 0;
00131
00132  _sim = new DropletSim();
00133  SimSetupData setupData(
00134   _simSettings.numRowTiles,
00135   _simSettings.numColTiles,
00136   _simSettings.tileLength,
00137   _simSettings.dropletRadius,
00138   _simSettings.fps,
00139   true);
00140  _sim->Init(setupData);
00141
00142  if (_simSettings.projecting)
00143  {
00144   QString fileDir = QString(DEFAULT_ASSETDIR).append(
      DEFAULT_PROJECTDIR);
00145   QFileInfo test(QString(fileDir).append(_simSettings.projTexture));
00146   if (test.exists())
00147   {
00148    qDebug() << "Found projection texture" << test.absoluteFilePath();
00149    if (_sim->SetUpProjector(fileDir.toStdString(),_simSettings.
      projTexture.toStdString()) == DS_WARNING)
00150    {
00151     qDebug() << "Error setting up projector";
00152     _simSettings.projecting = false;
00153     _simState.projTextureChanged = false;
00154    } else {
00155     _simState.projTexture = QImage(test.absoluteFilePath());
00156     _simState.projTextureChanged = true;
00157    }
00158   } else {
00159    qDebug() << "Error: file not found" << test.absoluteFilePath();
00160    _simSettings.projecting = false;
00161    _simState.projTextureChanged = false;
00162
00163   }
00164  }
00165
00166  btCollisionShape *floorShape = new btStaticPlaneShape(btVector3(0, 0, 1) , 0);
```

```
00167  btCollisionShape *xWallShape = new btBoxShape(btVector3(
00168   btScalar(_simSettings.tileLength * _simSettings.
      numRowTiles),
00169   btScalar(_simSettings.wallWidth),
00170   btScalar(_simSettings.wallHeight)));
00171  btCollisionShape *yWallShape = new btBoxShape(btVector3(
00172   btScalar(_simSettings.wallWidth),
00173   btScalar(_simSettings.tileLength * _simSettings.
      numColTiles),
00174   btScalar(_simSettings.wallHeight)));
00175
00176  /*
00177  btCollisionShape *dropletShape = new btCylinderShapeZ(btVector3(
00178  _simSettings.dropletRadius,
00179  _simSettings.dropletRadius,
00180  _simSettings.dropletRadius * 0.8242));
00181  */
00182
00183
00184  _simSettings.dropletOffset = _simStatus.dropletOffset;
00185  _simStatus.dropletShape->setLocalScaling(btVector3(
00186   _simSettings.dropletRadius,
00187   _simSettings.dropletRadius,
00188   _simSettings.dropletRadius));
00189
00190
00191  _sim->AddCollisionShape(floorShape, &_simStatus.btFloorShapeID);
00192  _sim->AddCollisionShape(xWallShape, &_simStatus.btXWallShapeID);
00193  _sim->AddCollisionShape(yWallShape, &_simStatus.btYWallShapeID);
00194  _sim->AddCollisionShape(_simStatus.dropletShape, &_simStatus.btDropletShapeID);
00195
00196  // Create the floor and walls
00197  //_sim->CreateFloor(_simStatus.btFloorShapeID, _simStatus.btXWallShapeID, _simStatus.btYWallShapeID);
00198  // setupSimObjects()
00199
00200
00201  _dropletColors = new std::vector<unsigned char *>();
00202  _dropletPos = new std::vector<GPSInfo *>();
00203
00204  _dropletComm = new std::vector<DropletCommData *>();
00205
00206  _objectPos = new std::vector<GPSInfo *>();
00207
00208  _timer = (DropletTimeControl *)malloc(sizeof(DropletTimeControl));
00209
00210  _simState.dropletData.clear();
00211  _simState.dynamicObjectData.clear();
00212  _simState.staticObjectData.clear();
00213  _simState.collisionShapes.clear();
00214
00215
00216  /*
00217  if(_simSettings.floorFile.operator==("Default (Rectangle)"))
00218  {
00219   createArena();
00220  }
00221  else
00222  {
00223   // BUG: Reading the arena from a file after initializing the simulator means the arena size inside the
      arena is wrong
00224   QString fileName = QString(DEFAULT_ASSETDIR).append(DEFAULT_FLOORDIR).append(_simSettings.floorFile);
00225   createArena(fileName);
00226  }*/
00227  qDebug() << "floor file" << _simSettings.floorFile;
00228  createArena();
00229
00230
00231
00232
00233
00234  if (_simSettings.startingObjects.count() > 0)
00235  {
00236   float floorWidth = _simSettings.tileLength *
      _simSettings.numColTiles;
00237   float floorLength = _simSettings.tileLength *
      _simSettings.numRowTiles;
00238   float posRangeWidth = floorWidth / 2.0f;
00239   float posRangeLength = floorLength / 2.0f;
00240   foreach(QStringList list,_simSettings.startingObjects)
00241   {
00242    QString iType = list[0].toLower();
00243    if (_objectNames.multiple.contains(iType))
00244    {
00245     if (list.count() >= 2)
00246     {
00247      int num = list[1].toInt();
00248      object_t oType = Cube;
```

```
00249       float radius = DEFAULT_OBJECT_RADIUS;
00250
00251       if (list.count() >= 3)
00252       {
00253        float temp = list[2].toFloat();
00254        if (temp > 0)
00255         radius = temp;
00256       }
00257
00258       if (iType == QString("spheres"))
00259       {
00260        oType = Sphere;
00261       } else if (iType == QString("cubes"))
00262       {
00263        oType = Cube;
00264        radius *= 2.0;
00265       }
00266       float mass = DEFAULT_OBJECT_MASS;
00267       float friction = DEFAULT_OBJECT_MASS;
00268       if (list.count() >= 4)
00269       {
00270        mass = list[3].toFloat();
00271       }
00272
00273       if (list.count() >= 5)
00274       {
00275        friction = list[4].toFloat();
00276       }
00277
00278       for(int i = 0; i < num; i++)
00279       {
00280        // prevents index out of bound errors
00281        int tileIndex = rand()%_tilePositions.size();
00282
00283        //qDebug() << tileIndex;
00284        //float xPos = getRandomf(-posRangeWidth + _simSettings.dropletRadius, posRangeWidth -
      _simSettings.dropletRadius);
00285        //float yPos = getRandomf(-posRangeLength + _simSettings.dropletRadius, posRangeLength -
      _simSettings.dropletRadius);
00286        /*float xPos = getRandomf(_tilePositions->at(tileIndex).x - _simSettings.tileLength/2.0 +
      _simSettings.dropletRadius,
00287        _tilePositions->at(tileIndex).x + _simSettings.tileLength/2.0 - _simSettings.dropletRadius);
00288        float yPos = getRandomf(_tilePositions->at(tileIndex).y - _simSettings.tileLength/2.0 +
      _simSettings.dropletRadius,
00289        _tilePositions->at(tileIndex).y + _simSettings.tileLength/2.0 - _simSettings.dropletRadius);
00290        */
00291
00292        float xPos = getRandomf(_tilePositions[tileIndex].x -
      _simSettings.tileLength/2.0 + _simSettings.
      dropletRadius,
00293         _tilePositions[tileIndex].x + _simSettings.
      tileLength/2.0 - _simSettings.dropletRadius);
00294        float yPos = getRandomf(_tilePositions[tileIndex].y -
      _simSettings.tileLength/2.0 + _simSettings.
      dropletRadius,
00295         _tilePositions[tileIndex].y + _simSettings.
      tileLength/2.0 - _simSettings.dropletRadius);
00296
00297        // qDebug() << QString("Added droplet at x: %1 y: %2").arg(xPos).arg(yPos);
00298        addObject(oType, xPos,yPos,radius,mass,friction);
00299       }
00300      }
00301     } else if (_objectNames.single.contains(iType))
00302     {
00303      if (list.count() >= 3)
00304      {
00305       object_t oType = Cube;
00306       float radius = DEFAULT_OBJECT_RADIUS;
00307
00308       if (list.count() >= 4)
00309       {
00310        float temp = list[3].toFloat();
00311        if (temp > 0)
00312         radius = temp;
00313       }
00314
00315       if (iType == QString("sphere"))
00316       {
00317        oType = Sphere;
00318       } else if (iType == QString("cube"))
00319       {
00320        oType = Cube;
00321        radius *= 2.0;
00322       }
00323       float xPos = list[1].toFloat();
00324       float yPos = list[2].toFloat();
00325       float mass = DEFAULT_OBJECT_MASS;
```

```
00326       float friction = DEFAULT_OBJECT_MASS;
00327       if (list.count() >= 4)
00328       {
00329        mass = list[3].toFloat();
00330       }
00331
00332       if (list.count() >= 5)
00333       {
00334        friction = list[4].toFloat();
00335       }
00336
00337       addObject(oType, xPos,yPos,radius,mass,friction);
00338
00339     }
00340    }
00341   }
00342  }
00343
00344  _sim->Step();
00345
00346
00347
00348  emit simulationUpdated(_simState);
00349  emit arenaChanged(_simSettings);
00350  emit ratesChanged(_simRates);
00351
00352
00353  float floorWidth = _simSettings.tileLength * _simSettings.
      numColTiles;
00354  float floorLength = _simSettings.tileLength *
      _simSettings.numRowTiles;
00355  float posRangeWidth = floorWidth / 2.0f;
00356  float posRangeLength = floorLength / 2.0f;
00357
00358  if (_simSettings.startingDroplets.count() > 0)
00359  {
00360   float floorWidth = _simSettings.tileLength *
      _simSettings.numColTiles;
00361   float floorLength = _simSettings.tileLength *
      _simSettings.numRowTiles;
00362   float posRangeWidth = floorWidth / 2.0f;
00363   float posRangeLength = floorLength / 2.0f;
00364   foreach(QStringList list,_simSettings.startingDroplets)
00365   {
00366    if (list.count() >= 2)
00367    {
00368
00369     droplet_t dType = March;
00370
00371     QString iType = list[0].toLower();
00372
00373     if (iType == QString("march"))
00374     {
00375      dType = March;
00376     } else if (iType == QString("rainbow"))
00377     {
00378      dType = Rainbow;
00379     } else if (iType == QString("turntest"))
00380     {
00381      dType = TurnTest;
00382     } else if (iType == QString("rgbsense"))
00383     {
00384      dType = RGBSense;
00385     } else if (iType == QString("randomwalk"))
00386     {
00387      dType = RandomWalk;
00388     } else if (iType == QString("stickpullers"))
00389     {
00390      dType = StickPullers;
00391     } else if (iType == QString("commtest"))
00392     {
00393      dType = CommTest;
00394     } else if (iType == QString("powertest"))
00395     {
00396      dType = PowerTest;
00397     } else if (iType == QString("granola"))
00398     {
00399      dType = Granola;
00400     } else if (iType == QString("stickpullersupdated"))
00401     {
00402      dType = StickPullersUpdated;
00403     } else if (iType == QString("ants"))
00404     {
00405      dType = Ants;
00406     } else if (iType == QString("customone"))
00407     {
00408      dType = CustomOne;
```

