Platform Perils

MIS/MID

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:

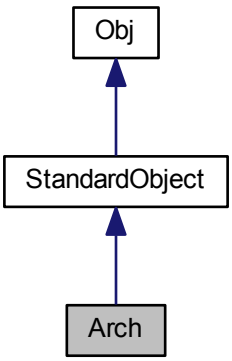
Chapter 3

Class Documentation

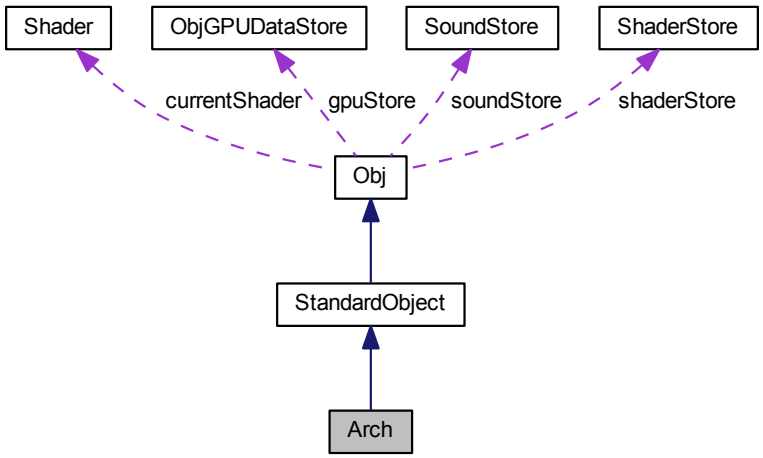
3.1 Arch Class Reference

```
#include <Arch.h>
```

Inheritance diagram for Arch:



Collaboration diagram for Arch:



Public Member Functions

- Arch (float x, float y, bool faceRight=false)

  Arch constructor.

Additional Inherited Members

3.1.1 Detailed Description

The Arch class implements the arch used to signify the stage goal.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Arch::Arch (float *x*, float *y*, bool *faceRight* = `false`)

Arch constructor.

Parameters

x	The x coordinate of the arch.
y	The y coordinate of the arch.
faceRight	True if arch should face to the right (default false, i.e. facing left).

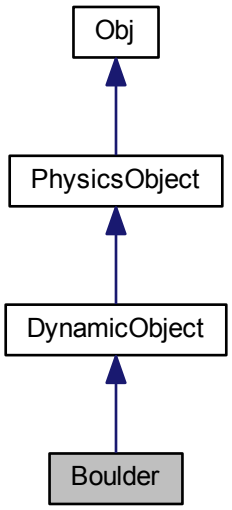
The documentation for this class was generated from the following file:

- Arch.h

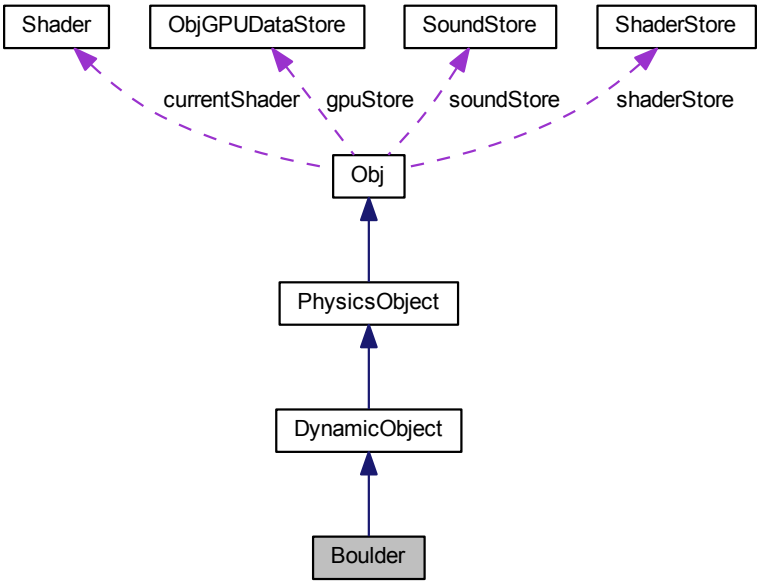
3.2 Boulder Class Reference

`#include <Boulder.h>`

Inheritance diagram for Boulder:



Collaboration diagram for Boulder:



Public Member Functions

- Boulder (float x, float y)
  Boulder constructor.

Additional Inherited Members

3.2.1 Detailed Description

The Boulder class implements the boulder hazard.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Boulder::Boulder (float *x*, float *y*)

Boulder constructor.

Parameters

x	The x coordinate of the boulder.
y	The y coordinate of the boulder.

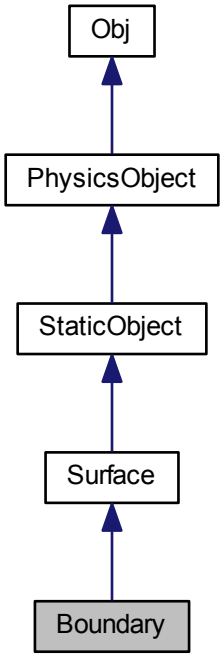
The documentation for this class was generated from the following file:

- Boulder.h

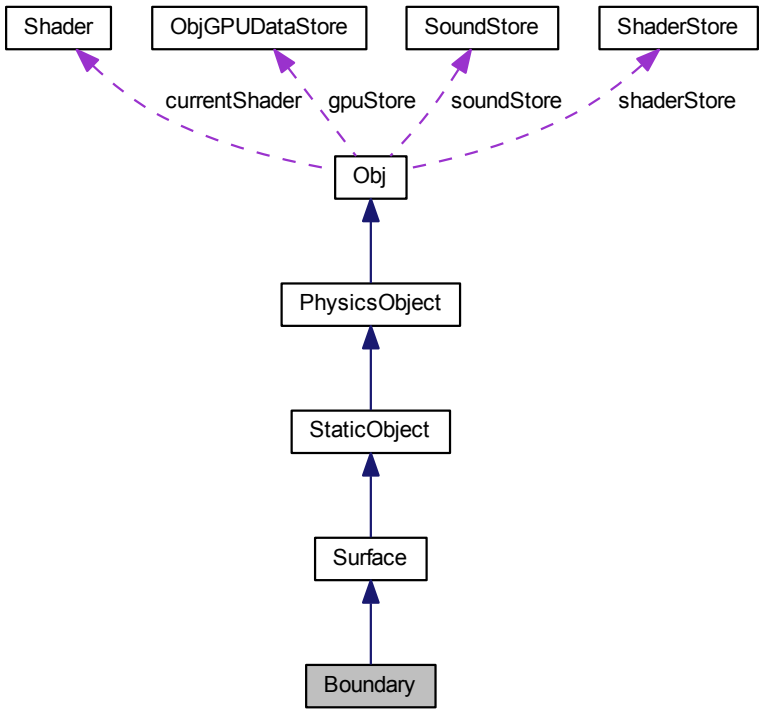
3.3 Boundary Class Reference

```
#include <Boundary.h>
```

Inheritance diagram for Boundary:



Collaboration diagram for Boundary:



Public Member Functions

- Boundary (float x1, float x2, float y, BSurface surfaceType)

  Boulder constructor.

Additional Inherited Members

3.3.1 Detailed Description

The Boundary class implements the boundary that is used to create a surface which extends into the distance (i.e. creates the scene).

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Boundary::Boundary (float *x1*, float *x2*, float *y*, BSurface *surfaceType*)

Boulder constructor.

Parameters

x1	The leftmost extent of the boundary.
x2	The rightmost extent of the boundary.
y	The y-coordinate of the boundary.
surfaceType	The type of surface.

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

- Boundary.h

3.4 Camera Class Reference

`#include <Camera.h>`

Public Member Functions

- Camera ()

  Camera constructor.
- void zoomIn ()

  Zoom the view in.
- void zoomOut ()

  Zoom the view out.

Public Attributes

- glm::vec3 up

  The up vector.
- float zoom

  Current zoom level.

3.4.1 Detailed Description

The Camera class controls the view of the stage.

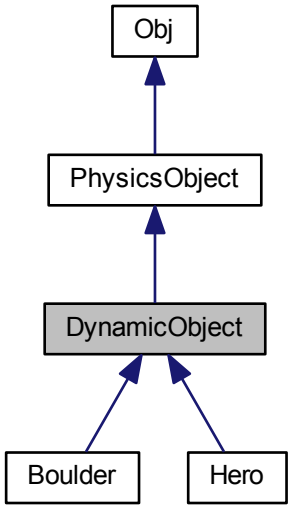
The documentation for this class was generated from the following file:

- Camera.h

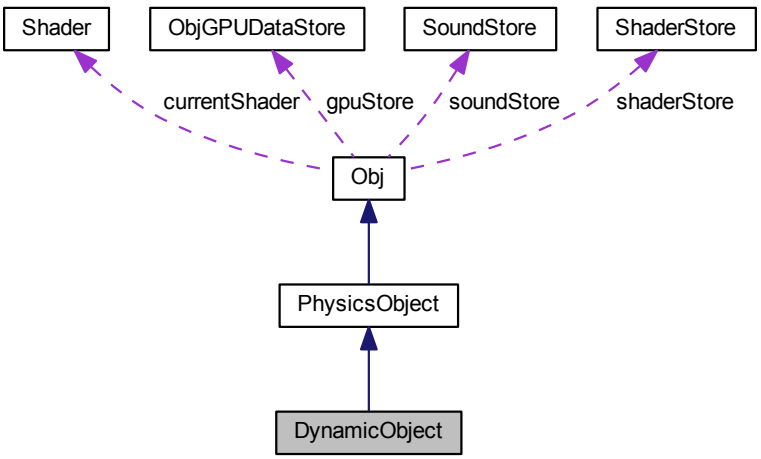
3.5 DynamicObject Class Reference

`#include <DynamicObject.h>`

Inheritance diagram for DynamicObject:



Collaboration diagram for DynamicObject:



Public Member Functions

- DynamicObject ()

  DynamicObject default constructor.
- DynamicObject (float x, float y, float scale, float mass, float elast, float fric, int type, std::string gpuPath, std::string vPath, std::string fPath)

  DynamicObject constructor.

