G53GRA.Framework

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 Animation Class Reference

`#include <Animation.h>`

Public Member Functions

- virtual void Update (const double &deltaTime)=0

3.1.1 Detailed Description

Class to be subclassed alongside DisplayableObject for all animated objects to be displayed in a Scene

Contains Update method that must be overloaded. Update(float deltaTime) is called from Scene.

Author

wil

3.1.2 Member Function Documentation

3.1.2.1 virtual void Animation::Update (const double & *deltaTime*) [pure virtual]

Called each frame to update. Must be defined!

Use this to update animation sequence.

Parameters

deltaTime	change in time since previous call

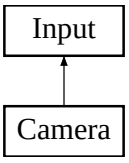
The documentation for this class was generated from the following file:

- Framework/Interface/Animation.h

3.2 Camera Class Reference

`#include <Camera.h>`

Inheritance diagram for Camera:



Public Member Functions

- Camera ()
- void **GetEyePosition** (float &x, float &y, float &z) const
- void **GetViewDirection** (float &x, float &y, float &z) const
- void **GetForwardVector** (float &x, float &y, float &z) const
- void **GetRightVector** (float &x, float &y, float &z) const
- void **GetUpVector** (float &x, float &y, float &z) const
- virtual void Update (const double &deltaTime)
- virtual void Reset ()
- virtual void SetViewport ()
- void HandleKey (unsigned char key, int state, int x, int y)
- void HandleSpecialKey (int key, int state, int x, int y)
- void HandleMouse (int button, int state, int x, int y)
- void HandleMouseDrag (int x, int y)
- void HandleMouseMove (int x, int y)
- virtual void SetupCamera ()

3.2.1 Detailed Description

This class implements the base Camera functionality. It controls the position and view direction of the camera in your Scene. You may add functionality by creating a new class that inherits `Camera`, e.g.

```
class MyCamera : public Camera
```

. You should avoid editting this class directly.

Note that

`Camera`

extends the virtual class Input, so will be passed key and mouse input by the window

Author

wil

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Camera::Camera ( )

Constructor for Camera to set up viewing properties in rendering window

3.2.3 Member Function Documentation

3.2.3.1 void Camera::HandleKey (unsigned char *key*, int *state*, int *x*, int *y*) [virtual]

Captures input from

`wasd`

-keys used for camera movement.

Spacebar #reset() reset}s the camera.

Reimplemented from Input.

3.2.3.2 void Camera::HandleMouse (int *button*, int *state*, int *x*, int *y*) [virtual]

Captures button click. Sets button to 0 if last mouse button is released. Saves current position of mouse at click.

See also

HandleMouseDrag(int, int)

Reimplemented from Input.

3.2.3.3 void Camera::HandleMouseDrag (int *x*, int *y*) [virtual]

Called when mouse is moved (while a button is pressed down). Functionality currently only for **LEFT** mouse click.

Calculates difference in mouse position since last call and adjusts camera view accordingly. Sensitivity is fixed at 0.01f.

See also

HandleMouse(int, int, int, int)

Reimplemented from Input.

3.2.3.4 void Camera::HandleMouseMove (int *x*, int *y*) [inline], [virtual]

Called when mouse is moved in rendering window when no mouse button is pressed

See also

HandleMouse(int button, int state, int x, int y)
HandleMouseDrag(int x, int y)

Parameters

x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

Reimplemented from Input.

3.2.3.5 void Camera::HandleSpecialKey (int *key*, int *state*, int *x*, int *y*) [inline], [virtual]

Called when keyboard input is received from special (non-ASCII) characters.

key constants are named GLUT_KEY_∗ where ∗ is the key. For example:

(arrow keys) GLUT_KEY_UP, GLUT_KEY_DOWN, GLUT_KEY_LEFT, GLUT_KEY_RIGHT,

GLUT_KEY_PAGE_UP, GLUT_KEY_PAGE_DOWN, GLUT_KEY_HOME, GLUT_KEY_END,

GLUT_KEY_F1, GLUT_KEY_F2, etc.

Example implementation:

```
void MyObject::HandleKey(unsigned int key, int state, int x, int y){
    if (state == 1){ // if key pressed down
      switch(key){ // special key
          case GLUT_KEY_LEFT:
                glTranslate(-1.f,0.f,0.f); // go left
              break;
          case GLUT_KEY_RIGHT:
                glTranslate(1.f,0.f,0.f); // go right
              break;
      }
   }
}
```

See also

#HandleKey(char key, int state, int x, int y)

Parameters

key	coded keyboard input
state	1 if key down, 0 if key up
x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

Reimplemented from Input.