```
00409     } else if (iType == QString("customtwo"))
00410     {
00411      dType = CustomTwo;
00412     } else if (iType == QString("customthree"))
00413     {
00414      dType = CustomThree;
00415     } else if (iType == QString("customfour"))
00416     {
00417      dType = CustomFour;
00418     } else if (iType == QString("customfive"))
00419     {
00420      dType = CustomFive;
00421     } else if (iType == QString("customsix"))
00422     {
00423      dType = CustomSix;
00424     } else if (iType == QString("customseven"))
00425     {
00426      dType = CustomSeven;
00427     } else if (iType == QString("customeight"))
00428     {
00429      dType = CustomEight;
00430     } else if (iType == QString("customnine"))
00431     {
00432      dType = CustomNine;
00433     } else if (iType == QString("customten"))
00434     {
00435      dType = CustomTen;
00436     } else
00437     {
00438      dType = March;
00439     }
00440
00441     if (list.count() == 2)
00442     {
00443      int num = list[1].toInt();
00444      for(int i = 0; i < num; i++)
00445      {
00446       // prevents index out of bound errors
00447       int tileIndex = rand()%_tilePositions.size();
00448
00449       //qDebug() << tileIndex;
00450       //float xPos = getRandomf(-posRangeWidth + _simSettings.dropletRadius, posRangeWidth -
      _simSettings.dropletRadius);
00451       //float yPos = getRandomf(-posRangeLength + _simSettings.dropletRadius, posRangeLength -
      _simSettings.dropletRadius);
00452       /*float xPos = getRandomf(_tilePositions->at(tileIndex).x - _simSettings.tileLength/2.0 +
      _simSettings.dropletRadius,
00453       _tilePositions->at(tileIndex).x + _simSettings.tileLength/2.0 - _simSettings.dropletRadius);
00454       float yPos = getRandomf(_tilePositions->at(tileIndex).y - _simSettings.tileLength/2.0 +
      _simSettings.dropletRadius,
00455       _tilePositions->at(tileIndex).y + _simSettings.tileLength/2.0 - _simSettings.dropletRadius);
00456       */
00457
00458       float xPos = getRandomf(_tilePositions[tileIndex].x -
      _simSettings.tileLength/2.0 + _simSettings.
      dropletRadius,
00459        _tilePositions[tileIndex].x + _simSettings.
      tileLength/2.0 - _simSettings.dropletRadius);
00460       float yPos = getRandomf(_tilePositions[tileIndex].y -
      _simSettings.tileLength/2.0 + _simSettings.
      dropletRadius,
00461        _tilePositions[tileIndex].y + _simSettings.
      tileLength/2.0 - _simSettings.dropletRadius);
00462
00463       // qDebug() << QString("Added droplet at x: %1 y: %2").arg(xPos).arg(yPos);
00464       addDroplet(xPos,yPos,dType);
00465       //qDebug() << "Adding a droplet";
00466      }
00467     } else if (list.count() == 3)
00468     {
00469      float xPos = list[1].toFloat();
00470      float yPos = list[2].toFloat();
00471      addDroplet(xPos,yPos,dType);
00472     }
00473    }
00474
00475   }
00476  }
00477
00478  // step the simulator
00479  _sim->Step();
00480  _simStatus.paused = false;
00481  Update(1000.0/_simSettings.fps);
00482  _simStatus.paused = isPaused;
00483  // restore state
00484  if (!isPaused)
00485   resume();
```

```
00486 }
00487
00488 void SimInterface::loadTilePositions()
00489 {
00490  //Floor Tiles
00491  _tilePositions.clear();
00492  _wallBools.clear();
00493  for(int i = 0; i < _simSettings.numColTiles; i++)
00494  {
00495   for(int j = 0; j < _simSettings.numRowTiles; j++)
00496   {
00497    vec2 currentTile;
00498    currentTile.x = ((float)i - (_simSettings.numColTiles-1)/2.0f)*
      _simSettings.tileLength;
00499    currentTile.y = ((float)j - (_simSettings.numRowTiles-1)/2.0f) *
      _simSettings.tileLength;
00500    //currentTile.x = i * _simSettings.tileLength - _simSettings.numColTiles/2.0f;
00501    //currentTile.y = j * _simSettings.tileLength -  _simSettings.numRowTiles/2.0f;
00502
00503    _tilePositions.push_back(currentTile);
00504
00505    //currentWallBool represents which sides of the tile need walls
00506    vec4 currentWallBool;
00507    currentWallBool.v[0] = 0;
00508    currentWallBool.v[1] = 0;
00509    currentWallBool.v[2] = 0;
00510    currentWallBool.v[3] = 0;
00511
00512    // Top Wall
00513    if(j == _simSettings.numRowTiles - 1)
00514    {
00515     currentWallBool.v[0] = 1;
00516    }
00517    // Right Wall
00518    if(i == _simSettings.numColTiles - 1)
00519    {
00520     currentWallBool.v[1] = 1;
00521    }
00522    // Bottom Wall
00523    if(j == 0)
00524    {
00525     currentWallBool.v[2] = 1;
00526    }
00527    // Left Wall
00528    if(i == 0)
00529    {
00530     currentWallBool.v[3] = 1;
00531    }
00532    _wallBools.push_back(currentWallBool);
00533
00534   }
00535  }
00536 }
00537
00538 void SimInterface::loadTilePositions(QString filename)
00539 {
00540  _simSettings.floorFile = QString(filename).append(".txt");
00541  _tilePositions.clear();
00542  _wallBools.clear();
00543  if(filename.operator!=("Default (Rectangle)"))
00544  {
00545   QFile file(QString(DEFAULT_ASSETDIR).append(DEFAULT_FLOORDIR).append(
      _simSettings.floorFile));
00546
00547   if (file.exists())
00548   {
00549    if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
00550    {
00551     QTextStream in(&file);
00552     while (!in.atEnd())
00553     {
00554      QString line = in.readLine(0);
00555      if (line.count() > 0)
00556      {
00557       if (!line.startsWith("#"))
00558       {
00559        QStringList list = line.split(" ",QString::SkipEmptyParts);
00560        //Store tile position in a QVector of tile positions
00561        vec2 currentTile;
00562        currentTile.x = list[0].toFloat() * _simSettings.tileLength;
00563        currentTile.y = list[1].toFloat() * _simSettings.tileLength;
00564        _tilePositions.push_back(currentTile);
00565
00566        //currentWallBool represents which sides of the tile need walls
00567        vec4 currentWallBool;
00568        currentWallBool.v[0] = 0;
00569        currentWallBool.v[1] = 0;
```

```
00570         currentWallBool.v[2] = 0;
00571         currentWallBool.v[3] = 0;
00572         // Top Wall
00573         if(list[2] == QString("yes"))
00574         {
00575          currentWallBool.v[0] = 1;
00576         }
00577         // Right Wall
00578         if(list[3] == QString("yes"))
00579         {
00580          currentWallBool.v[1] = 1;
00581         }
00582         // Bottom Wall
00583         if(list[4] == QString("yes"))
00584         {
00585          currentWallBool.v[2] = 1;
00586         }
00587         // Left Wall
00588         if(list[5] == QString("yes"))
00589         {
00590          currentWallBool.v[3] = 1;
00591         }
00592         //Store the wallBools
00593         // Needs to match up with _tilePositions indices wise
00594         // i.e. _tilePositions[0] and _wallBools[0] refer to same tile
00595         _wallBools.push_back(currentWallBool);
00596        }
00597       }
00598      }
00599
00600    }
00601    else
00602    {
00603     qDebug() << "Error: " << file.error();
00604     file.close();
00605    }
00606    file.close();
00607    //Find boundin box custom arena lies in
00608    float xMin = std::numeric_limits<float>::max();
00609    float xMax = std::numeric_limits<float>::min();
00610    float yMin = std::numeric_limits<float>::max();
00611    float yMax = std::numeric_limits<float>::min();
00612    foreach(vec2 tilePosition, _tilePositions){
00613     xMin = qMin(xMin, tilePosition.x);
00614     xMax = qMax(xMax, tilePosition.x);
00615     yMin = qMin(yMin, tilePosition.y);
00616     yMax = qMax(yMax, tilePosition.y);
00617    }
00618    //For projection purposes
00619    // min and max are at the center of each tile, so xMax - xMin isn't accounting for a full tileLength
00620    _simSettings.numColTiles = qAbs(xMax - xMin)/
      _simSettings.tileLength + 1;
00621    _simSettings.numRowTiles = qAbs(yMax - yMin)/
      _simSettings.tileLength + 1;
00622    //xOffset and yOffset represent the center of the custom arena
00623    float xOffset = (xMax + xMin)/2;
00624    float yOffset = (yMax + yMin)/2;
00625    for(int i = 0; i < _tilePositions.size(); i++)
00626    {
00627     //Modify tilePosition to center the arena to the origin
00628     _tilePositions[i].x -= xOffset;
00629     _tilePositions[i].y -= yOffset;
00630    }
00631   }
00632   else
00633   {
00634    qDebug() << "Error: file does not exist - using compiled defaults";
00635   }
00636  }
00637 }
00638
00639 void SimInterface::createArena()
00640 {
00641  if(_simSettings.floorFile.operator==("Default (Rectangle)"))
00642  {
00643   loadTilePositions();
00644  }
00645  vec3 tileScale;
00646  tileScale.x = _simSettings.tileLength;
00647  tileScale.y = _simSettings.tileLength;
00648  tileScale.z = 0.1f;
00649  vec3 hWallScale;
00650  hWallScale.x = _simSettings.tileLength;
00651  hWallScale.y = _simSettings.wallWidth;
00652  hWallScale.z = _simSettings.wallHeight;
00653  vec3 vWallScale;
00654  vWallScale.x = _simSettings.wallWidth;
```

