DropletConsole

Generated by Doxygen 1.8.3.1

Mon May 13 2013 02:38:16

Contents

1 DropletConsole

1.1 Introduction

DropletConsole 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 DropletConsole.

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

DropletConsole 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, DropletConsole is not compatible with Qt 5.0 and will not build or run correctly with it.

1.3.3 Step 3: Building DropletConsole

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

1. Navigate to the project folder found in `DropletSimulator/DropletSimDemos/Droplet-Console/vs2010/`
2. Open the Visual Studio solution file `DropletConsole.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 DropletConsole. 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

DropletConsole 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 DropletConsole 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, DropletConsole is not compatible with Qt 5.0 and will not run correctly with it.

2.3 Building DropletConsole

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

1. Navigate to the project folder found in `DropletSimulator/DropletSimDemos/Droplet-Console/vs2010/`
2. Open the Visual Studio solution file `DropletConsole.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 DropletConsole 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 DropletConsole 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/-DropletConsole/docs/Doxyfile`
3. Select the "Run" tab
4. Click "Run Doxygen"

This will create two directories inside `DropletSimulator/DropletSimDemos/DropletConsole/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/DropletConsole/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

DropletConsole 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\DropletConsole\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\DropletConsole\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 DropletConsole 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\DropletConsole\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\DropletConsole\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 DropletConsole 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 File Index

7.1 File List

Here is a list of all files with brief descriptions:

8 File Documentation

8.1 build.dox File Reference

8.2 doxgyen.dox File Reference

8.3 features.dox File Reference

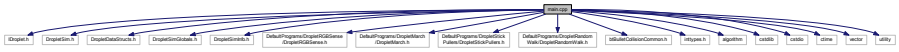
8.4 files.dox File Reference

8.5 main.cpp File Reference

This file is the main file of the console project that runs the simulator without a GUI. This project compiles on windows and UNIX systems. It also provides a good example of how to setup and run the simulator.

```
#include <IDroplet.h>
#include <DropletSim.h>
#include <DropletDataStructs.h>
#include <DropletSimGlobals.h>
#include <DropletSimInfo.h>
#include <DefaultPrograms/DropletRGBSense/DropletRGBSense.h>
#include <DefaultPrograms/DropletMarch/DropletMarch.h>
#include <DefaultPrograms/DropletStickPullers/DropletStickPullers.h>
#include <DefaultPrograms/DropletRandomWalk/DropletRandomWalk.h>
#include <btBulletCollisionCommon.h>
#include <inttypes.h>
#include <algorithm>
#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <vector>
#include <utility>
```

Include dependency graph for main.cpp:



Macros

- #define DEFAULT_COL_TILES 4
- #define DEFAULT_DROPLET_RADIUS 2.0f
- #define DEFAULT_FPS 60.0f
- #define DEFAULT_NUM_DROPLETS 1
- #define DEFAULT_PRINT_INTERVAL .5
- #define DEFAULT_ROW_TILES 4
- #define DEFAULT_TILE_LENGTH 24.0f
- #define DEFAULT_TOTAL_TIMESTEPS 5000
- #define DEFAULT_WALL_HEIGHT 5.0f
- #define DEFAULT_WALL_WIDTH 5.0f

Functions

- float getRandomf (float min, float max)

  gets a random float within the spcified range.
- void initSimulator (void)

  initializes the sim with default parameters and sets up the projector image and physics objects.
- int main (int argc, char ∗argv[])

  main function:
- void PrintComputation (std::vector< DropletCompData ∗ > ∗comp, FILE ∗fp)
- void PrintStatus (std::vector< unsigned char ∗ > ∗colors, std::vector< GPSInfo ∗ > ∗xyVals, std::vector< DropletCommData ∗ > ∗comm, FILE ∗file)

  prints sim info into a specified file. The params given to the function are vectors filled with the info of that type for all the droplets.
- void PrintWaitStatus (std::vector< GPSInfo ∗ > ∗gpsInfo, std::vector< DropletCommData ∗ > ∗commData, std::vector< DropletActuatorData ∗ > ∗actData, FILE ∗file)
- void setupSimObjects ()

  creates the arena and adds droplets with physics model. Droplets are added in random locations between the bounding rectangle of the arena. Droplets are created with a mass of 1 gram and .8 friction.