Additional Inherited Members

3.5.1 Detailed Description

The DynamicObject class is derived from the PhysicsObject class. This type of object is subject to all physics calculations.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 DynamicObject::DynamicObject (float *x,* float *y,* float *scale,* float *mass,* float *elast,* float *fric,* int *type,* std::string *gpuPath,* std::string *vPath,* std::string *fPath*)

DynamicObject constructor.

Parameters

x	The x-coordinate of the dynamic object
y	The y-coordinate of the dynamic object
scale	Scalar factor by which to scale the object during GPU rendering
mass	Mass of the object
elast	Elasticity of the object
fric	Friction factor of the object
type	Type of object (different values affect collision routines)
gpuPath	The path to the obj file for the object
vPath	The path to the vertex shader for the object
fPath	The path to the fragment shader for the object

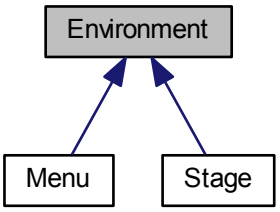
The documentation for this class was generated from the following file:

- DynamicObject.h

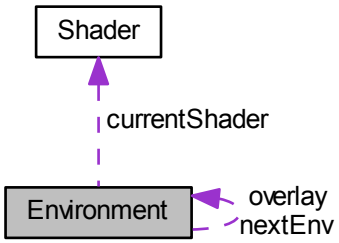
3.6 Environment Class Reference

```
#include <Environment.h>
```

Inheritance diagram for Environment:



Collaboration diagram for Environment:



Public Member Functions

- virtual ~Environment ()

  Virtual destructor.
- virtual void updateEnvironment (double dt)=0

  Pure virtual function (i.e. defined by derived classes) for updating the environment.
- virtual void drawEnvironment ()=0

  Pure virtual function (i.e. defined by derived classes) for drawing the environment.
- void changeShader (Shader *nextShader)

  Changes the shader program.
- virtual void processKB (int key, int scancode, int action, int mods)=0

  Pure virtual function (i.e. defined by derived classes) for keyboard input processing.
- virtual void processContinuousInput ()=0

  Pure virtual function (i.e. defined by derived classes) for continuous input processing.
- virtual bool processMousePosition (float xpos, float ypos)=0

  Pure virtual function (i.e. defined by derived classes) for mouse position change processing.
- virtual void processMouseClick (int button, int action, int mods)=0

  Pure virtual function (i.e. defined by derived classes) for mouse input processing.
- virtual void updateScreenSize ()=0

  Pure virtual function (i.e. defined by derived classes) for updating screen size.

Public Attributes

- int keyStates [GLFW_KEY_LAST] = {0}

  Array that keeps track of keyboard keys currently pressed down.
- Environment * overlay

  Environment to overlay the current environment (in-game menu).
- Environment * nextEnv

  Next environment (if non-null takes effect on next frame).

Static Public Attributes

- static float screenWidth

  The current width of the screen in pixels.
- static float screenHeight

  The current height of the screen in pixels.

Protected Attributes

- std::map< std::string, ObjGPUData * > gpuMap

  Stored object data used by the GPU (i.e. meshes/texture mappings/etc.).
- std::map< std::string, Sound * > soundMap

  Stored sound data.
- std::map< std::string, Shader * > shaderMap

  Stored shaders.
- glm::mat4 mat_Projection

  Projection matrix.
- glm::mat4 mat_View

  View matrix.
- Shader * currentShader

  Pointer to the shader that is currently bound.
- float mouseX

  Current position of the mouse x-coordinate.
- float mouseY

  Current position of the mouse y-coordinate.

3.6.1 Detailed Description

The Environment class is an abstract class that holds information about the current game state and provides function calls to update and draw the game to the screen. The Stage and Menu classes are derived from this class.

3.6.2 Member Function Documentation

3.6.2.1 void Environment::changeShader (Shader * *nextShader*)

Changes the shader program.

Parameters

nextShader	New shader to be bound.

3.6.2.2 virtual void Environment::processKB (int *key*, int *scancode*, int *action*, int *mods*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for keyboard input processing.

Parameters

key	Key to which the action corresponds.
scancode	System specific key code.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implemented in Menu, and Stage.

3.6.2.3 virtual void Environment::processMouseClick (int *button*, int *action*, int *mods*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for mouse input processing.

Parameters

button	Mouse button to which the action corresponds.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implemented in Menu, and Stage.

3.6.2.4 virtual bool Environment::processMousePosition (float *xpos*, float *ypos*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for mouse position change processing.

Parameters

xpos	Mouse cursor x-position
ypos	Mouse cursor y-position

Implemented in Menu, and Stage.

3.6.2.5 virtual void Environment::updateEnvironment (double *dt*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for updating the environment.

Parameters

dt	Time step to be used for the update (time since last update) in milliseconds.

Implemented in Menu, and Stage.

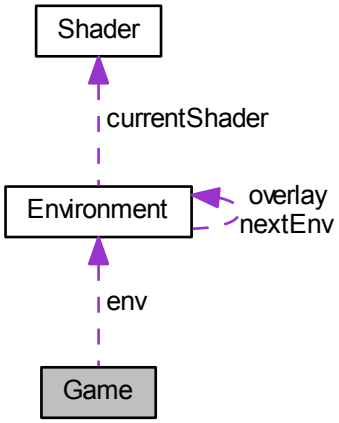
The documentation for this class was generated from the following file:

- Environment.h

3.7 Game Class Reference

`#include <Game.h>`

Collaboration diagram for Game:



Public Member Functions

- Game ()
  Game class constructor.
- ~Game ()
  Game class destructor.
- void run ()
  Runs the game until the application is terminated (infinite loop)
- void framebuffer_size_callback (GLFWwindow *, int, int)
  GLFW window resize callback.
- void key_callback (GLFWwindow *, int, int, int, int)
  GLFW keyboard input callback.
- void mouse_pos_callback (GLFWwindow *, float, float)
  GLFW mouse position change callback.
- void mouse_button_callback (GLFWwindow *, int, int, int)
  GLFW mouse button input callback.

Private Attributes

- GLFWwindow * window

  Reference to the GLFWwindow (the window)

- Environment * env

  Reference to the current Environment.

- double timeLast

  Last time that was polled; used for framerate control.

- double timeElapsed

  Time elapsed since last polling of time; used for framerate control.

- float winX

  Stores x-coordinate maximum of the window.

- float winY

  Stores y-coordinate maximum of the window.

3.7.1 Detailed Description

The Game class is a representation of the game on the highest level. It handles all exchanges between the user and the game code. It keeps a reference to the game window as well as the current environment of the game (main menu, stage, etc) and acts as a bridge between the two. User inputs are intercepted through GLFW callbacks in this class and passed on for processing by the current game environment. This class is also responsible for "running" the game and sends requests for the game environment to be updated and drawn to the screen at regular intervals (framerate is controlled).

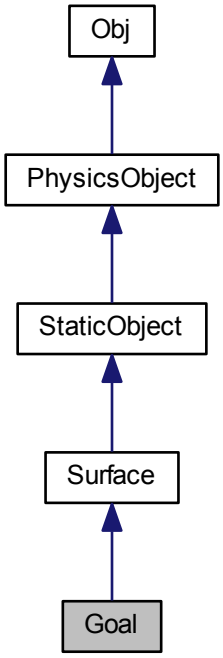
The documentation for this class was generated from the following file:

- Game.h

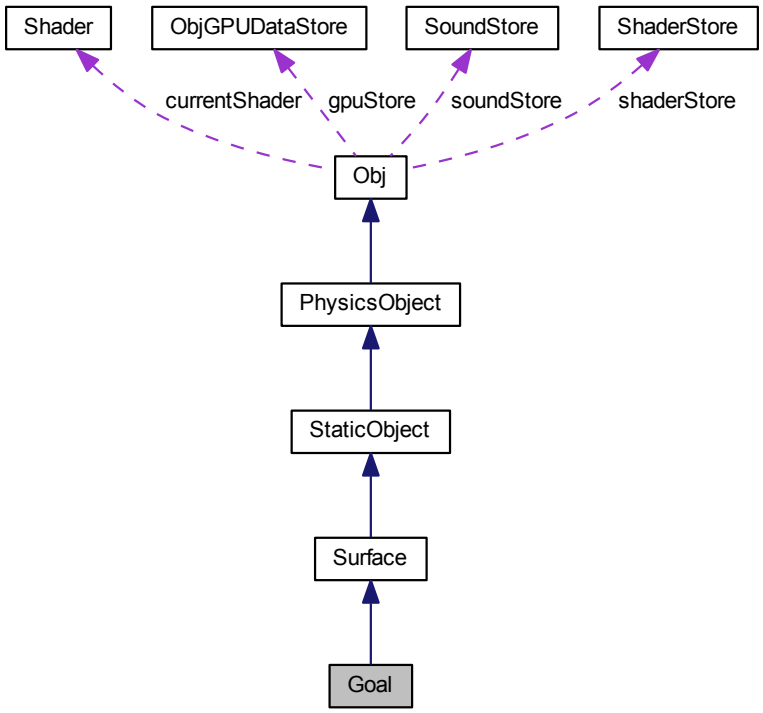
3.8 Goal Class Reference

```
#include <Goal.h>
```

Inheritance diagram for Goal:



Collaboration diagram for Goal:



Public Member Functions

- Goal (float x1, float x2, float ymid, float thickness=50.0f)
  Goal constructor.

Additional Inherited Members

3.8.1 Detailed Description

The Goal class implements the checkered goal platform.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Goal::Goal (float *x1*, float *x2*, float *ymid*, float *thickness* = `50.0f`)

Goal constructor.

Parameters

x1	The leftmost extent of the goal.
x2	The rightmost extent of the goal.
ymid	The y-coordinate of the goal.
thickness	The vertical thickness of the goal (default = 50.0).

