Traffic Simulation

Sándor Balázs
Version 1.3
Tue Dec 21 2021

Table of Contents

Table of contents

The Traffic Simulation application main files.

The Traffic Simulator application source code and resource files.

The folders in the project:

Folders and files for the running of the application.

3D Objects:

The 3D models and textures for the application.

Log:

Log files folder for the application.

Saves:

The saved map files for the application.

Config.csv:

Start configurations for the application.

imgui.ini

Start configurations for the application GUI.

Source folders for compiling:

Control:

The Control functions source files.

Model:

The model functions source files.

View:

The view functions source files.

Hierarchical Index

Class Hierarchy

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

Animator 10
AttributeData 24
BufferObject< target, usage > 26
BufferObject< BufferType::ElementArray > 26
Camera 30
ConfigLoader 49
ConfigSaver 53
Dijkstra 56
Edge 63
ElementInfo< T > 426
ElementInfo< std::array< T, N > > 426
ElementInfo< std::vector< T > > 426
ElementInfo< T[N]> 426
EventListener 70
std::false_type
    HasContiguousStorage< T > 130

fpsCounter 87
GLExtractPrimitiveType< T, C > ..... **Hiba! A könyvjelző nem létezik.**
GLExtractPrimitiveType< T, IsValidBaseType< T > > 426
Graph 93
GUI 103
ObjParser::IndexedVert 134
Logger 135
MapLoader 139
MapSaver 145
Mesh 152
Object3D 154
    Object3Dvehicle 200

Object3Droad 178
ObjectStorage 227
ObjParser 241
Point 246
ProgramObject 256
Render 262
reverseMap 294
ShaderObject 298
TextureObject< type > 301
TrafficSimulator ..... **Hiba! A könyvjelző nem létezik.**
std::true_type
    HasContiguousStorage< T[N]> 133
    HasContiguousStorage< std::array< T, N > > 131

HasContiguousStorage< std::vector< T > > 132

Vehicle 304
Mesh::Vertex 152
VertexArrayObject 317
WorkingWindow ..... **Hiba! A könyvjelző nem létezik.**
WorkWindow 320

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

Animator (The animation controller class) 10
AttributeData 24
BufferObject< target, usage > 26
Camera (The own OpenGL camera class) 30
ConfigLoader (The configuration loader class) 49
ConfigSaver (Configuration saver class) 53
Dijkstra (Dijkstra algorithm implementation) 56
Edge (The abstract Edge implementation) 63
EventListener (Event listener implementation. Contact: `sandorbalazs9402@gmail.com`) 70
fpsCounter (FPS counter, logger and limiter static class) 87
Graph (The abstract Edge implementation) 93
GUI (GUI Descriptor class. Contact: `sandorbalazs9402@gmail.com`) 103
HasContiguousStorage< T > 130
HasContiguousStorage< std::array< T, N > > 131
HasContiguousStorage< std::vector< T > > 132
HasContiguousStorage< T[N]> 133
ObjParser::IndexedVert 134
Logger (Logging system implementation) 135
MapLoader (Map loader class) 139
MapSaver (Map saver mechanism implementation) 145
Mesh 152
Object3D (The 3D renderable object representation) 154
Object3Droad (The parametric generated road implementation) 178
Object3Dvehicle (The renderable vehicle representation) 200
ObjectStorage (Load and store the 3D Models and Textures and create the render objects) 227
ObjParser 241
Point (The abstract point implementation) 246
ProgramObject 256
Render (The 3D Scene render implementation) 262
reverseMap (The reverse map implementation for the Dijkstra algorithm) 294
ShaderObject 298
TextureObject< type > 301
Vehicle (Vehicle model representation) 304
VertexArrayObject 317
WorkWindow (The OpenGL 3D window implementation) 320

File Index

File List

Here is a list of all files with brief descriptions:

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/resource.h 372
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/TrafficSimulator.cpp 374
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigLoader.cp p 336
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigLoader.h 337
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigSaver.cpp 339
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigSaver.h ..340
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/Logger.cpp 342
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/Logger.h 343
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapLoader.cpp 345
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapLoader.h ...346
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapSaver.cpp ..348
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapSaver.h 349
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Dijkstra.cpp 351
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Dijkstra.h 352
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Edge.cpp 354
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Edge.h 355
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Graph.cpp 357
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Graph.h 358
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Point.cpp 360
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Point.h 361
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/reverseMap.cpp .364
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/reverseMap.h 365
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Vehicle.cpp 367
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Vehicle.h 368
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Animator.cpp 376
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Animator.h 377
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Camera.cpp 380
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Camera.h 381
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/EventListener.cpp 384
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/EventListener.h 385
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/fpsCounter.cpp 388
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/fpsCounter.h 389
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/GUI.cpp 391
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/GUI.h 392
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/GUIWindows.cpp .396
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/RayPicking.cpp 414
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Render.cpp 415
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Render.h 416