```
00655  vWallScale.y = _simSettings.tileLength;
00656  vWallScale.z = _simSettings.wallHeight;
00657
00658  for(int i = 0; i < _tilePositions.size(); i++)
00659   {
00660    addFloor(_tilePositions[i].x, _tilePositions[i].y, 0, tileScale);
00661    //Top Wall
00662    if(_wallBools[i].v[0] == 1)
00663    {
00664     float xPos = _tilePositions[i].x;
00665     float yPos = _tilePositions[i].y + _simSettings.
       tileLength/2.0f;
00666     addWall(xPos, yPos,0,hWallScale);
00667    }
00668    //Right Wall
00669    if(_wallBools[i].v[1] == 1)
00670    {
00671     float xPos = _tilePositions[i].x + _simSettings.
       tileLength/2.0f;
00672     float yPos = _tilePositions[i].y;
00673     addWall(xPos,yPos,0,vWallScale);
00674    }
00675    //Bottom Wall
00676    if(_wallBools[i].v[2] == 1)
00677    {
00678     float xPos = _tilePositions[i].x;
00679     float yPos = _tilePositions[i].y - _simSettings.
       tileLength/2.0f;
00680     addWall(xPos,yPos,0,hWallScale);
00681    }
00682    //Left Wall
00683    if(_wallBools[i].v[3] == 1)
00684    {
00685     float xPos = _tilePositions[i].x -_simSettings.
       tileLength/2.0f;
00686     float yPos = _tilePositions[i].y;
00687     addWall(xPos,yPos,0,vWallScale);
00688    }
00689   }
00690 }
00691
00692 void SimInterface::addDroplet( float x, float y,
       droplet_t dType, int dropletID )
00693 {
00694  //qDebug() << QString("Added droplet at x: %1 y: %2").arg(x).arg(y);
00695  bool isPaused = _simStatus.paused;
00696  _simStatus.paused = true;
00697  unsigned char *tmp1 = (unsigned char *)malloc(sizeof(unsigned char) * 3);
00698  _dropletColors->push_back(tmp1);
00699  GPSInfo *tmp2 = (GPSInfo *)malloc(sizeof(GPSInfo));
00700  _dropletPos->push_back(tmp2);
00701
00702  DropletCommData *tmp3 = (DropletCommData *)malloc(sizeof(DropletCommData));
00703  _dropletComm->push_back(tmp3);
00704
00705  dropletStruct_t droplet;
00706
00707  if (dropletID == 0)
00708  {
00709   // make up a droplet ID
00710   droplet.dropletID = _simState.dropletData.count() + 1;
00711  } else {
00712
00713   droplet.dropletID = dropletID;
00714  }
00715
00716
00717  /*droplet.commData.commChannels[j].lastMsgInTimestamp = 0;
00718  droplet.commData.commChannels[j].inMsgLength = 0;
00719  droplet.commData.commChannels[i].lastMsgOutTimestamp = 0;
00720  droplet.commData.commChannels[i].outMsgLength = 0;*/
00721
00722  vec3 zero = {0.0f,0.0f,0.0f};
00723  droplet.color = vec3imake(0,0,0);
00724  droplet.origin = vec3make(x,y,0);
00725  droplet.rotation = zero;
00726  droplet.commData.sendActive = false;
00727  droplet.changed = true;
00728
00729  _simState.dropletData.append(droplet);
00730
00731  // Set up the simulator/physics model
00732  ObjectPhysicsData *dropletPhyDat = (ObjectPhysicsData *)malloc(sizeof(ObjectPhysicsData));
00733  dropletPhyDat->colShapeIndex = _simStatus.btDropletShapeID;
00734  dropletPhyDat->mass = DEFAULT_DROPLET_MASS;
00735  dropletPhyDat->localInertia = btVector3(0.0, 0.0, 0.0);
00736  dropletPhyDat->friction = DEFAULT_DROPLET_FRICTION;
```

```
00737
00738
00739
00740  //IDroplet *newDroplet = new DropletMarch(dropletPhyDat);
00741  IDroplet *newDroplet = newDropletOfType(dType,dropletPhyDat);
00742
00743  //  sim.AddDroplet(newDroplet, std::make_pair(0.0f, 0.0f), 0.0f);
00744  _sim->AddDroplet(
00745   newDroplet,
00746   std::make_pair(droplet.origin.x,droplet.origin.y),
00747   (rand() % 3600) / 10.0f
00748   );
00749  _simStatus.paused = isPaused;
00750
00751 }
00752
00753 //Adding collision objects. Returns the index of the corresponding shape in the list
00754 //of collision objects.
00755 int SimInterface::addNewShape(object_t objectType, float radius,
      vec3 scale){
00756  if(objectType == Sphere){
00757   if(_simState.collisionShapes.count() > 0){
00758    collisionShapeStruct_t shape;
00759    int i;
00760    for(i = 0; i < _simState.collisionShapes.count(); i++){
00761     shape = _simState.collisionShapes.at(i);
00762     if((shape.oType == Sphere) && (shape.objectRadius == radius)){
00763      return i;
00764     }
00765    }
00766   }
00767
00768   //Add new shape to list
00769   vec3 one = {1.0, 1.0, 1.0};
00770   collisionShapeStruct_t newShape;
00771   _simState.collisionShapes.append(newShape);
00772
00773   _simState.collisionShapes[_simState.
      collisionShapes.count() - 1].oType = Sphere;
00774   _simState.collisionShapes[_simState.
      collisionShapes.count() - 1].objectRadius = radius;
00775   _simState.collisionShapes[_simState.
      collisionShapes.count() - 1].scale = one;
00776
00777   // Add collision shape
00778   btCollisionShape *sphereShape = new btSphereShape(
00779    btScalar(radius));
00780
00781   // Get indeces
00782   _sim->AddCollisionShape(sphereShape, &(_simState.collisionShapes[
      _simState.collisionShapes.count() - 1].collisionID));
00783
00784   return _simState.collisionShapes.count() - 1;
00785  }
00786
00787  if((objectType == Cube) || (objectType == Wall) || (objectType == Floor)){
00788   if(_simState.collisionShapes.count() > 0){
00789    collisionShapeStruct_t shape;
00790    int i;
00791    for(i = 0; i < _simState.collisionShapes.count(); i++){
00792     shape = _simState.collisionShapes.at(i);
00793     if(((shape.oType == Cube) && (objectType == Cube)) && (shape.
      scale.x == scale.x) && (shape.scale.y == scale.y) && (shape.scale.
      z == scale.z)){
00794      return i;
00795     }
00796     if(((shape.oType == Wall) && (objectType == Wall)) && (shape.
      scale.x == scale.x) && (shape.scale.y == scale.y) && (shape.scale.
      z == scale.z)){
00797      return i;
00798     }
00799     if(((shape.oType == Floor) && (objectType == Floor)) && (shape.
      scale.x == scale.x) && (shape.scale.y == scale.y) && (shape.scale.
      z == scale.z)){
00800      return i;
00801     }
00802    }
00803   }
00804
00805   //Add new shape to list
00806   vec3 one = {1.0, 1.0, 1.0};
00807   collisionShapeStruct_t newShape;
00808   _simState.collisionShapes.append(newShape);
00809
00810   if(objectType == Cube){_simState.collisionShapes[
      _simState.collisionShapes.count() - 1].oType = Cube;}
00811   if(objectType == Wall){_simState.collisionShapes[
```

```
      _simState.collisionShapes.count() - 1].oType = Wall;}
00812   if(objectType == Floor){_simState.collisionShapes[
      _simState.collisionShapes.count() - 1].oType = Floor;}
00813   _simState.collisionShapes[_simState.
      collisionShapes.count() - 1].objectRadius = one.x;
00814   _simState.collisionShapes[_simState.
      collisionShapes.count() - 1].scale = scale;
00815
00816   // Add collision shape
00817   btCollisionShape *cubeShape = new btBoxShape(btVector3(
00818    btScalar(0.5*scale.x),
00819    btScalar(0.5*scale.y),
00820    btScalar(0.5*scale.z)));
00821
00822   // Get indeces
00823   _sim->AddCollisionShape(cubeShape, &(_simState.collisionShapes[
      _simState.collisionShapes.count() - 1].collisionID));
00824
00825   return _simState.collisionShapes.count() - 1;
00826  }
00827
00828  return NULL;
00829 }
00830
00831 void SimInterface::addSphere( float x, float y, int objectID, float radius, float
      mass, float friction)
00832 {
00833
00834  bool isPaused = _simStatus.paused;
00835  _simStatus.paused = true;
00836  GPSInfo *tmp2 = (GPSInfo *)malloc(sizeof(GPSInfo));
00837  _objectPos->push_back(tmp2);
00838  objectStruct_t object;
00839
00840  if (objectID == 0)
00841  {
00842   // make up an object ID
00843   object.objectID = _simState.dynamicObjectData.count() + 1;
00844  } else {
00845
00846   object.objectID = objectID;
00847  }
00848
00849  vec3 scale = {1.0f,1.0f,1.0f};
00850  vec4 zero = {0.0f,0.0f,0.0f,0.0f};
00851  object.color = vec3imake(rand()%256,rand()%256,rand()%256);
00852  // object.origin.z = 0.5*scale.z;
00853  object.origin = vec3make(x,y,radius);
00854
00855  object.objectRadius = radius;
00856  object.scale = scale;
00857  object.quaternion = zero;
00858  object.changed = true;
00859  object.oType = Sphere;
00860
00861  _simState.dynamicObjectData.append(object);
00862
00863  //Get shape colShapeIndex
00864  int colIndex = addNewShape(object.oType, object.objectRadius, scale);
00865
00866  // Set up the simulator/physics model
00867  ObjectPhysicsData *objectPhyDat = (ObjectPhysicsData *)malloc(sizeof(ObjectPhysicsData));
00868  //objectPhyDat->colShapeIndex = _simStatus.btSphereShapeID;
00869  objectPhyDat->colShapeIndex = _simState.collisionShapes[colIndex].collisionID;
00870  objectPhyDat->mass = mass;
00871  objectPhyDat->localInertia = btVector3(0.0, 0.0, 0.0);
00872  objectPhyDat->friction = friction;
00873
00874  //Create physical object
00875  DSimPhysicalObject *newobject = new DSimPhysicalObject(objectPhyDat);
00876
00877  //Add to sim
00878  _sim->AddPhysicalObject(
00879   newobject,
00880   std::make_pair(object.origin.x,object.origin.y),
00881   0.0f
00882   );
00883
00884  _simStatus.paused = isPaused;
00885 }
00886
00887 void SimInterface::addCube( float x, float y, int objectID,
      vec3 scale, float mass, float friction)
00888 {
00889
00890  bool isPaused = _simStatus.paused;
00891  _simStatus.paused = true;
```