Variables

- int btDropletShapeID

  used to store IDs of bullet collision shapes.
- int btFloorShapeID
- int btXWallShapeID
- int btYWallShapeID
- float dropletRadius
- float fps
- int numColTiles
- int numDroplets
- int numRowTiles
- int numSteps

  simulator params: numRowTiles and numColTiles refer to the dimensions of the arena numSteps is the number of steps to run the simulator fps determines the steps simulated per second, not the rendering rate printInterval determines how many steps to compute before the next print
- float printInterval
- DropletSim sim

  simulator instance.
- DropletSimInfo simInfo

  sim info instance. This class is used to gather all info on the droplets stored in DropletDataStructs
- float tileLength
- float wallHeight
- float wallWidth

8.5.1 Detailed Description

This file is the main file of the console project that runs the simulator without a GUI. This project compiles on windows and UNIX systems. It also provides a good example of how to setup and run the simulator. Command line options: -n number of droplets -f output file -t number of steps to run the simulator -p print interval

Definition in file main.cpp.

8.5.2 Macro Definition Documentation

8.5.2.1 #define DEFAULT_COL_TILES 4

Definition at line 38 of file main.cpp.

8.5.2.2 #define DEFAULT_DROPLET_RADIUS 2.0f

Definition at line 41 of file main.cpp.

8.5.2.3 #define DEFAULT_FPS 60.0f

Definition at line 44 of file main.cpp.

8.5.2.4 #define DEFAULT_NUM_DROPLETS 1

Definition at line 39 of file main.cpp.

8.5.2.5 #define DEFAULT_PRINT_INTERVAL .5

Definition at line 46 of file main.cpp.

8.5.2.6 #define DEFAULT_ROW_TILES 4

Definition at line 37 of file main.cpp.

8.5.2.7 #define DEFAULT_TILE_LENGTH 24.0f

Definition at line 40 of file main.cpp.

8.5.2.8 #define DEFAULT_TOTAL_TIMESTEPS 5000

Definition at line 45 of file main.cpp.

8.5.2.9 #define DEFAULT_WALL_HEIGHT 5.0f

Definition at line 42 of file main.cpp.

8.5.2.10 #define DEFAULT_WALL_WIDTH 5.0f

Definition at line 43 of file main.cpp.

8.5.3 Function Documentation

8.5.3.1 float getRandomf (float *min*, float *max*)

gets a random float within the spcified range.

Parameters

min	The minimum.
max	The maximum.

Returns

The random float value.

Definition at line 94 of file main.cpp.

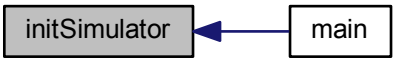
Here is the caller graph for this function:

8.5.3.2 void initSimulator (void)

initializes the sim with default parameters and sets up the projector image and physics objects.

Definition at line 106 of file main.cpp.

Here is the caller graph for this function:



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

main function:

1. gathers command line input arguments.
2. initialize the simulator
3. set up the physcis objects
4. allocate memory for vectors containing output info
5. steps the simulator for a given number of steps and prints info

clean up simulator and free memory

Parameters

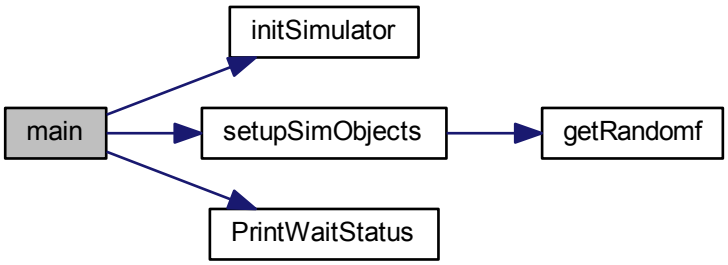
argc	Number of command-line arguments.
argv	Array of command-line argument strings.

Returns

Exit-code for the process - 0 for success, else an error code.

Definition at line 301 of file main.cpp.

Here is the call graph for this function:



8.5.3.4 void PrintComputation (std::vector< DropletCompData * > * *comp*, FILE * *fp*)

Definition at line 267 of file main.cpp.

8.5.3.5 void PrintStatus (std::vector< unsigned char * > * *colors*, std::vector< GPSInfo * > * *xyVals*, std::vector< DropletCommData * > * *comm*, FILE * *file*)

prints sim info into a specified file. The params given to the function are vectors filled with the info of that type for all the droplets.