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/WorkWindow.cpp 459
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/WorkWindow.h 460
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3D.cpp 397
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3D.h 398
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Droad.cpp 402
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Droad.h 403
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Dvehicle.cpp 407
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Dvehicle.h 408
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/ObjectStorage.cpp 410
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/ObjectStorage.h 411
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/BufferObject.h 420
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/BufferObject.inl 424
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/GLconversions.cpp 425
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/GLconversions.hpp 426
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/Mesh_OGL3.cpp 436
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/Mesh_OGL3.h 437
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ObjParser_OGL3.cpp 439
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ObjParser_OGL3.h 440
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ProgramObject.cpp 442
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ProgramObject.h 443
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ProgramObject.inl 446
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ShaderObject.cpp 447
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ShaderObject.h 448
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/TextureObject.h 450
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/TextureObject.inl 453
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/VertexArrayObject.cpp 454
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/VertexArrayObject.h 455

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/VertexArrayObject.inl 458

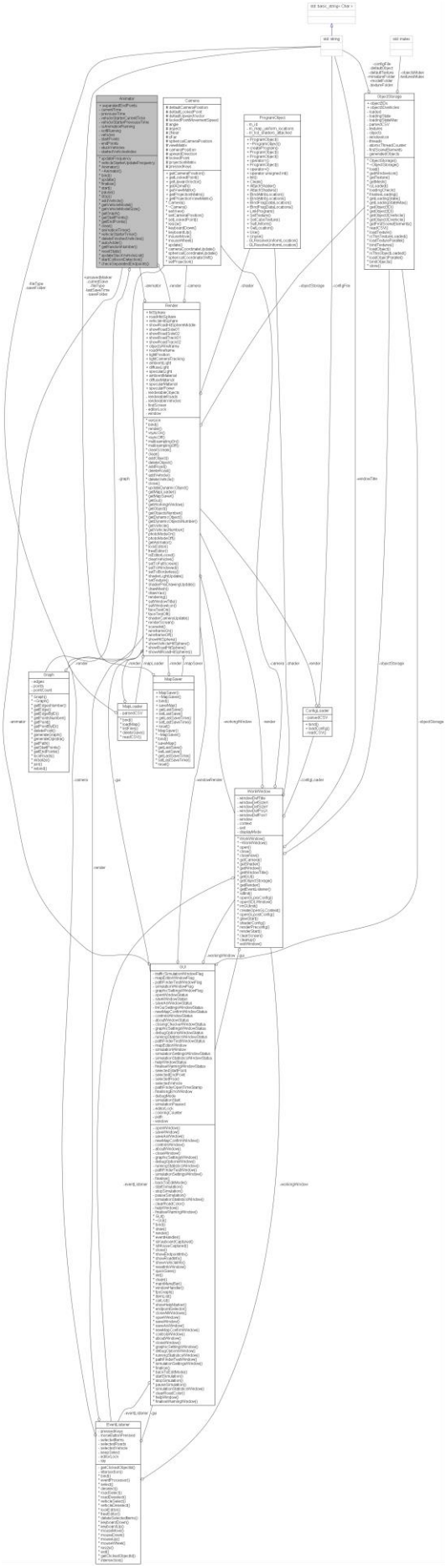
Data Structure Documentation

Animator Class Reference

The animation controller class.

```
#include <Animator.h>
```

Collaboration diagram for Animator:



Data Fields

- std::map< size_t, std::set< size_t > > **separatedEndPoints**
  The separated endpoints.

Private Attributes

- **Render** * **render** = NULL
  Pointer for the current active render.

- **Graph** * **graph** = NULL
  Pointer for the current active generated graph.

- Uint32 **currentTime** = 0
  The current time for animation timing.

- Uint32 **prevouseTime** = 0
  The previous time for animation timing.

- Uint32 **vehicleStarterCurrentTime** = 0
  The current time for vehicle start timing.

- Uint32 **vehicleStarterPrevouseTime** = 0
  The previous time for vehicle start timing.

- bool **isAnimationRunning** = false
  Running animation flag.

- bool **softRunning** = false
  Soft running flag for one vehicle animation.

- std::vector< **Vehicle** > **vehicles**
  The container for the vehicle models.

- std::vector< size_t > **startPoints**
  The current startpoints from the graph.

- std::vector< size_t > **endPoints**
  The current endpoint from the graph.

- std::set< size_t > **stuckVehicles**
  List of the current stopped vehicles.

- size_t **startedVehiclesIndex** = 0
  The started vehicle counter for the secvential starting.

Traffic Simulation

- static int **updateFrequency** = 20
  Static animator update frequency.

- static int **vehicleStarterUpdateFrequency** = 1000
  Static animator starter update frequency.

- **Animator** (void)
  Animator *constructor.*

- **~Animator** (void)
  Animator *destructor.*

- void **bind** (**Render** ***render**)
  Prepare the ***Animator*** *class.*

- void **update** ()
  The animation update.

- void **finalize** ()
  Prepare the graph representation and all variable for the animation and the vehicle path finding.

- void **start** ()
  Start the animation running.

- void **pause** ()
  Pause the animation.

- void **stop** ()
  Stop the animation running.