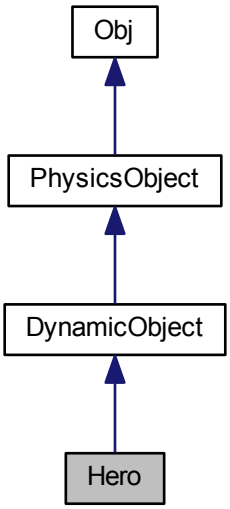
The documentation for this class was generated from the following file:

- Goal.h

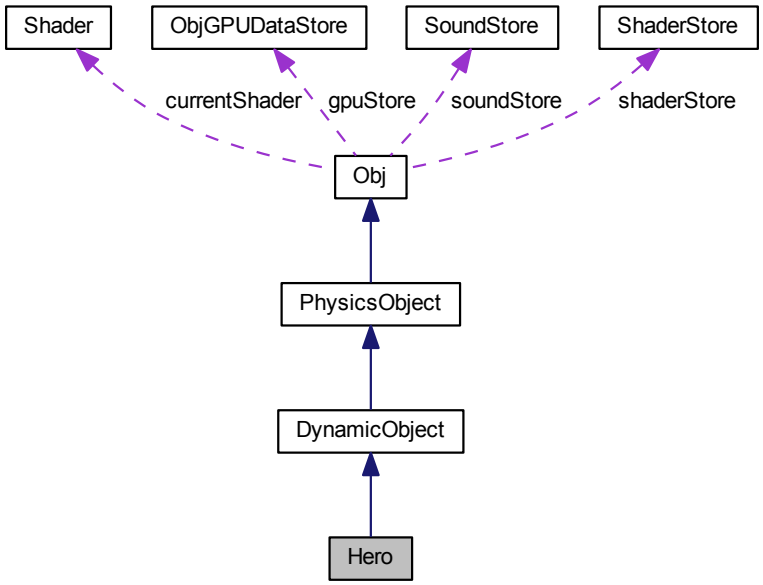
3.9 Hero Class Reference

```
#include <Hero.h>
```

Inheritance diagram for Hero:



Collaboration diagram for Hero:



Public Member Functions

- Hero (float x, float y)

  Hero constructor.
- void death ()

  Kills the hero and resets the stage.
- void jump ()

  Makes the hero jump.
- void win ()

  Play stage win fanfare and set the levelWin flag.

Public Attributes

- cpVect startPos

  Hero starting position.
- bool dead

  Flag indicates if the hero is dead.
- bool canJump

  Flag indicates if the hero is currently allowed to jump.
- cpVect relVel

  Velocity relative to the surface the hero is currently on (undefined if not on surface)
- bool levelWin

  Flag indicates if hero has reached the goal.

Additional Inherited Members

3.9.1 Detailed Description

The Hero class implements the hero.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Hero::Hero (float *x*, float *y*)

Hero constructor.

Parameters

x	The x coordinate of the hero.
y	The y coordinate of the hero.

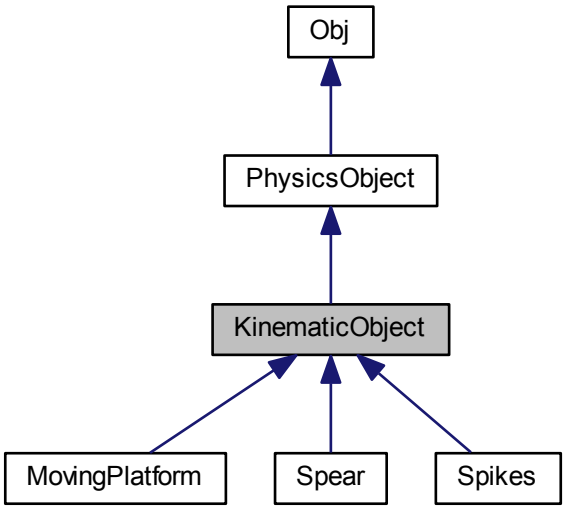
The documentation for this class was generated from the following file:

- Hero.h

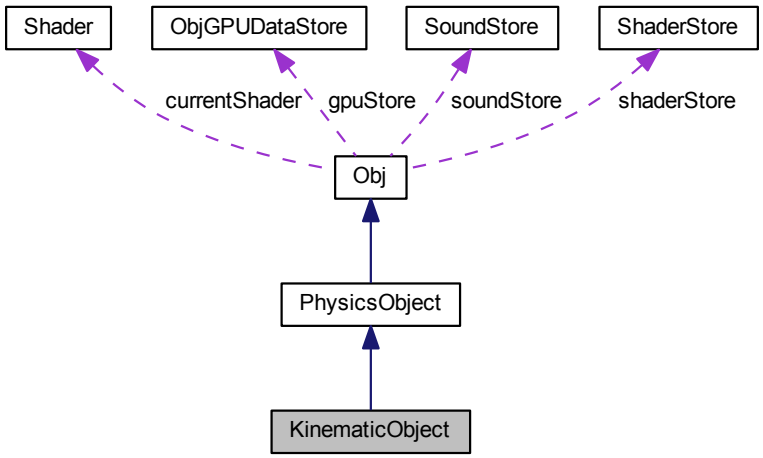
3.10 KinematicObject Class Reference

`#include <KinematicObject.h>`

Inheritance diagram for KinematicObject:



Collaboration diagram for KinematicObject:



Public Member Functions

- KinematicObject ()

  KinematicObject default constructor.
- KinematicObject (float x, float y, float scale, float elast, float fric, int type, std::string gpuPath, std::string vPath, std::string fPath)

  KinematicObject constructor.
- virtual void update (float dt)

  Updates kinematic object (position, velocity, etc)

Additional Inherited Members

3.10.1 Detailed Description

The KinematicObject class is derived from the PhysicsObject class. This type of object is identical to static objects except that it can be moved by directly changing position, velocity, force, etc.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 KinematicObject::KinematicObject (float *x,* float *y,* float *scale,* float *elast,* float *fric,* int *type,* std::string *gpuPath,* std::string *vPath,* std::string *fPath*)

KinematicObject constructor.

Parameters

x	The x-coordinate of the dynamic object

Parameters

y	The y-coordinate of the dynamic object
scale	Scalar factor by which to scale the object during GPU rendering
elast	Elasticity of the object
fric	Friction factor of the object
type	Type of object (different values affect collision routines)
gpuPath	The path to the obj file for the object
vPath	The path to the vertex shader for the object
fPath	The path to the fragment shader for the object

3.10.3 Member Function Documentation

3.10.3.1 virtual void KinematicObject::update (float *dt*) [virtual]

Updates kinematic object (position, velocity, etc)

Parameters

dt	The timestep to move the object through.

Reimplemented in MovingPlatform.

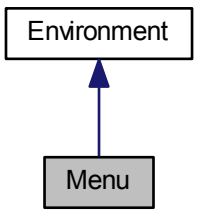
The documentation for this class was generated from the following file:

- KinematicObject.h

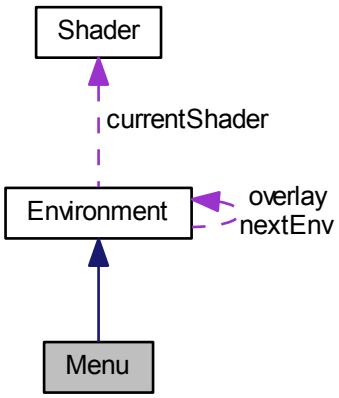
3.11 Menu Class Reference

`#include <Menu.h>`

Inheritance diagram for Menu:



Collaboration diagram for Menu:



Public Member Functions

- Menu (bool inGame)

  Menu constructor.
- ~Menu ()

  Menu destructor.
- void updateEnvironment (double dt)

  Function for updating the environment.
- void drawEnvironment ()

  Function for drawing the environment.
- void processKB (int key, int scancode, int action, int mods)

  Function for keyboard input processing.
- void processContinuousInput ()

  Function for continuous input processing.
- bool processMousePosition (float xpos, float ypos)

  Function for mouse position change processing.
- void processMouseClick (int button, int action, int mods)

  Function for mouse input processing.
- void ShowSubMenu (int level, bool bShow=true)

  Displays main menu.
- void showLevelSubMenu (bool bShow)

  Displays level select menu.
- void updateScreenSize ()

  Changes layout due to screen size update.
- void addNewItem (char *szNormal, char *szHover, char *szSelected, int posX, int posY, MenuItemHandler handler, void *param, short level, bool active=false, bool fixed=false)

  Adds button to the menu.

Private Member Functions

- void hover_menuitem (MenuItem *)

  Called when mouse position is changes to detect hover over buttons.
- void click_menuitem (MenuItem *)

  Called when mouse is clicked to detect click on buttons.

Private Attributes

- std::vector< MenuItem * > vecMenuItem

  Stores menu buttons.

Additional Inherited Members

3.11.1 Detailed Description

The Menu class is derived from the Environment class and holds and handles changes to the game state when the user is not playing a stage (i.e. is in a menu of some kind).

3.11.2 Member Function Documentation

3.11.2.1 void Menu::processKB (int *key*, int *scancode*, int *action*, int *mods*) [virtual]

Function for keyboard input processing.

Parameters

key	Key to which the action corresponds.
scancode	System specific key code.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implements Environment.

3.11.2.2 void Menu::processMouseClick (int *button*, int *action*, int *mods*) [virtual]

Function for mouse input processing.

Parameters

button	Mouse button to which the action corresponds.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implements Environment.

3.11.2.3 bool Menu::processMousePosition (float *xpos*, float *ypos*) [virtual]

Function for mouse position change processing.

Parameters

xpos	Mouse cursor x-position
ypos	Mouse cursor y-position

Implements Environment.

3.11.2.4 void Menu::showLevelSubMenu (bool *bShow*)

Displays level select menu.

Parameters

bShow	Flag specifies whether buttons should be drawn.

3.11.2.5 void Menu::ShowSubMenu (int *level*, bool *bShow* = `true`)

Displays main menu.

Parameters

level	Menu depth.
bShow	Flag specifies whether buttons should be drawn.

3.11.2.6 void Menu::updateEnvironment (double *dt*) [virtual]

Function for updating the environment.

Parameters

dt	Time step to be used for the update (time since last update) in milliseconds.

Implements Environment.