Parameters

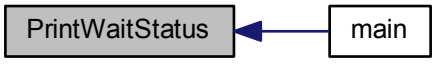
in,out	*colors*	If non-null, the colors.
in,out	*xyVals*	If non-null, the xy vals.
in,out	*comm*	If non-null, the communications.
in,out	*file*	If non-null, the file.

Definition at line 207 of file main.cpp.

8.5.3.6 void PrintWaitStatus (std::vector< GPSInfo * > * *gpsInfo*, std::vector< DropletCommData * > * *commData*, std::vector< DropletActuatorData * > * *actData*, FILE * *file*)

Definition at line 242 of file main.cpp.

Here is the caller graph for this function:

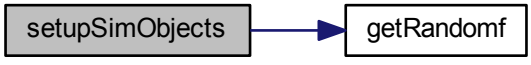


8.5.3.7 void setupSimObjects ( )

creates the arena and adds droplets with physics model. Droplets are added in random locations between the bounding rectangle of the arena. Droplets are created with a mass of 1 gram and .8 friction.

Definition at line 162 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

8.5.4 Variable Documentation

8.5.4.1 int btXWallShapeID btYWallShapeID btFloorShapeID btDropletShapeID

used to store IDs of bullet collision shapes.

Definition at line 81 of file main.cpp.

8.5.4.2 int btFloorShapeID

Definition at line 81 of file main.cpp.

8.5.4.3 int btXWallShapeID

Definition at line 81 of file main.cpp.

8.5.4.4 int btYWallShapeID

Definition at line 81 of file main.cpp.

8.5.4.5 float dropletRadius

Definition at line 73 of file main.cpp.

8.5.4.6 float fps

Definition at line 73 of file main.cpp.

8.5.4.7 int numColTiles

Definition at line 72 of file main.cpp.

8.5.4.8 int numDroplets

Definition at line 72 of file main.cpp.

8.5.4.9 int numRowTiles

Definition at line 72 of file main.cpp.

8.5.4.10 int numRowTiles numColTiles numDroplets numSteps

simulator params: numRowTiles and numColTiles refer to the dimensions of the arena numSteps is the number of steps to run the simulator fps determines the steps simulated per second, not the rendering rate printInterval determines how many steps to compute before the next print

float tileLength, dropletRadius, wallWidth, wallHeight, fps, printInterval;

Definition at line 72 of file main.cpp.

8.5.4.11 float printInterval

Definition at line 73 of file main.cpp.

8.5.4.12 DropletSim sim

simulator instance.

Definition at line 53 of file main.cpp.

8.5.4.13 DropletSimInfo simInfo

sim info instance. This class is used to gather all info on the droplets stored in DropletDataStructs

Definition at line 59 of file main.cpp.

8.5.4.14 float tileLength

Definition at line 73 of file main.cpp.

8.5.4.15 float wallHeight

Definition at line 73 of file main.cpp.

8.5.4.16 float wallWidth

Definition at line 73 of file main.cpp.