- void **addVehicle** (size_t startPoint, bool singleVehicle)
  Vehicle *adder method.*

- **Vehicle** * **getVehicleModel** (size_t modelID)
  Getter for an abstract vehicle.

- size_t **getVehicleModelSize** ()
  Getter for the generated abstract vehicles number.

- **Graph** * **getGraph** ()
  Getter for the generated graph.

- std::vector< size_t > **getStartPoints** ()
  Getter for the current startpoints from the graph.

- std::vector< size_t > **getEndPoints** ()

Getter for the current endpoint from the graph.

- void **clear** ()
  Delete all abstract vehicle.

- bool **animationTimer** ()
  The animation timer calculator.

- bool **vehicleStarterTimer** ()
  The vehicle adding timer calculator.

- void **deleteFinishedVehicles** ()
  Delete the finished vehicles.

- void **autoAdder** ()
  Timed automate timed secvencial vehicle adder.

- int **getRandomNumber** (int min, int max)
  Random generator.

- void **resetStats** ()
  Reset the statistic counters.

- void **updateStuckVehicleList** ()
  Update the stuck vehicles list.

- bool **startCollisionDetection** ()
  Check the start road to collisions.

- void **checkSeparatedEndpoints** ()
  List the separated endpoints.

Detailed Description

The animation controller class.

The animation controller class. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.12.15.

Constructor & Destructor Documentation

Animator::Animator (void)

Animator constructor.

Empty.

Parameters

Empty.	

Animator::~Animator (void)

Animator destructor.

Empty.

Parameters

Empty.	

Member Function Documentation

void Animator::addVehicle (size_t *startPoint*, bool *singleVehicle*)

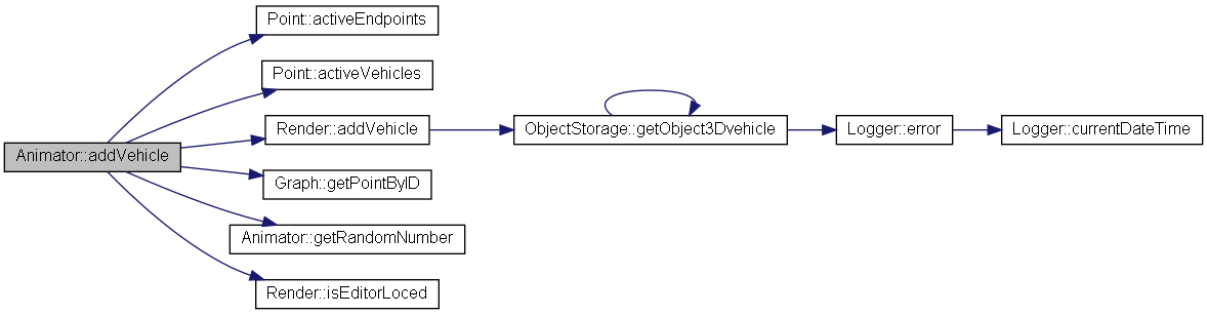
Vehicle adder method.

Create the 3D and the abstract vehicle too.

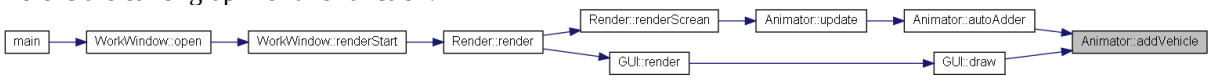
Parameters

startPoint	The startable vehicle start point.
singleVehicle	The startable vehicle destination point.

Here is the call graph for this function:



Here is the caller graph for this function:



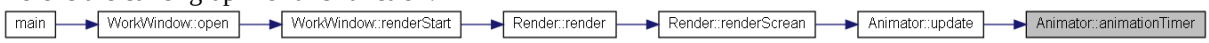
bool Animator::animationTimer () `[private]`

The animation timer calculator.

Returns

The clock tick for the animation timing.