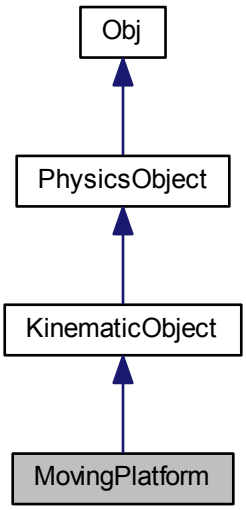
The documentation for this class was generated from the following file:

- Menu.h

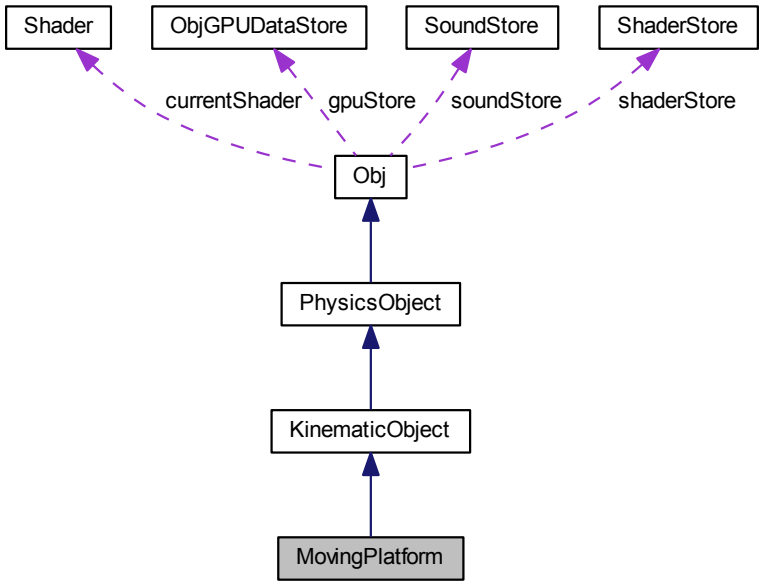
3.12 MovingPlatform Class Reference

`#include <MovingPlatform.h>`

Inheritance diagram for MovingPlatform:



Collaboration diagram for MovingPlatform:



Public Member Functions

- MovingPlatform (float w, float speed, std::vector< cpVect > &path, float thickness=50.0f)

Creates a new moving platform.

- void update (float dt)

  Updates platform position.

Private Attributes

- int dir

  Direction of movement.

- float speed

  Speed of movement.

- int destinationNode

  Node that the platform is headed to.

- std::vector< cpVect > path

  The path the platform takes.

Additional Inherited Members

3.12.1 Detailed Description

The MovingPlatform class implements moving platforms.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 MovingPlatform::MovingPlatform (float *w*, float *speed*, std::vector< cpVect > & *path*, float *thickness* = `50.0f`)

Creates a new moving platform.

Parameters

w	Width of platform
speed	Speed of platform
path	The path the platform takes
thickness	The vertical thickness of the platform

3.12.3 Member Function Documentation

3.12.3.1 void MovingPlatform::update (float *dt*) `[virtual]`

Updates platform position.

Parameters

dt	The timestep to move the platform through.

Reimplemented from KinematicObject.

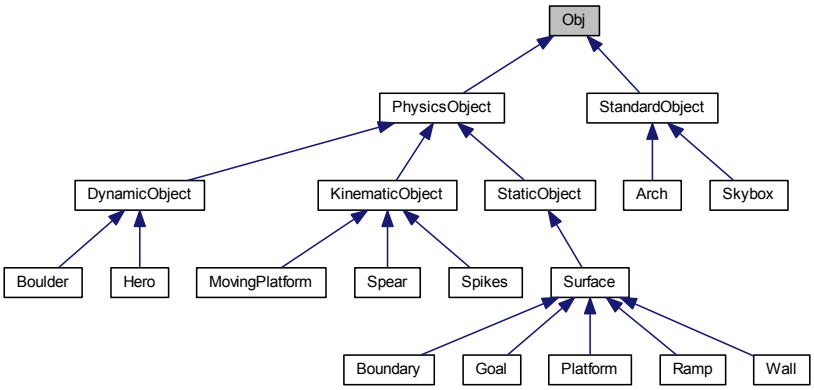
The documentation for this class was generated from the following file:

- MovingPlatform.h

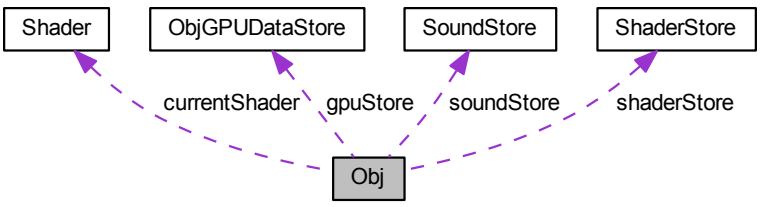
3.13 Obj Class Reference

`#include <Obj.h>`

Inheritance diagram for Obj:



Collaboration diagram for Obj:



Public Member Functions

- void render (glm::vec3 pos, float angle)

  Renders an object.

Public Attributes

- std::vector< ObjGPUData * > gpuDataList

  Vector of meshes associated with this object.
- std::vector< Shader * > shaderList

  Vector of shaders associated with this object.
- bool transformOverrides = false

  Flag to activate override transformations.
- std::vector< glm::vec3 > translationOverrideList

  List of translation overrides (one for each mesh).
- std::vector< float > rotationOverrideList

  List of rotation overrides (one for each mesh).
- std::vector< glm::mat4 > shearOverrideList

  List of shear overrides (one for each mesh).
- float height

  Height of the object.
- float width

  Width of the object.
- glm::vec3 modelScale

  Initial scaling matrix to be applied the model.
- bool draw = true

  Flag for whether the object should be drawn or not.

Static Public Attributes

- static glm::mat4 matProjection

  The projection matrix.
- static glm::mat4 matView

  The view matrix.
- static glm::vec4 primaryLightPos

  The primary light position.
- static glm::vec3 primaryLightLa

  The primary ambient light vector.
- static glm::vec3 primaryLightLd

  The primary diffuse light vector.
- static glm::vec3 primaryLightLs

  The primary spectral light vector.
- static ObjGPUDataStore gpuStore

  Collection of loaded meshes.
- static ShaderStore shaderStore

  Collection of loaded shaders.
- static SoundStore soundStore

  Collection of loaded sounds.
- static Shader * currentShader

  The shader currently being used.

3.13.1 Detailed Description

The Obj class acts as a base class for static, dynamic, and kinematic objects. It holds the physics data (Chipmunk 2D) and gpu data of an object.

3.13.2 Member Function Documentation

3.13.2.1 void Obj::render (glm::vec3 *pos*, float *angle*)

Renders an object.

Parameters

pos	Object position
angle	Rotation about z axis

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

- Obj.h

3.14 ObjGPUData Class Reference

`#include <ObjGPUData.h>`

Classes

- class **Material**
  Class for storing material information loaded from .mtl file.

Public Member Functions

- ObjGPUData (const char *objFile, float angle=0.0f, bool scalingMode=true)
  ObjGPUData constructor.

Public Attributes

- std::vector< glm::vec3 > vList
  Stores vertex coordinates loaded from .obj file.
- std::vector< glm::vec2 > vTextureList
  Stores texture coordinates loaded from .obj file.
- std::vector< glm::vec3 > vNormalList
  Stores vertex normals loaded from .obj file.
- std::vector< glm::vec4 > vTangentList
  Stores vertex tangents (calculated)
- std::vector< GLuint > fList
  Stores faces loaded from .obj file.
- std::vector< int > materialIndices
  Marks divisions of different materials given in the .obj file (and defined in .mtl file)
- std::vector< Material > materials
  Stores material information loaded from .mtl file.
- GLuint vertexArrayObj

Name to bind the vertex array object.

- GLuint elementBuffer

  Name to bind the element buffer object.
- GLuint vertexBuffer

  Name to bind the vertex buffer object.
- GLuint textureBuffer

  Name to bind the texture coordinate buffer object.
- GLuint normalBuffer

  Name to bind the vertex normal buffer object.
- GLuint tangentBuffer

  Name to bind the vertex tangent buffer object.
- glm::mat4 unitScale

  Scaling factor to adjust object to size 1.0 in y-axis (height)
- glm::mat4 rotation

  Rotation about y-axis to adjust objects initial rotational centering (if required: this is what the optional constructor argument sets)
- float whRatio

  Ratio of maximum x-axis vertex separation (width) to maximum y-axis vertex separation (height)
- GLint texturePlane

  Used for surface shader; specifies which plane the face falls in for texturing.
- GLboolean parallax = false

  Used for surface shader; flag specifies if parallax mapping should be used.

Private Member Functions

- void loadObject (const char *fileName)

  Parses .obj and .mtl files and stores the data.
- dataType getDataType (std::string dataTypeString)

  Converts string to .obj file datatype (dataType).
- mtlDataType getMtlDataType (std::string dataTypeString)

  Converts string to .mtl file datatype (mtlDataType).

3.14.1 Detailed Description

The ObjGPUData class loads and stores data used by the GPU to render objects. Each object is defined by three files which are loaded by this class: an object file (.obj) which contains information about vertices, faces, normals, and texture coordinates; a material file (.mtl) which contains texture and lighting information; and an image file(s) (.dds) which contain the texture images.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 ObjGPUData::ObjGPUData (const char * *objFile*, float *angle* = `0.0f`, bool *scalingMode* = `true`)

ObjGPUData constructor.

Parameters

objFile	Object and material file path (these should have the same name) without extension
angle	Initial y-axis rotation in radians (optional: default 0.0)
scalingMode	Flag specifies whether loaded object should be scaled to unit height (default = true)

3.14.3 Member Function Documentation

3.14.3.1 dataType ObjGPUData::getDataType (std::string *dataTypeString*) [private]

Converts string to .obj file datatype (dataType).

Parameters

dataTypeString	The string to be converted.

Returns

Equivalent dataType value.

3.14.3.2 mtlDataType ObjGPUData::getMtlDataType (std::string *dataTypeString*) [private]

Converts string to .mtl file datatype (mtlDataType).

Parameters

dataTypeString	The string to be converted.

Returns

Equivalent mtlDataType value.

3.14.3.3 void ObjGPUData::loadObject (const char * *fileName*) [private]

Parses .obj and .mtl files and stores the data.

Parameters

fileName	Name (without extension) of the .obj and .mtl files

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

- ObjGPUData.h

3.15 ObjGPUDataStore Class Reference

```
#include <ObjGPUDataStore.h>
```

Public Member Functions

- ObjGPUDataStore ()

  Constructor.
- ObjGPUData * add (std::string path, float angle=0.0, bool scalingMode=true)

  Adds a new mesh/texture to the store: if not in the store, it loads and stores; if already present, it does nothing.
- ObjGPUData * get (std::string path)

  Retrieves gpu data from the store.