```
00892  GPSInfo *tmp2 = (GPSInfo *)malloc(sizeof(GPSInfo));
00893  _objectPos->push_back(tmp2);
00894  objectStruct_t object;
00895
00896  if (objectID == 0)
00897  {
00898   // make up an object ID
00899   object.objectID = _simState.dynamicObjectData.count() + 1;
00900  } else {
00901
00902   object.objectID = objectID;
00903  }
00904
00905  vec4 zero = {0,0,0,0};
00906  object.color = vec3imake(rand()%256,rand()%256,rand()%256);
00907  // object.origin.z = 0.5*scale.z;
00908  object.origin = vec3make(x,y,0.5*scale.z);
00909
00910  object.objectRadius = 1.0;
00911  object.scale = scale;
00912  object.quaternion = zero;
00913  object.changed = true;
00914  object.oType = Cube;
00915
00916  _simState.dynamicObjectData.append(object);
00917
00918  //Get shape colShapeIndex
00919  int colIndex = addNewShape(object.oType, object.objectRadius, scale);
00920
00921  // Set up the simulator/physics model
00922  ObjectPhysicsData *objectPhyDat = (ObjectPhysicsData *)malloc(sizeof(ObjectPhysicsData));
00923  //objectPhyDat->colShapeIndex = _simStatus.btCubeShapeID;
00924  objectPhyDat->colShapeIndex = _simState.collisionShapes.at(colIndex).collisionID;
00925  objectPhyDat->mass = mass;
00926  objectPhyDat->localInertia = btVector3(0.0, 0.0, 0.0);
00927  objectPhyDat->friction = friction;
00928
00929  //Create physical object
00930  DSimPhysicalObject *newobject = new DSimPhysicalObject(objectPhyDat);
00931
00932  //Add to sim
00933  _sim->AddPhysicalObject(
00934   newobject,
00935   std::make_pair(object.origin.x,object.origin.y),
00936   0.0f
00937   );
00938
00939  _simStatus.paused = isPaused;
00940 }
00941
00942 void SimInterface::addWall( float x, float y, int objectID,
      vec3 scale)
00943 {
00944
00945  bool isPaused = _simStatus.paused;
00946  _simStatus.paused = true;
00947  //GPSInfo *tmp2 = (GPSInfo *)malloc(sizeof(GPSInfo));
00948  //_objectPos->push_back(tmp2);
00949  objectStruct_t object;
00950
00951  if (objectID == 0)
00952  {
00953   // make up an object ID
00954   object.objectID = _simState.staticObjectData.count() + 1;
00955  } else {
00956
00957   object.objectID = objectID;
00958  }
00959
00960  vec4 zero = {0,0,0,0};
00961  object.color = vec3imake(rand()%256,rand()%256,rand()%256);
00962  object.origin = vec3make(x,y,0.5*scale.z);
00963  object.objectRadius = 1.0;
00964  object.scale = scale;
00965  object.quaternion = zero;
00966  object.changed = true;
00967  object.oType = Wall;
00968
00969  _simState.staticObjectData.append(object);
00970
00971  //Get shape colShapeIndex
00972  int colIndex = addNewShape(object.oType, object.objectRadius, scale);
00973
00974  // Set up the simulator/physics model
00975  ObjectPhysicsData *objectPhyDat = (ObjectPhysicsData *)malloc(sizeof(ObjectPhysicsData));
00976  objectPhyDat->colShapeIndex = _simState.collisionShapes[colIndex].collisionID;
00977  objectPhyDat->mass = DEFAULT_WALL_MASS;
```

```
00978  objectPhyDat->localInertia = btVector3(0.0, 0.0, 0.0);
00979  objectPhyDat->friction = DEFAULT_WALL_FRICTION;
00980
00981  //Create physical object
00982  DSimPhysicalObject *newobject = new DSimPhysicalObject(objectPhyDat);
00983
00984  //Add to sim
00985  _sim->AddPhysicalObject(
00986   newobject,
00987   std::make_pair(object.origin.x,object.origin.y),
00988   object.origin.z,
00989   0.0f
00990   );
00991
00992  _simStatus.paused = isPaused;
00993 }
00994
00995 void SimInterface::addFloor( float x, float y, int objectID,
      vec3 scale)
00996 {
00997
00998  bool isPaused = _simStatus.paused;
00999  _simStatus.paused = true;
01000  //GPSInfo *tmp2 = (GPSInfo *)malloc(sizeof(GPSInfo));
01001  //_objectPos->push_back(tmp2);
01002  objectStruct_t object;
01003
01004  if (objectID == 0)
01005  {
01006   // make up an object ID
01007   object.objectID = _simState.staticObjectData.count() + 1;
01008  } else {
01009
01010   object.objectID = objectID;
01011  }
01012  vec4 zero = {0,0,0,0};
01013  object.color = vec3imake(rand()%256,rand()%256,rand()%256);
01014  // object.origin.z = 0.5*scale.z;
01015  object.origin = vec3make(x,y,0);
01016  object.objectRadius = 1.0;
01017  object.scale = scale;
01018  object.quaternion = zero;
01019  object.changed = true;
01020  object.oType = Floor;
01021
01022  _simState.staticObjectData.append(object);
01023
01024  //Get shape colShapeIndex
01025  int colIndex = addNewShape(object.oType, object.objectRadius, scale);
01026
01027  // Set up the simulator/physics model
01028  ObjectPhysicsData *objectPhyDat = (ObjectPhysicsData *)malloc(sizeof(ObjectPhysicsData));
01029  objectPhyDat->colShapeIndex = _simState.collisionShapes[colIndex].collisionID;
01030  objectPhyDat->mass = DEFAULT_FLOOR_MASS;
01031  objectPhyDat->localInertia = btVector3(0.0, 0.0, 0.0);
01032  objectPhyDat->friction = DEFAULT_FLOOR_FRICTION;
01033
01034  //Create physical object
01035  DSimPhysicalObject *newobject = new DSimPhysicalObject(objectPhyDat);
01036
01037  //Add to sim
01038  _sim->AddPhysicalObject(
01039   newobject,
01040   std::make_pair(object.origin.x,object.origin.y),
01041   object.origin.z,
01042   0.0f
01043   );
01044
01045  _simStatus.paused = isPaused;
01046 }
01047
01048
01049 void SimInterface::addObject(object_t kind, float x, float y, float radius,
      float mass, float friction)
01050 {
01051  switch(kind)
01052  {
01053  case Sphere:
01054   addSphere(x,y,0,radius, mass, friction);
01055   break;
01056  case Cube:
01057   addCube(x,y,0,vec3make(radius,radius,radius), mass, friction);
01058   break;
01059  }
01060 }
01061
01062 float SimInterface::getRandomf(float min, float max)
```

```
01063 {
01064  uint randNum = rand();
01065  float range = max - min;
01066  return (min + ((range * randNum) / RAND_MAX));
01067 }
01068
01069
01070 void SimInterface::Update(float timeSinceLastUpdate)
01071 {
01072  if (_sim != NULL)
01073  {
01074   if (!_simStatus.paused)
01075   {
01076    // float timeInSec = 1.0f/_simSettings.fps;
01077    _sim->Step();
01078    _simInfo.GetDropletColors(_dropletColors,*_sim);
01079    _simInfo.GetDropletPositions(_dropletPos,*_sim);
01080
01081    _simInfo.GetCommData(_dropletComm,*_sim);
01082
01083    _simInfo.GetObjectPositions(_objectPos, *_sim);
01084
01085    _simStatus.runTime = _simInfo.GetTotalST(*_sim);
01086
01087
01088   }
01089
01090
01091
01092   std::vector<GPSInfo *>::iterator it;
01093   it = _dropletPos->begin();
01094   std::vector<DropletCommData *>::iterator commIt;
01095   commIt = _dropletComm->begin();
01096
01097
01098   for(int i = 0 ; i < _dropletPos->size(); i++)
01099   {
01100
01101    dropletStruct_t droplet = _simState.dropletData[i];
01102    GPSInfo *gpsInfo = *it;
01103    DropletCommData *dropletCommData = *commIt;
01104    bool posChanged = false;
01105
01106
01107    glm::quat quaternion = glm::angleAxis(gpsInfo->rotA * 180.f / SIMD_PI,gpsInfo->rotX,gpsInfo->rotY,
      gpsInfo->rotZ);
01108    glm::vec3 rotation = glm::eulerAngles(quaternion);
01109
01110    vec4 newQuaternion = {
01111     quaternion.x,
01112     quaternion.y,
01113     quaternion.z,
01114     quaternion.w
01115    };
01116
01117    vec3 newRotation = {
01118     rotation.x,
01119     rotation.y,
01120     rotation.z
01121    };
01122
01123
01124    vec3 newOrigin = {
01125     gpsInfo->posX,
01126     gpsInfo->posY,
01127     gpsInfo->posZ
01128    };
01129
01130    vec3i newColor = {
01131     _dropletColors->at(i)[0],
01132     _dropletColors->at(i)[1],
01133     _dropletColors->at(i)[2]
01134    };
01135
01136    droplet.commData.sendActive = dropletCommData->sendActive;
01137    int j;
01138    for (j = 0; j < 6; j++)
01139    {
01140     droplet.commData.commChannels[j].lastMsgInTimestamp =
01141      dropletCommData->commChannels[j].lastMsgInTimestamp;
01142     droplet.commData.commChannels[j].inMsgLength =
01143      dropletCommData->commChannels[j].inMsgLength;
01144     droplet.commData.commChannels[j].lastMsgOutTimestamp =
01145      dropletCommData->commChannels[j].lastMsgOutTimestamp;
01146     droplet.commData.commChannels[j].outMsgLength =
01147      dropletCommData->commChannels[j].outMsgLength;
01148     memcpy(droplet.commData.commChannels[j].inBuf,
```

```
01149       dropletCommData->commChannels[j].inBuf, IR_BUFFER_SIZE);
01150      memcpy(droplet.commData.commChannels[j].outBuf,
01151       dropletCommData->commChannels[j].outBuf, IR_BUFFER_SIZE);
01152     }
01153
01154     if (newQuaternion.v != droplet.quaternion.v)
01155     {
01156      droplet.quaternion = newQuaternion;
01157      droplet.rotation = newRotation;
01158      droplet.changed = true;
01159
01160     }
01161
01162     if (newOrigin.v != droplet.origin.v)
01163     {
01164      droplet.origin = newOrigin;
01165      droplet.changed = true;
01166     }
01167
01168     if (droplet.color.v != newColor.v)
01169     {
01170      droplet.color = newColor;
01171      droplet.changed = true;
01172     }
01173     if (droplet.changed == true)
01174     {
01175      _simState.dropletData[i]=droplet;
01176     }
01177
01178     it++;
01179     commIt++;
01180    }
01181
01182    //Update object positions
01183    std::vector<GPSInfo *>::iterator itobj;
01184    itobj = _objectPos->begin();
01185
01186    for(int i = 0 ; i < _objectPos->size(); i++)
01187    {
01188
01189     objectStruct_t object = _simState.dynamicObjectData[i];
01190     GPSInfo *gpsInfo = *itobj;
01191     bool posChanged = false;
01192
01193     glm::quat quaternion = glm::angleAxis(gpsInfo->rotA * 180.f / SIMD_PI,gpsInfo->rotX,gpsInfo->rotY,
      gpsInfo->rotZ);
01194     glm::vec3 rotation = glm::eulerAngles(quaternion);
01195
01196     vec4 newQuaternion = {
01197      quaternion.x,
01198      quaternion.y,
01199      quaternion.z,
01200      quaternion.w
01201     };
01202
01203     vec3 newRotation = {
01204      rotation.x,
01205      rotation.y,
01206      rotation.z
01207     };
01208
01209
01210     vec3 newOrigin = {
01211      gpsInfo->posX,
01212      gpsInfo->posY,
01213      gpsInfo->posZ
01214     };
01215
01216
01217     if (newQuaternion.v != object.quaternion.v)
01218     {
01219      object.quaternion = newQuaternion;
01220      object.rotation = newRotation;
01221      object.changed = true;
01222
01223     }
01224
01225     if (newOrigin.v != object.origin.v)
01226     {
01227      object.origin = newOrigin;
01228      object.changed = true;
01229     }
01230
01231     if (object.changed == true)
01232     {
01233      _simState.dynamicObjectData[i]=object;
01234     }
```