8.6 main.cpp

```
00001
00015 #include <IDroplet.h>
00016 #include <DropletSim.h>
00017 #include <DropletDataStructs.h>
00018 #include <DropletSimGlobals.h>
00019 #include <DropletSimInfo.h>
00020
00021 #include <DefaultPrograms/DropletRGBSense/DropletRGBSense.h>
00022 #include <DefaultPrograms/DropletMarch/DropletMarch.h>
00023 #include <DefaultPrograms/DropletStickPullers/DropletStickPullers.h>
00024 #include <DefaultPrograms/DropletRandomWalk/DropletRandomWalk.h>
00025
00026 #include <btBulletCollisionCommon.h>
00027
00028 #include <inttypes.h>
00029 #include <algorithm>
00030 #include <cstdlib>
00031 #include <cstdio>
00032 #include <ctime>
00033 #include <vector>
00034 #include <utility>
00035
00036 // Droplet Simulator specific constants.
00037 #define DEFAULT_ROW_TILES 4
00038 #define DEFAULT_COL_TILES 4
00039 #define DEFAULT_NUM_DROPLETS 1
00040 #define DEFAULT_TILE_LENGTH 24.0f
00041 #define DEFAULT_DROPLET_RADIUS 2.0f
00042 #define DEFAULT_WALL_HEIGHT 5.0f // in cm
00043 #define DEFAULT_WALL_WIDTH 5.0f
00044 #define DEFAULT_FPS 60.0f // 60 frames per second
00045 #define DEFAULT_TOTAL_TIMESTEPS 5000
00046 #define DEFAULT_PRINT_INTERVAL .5 // prints output to file for each .5 secs in sim time
00047
00048
00053 DropletSim sim;
00054
00059 DropletSimInfo simInfo;
00060
00072 int numRowTiles, numColTiles, numDroplets,
      numSteps;
00073 float tileLength, dropletRadius, wallWidth,
      wallHeight, fps, printInterval;
00074
00081 int btXWallShapeID, btYWallShapeID, btFloorShapeID,
```

```
      btDropletShapeID;
00082
00094 float getRandomf(float min, float max)
00095 {
00096  float range = max - min;
00097  return min + (range * ((float)rand() / RAND_MAX));
00098 }
00099
00106 void initSimulator(void)
00107 {
00108  // Set up required values
00109  // TODO : Setting up these values should go in a different function
00110  numRowTiles  = DEFAULT_ROW_TILES;
00111  numColTiles  = DEFAULT_COL_TILES;
00112  // numDroplets  = DEFAULT_NUM_DROPLETS;
00113  tileLength  = DEFAULT_TILE_LENGTH;
00114  dropletRadius = DEFAULT_DROPLET_RADIUS;
00115  wallHeight  = DEFAULT_WALL_HEIGHT;
00116  wallWidth  = DEFAULT_WALL_WIDTH;
00117  fps    = DEFAULT_FPS;
00118
00119  // Initialize Simulator
00120  SimSetupData setupData(
00121   numRowTiles,
00122   numColTiles,
00123   tileLength,
00124   dropletRadius,
00125   fps,
00126   true);
00127  sim.Init(setupData);
00128
00129  // NOTE : Set up the projector ONLY if we need it! It slows the simulation down.
00130  sim.SetUpProjector(std::string(".//"), "ScaledArena.bmp");
00131
00132  // Set up simulator phyiscs objects
00133  btCollisionShape *floorShape = new btStaticPlaneShape(btVector3(0, 0, 1) , 0);
00134  btCollisionShape *xWallShape = new btBoxShape(btVector3(
00135   btScalar(tileLength * numRowTiles),
00136   btScalar(wallWidth),
00137   btScalar(wallHeight)));
00138  btCollisionShape *yWallShape = new btBoxShape(btVector3(
00139   btScalar(wallWidth),
00140   btScalar(tileLength * numColTiles),
00141   btScalar(wallHeight)));
00142  btCollisionShape *dropletShape = new btCylinderShape(btVector3(
00143   dropletRadius,
00144   dropletRadius,
00145   0.5f));
00146
00147  sim.AddCollisionShape(floorShape, &btFloorShapeID);
00148  sim.AddCollisionShape(xWallShape, &btXWallShapeID);
00149  sim.AddCollisionShape(yWallShape, &btYWallShapeID);
00150  sim.AddCollisionShape(dropletShape, &btDropletShapeID);
00151
00152 }
00153
00162 void setupSimObjects()
00163 {
00164  // Create the floor and walls
00165  sim.CreateFloor(btFloorShapeID, btXWallShapeID,
      btYWallShapeID);
00166
00167  float floorWidth = tileLength * numColTiles;