Private Attributes

- std::map< std::string, ObjGPUData * > gpuMap

  Map translates object file name to stored data.

3.15.1 Detailed Description

The ObjGPUDataStore class acts as a storage class for loaded object mesh/texture data.

3.15.2 Member Function Documentation

3.15.2.1 ObjGPUData* ObjGPUDataStore::add (std::string *path*, float *angle* = `0.0`, bool *scalingMode* = `true`)

Adds a new mesh/texture to the store: if not in the store, it loads and stores; if already present, it does nothing.

Parameters

path	Path to object file
angle	Initial y-axis rotation in radians (optional: default 0.0)
scalingMode	Flag specifies if object should be scaled to unit height when loaded (default true)

3.15.2.2 ObjGPUData* ObjGPUDataStore::get (std::string *path*)

Retrieves gpu data from the store.

Parameters

path	Path to object file to search for

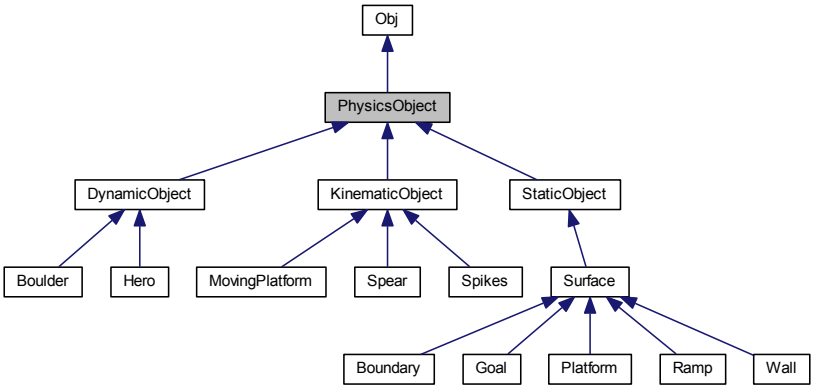
The documentation for this class was generated from the following file:

- ObjGPUDataStore.h

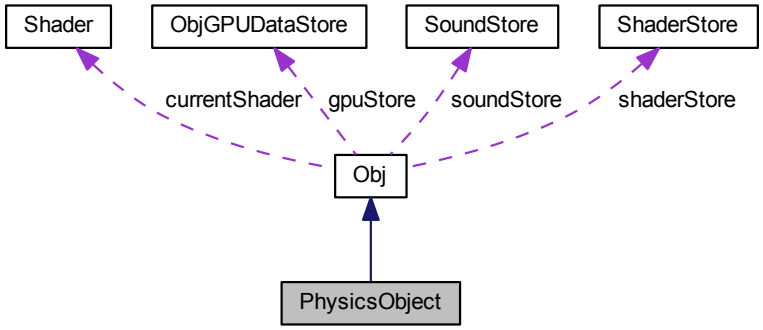
3.16 PhysicsObject Class Reference

`#include <PhysicsObject.h>`

Inheritance diagram for PhysicsObject:



Collaboration diagram for PhysicsObject:



Public Member Functions

- virtual ~PhysicsObject ()

  PhysicsObject constructor.
- void render ()

  Renders the object.

Public Attributes

- cpBody * body

  Pointer to Chipmunk 2D body associated with the object.
- cpShape * shape

  Pointer to Chipmunk 2D shape associated with the object.
- cpVect standingNormal

  Normal used in collision handler to determine if hero can jump.
- cpVect deathNormal

  Normal used in collision handler to determine if hero should be killed.

Static Public Attributes

- static cpSpace * space

  Pointer to current Chipmunk 2D space.

3.16.1 Detailed Description

The PhysicsObject class is derived from the Obj class and is the parent to all objects associated with Chipmunk2D.

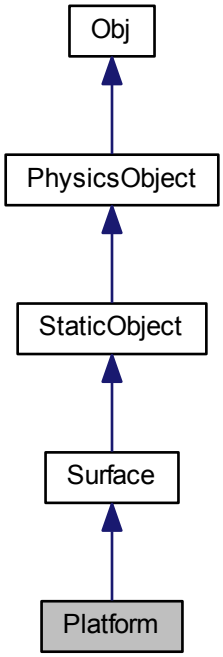
The documentation for this class was generated from the following file:

- PhysicsObject.h

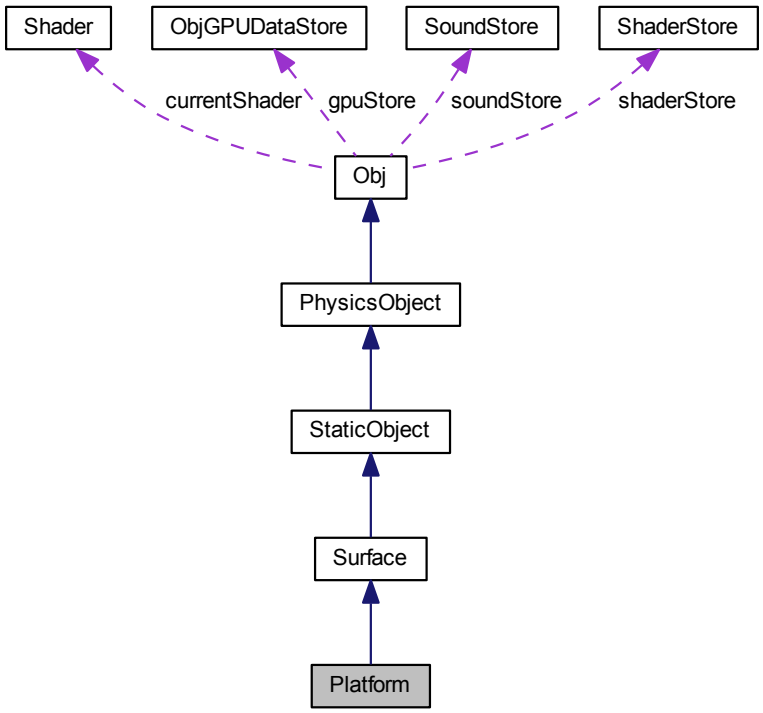
3.17 Platform Class Reference

`#include <Platform.h>`

Inheritance diagram for Platform:



Collaboration diagram for Platform:



Public Member Functions

- Platform (float x1, float x2, float ymid, float thickness=50.0f)

  Platform constructor.

Additional Inherited Members

3.17.1 Detailed Description

The Platform class implements platform objects.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 Platform::Platform (float *x1*, float *x2*, float *ymid*, float *thickness* = `50.0f`)

Platform constructor.

Parameters

x1	The leftmost extent of the platform.
x2	The rightmost extent of the platform.
ymid	The y-coordinate of the platform.
thickness	The vertical thickness of the platform (default = 50.0).

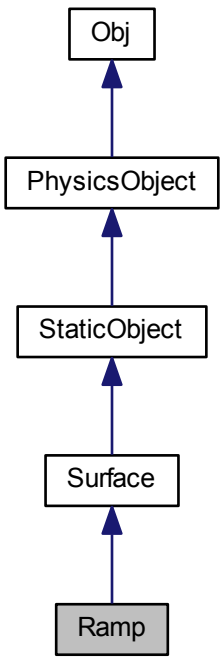
The documentation for this class was generated from the following file:

- Platform.h

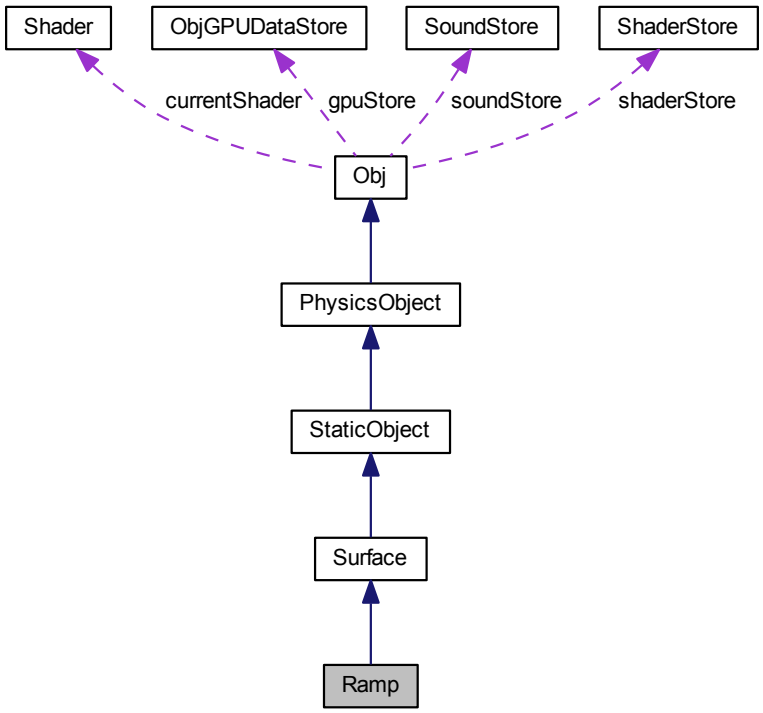
3.18 Ramp Class Reference

```
#include <Ramp.h>
```

Inheritance diagram for Ramp:



Collaboration diagram for Ramp:



Public Member Functions

- Ramp (float x1, float x2, float y1, float y2, float thickness=50.0f)

  Ramp constructor.

Additional Inherited Members

3.18.1 Detailed Description

The Ramp class implements ramp objects.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 Ramp::Ramp (float *x1*, float *x2*, float *y1*, float *y2*, float *thickness* = `50.0f`)

Ramp constructor.

Parameters

x1	The leftmost extent of the ramp.
x2	The rightmost extent of the ramp.
y1	The bottom y-coordinate of the ramp.
y2	The top y-coordinate of the ramp.
thickness	The thickness of the ramp (default = 50.0).

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

- Ramp.h

3.19 Shader Class Reference

`#include <Shader.h>`

Public Member Functions

- Shader (const char *vShader, const char *fShader)
  Shader constructor.

Public Attributes

- GLuint shaderProgram
  Name to bind the shader program.
- std::map< std::string, GLuint > uniformIDMap
  Stores names which bind the shader uniform IDs.