```
01235    itobj++;
01236   }
01237
01238   _simState.simTime = _simStatus.runTime;
01239   _simState.realTime = _simInfo.GetTotalRT(*_sim);
01240   _simState.timeRatio = _simInfo.GetTimeRatio(*_sim);
01241   emit simulationUpdated(_simState);
01242
01243   if (_simState.projTextureChanged)
01244    _simState.projTextureChanged = false;
01245  }
01246 }
01247
01248
01249 void SimInterface::pause()
01250 {
01251  _simStatus.paused = true;
01252  if (_timerID != 0)
01253  {
01254   killTimer(_timerID);
01255   _timerID = 0;
01256  }
01257  emit pauseChanged(_simStatus.paused);
01258 }
01259 void SimInterface::resume()
01260 {
01261  _simStatus.paused = false;
01262  _updateTimer.start();
01263  _timerID = startTimer(0);
01264  emit pauseChanged(_simStatus.paused);
01265 }
01266
01267 void SimInterface::togglePause()
01268 {
01269  if (_simStatus.paused)
01270  {
01271   resume();
01272  } else
01273  {
01274   pause();
01275  }
01276 }
01277
01278 bool SimInterface::isPaused()
01279 {
01280  return _simStatus.paused;
01281 }
01282
01283
01284 IDroplet* SimInterface::newDropletOfType(droplet_t dType,
     ObjectPhysicsData *dropletPhyDat)
01285 {
01286  IDroplet* result = NULL;
01287
01288  switch (dType)
01289  {
01290  case RGBSense:
01291   result = new DropletRGBSense(dropletPhyDat);
01292   break;
01293  case RandomWalk:
01294   result = new DropletRandomWalk(dropletPhyDat);
01295   break;
01296  case March:
01297   result = new DropletMarch(dropletPhyDat);
01298   break;
01299  case Rainbow:
01300   result = new DropletRainbow(dropletPhyDat);
01301   break;
01302  case StickPullers:
01303   result = new DropletStickPullers(dropletPhyDat);
01304   break;
01305  case TurnTest:
01306   result = new DropletTurnTest(dropletPhyDat);
01307   break;
01308  case CommTest:
01309   result = new DropletCommTest(dropletPhyDat);
01310   break;
01311  case PowerTest:
01312   result = new DropletPowerTest(dropletPhyDat);
01313   break;
01314  case Granola:
01315   result = new DropletGranola(dropletPhyDat);
01316   break;
01317  case StickPullersUpdated:
01318   result = new DropletStickPullersUpdated(dropletPhyDat);
01319   break;
01320  case Ants:
```

```
01321   result = new DropletAnts(dropletPhyDat);
01322   break;
01323  case CustomOne:
01324   result = new DropletCustomOne(dropletPhyDat);
01325   break;
01326  case CustomTwo:
01327   result = new DropletCustomTwo(dropletPhyDat);
01328   break;
01329  case CustomThree:
01330   result = new DropletCustomThree(dropletPhyDat);
01331   break;
01332  case CustomFour:
01333   result = new DropletCustomFour(dropletPhyDat);
01334   break;
01335  case CustomFive:
01336   result = new DropletCustomFive(dropletPhyDat);
01337   break;
01338  case CustomSix:
01339   result = new DropletCustomSix(dropletPhyDat);
01340   break;
01341  case CustomSeven:
01342   result = new DropletCustomSeven(dropletPhyDat);
01343   break;
01344  case CustomEight:
01345   result = new DropletCustomEight(dropletPhyDat);
01346   break;
01347  case CustomNine:
01348   result = new DropletCustomNine(dropletPhyDat);
01349   break;
01350  case CustomTen:
01351   result = new DropletCustomTen(dropletPhyDat);
01352   break;
01353  default:
01354   result = new DropletMarch(dropletPhyDat);
01355
01356  }
01357  return result;
01358 }
01359
01360 void SimInterface::reset()
01361 {
01362  Init();
01363
01364 }
01365
01366 simSetting_t SimInterface::getSimulatorSettings()
01367 {
01368  return _simSettings;
01369 }
01370
01371 simState_t SimInterface::getSimulatorState()
01372 {
01373  return _simState;
01374 }
01375
01376 void SimInterface::updateTiming()
01377 {
01378  if (_simRates.limitRate)
01379  {
01380   if (_simSettings.fps > 0 && _simRates.timeScale > 0)
01381   {
01382    float rate = 1000.0 / (_simSettings.fps * _simRates.
      timeScale);
01383    if (rate != _targetUpdateTime)
01384    {
01385     _targetUpdateTime = rate;
01386     qDebug() << "Set update goal for" <<  _targetUpdateTime << "ms";
01387    }
01388   }
01389  } else
01390  {
01391   _targetUpdateTime = 0;
01392  }
01393 }
01394
01395 void SimInterface::timerEvent ( QTimerEvent * event )
01396 {
01397  qint64 nsElapsed = _updateTimer.nsecsElapsed();
01398  _timeUntilNextUpdate += nsElapsed / 1000000.0;
01399  _updateTimer.start();
01400
01401  if (_timeUntilNextUpdate >= _targetUpdateTime)
01402  {
01403   Update(1000.0/_simSettings.fps);
01404   if (_targetUpdateTime != 0)
01405   {
01406    _timeUntilNextUpdate -= _targetUpdateTime;
```

```
01407   } else {
01408    _timeUntilNextUpdate = 0;
01409   }
01410  }
01411
01412
01413
01414 }
01415
01416 void SimInterface::loadArena(simSetting_t arena)
01417 {
01418  _simSettings = arena;
01419  Init();
01420 }
01421
01422 simSetting_t SimInterface::getDefaultSettings()
01423 {
01424  simSetting_t newSettings;
01425  newSettings.dropletRadius = DEFAULT_DROPLET_RADIUS;
01426  newSettings.fps = DEFAULT_FPS;
01427  newSettings.numColTiles = DEFAULT_COL_TILES;
01428  newSettings.numRowTiles = DEFAULT_ROW_TILES;
01429  newSettings.tileLength = DEFAULT_TILE_LENGTH;
01430  newSettings.wallWidth = DEFAULT_WALL_WIDTH;
01431  newSettings.wallHeight = DEFAULT_WALL_HEIGHT;
01432  newSettings.dropletRadius = DEFAULT_DROPLET_RADIUS;
01433  newSettings.dropletOffset = 0.0f;
01434  newSettings.projecting = false;
01435  newSettings.projTexture = QString("none.bmp");
01436  newSettings.floorFile = QString("Default (Rectangle)");
01437  return newSettings;
01438 }
01439
01440 simSetting_t SimInterface::getCurrentSettings()
01441 {
01442  return _simSettings;
01443 }
01444
01445 void SimInterface::makeDropletCollisionShapeFromFile(QString
      fileName)
01446 {
01447  if (_simStatus.dropletShape != NULL)
01448  {
01449   delete _simStatus.dropletShape;
01450   _simStatus.dropletShape = NULL;
01451  }
01452  btConvexHullShape *shape = new btConvexHullShape();
01453
01454  QVector<btVector3> points;
01455  float highest = std::numeric_limits<float>::min();
01456  float lowest = std::numeric_limits<float>::max();
01457  float range;
01458  btVector3 offset;
01459
01460  QString inLine;
01461  int n;
01462
01463  QFile file(fileName);
01464
01465
01466  if (file.exists())
01467  {
01468   if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
01469   {
01470    QTextStream in(&file);
01471
01472    while (!in.atEnd())
01473    {
01474     inLine = in.readLine(0);
01475     if(inLine.length() == 0)
01476      continue;
01477     // qDebug() << inLine;
01478
01479     QStringList list = inLine.split(" ",QString::SkipEmptyParts);
01480
01481     if(list[0] == QString("v") && list.size() == 4)
01482     {
01483      btVector3 vertex;
01484      float z = list[2].toFloat();
01485
01486      if (z > highest)
01487       highest = z;
01488      if (z < lowest)
01489       lowest = z;
01490
01491      vertex.setX(list[1].toFloat());
01492      vertex.setY(list[3].toFloat());
```