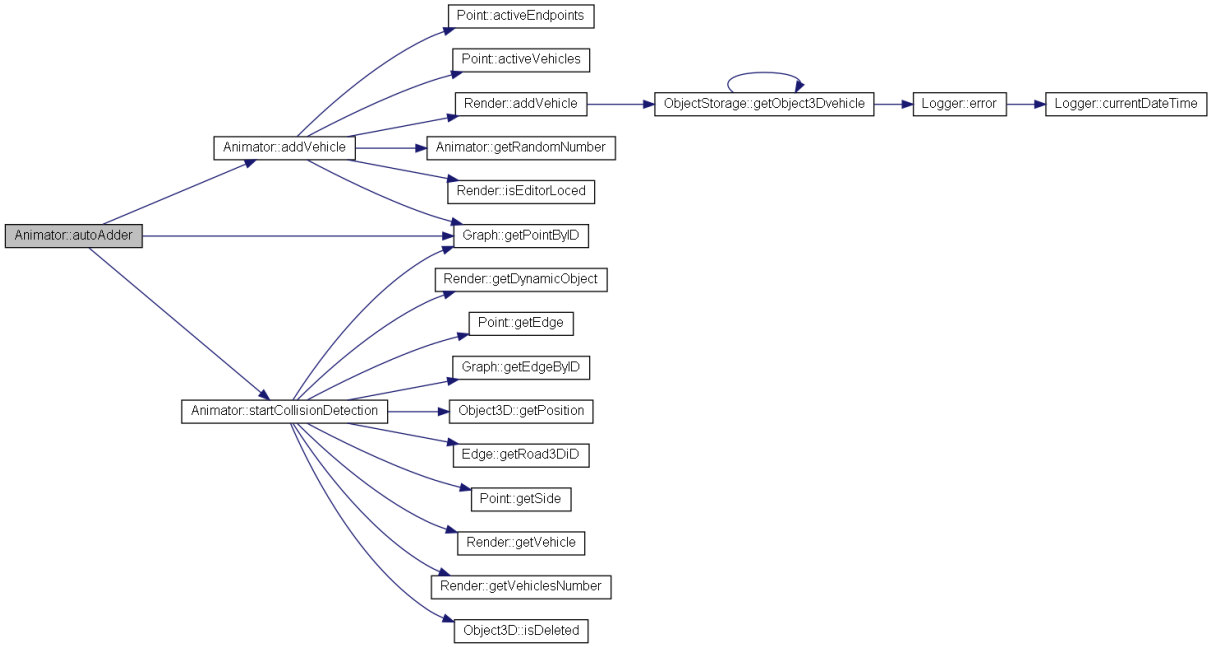
Here is the caller graph for this function:



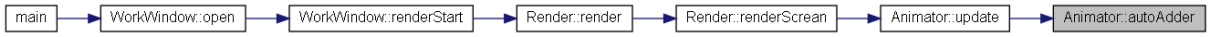
void Animator::autoAdder () [private]

Timed automate timed secvencial vehicle adder.

Here is the call graph for this function:



Here is the caller graph for this function:



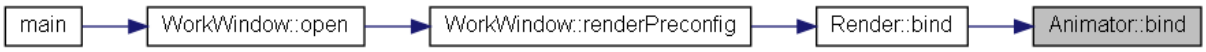
void Animator::bind (Render * *render*)

Prepare the **Animator** class.

Parameters

render	The current render object.

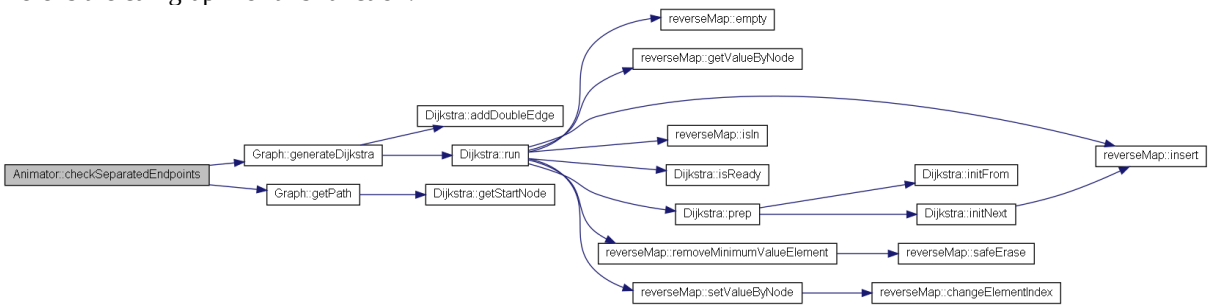
Here is the caller graph for this function:



void Animator::checkSeparatedEndpoints () [private]

List the separated endpoints.

Here is the call graph for this function:



Here is the caller graph for this function:

void Animator::clear ()

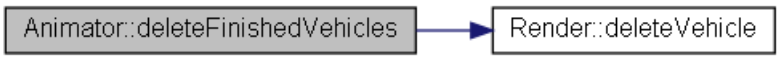
Delete all abstract vehicle.

Here is the caller graph for this function:

void Animator::deleteFinishedVehicles () [private]

Delete the finished vehicles.

Here is the call graph for this function:

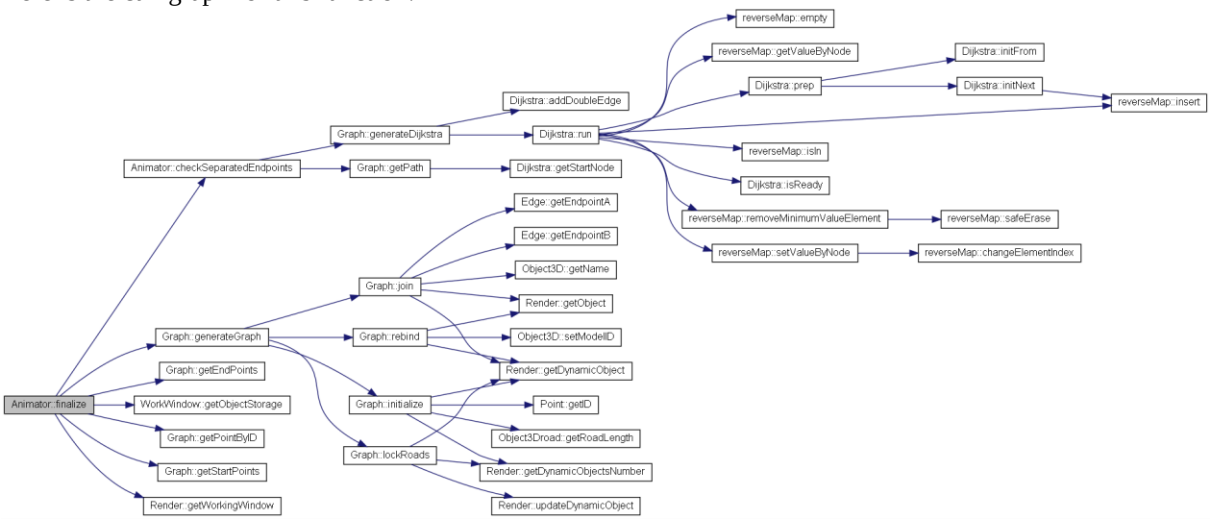


Here is the caller graph for this function:

void Animator::finalize ()

Prepare the graph representation and all variable for the animation and the vehicle path finding.

Here is the call graph for this function:



Here is the caller graph for this function:

std::vector< size_t > Animator::getEndPoints ()

Getter for the current endpoint from the graph.

Graph * Animator::getGraph ()

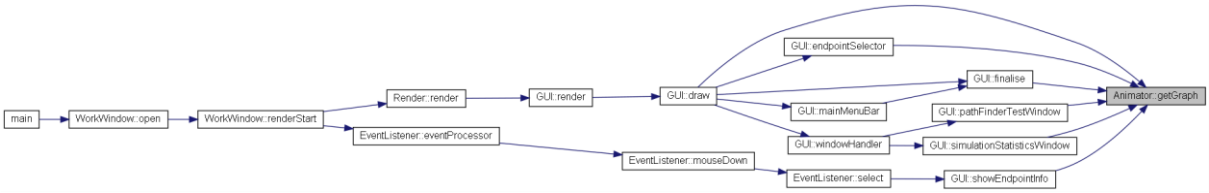
Getter for the generated graph.

The graph generated in the finalize method.

Returns

The generated graph

Here is the caller graph for this function:



int Animator::getRandomNumber (int *min*, int *max*) `[private]`

Random generator.

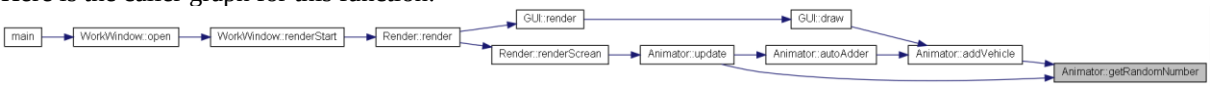
Parameters

min	The needed minimum random number.
max	The needed maximum random number.

Returns

The generated random number.

Here is the caller graph for this function:



std::vector< size_t > Animator::getStartPoints ()

Getter for the current startpoints from the graph.

Vehicle * Animator::getVehicleModel (size_t *modelID*)

Getter for an abstract vehicle.

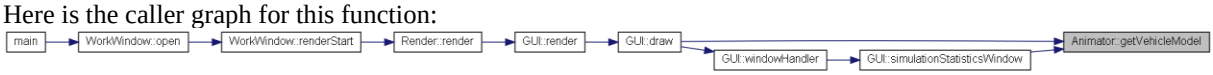
Parameters

modelID	The needed abstract vehicle.

Returns

The abstract vehicle.

Here is the caller graph for this function:



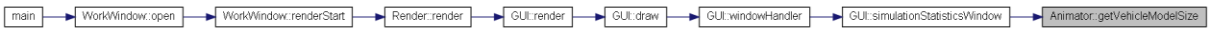
size_t Animator::getVehicleModelSize ()

Getter for the generated abstract vehicles number.

Returns

The generated abstract vehicles number.

Here is the caller graph for this function:



void Animator::pause ()

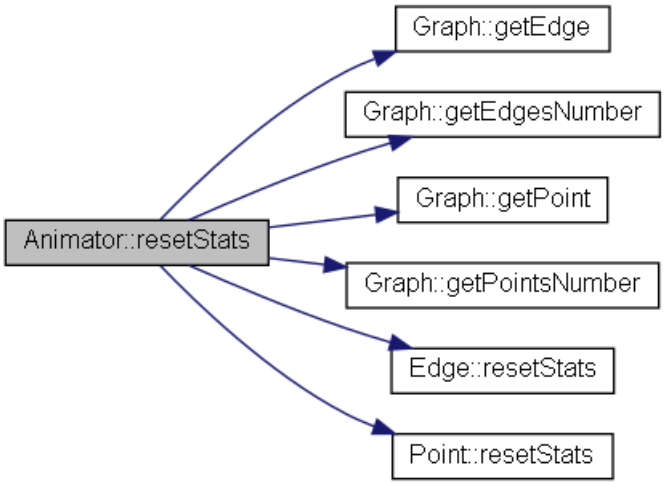
Pause the animation.