3.2.3.6 void Camera::Reset ( ) [virtual]

Resets Camera vectors to default values. Sets position of camera at (0,0) in x,y-plane and puts z-position at

`0.5*height/tan(pi/6)`

which puts the coordinate width and height of window into view (if projection is in perspective view, and both

`fovy = 60`

° and

`aspect = width/height`

)

`width`

and

`height`

refer to the window size of

`Scene`

parent.

3.2.3.7 void Camera::SetupCamera ( ) [virtual]

Called by Scene to position camera.

Sets up position (eye), look at (

`cen=`

eye

`+`

view) and up vector (up) and updates Scene viewing.

See also

#Update(float)
Reset()

3.2.3.8 void Camera::SetViewport ( ) [virtual]

Sets the window viewport of the scene

3.2.3.9 void Camera::Update (const double & *deltaTime*) [virtual]

Update the position of the camera and look-at vectors based on keyboard input.

Parameters

deltaTime	change in time since previous call (unused)

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

- Framework/Utility/Camera.h
- Framework/Utility/Camera.cpp

3.3 DisplayableObject Class Reference

`#include <DisplayableObject.h>`

Public Member Functions

- DisplayableObject ()
- virtual void Display ()=0
- void position (float x, float y, float z)
- void size (float s)
- void size (float sx, float sy, float sz)
- void orientation (float rx, float ry, float rz)
- float * size ()
- float * orientation ()
- float * position ()

Protected Attributes

- float pos [3]
- float scale [3]
- float rotation [3]

3.3.1 Detailed Description

Virtual class to be inherited by all objects to be displayed in Scene

Contains purely virtual Display method that must be overloaded. Display() is called from Scene.

Author

wil

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DisplayableObject::DisplayableObject ( ) [inline]

Default constructor

Initialises position, size and orientation to origin in World Space.

3.3.3 Member Function Documentation

3.3.3.1 virtual void DisplayableObject::Display ( ) [pure virtual]

Virtual method. Called from Scene parent.

Must be overloaded by your DisplayableObject subclass. Contains all rendering commands.

3.3.3.2 void DisplayableObject::orientation (float *rx*, float *ry*, float *rz*) [inline]

set orientation in World Space

3.3.3.3 float* DisplayableObject::orientation ( ) [inline]

Get orientation in World Space

3.3.3.4 void DisplayableObject::position (float *x*, float *y*, float *z*) [inline]

set World Space position

3.3.3.5 float* DisplayableObject::position ( ) [inline]

Get World Space position

3.3.3.6 void DisplayableObject::size (float *s*) [inline]

set size in World Space (single value)

3.3.3.7 void DisplayableObject::size (float *sx*, float *sy*, float *sz*) [inline]

set size in World Space (seperate axes)

3.3.3.8 float* DisplayableObject::size ( ) [inline]

Get size in World Space

3.3.4 Member Data Documentation

3.3.4.1 float DisplayableObject::pos[3] [protected]

float[] containing World Space position coordinates

3.3.4.2 float DisplayableObject::rotation[3] [protected]

float[] containing angles of orientation in World Space for x, y, and z axes

3.3.4.3 float DisplayableObject::scale[3] [protected]

float[] containing relative World Space scaling values for x,y,z

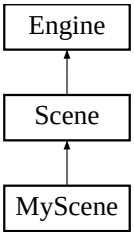
The documentation for this class was generated from the following file:

- Framework/Interface/DisplayableObject.h

3.4 Engine Class Reference

`#include <Engine.h>`

Inheritance diagram for Engine:



Public Member Functions

- Engine (int argc, char **argv, const char *title, const int &windowWidth, const int &windowHeight)
- virtual void Run ()

Static Public Member Functions

- static Engine * GetCurrentInstance ()
- static const char * GetWindowTitle ()
- static int GetWindowID ()
- static int GetWindowWidth ()
- static int GetWindowHeight ()

Protected Member Functions

- virtual void **Initialise** ()=0
- virtual void **Draw** ()=0
- virtual void **Reshape** (int w, int h)=0
- virtual void **Update** (const double &)=0
- virtual void **HandleKey** (unsigned char key, int state, int x, int y)=0
- virtual void **HandleSpecialKey** (int key, int state, int x, int y)=0
- virtual void **HandleMouse** (int button, int state, int x, int y)=0
- virtual void **HandleMouseDrag** (int x, int y)=0
- virtual void **HandleMouseMove** (int x, int y)=0