3.19.1 Detailed Description

The Shader class is used to build shaders. All of the names/identifiers required to bind/use the shader afterwards are stored.

3.19.2 Constructor & Destructor Documentation

3.19.2.1 Shader::Shader (const char * *vShader*, const char * *fShader*)

Shader constructor.

Parameters

vShader	Vertex shader path
fShader	Fragment shader path

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

- Shader.h

3.20 ShaderStore Class Reference

`#include <ShaderStore.h>`

Public Member Functions

- ShaderStore ()

  Constructor.
- Shader * add (std::string vpath, std::string fpath)

  Adds a new shader to the store: if not in the store, it loads and stores; if already present, it does nothing.
- Shader * get (std::string vpath, std::string fpath)

  Retrieves gpu data from the store.

Private Attributes

- std::map< std::string, Shader * > shaderMap

  Map translates sound file name to shader.

3.20.1 Detailed Description

The ShaderStore class acts as a storage class for loaded shaders.

3.20.2 Member Function Documentation

3.20.2.1 Shader* ShaderStore::add (std::string *vpath*, std::string *fpath*)

Adds a new shader to the store: if not in the store, it loads and stores; if already present, it does nothing.

Parameters

vpath	Path to vertex shader
fpath	Path to fragment shader

3.20.2.2 Shader* ShaderStore::get (std::string *vpath*, std::string *fpath*)

Retrieves gpu data from the store.

Parameters

vpath	Path to vertex shader
fpath	Path to fragment shader

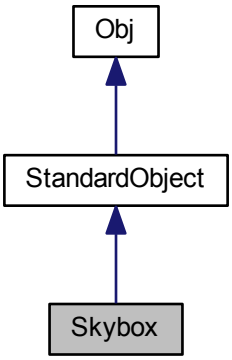
The documentation for this class was generated from the following file:

- ShaderStore.h

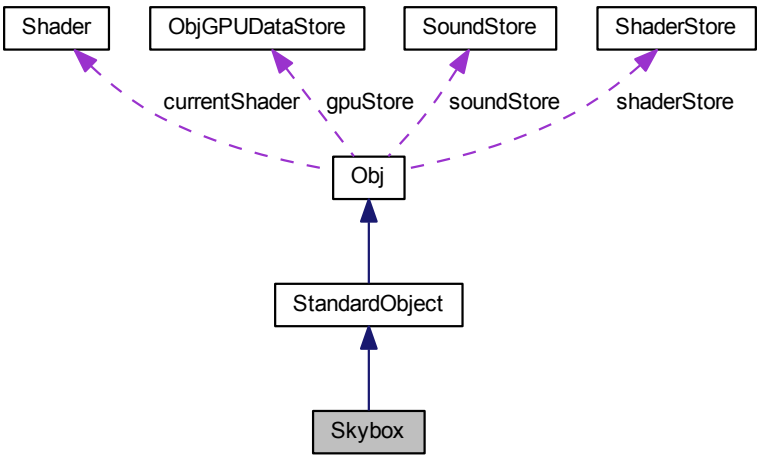
3.21 Skybox Class Reference

```
#include <Skybox.h>
```

Inheritance diagram for Skybox:



Collaboration diagram for Skybox:



Public Member Functions

- Skybox (float x, float y, int bgNum)
  Skybox constructor.

Additional Inherited Members

3.21.1 Detailed Description

The Skybox class implements the skybox used to provide the backdrop of the stage.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 Skybox::Skybox (float *x*, float *y*, int *bgNum*)

Skybox constructor.

Parameters

x	The x coordinate of the skybox.
y	The y coordinate of the skybox.
bgNum	Int corresponding to the desired backdrop.

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

- Skybox.h

3.22 Sound Class Reference

`#include <Sound.h>`

Public Member Functions

- Sound (std::string path)
  Sound constructor.
- ~Sound ()
  Sound destructor.
- void play (int loop=0)
  Plays the sound data contained in the class.
- void stop ()
  Stops playing sound.

Private Attributes

- ALuint audioBuffer
  Binding id for sound data storage.
- ALuint audioSource
  Binding id for position, velocity, etc. of sound source.
- bool loaded
  Flag specifies if sound is loaded.

3.22.1 Detailed Description

The Sound class loads and stores sound data.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 Sound::Sound (std::string *path*)

Sound constructor.

Parameters

path	Path to the sound file that should be loaded.

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

- Sound.h

3.23 SoundStore Class Reference

`#include <SoundStore.h>`

Public Member Functions

- SoundStore ()
  Constructor.
- Sound * add (std::string path)
  Adds a new sound to the store: if not in the store, it loads and stores; if already present, it does nothing.
- Sound * get (std::string path)
  Retrieves gpu data from the store.

Private Attributes

- std::map< std::string, Sound * > soundMap
  Map translates sound file name to stored data.

3.23.1 Detailed Description

The SoundStore class acts as a storage class for loaded sounds.

3.23.2 Member Function Documentation

3.23.2.1 Sound* SoundStore::add (std::string *path*)

Adds a new sound to the store: if not in the store, it loads and stores; if already present, it does nothing.

Parameters

path	Path to sound file

3.23.2.2 Sound* SoundStore::get (std::string *path*)

Retrieves gpu data from the store.

Parameters

path	Path to sound file

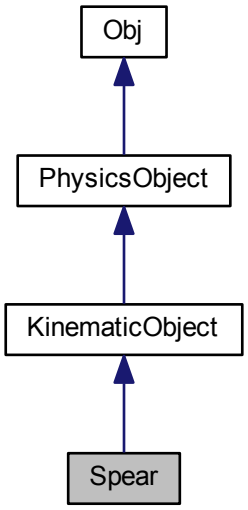
The documentation for this class was generated from the following file:

- SoundStore.h

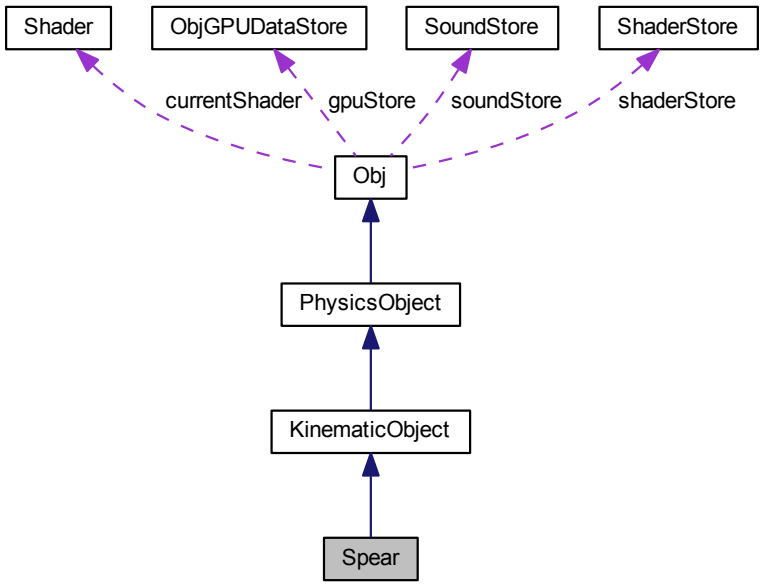
3.24 Spear Class Reference

```
#include <Spear.h>
```

Inheritance diagram for Spear:



Collaboration diagram for Spear:



Public Member Functions

- Spear (float x, float y, float rotation=0.0f)

  Spear constructor.

Additional Inherited Members

3.24.1 Detailed Description

The Spear class implements the spear hazard.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 Spear::Spear (float *x*, float *y*, float *rotation* = `0.0f`)

Spear constructor.

Parameters

x	The x coordinate of the spear.
y	The y coordinate of the spear.
rotation	The rotation of the spear about the z-axis.

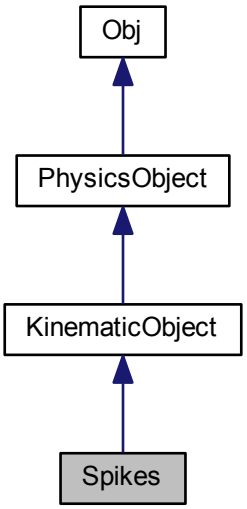
The documentation for this class was generated from the following file:

- Spear.h

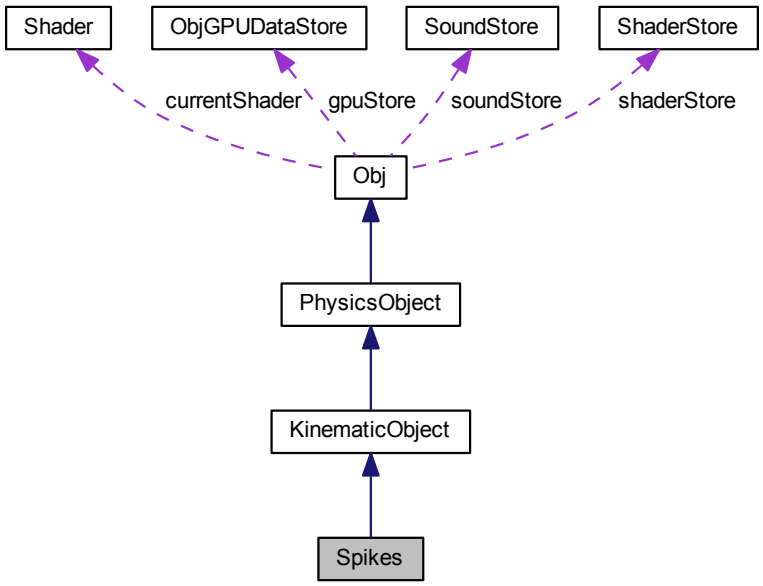
3.25 Spikes Class Reference

```
#include <Spikes.h>
```

Inheritance diagram for Spikes:



Collaboration diagram for Spikes:



Public Member Functions

- Spikes (float x, float y, float rotation=0.0f)

  Spikes constructor.

Additional Inherited Members

3.25.1 Detailed Description

The Spikes class implements the spikes hazard.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 Spikes::Spikes (float *x*, float *y*, float *rotation* = `0.0f`)

Spikes constructor.

Parameters

x	The x coordinate of the spikes.
y	The y coordinate of the spikes.
rotation	The rotation of the spikes about the z-axis.