Here is the caller graph for this function:

void Animator::resetStats () [private]

Reset the statistic counters.

Here is the call graph for this function:

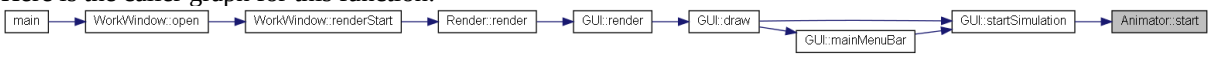


Here is the caller graph for this function:

void Animator::start ()

Start the animation running.

Here is the caller graph for this function:



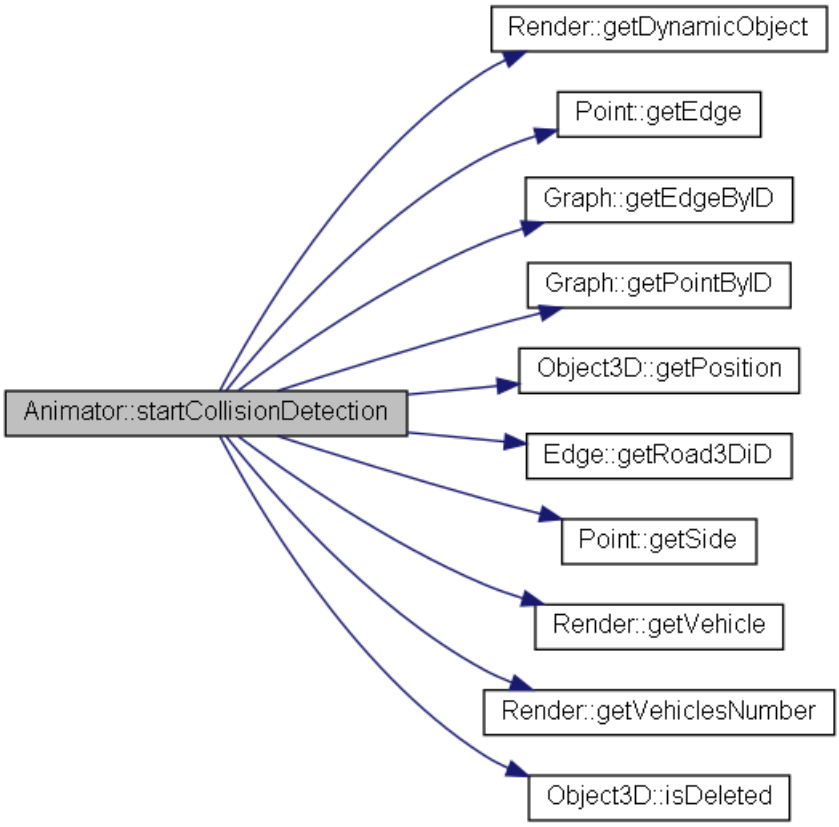
bool Animator::startCollisionDetection () [private]

Check the start road to collisions.

Returns

Return true if the start road is not free.

Here is the call graph for this function:

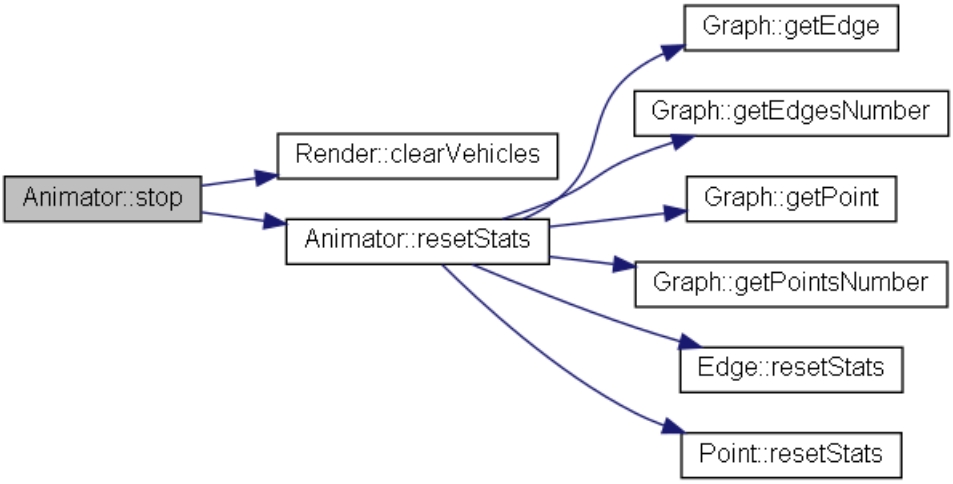


Here is the caller graph for this function:

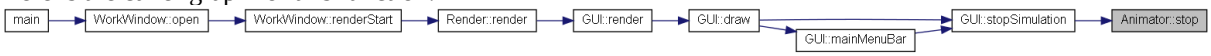
void Animator::stop ()

Stop the animation running.