Static Protected Member Functions

- static void InitFunc ()
- static void DrawFunc ()
- static void IdleFunc ()
- static void ResizeFunc (int _width, int _height)
- static void KeyDownFunc (unsigned char key, int x, int y)
- static void KeyUpFunc (unsigned char key, int x, int y)
- static void SpecialKeyDownFunc (int key, int x, int y)
- static void SpecialKeyUpFunc (int key, int x, int y)
- static void MouseFunc (int button, int state, int x, int y)
- static void MouseMotionFunc (int x, int y)
- static void PassiveMouseMotionFunc (int x, int y)
- static int CheckGLError ()

Static Protected Attributes

- static Engine * current = 0
- static const char * **windowTitle** = ""
- static int **windowID** = 0
- static int **windowWidth** = 0
- static int **windowHeight** = 0
- static int **time** = 0

3.4.1 Detailed Description

Base Engine for the framework. Handles windowing and freeglut/OpenGL contexts. Engine is static.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Engine::Engine (int *argc,* char ** *argv,* const char * *title,* const int & *windowWidth,* const int & *windowHeight*)

Constructor takes in command line arguments, a title and initial window dimensions.

3.4.3 Member Function Documentation

3.4.3.1 int Engine::CheckGLError ( ) [static], [protected]

Iterates through OpenGL error list and dumps error information to console.

Call this method in a draw loop.

3.4.3.2 void Engine::DrawFunc ( ) [static], [protected]

Calls Draw and handles double buffering

3.4.3.3 Engine * Engine::GetCurrentInstance ( ) [static]

Returns self.

3.4.3.4 int Engine::GetWindowHeight ( ) [static]

Returns window height

3.4.3.5 int Engine::GetWindowID ( ) [static]

Returns window id

3.4.3.6 const char * Engine::GetWindowTitle ( ) [static]

Returns window title

3.4.3.7 int Engine::GetWindowWidth ( ) [static]

Returns window width

3.4.3.8 void Engine::IdleFunc ( ) [static], [protected]

Checks runtime between successive calls and runs Update before pushing redisplay

3.4.3.9 void Engine::InitFunc ( ) [static], [protected]

Sets up default initial functionality for window

3.4.3.10 void Engine::KeyDownFunc (unsigned char *key,* int *x,* int *y*) [static], [protected]

Handles key press

3.4.3.11 void Engine::KeyUpFunc (unsigned char *key,* int *x,* int *y*) [static], [protected]

Handles key release

3.4.3.12 void Engine::MouseFunc (int *button,* int *state,* int *x,* int *y*) [static], [protected]

Handles mouse button click

3.4.3.13 void Engine::MouseMotionFunc (int *x,* int *y*) [static], [protected]

Handles mouse click and drag (active motion)

3.4.3.14 void Engine::PassiveMouseMotionFunc (int *x,* int *y*) [static], [protected]

Handles mouse movement (passive motion)

3.4.3.15 void Engine::ResizeFunc (int *_width,* int *_height*) [static], [protected]

Handles window resize

3.4.3.16 void Engine::Run ( ) [virtual]

Initial startup method. Sets up GL context and windowing functions to handle drawing, timing and input.

3.4.3.17 void Engine::SpecialKeyDownFunc (int *key,* int *x,* int *y*) [static], [protected]

Handles special key press

3.4.3.18 void Engine::SpecialKeyUpFunc (int *key,* int *x,* int *y*) [static], [protected]

Handles special key release

3.4.4 Member Data Documentation

3.4.4.1 Engine * Engine::current = 0 [static], [protected]

pointer to current window context

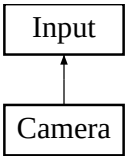
The documentation for this class was generated from the following files:

- Framework/Engine/Engine.h
- Framework/Engine/Engine.cpp

3.5 Input Class Reference

`#include <Input.h>`

Inheritance diagram for Input:



Public Member Functions

- virtual void HandleKey (unsigned char key, int state, int x, int y)
- virtual void HandleSpecialKey (int key, int state, int x, int y)
- virtual void HandleMouse (int button, int state, int x, int y)
- virtual void HandleMouseDrag (int x, int y)
- virtual void HandleMouseMove (int x, int y)

3.5.1 Detailed Description

Class for giving an coursework object input handling. Any class you want to have input handling should subclass the

```
public Input
```

. e.g.

```
MyObject : public DisplayableObject, public Input
```

Methods will be called from a Scene renderer to give input from keyboard and mouse

See also

Camera

Author

wil

3.5.2 Member Function Documentation

3.5.2.1 virtual void Input::HandleKey (unsigned char *key*, int *state*, int *x*, int *y*) [inline], [virtual]

Called when keyboard input is received from ASCII characters.