```
01493      vertex.setZ(z);
01494      points.append(vertex);
01495     }
01496    }
01497    if (highest >= 0)
01498     range = highest - lowest;
01499    else
01500     range = highest + lowest;
01501
01502    _simStatus.dropletOffset = -(lowest +
      DEFAULT_DROPLET_CENTER_OF_MASS * range);
01503    offset = btVector3(0,0,_simStatus.dropletOffset);
01504    qDebug() << "Droplet offset" << _simStatus.dropletOffset;
01505    foreach(btVector3 vert,points)
01506    {
01507     btVector3 newvert;
01508     newvert = vert + offset;
01509     shape->addPoint(newvert);
01510    }
01511
01512 #ifdef SIMPLIFY_COLLISION_SHAPES
01513    btShapeHull* hull = new btShapeHull(shape);
01514    btScalar margin = shape->getMargin();
01515    hull->buildHull(margin);
01516    btConvexHullShape* simplifiedConvexShape = new btConvexHullShape((btScalar *) hull->getVertexPointer(),
      hull->numVertices());
01517    delete shape;
01518    _simStatus.dropletShape = (btCollisionShape *) simplifiedConvexShape;
01519 #else
01520    _simStatus.dropletShape = (btCollisionShape *) shape;
01521 #endif
01522   } else {
01523    qDebug() << "Error: cannot read from" << fileName;
01524   }
01525  } else {
01526   qDebug() << "Error: file" << fileName << "does not exist";
01527  }
01528
01529
01530 }
01531
01532
01533 btCollisionShape* SimInterface::makeCollisionShapeFromFile(QString
      fileName)
01534 {
01535
01536  btConvexHullShape *shape = new btConvexHullShape();
01537
01538  QString inLine;
01539  int n;
01540
01541  QFile file(fileName);
01542
01543  if (file.exists())
01544  {
01545   if (file.open(QIODevice::ReadOnly  |QIODevice::Text))
01546   {
01547    QTextStream in(&file);
01548
01549    while (!in.atEnd())
01550    {
01551     inLine = in.readLine(0);
01552     if(inLine.length() == 0)
01553      continue;
01554     // qDebug() << inLine;
01555
01556     QStringList list = inLine.split(" ",QString::SkipEmptyParts);
01557
01558     if(list[0] == QString("v") && list.size() == 4)
01559     {
01560      btVector3 vertex;
01561      vertex.setX(list[1].toFloat());
01562      vertex.setY(list[3].toFloat());
01563      vertex.setZ(list[2].toFloat());
01564      shape->addPoint(vertex);
01565     }
01566     //obj files include face information, the following parses through
01567     //three vertices at a time (for each triangle) storing indices
01568    }
01569
01570
01571   } else {
01572    qDebug() << "Error: cannot read from" << fileName;
01573    return NULL;
01574   }
01575  } else {
01576   qDebug() << "Error: file" << fileName << "does not exist";
```

```
01577   return NULL;
01578  }
01579 #ifdef SIMPLIFY_COLLISION_SHAPES
01580  btShapeHull* hull = new btShapeHull(shape);
01581  btScalar margin = shape->getMargin();
01582  hull->buildHull(margin);
01583  btConvexHullShape* simplifiedConvexShape = new btConvexHullShape((btScalar *) hull->getVertexPointer(),
      hull->numVertices());
01584  delete shape;
01585  return (btCollisionShape *) simplifiedConvexShape;
01586 #else
01587  return (btCollisionShape *) shape;
01588 #endif
01589 }
01590
01591 void SimInterface::setUpdateRate(float rate)
01592 {
01593  _simRates.timeScale = rate;
01594  if (_simRates.timeScale < 0.1)
01595   _simRates.timeScale = 0.1;
01596  updateTiming();
01597  emit ratesChanged(_simRates);
01598
01599 }
01600
01601 void SimInterface::increaseUpdateRate()
01602 {
01603
01604  _simRates.timeScale -= 0.1;
01605  if (_simRates.timeScale < 0.1)
01606   _simRates.timeScale = 0.1;
01607  updateTiming();
01608  emit ratesChanged(_simRates);
01609 }
01610
01611 void SimInterface::decreaseUpdateRate()
01612 {
01613  _simRates.timeScale += 0.1;
01614  updateTiming();
01615  emit ratesChanged(_simRates);
01616
01617 }
01618
01619
01620 void SimInterface::enableUpdateLimit()
01621 {
01622  if (!_simRates.limitRate)
01623  {
01624   _simRates.limitRate = true;
01625   updateTiming();
01626   emit ratesChanged(_simRates);
01627  }
01628 }
01629
01630 void SimInterface::disableUpdateLimit()
01631 {
01632  if (_simRates.limitRate)
01633  {
01634   _simRates.limitRate = false;
01635   updateTiming();
01636   emit ratesChanged(_simRates);
01637  }
01638 }
01639
01640 void SimInterface::toggleUpdateLimit()
01641 {
01642  _simRates.limitRate = !_simRates.limitRate;
01643  updateTiming();
01644  emit ratesChanged(_simRates);
01645 }
01646
01647 simRate_t SimInterface::getSimulatorRate()
01648 {
01649  return _simRates;
01650 }
```

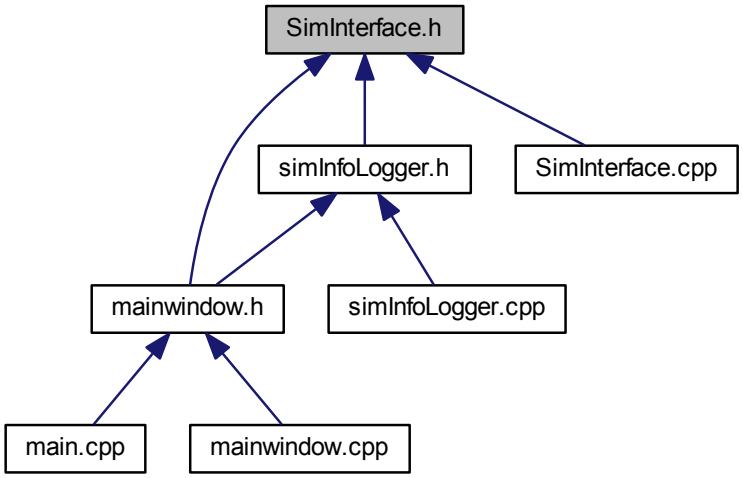
13.36 SimInterface.h File Reference

Declares the simulation interface class.

```
#include <QGlobal.h>
#include <QDebug>
#include <QObject>
#include <QVector>
#include <QTime>
#include <QImage>
#include <QString>
#include <QStringList>
#include <QFile>
#include <QFileInfo>
#include <QTextStream>
#include <QElapsedTimer>
#include <IDroplet.h>
#include <DropletSim.h>
#include <DropletSimInfo.h>
#include <DropletSimGlobals.h>
#include <DropletDataStructs.h>
#include <DefaultPrograms/DropletRGBSense/DropletRGBSense.h>
#include <DefaultPrograms/DropletRandomWalk/DropletRandomWalk.h>
#include <DefaultPrograms/DropletMarch/DropletMarch.h>
#include <DefaultPrograms/DropletRainbow/DropletRainbow.h>
#include <DefaultPrograms/DropletStickPullers/DropletStickPullers.h>
#include <DefaultPrograms/DropletTurnTest/DropletTurnTest.h>
#include <DefaultPrograms/DropletCommTest/DropletCommTest.h>
#include <DefaultPrograms/DropletPowerTest/DropletPowerTest.h>
#include <DefaultPrograms/DropletGranola/DropletGranola.h>
#include <DefaultPrograms/DropletStickPullersUpdated/DropletStickPullers-
Updated.h>
#include <DefaultPrograms/DropletAnts/DropletAnts.h>
#include <CustomPrograms/DropletCustomOne/DropletCustomOne.h>
#include <CustomPrograms/DropletCustomTwo/DropletCustomTwo.h>
#include <CustomPrograms/DropletCustomThree/DropletCustomThree.h>
#include <CustomPrograms/DropletCustomFour/DropletCustomFour.h>
#include <CustomPrograms/DropletCustomFive/DropletCustomFive.h>
#include <CustomPrograms/DropletCustomSix/DropletCustomSix.h>
#include <CustomPrograms/DropletCustomSeven/DropletCustomSeven.h>
#include <CustomPrograms/DropletCustomEight/DropletCustomEight.h>
#include <CustomPrograms/DropletCustomNine/DropletCustomNine.h>
#include <CustomPrograms/DropletCustomTen/DropletCustomTen.h>
#include <limits.h>
#include "defaults.h"
#include "structs.h"
#include <glm/glm.hpp>
#include <glm/gtc/quaternion.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <stdint.h>
#include <BulletCollision/CollisionShapes/btConvexHullShape.h>
#include <BulletCollision/CollisionShapes/btShapeHull.h>
```

Include dependency graph for SimInterface.h:

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



Classes

- class SimInterface

  Model that contains and controls the droplet simulator.

13.36.1 Detailed Description

Declares the simulation interface class.

Definition in file SimInterface.h.

13.37 SimInterface.h

```
00001
00007 #ifndef SIMULATORINTERFACE_H
00008 #define SIMULATORINTERFACE_H
00009
00010 #include <QGlobal.h>
00011 #include <QDebug>
00012 #include <QObject>
00013 #include <QVector>
00014 #include <QTime>
00015 #include <QImage>
00016 #include <QString>
00017 #include <QStringList>
00018 #include <QFile>
00019 #include <QFileInfo>
00020 #include <QTextStream>
00021 #include <QElapsedTimer>
00022
00023 #include <IDroplet.h>
00024 #include <DropletSim.h>
00025
00026 #include <DropletSimInfo.h>
00027 #include <DropletSimGlobals.h>
00028 #include <DropletDataStructs.h>
00029
00030 #include <DefaultPrograms/DropletRGBSense/DropletRGBSense.h>
00031 #include <DefaultPrograms/DropletRandomWalk/DropletRandomWalk.h>
00032 #include <DefaultPrograms/DropletMarch/DropletMarch.h>
00033 #include <DefaultPrograms/DropletRainbow/DropletRainbow.h>
```

```
00034 #include <DefaultPrograms/DropletStickPullers/DropletStickPullers.h>
00035 #include <DefaultPrograms/DropletTurnTest/DropletTurnTest.h>
00036 #include <DefaultPrograms/DropletCommTest/DropletCommTest.h>
00037 #include <DefaultPrograms/DropletPowerTest/DropletPowerTest.h>
00038 #include <DefaultPrograms/DropletGranola/DropletGranola.h>
00039 #include <DefaultPrograms/DropletStickPullersUpdated/DropletStickPullersUpdated.h>
00040 #include <DefaultPrograms/DropletAnts/DropletAnts.h>
00041
00042 #include <CustomPrograms/DropletCustomOne/DropletCustomOne.h>
00043 #include <CustomPrograms/DropletCustomTwo/DropletCustomTwo.h>
00044 #include <CustomPrograms/DropletCustomThree/DropletCustomThree.h>
00045 #include <CustomPrograms/DropletCustomFour/DropletCustomFour.h>
00046 #include <CustomPrograms/DropletCustomFive/DropletCustomFive.h>
00047 #include <CustomPrograms/DropletCustomSix/DropletCustomSix.h>
00048 #include <CustomPrograms/DropletCustomSeven/DropletCustomSeven.h>
00049 #include <CustomPrograms/DropletCustomEight/DropletCustomEight.h>
00050 #include <CustomPrograms/DropletCustomNine/DropletCustomNine.h>
00051 #include <CustomPrograms/DropletCustomTen/DropletCustomTen.h>
00052
00053 #include <limits.h>
00054
00055 #include "defaults.h"
00056 #include "structs.h"
00057
00058 #include <glm/glm.hpp>
00059 #include <glm/gtc/quaternion.hpp>
00060 #include <glm/gtc/matrix_transform.hpp>
00061
00062 #include <stdint.h>
00063
00064 #include <BulletCollision/CollisionShapes/btConvexHullShape.h>
00065 #include <BulletCollision/CollisionShapes/btShapeHull.h>
00066
00078 class SimInterface : public QObject
00079 {
00080  Q_OBJECT
00081
00082 public:
00083  SimInterface(QObject *parent = 0);
00084  ~SimInterface();
00085
00094  void Init();
00095
00107  float getRandomf(float min, float max);
00108
00117  bool isPaused(void);
00118
00127  simSetting_t getDefaultSettings();
00128
00137  simSetting_t getCurrentSettings();
00138
00147  simSetting_t getSimulatorSettings();
00148
00157  simState_t getSimulatorState();
00158
00159
00160
00167  void createArena();
00168
00177  simRate_t getSimulatorRate();
00178
00179 signals:
00180
00190  void arenaChanged(simSetting_t arena);
00191
00201  void simulationUpdated(simState_t simInfo);
00202
00212  void pauseChanged(bool paused);
00213
00225  void ratesChanged(simRate_t rates);
00226
00227 public slots:
00228
00236  void loadTilePositions();
00237
00252  void loadTilePositions(QString filename);
00253
00260  void pause(void);
00261
00268  void resume(void);
00269
00277  void togglePause(void);
00278
00288  void Update(float timeSinceLastUpdate);
00289
00296  void reset(void);
00297
```