Here is the call graph for this function:



Here is the caller graph for this function:

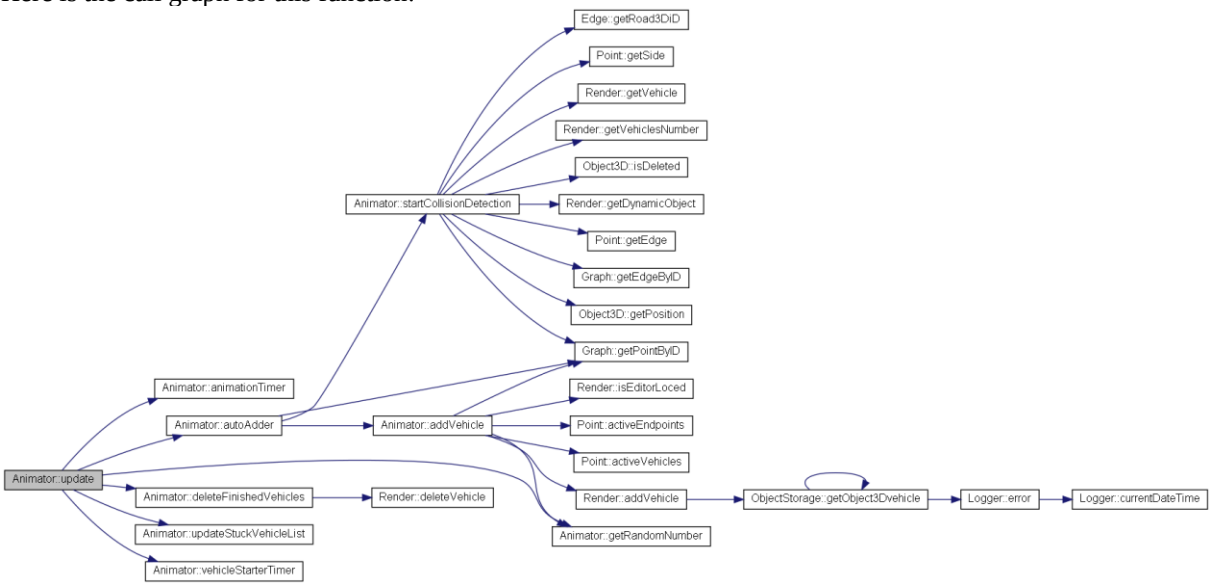


void Animator::update ()

The animation update.

Callable by every frame for the animation update.

Here is the call graph for this function:



Here is the caller graph for this function:

void Animator::updateStuckVehicleList () [private]

Update the stuck vehicles list.

;

Here is the caller graph for this function:

bool Animator::vehicleStarterTimer () [private]

The vehicle adding timer calculator.

Returns

The clock tick for the vehicle start timing.

Here is the caller graph for this function:

Field Documentation

Uint32 Animator::currentTime = 0 [private]

The current time for animation timing.

std::vector<size_t> Animator::endPoints [private]

The current endpoint from the graph.

Graph* Animator::graph = NULL [private]

Pointer for the current active generated graph.

bool Animator::isAnimationRunning = false `[private]`

Running animation flag.

Uint32 Animator::prevouseTime = 0 `[private]`

The previous time for animation timing.

Render* Animator::render = NULL `[private]`

Pointer for the current active render.

std::map<size_t,std::set<size_t> > Animator::separatedEndPoints

The separated endpoints.

bool Animator::softRunning = false `[private]`

Soft running flag for one vehicle animation.

size_t Animator::startedVehiclesIndex = 0 `[private]`

The started vehicle counter for the secvential starting.

std::vector<size_t> Animator::startPoints `[private]`

The current startpoints from the graph.

std::set<size_t> Animator::stuckVehicles `[private]`

List of the current stopped vehicles.

int Animator::updateFrequency = 20 `[static]`

Static animator update frequency.

std::vector<Vehicle> Animator::vehicles `[private]`

The container for the vehicle models.

Uint32 Animator::vehicleStarterCurrentTime = 0 `[private]`

The current time for vehicle start timing.

Uint32 Animator::vehicleStarterPrevouseTime = 0 `[private]`

The previous time for vehicle start timing.

int Animator::vehicleStarterUpdateFrequency = 1000 `[static]`

Static animator starter update frequency.

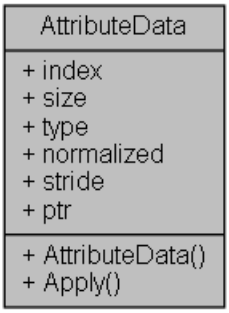
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**Animator.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**Animator.cpp**

AttributeData Struct Reference

`#include <VertexArrayObject.h>`

Collaboration diagram for AttributeData:



Public Member Functions

- **AttributeData** (GLuint pIndex, GLint pSize, GLenum pType, GLboolean pNormalized, GLsizei pStride, void *pPtr)
- void **Apply** ()

Data Fields

- GLuint **index** {}
- GLint **size** {}
- GLenum **type** {}
- GLboolean **normalized** {}
- GLsizei **stride** {}
- void * **ptr** {}