This includes keys ENTER (\n), RETURN (\cr), TAB (\t), BACKSPACE (\b), ESC (27), DELETE (127)

Note: check for RETURN and ENTER for cross-platform operability.

Example implementation:

```
  void MyObject:HandleKey(unsigned char key, int state, int mX, int mY){
      if (state == 1){ // if key pressed down
        switch(key){ // ASCII char
            case 'A':
              case 'a':
                  glTranslatef(-1.f,0.f,0.f); // go left
                break;
            case 'D':
              case 'd':
                  glTranslate(1.f,0.f,0.f); // go right
                break;
              case '\n':  // Windows and Linux
              case '\cr': // Mac OS X
                  // operation for using enter/return key
                break;
        }
    }
}
```

See also

HandleSpecialKey(int key, int state, int x, int y)

Parameters

key	ASCII character from keyboard input
state	1 if key down, 0 if key up
x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

Reimplemented in Camera.

3.5.2.2 virtual void Input::HandleMouse (int *button,* int *state,* int *x,* int *y*) [inline], [virtual]

Called when mouse is clicked up / down in the rendering window

button constants: GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON and GLUT_MIDDLE_BUTTON

See also

HandleMouseDrag(int x, int y)
HandleMouseMove(int x, int y)

Parameters

button	mouse button (GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON or GLUT_MIDDLE_BUTTON)
state	1 if mouse down, 0 if mouse up
x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

Reimplemented in Camera.

3.5.2.3 virtual void Input::HandleMouseDrag (int *x,* int *y*) [inline], [virtual]

Called when mouse is moved in rendering window while mouse button is held down

See also

HandleMouse(int button, int state, int x, int y)
HandleMouseMove(int x, int y)

Parameters

x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

Reimplemented in Camera.

3.5.2.4 virtual void Input::HandleMouseMove (int *x*, int *y*) [inline], [virtual]

Called when mouse is moved in rendering window when no mouse button is pressed

See also

HandleMouse(int button, int state, int x, int y)
HandleMouseDrag(int x, int y)

Parameters

x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

Reimplemented in Camera.

3.5.2.5 virtual void Input::HandleSpecialKey (int *key*, int *state*, int *x*, int *y*) [inline], [virtual]

Called when keyboard input is received from special (non-ASCII) characters.

key constants are named GLUT_KEY_$*$ where $*$ is the key. For example:

(arrow keys) GLUT_KEY_UP, GLUT_KEY_DOWN, GLUT_KEY_LEFT, GLUT_KEY_RIGHT,

GLUT_KEY_PAGE_UP, GLUT_KEY_PAGE_DOWN, GLUT_KEY_HOME, GLUT_KEY_END,

GLUT_KEY_F1, GLUT_KEY_F2, etc.

Example implementation:

```
void MyObject::HandleKey(unsigned int key, int state, int x, int y){
    if (state == 1){ // if key pressed down
      switch(key){ // special key
          case GLUT_KEY_LEFT:
                glTranslate(-1.f,0.f,0.f); // go left
              break;
          case GLUT_KEY_RIGHT:
                glTranslate(1.f,0.f,0.f); // go right
              break;
      }
   }
}
```

See also

#HandleKey(char key, int state, int x, int y)

Parameters

key	coded keyboard input
state	1 if key down, 0 if key up
x	X coordinate of mouse in rendering window
y	Y coordinate of mouse in rendering window

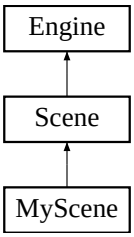
Reimplemented in Camera.

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

- Framework/Interface/Input.h

3.6 MyScene Class Reference

Inheritance diagram for MyScene:



Public Member Functions

- **MyScene** (int argc, char ∗∗argv, const char ∗title, const int &windowWidth, const int &windowHeight)

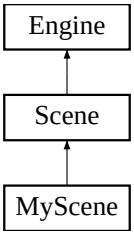
Additional Inherited Members

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

- Code/MyScene.h
- Code/MyScene.cpp

3.7 Scene Class Reference

Inheritance diagram for Scene:



Public Member Functions

- Scene (int argc, char ∗∗argv, const char ∗title, const int &windowWidth, const int &windowHeight)
- virtual ∼Scene ()

Static Public Member Functions

- static int GetWindowWidth ()
- static int GetWindowHeight ()
- static int GetTexture (std::string fileName)
- static Camera * GetCamera ()