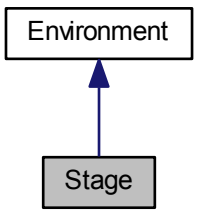
The documentation for this class was generated from the following file:

- Spikes.h

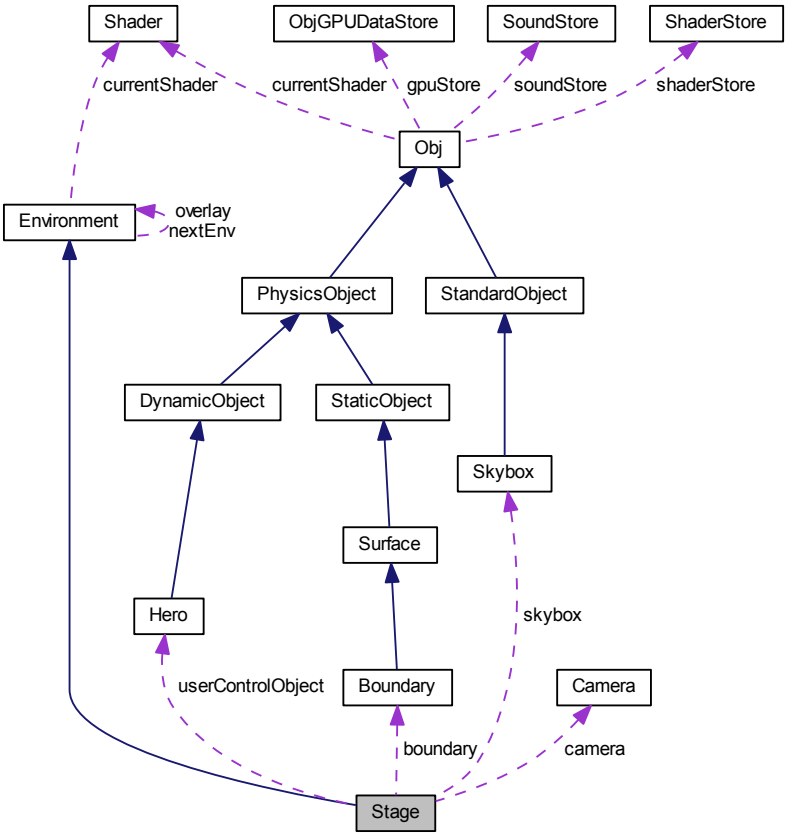
3.26 Stage Class Reference

```
#include <Stage.h>
```

Inheritance diagram for Stage:



Collaboration diagram for Stage:



Public Member Functions

- Stage (std::string stageName)
  Stage constructor.
- ~Stage ()
  Stage destructor.
- void updateEnvironment (double dt)
  Function for updating the environment.
- void drawEnvironment ()
  Function for drawing the environment.
- void processKB (int key, int scancode, int action, int mods)
  Function for keyboard input processing.
- void processContinuousInput ()
  Function for continuous input processing.
- bool processMousePosition (float xpos, float ypos)
  Function for mouse position change processing.
- void processMouseClick (int button, int action, int mods)
  Function for mouse input processing.
- bool checkCompletion ()
  Checks if stage is complete.
- void updateScreenSize ()
  Updates the projection matrix when screen size is changed.

Private Member Functions

- void drawObj (PhysicsObject currentObj, bool isBoundary=false)

  Draws an object.

Private Attributes

- cpSpace * envSpace

  Pointer to the chipmunk space associated with the stage.
- std::vector< PhysicsObject * > physicsObjects

  List of physics objects (excluding kinematic) in stage.
- std::vector< KinematicObject * > kinematicObjects

  List of kinematic objects in stage.
- std::vector< StandardObject * > standardObjects

  List of standard objects in stage.
- Skybox * skybox

  The skybox object provides the backdrop of the stage.
- Boundary * boundary

  The boundary object provides the surface setting.
- float stageTime

  Time elapsed since beginning the stage.
- Hero * userControlObject

  Pointer to the dynamic object that is controlled by the user (normally the hero object).
- Camera camera

  Used to control the view.
- int winTimer

  Countdown timer set when stage is won; determines when exit to menu occurs.
- std::string stageName

  Name of current stage file (used for reseting stage).

Additional Inherited Members

3.26.1 Detailed Description

The Stage class is derived from the Environment class and holds and handles changes to the game state when the user is playing a stage.

3.26.2 Member Function Documentation

3.26.2.1 bool Stage::checkCompletion ( )

Checks if stage is complete.

Returns

True if stage is complete, else false.

3.26.2.2 void Stage::drawObj (PhysicsObject *currentObj*, bool *isBoundary* = `false`) [private]

Draws an object.

Parameters

currentObj	Object that should be drawn.
isBoundary	Flag for boundaries (default = false); boundaries are drawn slightly differently.

3.26.2.3 void Stage::processKB (int *key,* int *scancode,* int *action,* int *mods*) [virtual]

Function for keyboard input processing.

Parameters

key	Key to which the action corresponds.
scancode	System specific key code.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implements Environment.

3.26.2.4 void Stage::processMouseClick (int *button,* int *action,* int *mods*) [virtual]

Function for mouse input processing.

Parameters

button	Mouse button to which the action corresponds.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implements Environment.

3.26.2.5 bool Stage::processMousePosition (float *xpos,* float *ypos*) [virtual]

Function for mouse position change processing.

Parameters

xpos	Mouse cursor x-position
ypos	Mouse cursor y-position

Implements Environment.

3.26.2.6 void Stage::updateEnvironment (double *dt*) [virtual]

Function for updating the environment.

Parameters

dt	Time step to be used for the update (time since last update) in milliseconds.

Implements Environment.

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

- Stage.h

3.27 StageLoader Class Reference

`#include <StageLoader.h>`

Public Member Functions

- StageLoader (std::string fileName, std::vector< PhysicsObject * > &physicsObjects, std::vector< KinematicObject * > &kinematicObjects, std::vector< StandardObject * > &standardObjects, Skybox *&skybox, Boundary *&boundary, Hero *&userControlObject)

  StageLoader constructor.

Private Member Functions

- std::string stripWhitespace (std::string str)

  Strips all whitespace from string.
- std::string checkField (std::string field)

  Checks if stageInfo contains a key; if not reports error.
- void reportError ()

  Reports error at current line.
- void getNextLine ()

  Retrieves next line.

Private Attributes

- std::string fileName

  Stage file path.
- int lineNo

  Current line number.
- std::string curLine

  Current line being parsed.
- std::string line

  Current line being parsed, stripped of whitespace.
- std::ifstream inFile

  File stream of the input file.
- std::map< std::string, std::string > stageInfo

  Map that stores stage information.

3.27.1 Detailed Description

The StageLoader class parses stage files to create levels at runtime.

3.27.2 Constructor & Destructor Documentation

3.27.2.1 StageLoader::StageLoader (std::string *fileName*, std::vector< PhysicsObject * > & *physicsObjects*, std::vector< KinematicObject * > & *kinematicObjects*, std::vector< StandardObject * > & *standardObjects*, Skybox *& *skybox*, Boundary *& *boundary*, Hero *& *userControlObject*)

StageLoader constructor.

Parameters

fileName	The path to the stage file.
physicsObjects	Reference to vector to be filled with physics objects.
kinematicObjects	Reference to vector to be filled with kinematic objects.
standardObjects	Reference to vector to be filled with standard objects.
skybox	Reference to Skybox to be populated by skybox object.
boundary	Reference to Boundary to be populated by boundary object.
userControlObject	Reference to Hero to be populated with the hero object.

3.27.3 Member Function Documentation

3.27.3.1 std::string StageLoader::checkField (std::string *field*) [private]

Checks if stageInfo contains a key; if not reports error.

Parameters

field	The field to be checked.

Returns

The value paired with the key.

3.27.3.2 std::string StageLoader::stripWhitespace (std::string *str*) [private]

Strips all whitespace from string.

Parameters

str	The string to be stripped.

Returns

Whitespace stripped string.

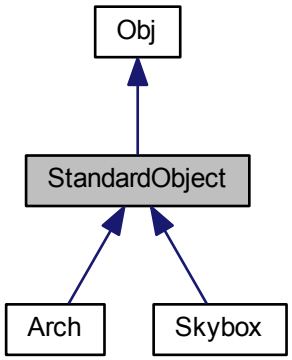
The documentation for this class was generated from the following file:

- StageLoader.h

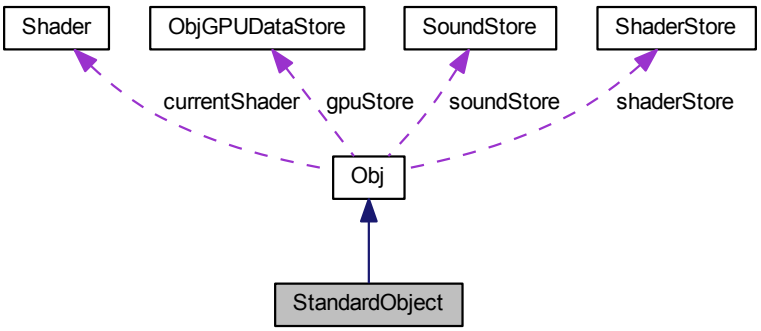
3.28 StandardObject Class Reference

`#include <StandardObject.h>`

Inheritance diagram for StandardObject:



Collaboration diagram for StandardObject:



Public Member Functions

- void render ()

  Renders the object.

Public Attributes

- glm::vec3 position

  Object position.
- float angle

  Object rotation about z-axis.

Additional Inherited Members

3.28.1 Detailed Description

The StandardObject class is derived from the Obj class and is the parent to all objects that do not take part in physics/collision calculations.