```
00310  void addDroplet(float x, float y, droplet_t dType = March, int dropletID = 0);
00311
00325  int addNewShape(object_t objectType, float radius, vec3 scale);
00326
00341  void addSphere( float x, float y, int objectID, float radius, float mass, float friction);
00342
00357  void addCube( float x, float y, int objectID, vec3 scale, float mass, float friction);
00358
00370  void addWall( float x, float y, int objectID, vec3 scale);
00371
00383  void addFloor( float x, float y, int objectID, vec3 scale);
00384
00399  void addObject(object_t kind, float x, float y, float radius = 1.0f, float mass =
       DEFAULT_OBJECT_MASS, float friction = DEFAULT_OBJECT_FRICTION);
00400
00410  void loadArena(simSetting_t arena);
00411
00421  void setUpdateRate(float rate);
00422
00430  void increaseUpdateRate(void);
00431
00439  void decreaseUpdateRate(void);
00440
00448  void enableUpdateLimit(void);
00449
00457  void disableUpdateLimit(void);
00458
00466  void toggleUpdateLimit(void);
00467
00468 protected:
00469
00478  void timerEvent ( QTimerEvent * event );
00479
00480
00481 private:
00494  IDroplet* newDropletOfType(droplet_t dType,ObjectPhysicsData *dropletPhyDat);
00495
00506  btCollisionShape* makeCollisionShapeFromFile(QString file);
00507
00516  void makeDropletCollisionShapeFromFile(QString file);
00517
00524  void teardownSim();
00525
00526  //variables
00527  DropletSim *_sim;
00528  DropletSimInfo _simInfo;
00529
00534  simSetting_t _simSettings;
00535  simRate_t _simRates;
00536
00537
00538
00539
00540  struct {
00541   bool paused;
00542   // TIMER
00543   //uint64_t runTimeSec;
00544   //double runTimeSubSec;
00545   //float runTime;
00546   double runTime;
00547   btCollisionShape *dropletShape;
00548   float dropletOffset;
00549   int btXWallShapeID;
00550   int btYWallShapeID;
00551   int btFloorShapeID;
00552   int btDropletShapeID;
00553   //int btCubeShapeID;
00554   //int btSphereShapeID;
00555  } _simStatus;
00556
00561  simState_t _simState;
00562
00569  void updateTiming();
00570
00571  std::vector<unsigned char *> *_dropletColors;
00572
00577  std::vector<GPSInfo *> *_dropletPos;
00578  std::vector<DropletCommData *> *_dropletComm;
00579
00584  std::vector<GPSInfo *> *_objectPos;
00585  DropletTimeControl *_timer;
00586
00591  QVector<vec2> _tilePositions;
00592  QVector<vec4> _wallBools;
00593
00594  // TIMER
00595  //QElapsedTimer _updateTimer,_realTime;
```

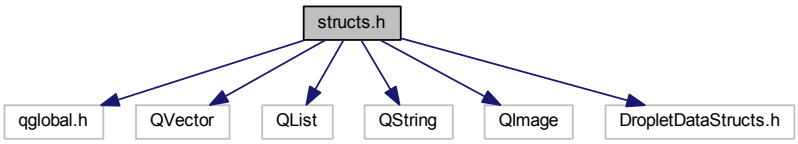
```
00596  QElapsedTimer _updateTimer;
00597
00602  int _timerID;
00603
00604  double _realTime;
00605
00610  double _simTimeScale;
00611
00616  double _targetUpdateTime;
00617
00622  double _timeUntilNextUpdate;
00623
00624  struct {
00625   QStringList single;
00626   QStringList multiple;
00627  } _objectNames;
00628
00629
00630 };
00631
00632 #endif // SIMULATORINTERFACE_H
```

13.38 structs.h File Reference

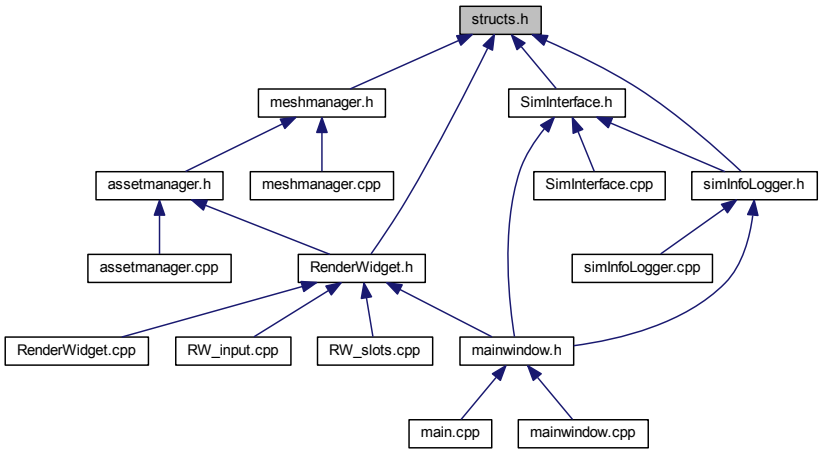
Declares global structs and types.

```
#include <qglobal.h>
#include <QVector>
#include <QList>
#include <QString>
#include <QImage>
#include <DropletDataStructs.h>
```

Include dependency graph for structs.h:



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



Classes

- struct collisionShapeStruct_t

  Defines the properties of a collision shape.
- struct dropletStruct_t

  Defines properties about droplets that are emitted from SimInterface.
- struct objectStruct_t

  Defines properties about physical objects that are emitted from SimInterface.
- struct simRate_t

  Defines the current state of rate limiting emitted by the SimInterface.
- struct simSetting_t

  Defines simulator settings that are passed in and out of SimInterface.
- struct simState_t

  Defines current state of the simulator that is emitted by SimInterface.
- union vec2

  Two element vector of floats.
- union vec3

  Three element vector of floats.
- union vec3i

  Three element vector of integers.
- union vec4

  Four element vector of floats.

Enumerations

- enum droplet_t {
  RGBSense, RandomWalk, March, Rainbow,
  StickPullers, TurnTest, CommTest, PowerTest,
  Granola, StickPullersUpdated, Ants, CustomOne,
  CustomTwo, CustomThree, CustomFour, CustomFive,
  CustomSix, CustomSeven, CustomEight, CustomNine,
  CustomTen, NUM_DROPLET_PROGRAMS }

Enumerated type defining possible droplet programs.

- enum object_t { Sphere, Cube, Wall, Floor }

  Enumerated type defining possible physical objects.

Functions

- Q_DECLARE_TYPEINFO (objectStruct_t, Q_PRIMITIVE_TYPE)
- Q_DECLARE_TYPEINFO (dropletStruct_t, Q_PRIMITIVE_TYPE)
- Q_DECLARE_TYPEINFO (collisionShapeStruct_t, Q_PRIMITIVE_TYPE)
- Q_DECLARE_TYPEINFO (simSetting_t, Q_PRIMITIVE_TYPE)
- Q_DECLARE_TYPEINFO (simState_t, Q_PRIMITIVE_TYPE)
- Q_DECLARE_TYPEINFO (simRate_t, Q_PRIMITIVE_TYPE)

13.38.1 Detailed Description

Declares global structs and types.

Definition in file structs.h.

13.38.2 Function Documentation

13.38.2.1 Q_DECLARE_TYPEINFO (objectStruct_t , Q_PRIMITIVE_TYPE)

13.38.2.2 Q_DECLARE_TYPEINFO (dropletStruct_t , Q_PRIMITIVE_TYPE)

13.38.2.3 Q_DECLARE_TYPEINFO (collisionShapeStruct_t , Q_PRIMITIVE_TYPE)

13.38.2.4 Q_DECLARE_TYPEINFO (simSetting_t , Q_PRIMITIVE_TYPE)

13.38.2.5 Q_DECLARE_TYPEINFO (simState_t , Q_PRIMITIVE_TYPE)

13.38.2.6 Q_DECLARE_TYPEINFO (simRate_t , Q_PRIMITIVE_TYPE)

13.39 structs.h