00168  float floorLength = tileLength * numRowTiles;
00169  float posRangeWidth = floorWidth / 2.0f;
00170  float posRangeLength = floorLength / 2.0f;
00171
00172  for(int i = 0; i < numDroplets; i++)
00173  {
00174   // Set up the simulator/physics model
00175   ObjectPhysicsData *dropletPhyDat = (ObjectPhysicsData *)malloc(sizeof(ObjectPhysicsData));
00176   dropletPhyDat->colShapeIndex = btDropletShapeID;
00177   dropletPhyDat->mass = 1.0;
00178   dropletPhyDat->localInertia = btVector3(0.0, 0.0, 0.0);
00179   dropletPhyDat->friction = 0.8f;
00180
00181
00182   //IDroplet *newDroplet = new DropletMarch(dropletPhyDat);
00183   //IDroplet *newDroplet = new DropletStickPullers(dropletPhyDat);
00184   IDroplet *newDroplet = new DropletRandomWalk(dropletPhyDat);
00185
00186   sim.AddDroplet(newDroplet, std::make_pair(
00187    getRandomf(-posRangeWidth + dropletRadius, posRangeWidth -
      dropletRadius),
00188    getRandomf(-posRangeLength + dropletRadius, posRangeLength -
      dropletRadius)),
00189    0.0f);
```

```
00190
00191  }
00192 }
00193
00207 void PrintStatus(std::vector<unsigned char *> *colors, std::vector<GPSInfo *> *xyVals,
00208      std::vector<DropletCommData *> *comm, FILE *file)
00209 {
00210
00211  std::vector<GPSInfo *>::iterator it;
00212  it = xyVals->begin();
00213  // NOTE : 'i' skips over tiles & walls and starts at droplets
00214  int j = 0;
00215
00216  for(unsigned int i = 0; i < xyVals->size(); i++)
00217  {
00218   GPSInfo *gpsInfo = *it;
00219   fprintf(file,"[Droplet %05u] Color (R, G , B) = (%03u, %03u, %03u) Postion (X, Y) = (%9.5f, %9.5f)\n",
00220    i + 1,
00221    colors->at(i)[0],
00222    colors->at(i)[1],
00223    colors->at(i)[2],
00224    gpsInfo->posX,
00225    gpsInfo->posY);
00226   if (comm->at(i)->sendActive) {
00227    for (unsigned int j = 0; j < 6; j++) {
00228     fprintf(file, "\tComm Channel %u(lastMsgOut, lastMsgIn, outMsgLen, inMsgLen) = (%5u, %5u, %3u, %3u)\n",
00229      j,
00230      comm->at(i)->commChannels[j].lastMsgOutTimestamp,
00231      comm->at(i)->commChannels[j].lastMsgInTimestamp,
00232      comm->at(i)->commChannels[j].outMsgLength,
00233      comm->at(i)->commChannels[j].inMsgLength);
00234    }
00235   }
00236   it++;
00237
00238  }
00239
00240 }
00241
00242 void PrintWaitStatus(
00243  std::vector<GPSInfo *> *gpsInfo,
00244  std::vector<DropletCommData *> *commData,
00245  std::vector<DropletActuatorData *> *actData,
00246  FILE *file)
00247 {
00248 #ifdef _WIN32
00249  unsigned int size = min(commData->size(), actData->size());
00250 #elif __linux__
00251  unsigned int size = std::min(commData->size(), actData->size());
00252 #endif
00253  for(unsigned int i = 0; i < size; i++)
00254  {
00255   fprintf(file, "D %05u (X, Y) %f %f (R, G, B) %u %u %u MTR %g SA %u\n",
00256    i,
00257    gpsInfo->at(i)->posX,
00258    gpsInfo->at(i)->posY,
00259    actData->at(i)->rOut,
00260    actData->at(i)->gOut,
00261    actData->at(i)->bOut,
00262    actData->at(i)->moveTimeRemaining,
00263    commData->at(i)->sendActive);
00264  }
00265 }
00266
00267 void PrintComputation(std::vector<DropletCompData *> *comp, FILE* fp)
00268 {
00269  fprintf(fp,"====== COMPUTATION ======\n");
00270  for(int i = 0; i < numDroplets; i++)
00271  {
00272   fprintf(fp, "[Droplet %i] Comp (leg1Power, leg2Power, leg3Power, CapacitorPower, DropletID) = (%i,\t%i,\t
      %i,\t%u,%u)\n",
00273    i + 1,
00274    comp->at(i)->leg1Power,
00275    comp->at(i)->leg2Power,
00276    comp->at(i)->leg3Power,
00277    comp->at(i)->capacitorPower,
00278    comp->at(i)->dropletID);
00279
00280  }
00281 }
00282
00301 int main(int argc, char *argv[])
00302 {
00303 #ifdef _WIN32