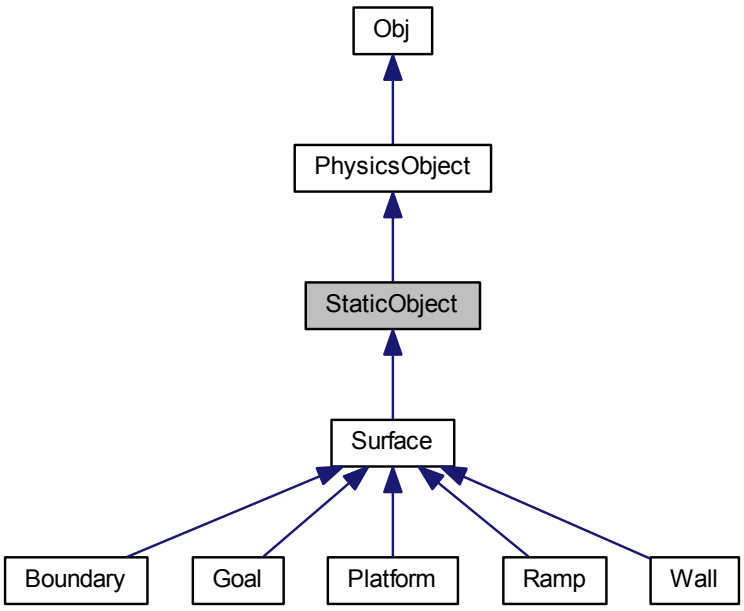
The documentation for this class was generated from the following file:

- StandardObject.h

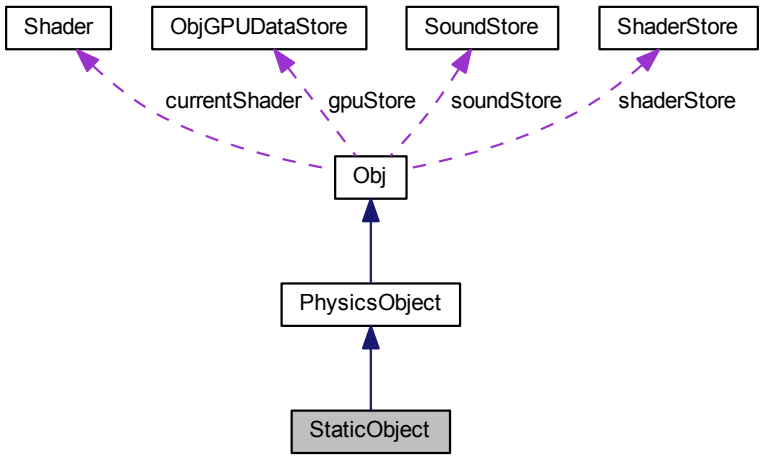
3.29 StaticObject Class Reference

`#include <StaticObject.h>`

Inheritance diagram for StaticObject:



Collaboration diagram for StaticObject:



Public Member Functions

- StaticObject ()

  StaticObject default constructor.
- StaticObject (float x, float y, float scale, float elast, float fric, int type, std::string gpuPath, std::string vPath, std::string fPath)

  StaticObject constructor.

Additional Inherited Members

3.29.1 Detailed Description

The StaticObject class is derived from the PhysicsObject class. This type of object is subject only to collision calculations and may not be moved once placed in the space.

3.29.2 Constructor & Destructor Documentation

3.29.2.1 StaticObject::StaticObject (float *x,* float *y,* float *scale,* float *elast,* float *fric,* int *type,* std::string *gpuPath,* std::string *vPath,* std::string *fPath*)

StaticObject constructor.

Parameters

x	The x-coordinate of the dynamic object
y	The y-coordinate of the dynamic object
scale	Scalar factor by which to scale the object during GPU rendering

Parameters

elast	Elasticity of the object
fric	Friction factor of the object
type	Type of object (different values affect collision routines)
gpuPath	The path to the obj file for the object
vPath	The path to the vertex shader for the object
fPath	The path to the fragment shader for the object

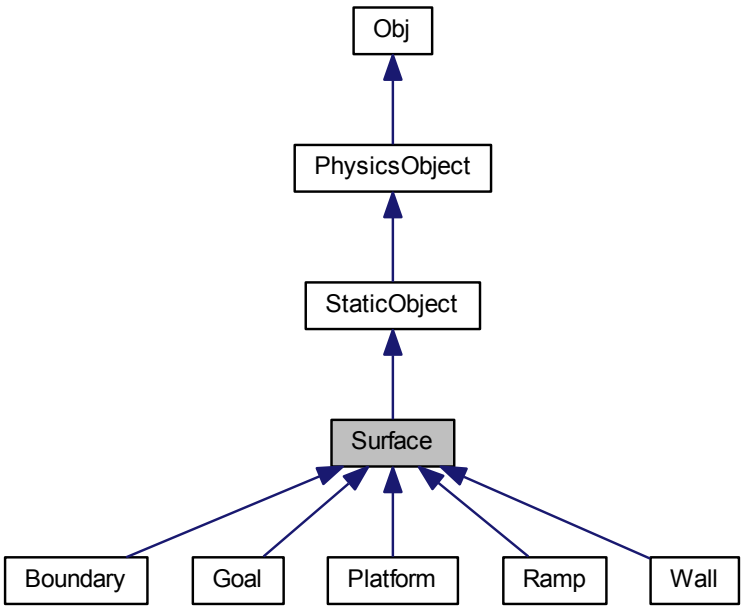
The documentation for this class was generated from the following file:

- StaticObject.h

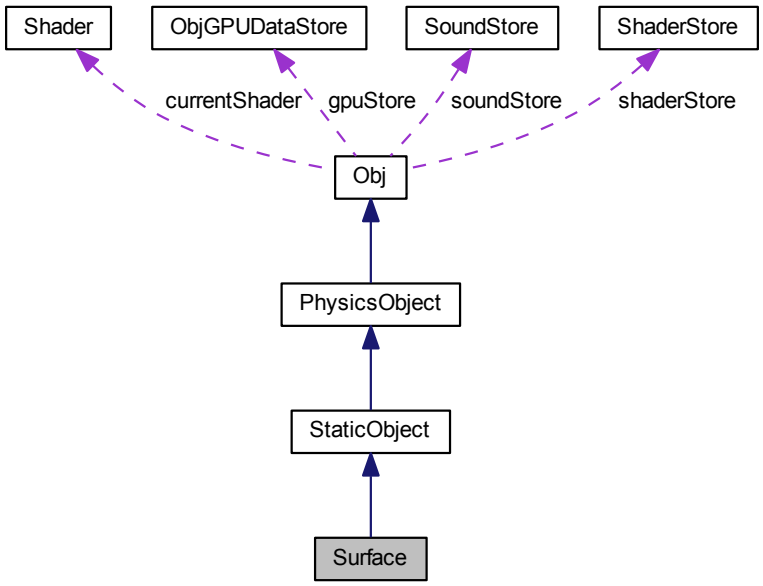
3.30 Surface Class Reference

`#include <Surface.h>`

Inheritance diagram for Surface:



Collaboration diagram for Surface:



Public Member Functions

- Surface (cpVect p1, cpVect p2, bool isRamp, float thickness=50.0f)

  Surface constructor.

Additional Inherited Members

3.30.1 Detailed Description

The Surface class is derived from the StaticObject class. Used primarily for stationary platforms.

3.30.2 Constructor & Destructor Documentation

3.30.2.1 Surface::Surface (cpVect *p1*, cpVect *p2*, bool *isRamp*, float *thickness* = `50.0f`)

Surface constructor.

Parameters

p1	Bottom left coordinate of bounding box
p2	Upper right coordinate of bounding box
isRamp	Flag specifies if surface is a ramp
thickness	Thickness of the surface

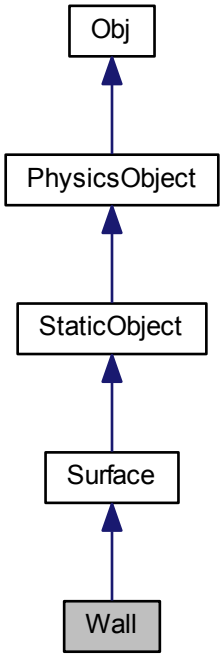
The documentation for this class was generated from the following file:

- Surface.h

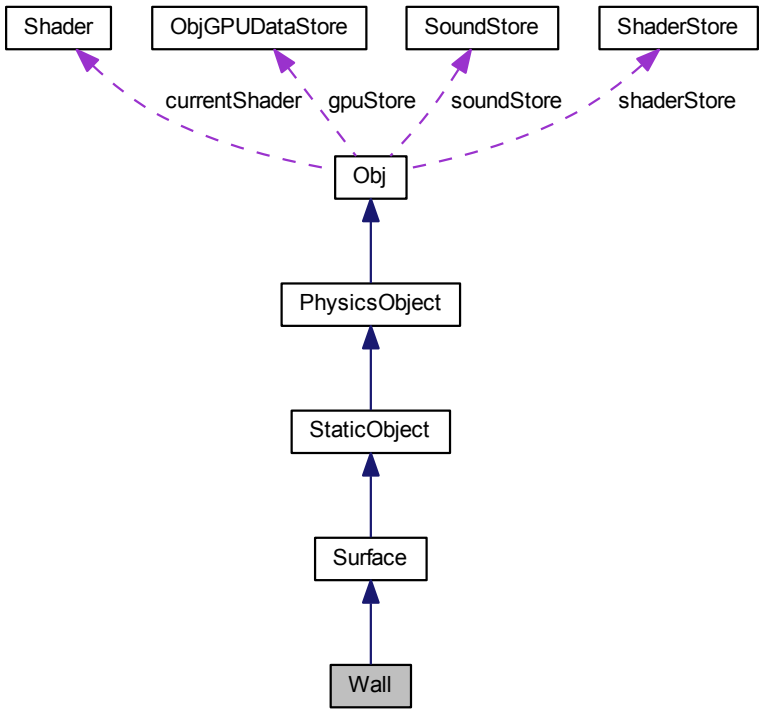
3.31 Wall Class Reference

```
#include <Wall.h>
```

Inheritance diagram for Wall:



Collaboration diagram for Wall:



Public Member Functions

- Wall (float y1, float y2, float xmid, float thickness=50.0f)
  Wall constructor.

Additional Inherited Members

3.31.1 Detailed Description

The Wall class implements wall objects.

3.31.2 Constructor & Destructor Documentation

3.31.2.1 Wall::Wall (float *y1*, float *y2*, float *xmid*, float *thickness* = `50.0f`)

Wall constructor.

Parameters

y1	The bottom y-coordinate of the wall.
y2	The top y-coordinate of the wall.
xmid	The x-coordinate of the wall.
thickness	The horizontal thickness of the wall (default = 50.0).

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

- Wall.h

Index