```
00001
00007 #ifndef STRUCTS_H
00008 #define STRUCTS_H
00009
00010 #include <qglobal.h>
00011 #include <QVector>
00012 #include <QList>
00013 #include <QString>
00014 #include <QImage>
00015 #include <DropletDataStructs.h>
00016
00031 union vec4
00032 {
00033     struct { float x, y, z, w; };
00034     struct { float r, g, b, a; };
00035     struct { float s, t, p, q; };
00036     float v[4];
00037 };
00038
00045 union vec3i
00046 {
00047     struct { unsigned char x, y, z; };
00048     struct { unsigned char r, g, b; };
00049     struct { unsigned char s, t, p; };
00050     unsigned char v[4];
00051 };
00052
00059 union vec3
00060 {
00061     struct { float x, y, z; };
00062     struct { float r, g, b; };
00063     struct { float s, t, p; };
```

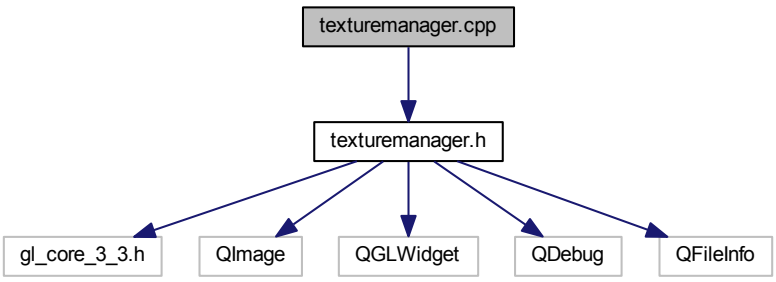
```
00064     float v[3];
00065 };
00066
00073 union vec2
00074 {
00075     struct { float x, y; };
00076     struct { float s, t; };
00077     float v[2];
00078 };
00079
00080
00093 static inline vec3 vec3make(float x, float y, float z)
00094 {
00095     vec3 v = { x, y, z };
00096     return v;
00097 }
00098
00111 static inline vec3i vec3imake(int x, int y, int z)
00112 {
00113     vec3i v = { x, y, z };
00114     return v;
00115 }
00116
00128 static inline vec2 vec2make(float x, float y)
00129 {
00130     vec2 v = { x, y};
00131     return v;
00132 }
00133
00147 static inline vec4 vec4make(float x, float y, float z, float w)
00148 {
00149     vec4 v = { x, y, z, w};
00150     return v;
00151 }
00152
00160 enum droplet_t {
00161  RGBSense,
00162  RandomWalk,
00163  March,
00164  Rainbow,
00165  StickPullers,
00166  TurnTest,
00167  CommTest,
00168  PowerTest,
00169  Granola,
00170  StickPullersUpdated,
00171  Ants,
00172  CustomOne,
00173  CustomTwo,
00174  CustomThree,
00175  CustomFour,
00176  CustomFive,
00177  CustomSix,
00178  CustomSeven,
00179  CustomEight,
00180  CustomNine,
00181  CustomTen,
00182  NUM_DROPLET_PROGRAMS
00183 };
00184
00185 /* object structs */
00186
00194 struct dropletStruct_t {
00195  int dropletID;
00196  vec3 origin;
00197  vec3 rotation;
00198  vec4 quaternion;
00199  vec3i color;
00200  DropletCommData commData;
00201  bool changed;
00202 };
00203
00211 enum object_t {
00212  Sphere,
00213  Cube,
00214  Wall,
00215  Floor
00216 };
00217
00225 struct objectStruct_t {
00226  int objectID;
00227  float objectRadius;
00228  object_t oType;
00229  vec3 scale;
00230  vec3 origin;
00231  vec3 rotation;
00232  vec4 quaternion;
```

```
00233  vec3i color;
00234  bool changed;
00235 };
00236
00237
00245 struct simSetting_t {
00246  QString arenaName;
00247  int numRowTiles;
00248  int numColTiles;
00249  float tileLength;
00250  float dropletRadius;
00251  float wallHeight;
00252  float wallWidth;
00253  float fps;
00254  float dropletOffset;
00255  bool projecting;
00256  QString projTexture;
00257  QString floorFile;
00258  QVector<QStringList> startingDroplets;
00259  QVector<QStringList> startingObjects;
00260 };
00261
00268 struct collisionShapeStruct_t {
00269  object_t oType;
00270  float objectRadius;
00271  vec3 scale;
00272  int collisionID;
00273 };
00274
00282 struct simState_t {
00283  //QTime simTime;
00284  //QTime realTime;
00285  double simTime;
00286  double realTime;
00287  double timeRatio;
00288  QImage projTexture;
00289  bool projTextureChanged;
00290  float dropletOffset;
00291  QVector<dropletStruct_t> dropletData;
00292  QVector<objectStruct_t> dynamicObjectData;
00293  QVector<objectStruct_t> staticObjectData;
00294  QList<collisionShapeStruct_t> collisionShapes;
00295 };
00296
00304 struct simRate_t {
00305  float timeScale;
00306  bool limitRate;
00307 };
00308  // end of globals
00310
00311 Q_DECLARE_TYPEINFO(objectStruct_t,Q_PRIMITIVE_TYPE);
00312 Q_DECLARE_TYPEINFO(dropletStruct_t,Q_PRIMITIVE_TYPE);
00313 Q_DECLARE_TYPEINFO(collisionShapeStruct_t,Q_PRIMITIVE_TYPE);
00314 Q_DECLARE_TYPEINFO(simSetting_t,Q_PRIMITIVE_TYPE);
00315 Q_DECLARE_TYPEINFO(simState_t,Q_PRIMITIVE_TYPE);
00316 Q_DECLARE_TYPEINFO(simRate_t,Q_PRIMITIVE_TYPE);
00317
00318 #endif
```

13.40 texturemanager.cpp File Reference

Implements the TextureManager class.

`#include "texturemanager.h"`
Include dependency graph for texturemanager.cpp:



13.40.1 Detailed Description

Implements the TextureManager class.

Definition in file texturemanager.cpp.

13.41 texturemanager.cpp

```
00001
00007 #include "texturemanager.h"
00008
00009 TextureManager::TextureManager()
00010 {
00011  _textureInfo.width = 0;
00012  _textureInfo.height = 0;
00013  _textureInfo.handle = 0;
00014  _hasLoaded = false;
00015  _fileSet = false;
00016  _fileName = "";
00017  ogl_LoadFunctions();
00018 }
00019
00020 TextureManager::~TextureManager()
00021 {
00022  freeTexture();
00023 }
00024
00025 bool TextureManager::setFile(QString fileName)
00026 {
00027  QFileInfo inf(fileName);
00028  if (inf.exists())
00029  {
00030   freeTexture();
00031   _fileName = fileName;
00032   _fileSet = true;
00033   return true;
00034  } else
00035  {
00036   return false;
00037  }
00038 }
00039
00040 bool TextureManager::loadFile(QString fileName)
00041 {
00042  if (setFile(fileName))
00043  {
00044   QImage temp;
00045   bool result = temp.load(fileName);
00046   if (!result)
00047   {
00048    qDebug() << "Could not load file" << fileName;
00049    return false;
00050   }
00051
```

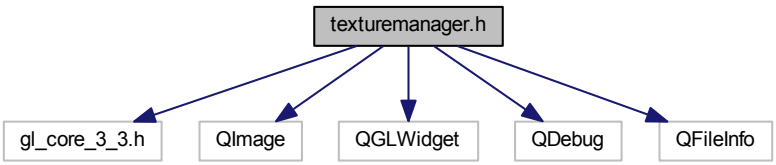
```
00052
00053
00054   QImage image = QGLWidget::convertToGLFormat(temp);
00055   _textureInfo.width = image.width();
00056   _textureInfo.height = image.height();
00057
00058   glGenTextures(1,&_textureInfo.handle);
00059   glBindTexture(GL_TEXTURE_2D, _textureInfo.handle);
00060   glTexImage2D(GL_TEXTURE_2D, 0, 4, _textureInfo.width, _textureInfo.height, 0,
      GL_RGBA, GL_UNSIGNED_BYTE, image.bits());
00061   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_LINEAR); // Linear Filtering
00062   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // Linear Filtering
00063   glGenerateMipmap(GL_TEXTURE_2D);
00064   glBindTexture(GL_TEXTURE_2D,0);
00065   qDebug() << "Bound texture to ID" << _textureInfo.handle;
00066   _hasLoaded = true;
00067
00068   return true;
00069  } else {
00070   return false;
00071  }
00072 }
00073 void TextureManager::bindTexture()
00074 {
00075  if (!_hasLoaded && _fileSet)
00076  {
00077   loadFile(_fileName);
00078  }
00079
00080
00081  if (_hasLoaded)
00082  {
00083   glBindTexture(GL_TEXTURE_2D,_textureInfo.handle);
00084  }
00085 }
00086 void TextureManager::unbindTexture()
00087 {
00088  glBindTexture(GL_TEXTURE_2D,0);
00089 }
00090
00091 void TextureManager::freeTexture()
00092 {
00093  if (_hasLoaded)
00094  {
00095   glDeleteTextures(1,&_textureInfo.handle);
00096   _hasLoaded = false;
00097   _textureInfo.handle = 0;
00098   _textureInfo.height = 0;
00099   _textureInfo.width = 0;
00100  }
00101 }
```

13.42 texturemanager.h File Reference

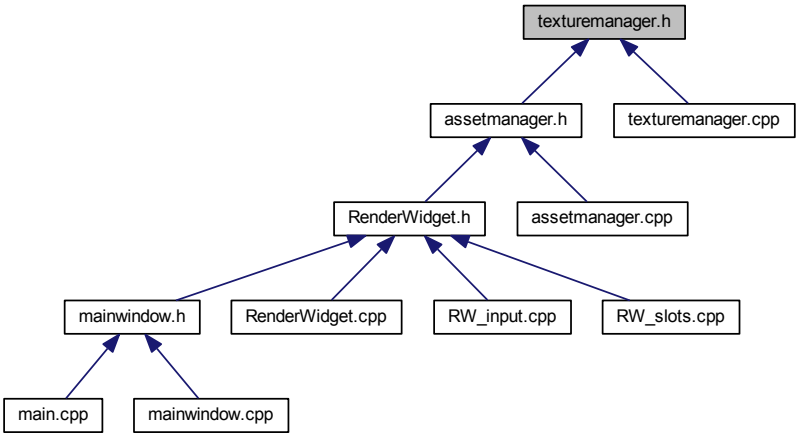
Declares the TextureManager class.

```
#include "gl_core_3_3.h"
#include <QImage>
#include <QGLWidget>
#include <QDebug>
#include <QFileInfo>
```

Include dependency graph for texturemanager.h:



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



Classes

- class TextureManager

  Container class for a texture.

Macros

- #define TEXTUREMANAGER_H

  A macro that defines texturemanager h.

13.42.1 Detailed Description

Declares the TextureManager class.

Definition in file texturemanager.h.

13.42.2 Macro Definition Documentation

13.42.2.1 #define TEXTUREMANAGER_H

A macro that defines texturemanager h.

Definition at line 15 of file texturemanager.h.

13.43 texturemanager.h

```
00001
00007 #ifndef TEXTUREMANAGER_H
00008
00015 #define TEXTUREMANAGER_H
00016
00017 #include "gl_core_3_3.h"
00018 #include <QImage>
00019 #include <QGLWidget>
00020 #include <QDebug>
00021 #include <QFileInfo>
00022
```

```
00030 class TextureManager
00031 {
00032
00033
00034 public:
00035
00042  TextureManager();
00043  ~TextureManager();
00044  bool loadFile(QString fileName);
00045
00052  void bindTexture();
00053
00060  void unbindTexture();
00061
00072  bool setFile(QString fileName);
00073  void freeTexture();
00074 private:
00075  struct {
00076   GLuint width, height;
00077   GLuint handle;
00078   bool valid;
00079  } _textureInfo;
00080  bool _hasLoaded;
00081  bool _fileSet;
00082  QString _fileName;
00083
00084 };
00085
00086 #endif // TEXTUREMANAGER_H
```

13.44 tutorial.dox File Reference

Index