00304  // for some reason, time_t is not simply an alias to an unsigned integer on Win32
00305  srand(static_cast<unsigned int>(time(0)));
00306 #else
```

```
00307  srand(time(0));
00308 #endif
00309
00310  FILE *outFile = NULL;
00311
00312  /* Process program arguments to select iterations and policy */
00313  /* Set default iterations if not supplied */
00314
00315  numDroplets = DEFAULT_NUM_DROPLETS;
00316  numSteps = DEFAULT_TOTAL_TIMESTEPS;
00317  printInterval = DEFAULT_PRINT_INTERVAL;
00318
00319  int i = 1;
00320  while (i < argc) {
00321   if (!strcmp(argv[i], "-n")) {
00322    i++;
00323    if (i < argc) {
00324     int temp = atol(argv[i]);
00325     if(temp < 1){
00326      fprintf(stderr, "Invalid number of droplets '%s'.\n",argv[i]);
00327      exit(EXIT_FAILURE);
00328     }
00329     numDroplets = temp;
00330     i++;
00331    } else {
00332     fprintf(stderr, "Invalid droplets value\n");
00333     exit(EXIT_FAILURE);
00334    }
00335   } else if (!strcmp(argv[i], "-f")) {
00336    i++;
00337    if (i < argc) {
00338 #ifdef _WIN32
00339     int result = fopen_s(&outFile,argv[i], "w");
00340     if (result != 0) {
00341      perror("Error: Cannot open specified file");
00342      exit(EXIT_FAILURE);
00343     }
00344 #else
00345     FILE *result = fopen(argv[i],"w");
00346     if (result == 0)
00347     {
00348      perror("Error: Cannot open specified file");
00349      exit(EXIT_FAILURE);
00350     }
00351     outFile = result;
00352 #endif
00353     i++;
00354    } else {
00355     fprintf(stderr, "Invalid filename\n");
00356     exit(EXIT_FAILURE);
00357    }
00358   } else if (!strcmp(argv[i], "-t")) {
00359    i++;
00360    if (i < argc) {
00361     int temp = atol(argv[i]);
00362     if(temp < 1){
00363      fprintf(stderr, "Invalid repetitions '%s'.\n",argv[i]);
00364      exit(EXIT_FAILURE);
00365     }
00366     numSteps = temp;
00367     i++;
00368    } else {
00369     fprintf(stderr, "Invalid repetitions\n");
00370     exit(EXIT_FAILURE);
00371    }
00372   }
00373   else if (!strcmp(argv[i], "-p")) {
00374    i++;
00375    if (i < argc) {
00376 #ifdef _WIN32
00377     // Windows returns a double for atof instead of a float
00378     float temp = static_cast<float>(atof(argv[i]));
00379 #else
00380     float temp = atof(argv[i]);
00381 #endif
00382     if (temp < 0) {
00383      fprintf(stderr, "Invalid print interval '%s'.\n",argv[i]);
00384      exit(EXIT_FAILURE);
00385     }
00386     printInterval = temp;
00387     i++;
00388    }
00389    else {
00390     fprintf(stderr, "Invalid print interval\n");
00391     exit(EXIT_FAILURE);
00392    }
00393   }
```

```
00394   else if (!strcmp(argv[i], "-?")) {
00395    fprintf(stdout,"usage: HelloDroplets -? | -n <number of droplets> -f <output filename> -t <number of steps> -p <logging interval>\n");
00396    exit(EXIT_SUCCESS);
00397   }else {
00398    fprintf(stdout, "Error: invalid option '%s'.\n",argv[i]);
00399    fprintf(stdout,"usage: HelloDroplets -? | -n <number of droplets> -f <output filename> -t <number of steps> -p <logging interval>\n");
00400    exit(EXIT_SUCCESS);
00401   }
00402
00403  }
00404
00405  if (outFile == NULL)
00406  {
00407 #ifdef _WIN32
00408   int result = fopen_s(&outFile,"output.txt", "w");
00409   if (result != 0) {
00410    perror("Error: Cannot open default output file for writing");
00411    exit(EXIT_FAILURE);
00412   }
00413 #else
00414   FILE *result = fopen("output.txt","w");
00415   if (result == 0)
00416   {
00417    perror("Error: Cannot open default output file for writing");
00418    exit(EXIT_FAILURE);
00419   }
00420   outFile = result;
00421 #endif