Protected Member Functions

- virtual void Initialise ()=0
- void Draw ()
- void Reshape (int w, int h)
- virtual void Projection ()
- void Update (const double &deltaTime)
- void **HandleKey** (unsigned char key, int state, int x, int y)
- void **HandleSpecialKey** (int key, int state, int x, int y)
- void **HandleMouse** (int button, int state, int x, int y)
- void **HandleMouseDrag** (int x, int y)
- void **HandleMouseMove** (int x, int y)
- void AddObjectToScene (DisplayableObject *obj)

Additional Inherited Members

3.7.1 Constructor & Destructor Documentation

3.7.1.1 Scene::Scene (int *argc,* char ** *argv,* const char * *title,* const int & *windowWidth,* const int & *windowHeight*)

Constructor, overrides Engine() and takes in command line arguments, a title and initial window dimensions.

3.7.1.2 Scene::~Scene ( ) [virtual]

Destructor: deletes all DisplayableObjects in the Scene

3.7.2 Member Function Documentation

3.7.2.1 void Scene::AddObjectToScene (DisplayableObject * *obj*) [protected]

Adds a DisplayableObject (includes Animations) to the Scene.

It is strongly recommended you do NOT attempt to override this method.

See also

#addObjectToScene(DisplayableObject, String)
#getObject(String)

Parameters

obj	DisplayableObject to be added to the scene.

3.7.2.2 void Scene::Draw ( ) [protected], [virtual]

This function will loop continuously until the program is exited. It will iterate through the objects and render in your Scene

The frequency at which

`Draw()`

is called per second is dependent on your hardware and scene, so for animation, you MUST use the Update method with

`deltaTime`

It is strongly recommended you do NOT attempt to override this method.

`Draw()`

should NEVER be called explicitly.

See also

Reshape()

Implements Engine.

3.7.2.3 static Camera* Scene::GetCamera ( ) [inline], [static]

Returns a pointer to the Camera

3.7.2.4 int Scene::GetTexture (std::string *fileName*) [static]

Input a .bmp bitmap image to bind to internal texture buffer

Returns

textureID fileName the name of the texture file you want to input

3.7.2.5 int Scene::GetWindowHeight ( ) [static]

Return window height

3.7.2.6 int Scene::GetWindowWidth ( ) [static]

Return window width

3.7.2.7 void Scene::Initialise ( ) [protected], [pure virtual]

This must be overloaded this to add DisplayableObjects to your scene.

See also

Projection()

Implements Engine.

3.7.2.8 void Scene::Projection ( ) [protected], [virtual]

Sets the projection properties of the scene. Override to change projection properties.

Default: Orthographic mode

3.7.2.9 void Scene::Reshape (int *w*, int *h*) [protected], [virtual]

Called when the window is resized, and handles resizing events.

You should this function to handle the window being resized. The default property is to update the projection parameters. You can access the window size parameters by the

`w`

and

`h`

variables. For example, you could overload this function to adjust the size of all objects in your

`Scene`

.

See also

Draw()
Projection()
Initialise()

Implements Engine.

3.7.2.10 void Scene::Update (const double & *deltaTime*) [protected], [virtual]

The update function for Camera and Animation. Calculates the time-delay since the last update and passes as a parameter to the respective class's

`Update()`

functions.

You should only override this class if you want to change how the animation update function works. This is not advised.

See also

Camera
Animation
Draw()

Implements Engine.

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

- Framework/Engine/Scene.h
- Framework/Engine/Scene.cpp

3.8 tagBITMAPFILEHEADER Struct Reference

Public Attributes

- WORD **bfType**
- DWORD **bfSize**
- WORD **bfReserved1**
- WORD **bfReserved2**
- DWORD **bfOffBits**

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

- Framework/Utility/Texture.cpp

3.9 tagBITMAPINFOHEADER Struct Reference

Public Attributes

- DWORD **biSize**
- LONG **biWidth**
- LONG **biHeight**
- WORD **biPlanes**
- WORD **biBitCount**
- DWORD **biCompression**
- DWORD **biSizeImage**
- LONG **biXPelsPerMeter**
- LONG **biYPelsPerMeter**
- DWORD **biClrUsed**
- DWORD **biClrImportant**

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

- Framework/Utility/Texture.cpp

3.10 Texture Class Reference

`#include <Texture.h>`

Public Member Functions

- int GetTexture (std::string fileName)

3.10.1 Detailed Description

Class for loading bitmap files into texture buffer and handling texture IDs

3.10.2 Member Function Documentation

3.10.2.1 int Texture::GetTexture (std::string *fileName*)

Loads a texture into memory and returns the id of the texture object created

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

- Framework/Utility/Texture.h
- Framework/Utility/Texture.cpp