00422
00423   fprintf(outFile, "N %d\n",numDroplets);
00424  }
00425
00426  initSimulator();
00427
00428  setupSimObjects();
00429
00430  //std::vector<unsigned char *> *dropletColors = new std::vector<unsigned char *>();
00431  std::vector<GPSInfo *> *dropletPos = new std::vector<GPSInfo *>;
00432  std::vector<DropletCommData *> *dropletComm = new std::vector<DropletCommData *>;
00433  std::vector<DropletActuatorData *> *dropletAct = new std::vector<DropletActuatorData *>();
00434  //std::vector<DropletCompData *> *dropletComp = new std::vector<DropletCompData *>();
00435
00436
00437  for(int i = 0; i < numDroplets; i++)
00438  {
00439   //unsigned char *tmp1 = (unsigned char *)malloc(sizeof(unsigned char) * 3);
00440   //dropletColors->push_back(tmp1);
00441   GPSInfo *tmp2 = (GPSInfo *)malloc(sizeof(GPSInfo));
00442   dropletPos->push_back(tmp2);
00443   DropletCommData *tmp3 = (DropletCommData *)malloc(sizeof(DropletCommData));
00444   dropletComm->push_back(tmp3);
00445   DropletActuatorData *tmp4 = (DropletActuatorData *)malloc(sizeof(DropletActuatorData));
00446   dropletAct->push_back(tmp4);
00447   /*DropletCompData *tmp5 = (DropletCompData *)malloc(sizeof(DropletCompData));
00448   dropletComp->push_back(tmp5);*/
00449  }
00450  double lastPrintTime, currentTime, realTime;
00451  lastPrintTime = -printInterval - 1; // makes it print at sim time 0
00452  sim.Step();
00453
00454  for(int i = 1; i <= numSteps; i++)
00455  {
00456   sim.Step();
00457   currentTime = simInfo.GetTotalST(sim);
00458   realTime = simInfo.GetTotalRT(sim);
00459   if ( (currentTime - lastPrintTime) > printInterval) {
00460    fprintf(outFile, "------ %.3f:SIM STEP %i ------\n", currentTime, i + 1);
00461    fprintf(outFile, "Real Time: %.3f\n", realTime);
00462    simInfo.GetDropletPositions(dropletPos,sim);
00463
00464    simInfo.GetCommData(dropletComm, sim);
00465    simInfo.GetActuationData(dropletAct, sim);
00466    //simInfo.GetCompData(dropletComp, sim);
00467    //PrintComputation(dropletComp, outFile);
00468    //PrintStatus(dropletColors,dropletPos, dropletComm, outFile);
00469    PrintWaitStatus(dropletPos, dropletComm, dropletAct, outFile);
00470
00471    fprintf(outFile,"\n");
00472    lastPrintTime = currentTime;
00473   }
00474  }
00475  fprintf(outFile, "------ SIMULATION ENDS ------\n");
00476  fclose(outFile);
00477  sim.Cleanup();
00478
```

```
00479  fprintf(stdout, "Simulation Complete\n");
00480
00481  //std::vector<unsigned char *>::reverse_iterator cit;
00482  //for(cit = dropletColors->rbegin(); cit != dropletColors->rend(); cit++)
00483  // free((unsigned char *)*cit);
00484
00485  std::vector<GPSInfo *>::reverse_iterator xyit;
00486  for(xyit = dropletPos->rbegin(); xyit != dropletPos->rend(); xyit++)
00487   free((GPSInfo *)*xyit);
00488
00489  std::vector<DropletCommData *>::reverse_iterator commit;
00490  for(commit = dropletComm->rbegin(); commit != dropletComm->rend(); commit++)
00491   free((DropletCommData *)*commit);
00492
00493  std::vector<DropletActuatorData *>::reverse_iterator act_rit;
00494  for(act_rit = dropletAct->rbegin(); act_rit != dropletAct->rend(); act_rit++)
00495   free((DropletActuatorData *)*act_rit);
00496
00497  /*std::vector<DropletCompData *>::reverse_iterator compit;
00498  for(compit = dropletComp->rbegin(); compit != dropletComp->rend(); compit++)
00499  free((DropletCompData *)*compit);*/
00500
00501  //delete dropletColors;
00502  //delete dropletPos;
00503  delete dropletComm;
00504  delete dropletAct;
00505  //delete dropletComp;
00506  return(EXIT_SUCCESS);
00507 }
```

8.7 main.dox File Reference

8.8 tutorial.dox File Reference

Index