Constructor & Destructor Documentation

AttributeData::AttributeData (GLuint *pIndex*, GLint *pSize*, GLenum *pType*, GLboolean *pNormalized*, GLsizei *pStride*, void * *pPtr*) `[inline]`

Member Function Documentation

void AttributeData::Apply () `[inline]`

Here is the caller graph for this function:

Field Documentation

GLuint AttributeData::index {}

GLboolean AttributeData::normalized {}

void* AttributeData::ptr {}

GLint AttributeData::size {}

GLsizei AttributeData::stride {}

GLenum AttributeData::type {}

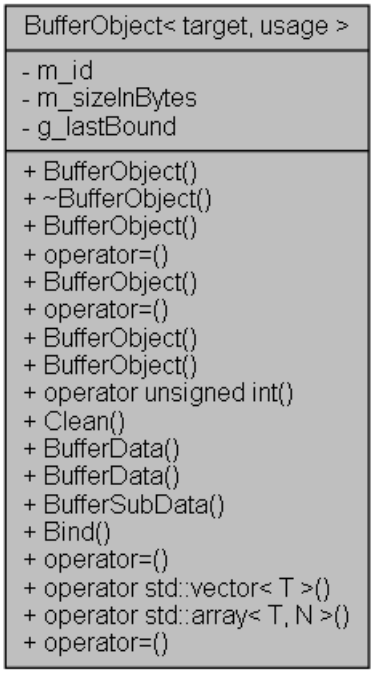
The documentation for this struct was generated from the following file:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**VertexArrayObject.h**

BufferObject< target, usage > Class Template Reference

`#include <BufferObject.h>`

Collaboration diagram for BufferObject< target, usage >:



Public Member Functions

- **BufferObject** ()
- **~BufferObject** ()
- **BufferObject** (const **BufferObject** &)=delete
- **BufferObject** & **operator=** (const **BufferObject** &)=delete
- **BufferObject** (**BufferObject** &&rhs)
- **BufferObject** & **operator=** (**BufferObject** &&rhs)
- template<typename T > **BufferObject** (const std::vector< T > &)
- template<typename T , size_t N> **BufferObject** (const std::array< T, N > &)
- **operator unsigned int** () const
- void **Clean** ()
- template<typename T > **IsContiguousContainer**< T > **BufferData** (const T &pArr)
- void **BufferData** (GLsizeiptr pSize, const GLvoid *pSource=nullptr)
- void **BufferSubData** (GLintptr pOffset, GLsizeiptr pSize, const GLvoid *pSource=nullptr)
- void **Bind** () const
- template<typename T > **BufferObject** & **operator=** (const T &pArr)
- template<typename T > **operator std::vector< T >** () const
- template<typename T , size_t N> **operator std::array< T, N >** () const
- template<typename T > **BufferObject**< target, usage > & **operator=** (const T &pArr)

Private Attributes

- GLuint **m_id** {}
- GLsizeiptr **m_sizeInBytes** {}

Static Private Attributes

- static GLuint **g_lastBound** = 0

Constructor & Destructor Documentation

template<BufferType target, BufferUsage usage> BufferObject< target, usage >::BufferObject `[inline]`

template<BufferType target, BufferUsage usage> BufferObject< target, usage >::~BufferObject `[inline]`

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> BufferObject< target, usage >::BufferObject (const BufferObject< target, usage > &) `[delete]`

template<BufferType target, BufferUsage usage> BufferObject< target, usage >::BufferObject (BufferObject< target, usage > && *rhs*) `[inline]`

template<BufferType target, BufferUsage usage> template<typename T > BufferObject< target, usage >::BufferObject (const std::vector< T > & *pArr*) `[inline], [explicit]`

Here is the call graph for this function:

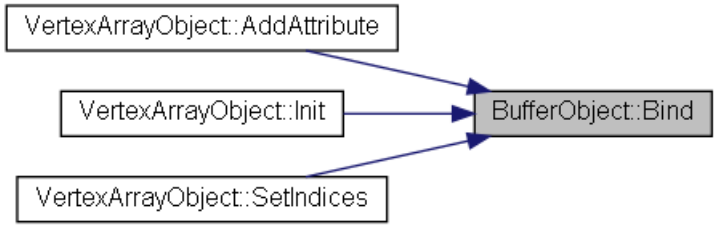
template<BufferType target, BufferUsage usage> template<typename T , size_t N> BufferObject< target, usage >::BufferObject (const std::array< T, N > & *pArr*) `[inline], [explicit]`

Here is the call graph for this function:

Member Function Documentation

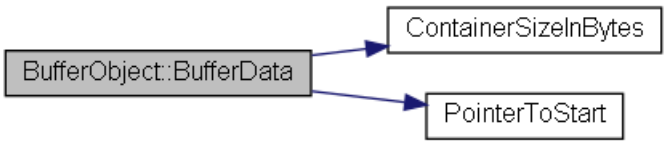
template<BufferType target, BufferUsage usage> void BufferObject< target, usage >::Bind `[inline]`

Here is the caller graph for this function:



template<BufferType target, BufferUsage usage> template<typename T > IsContiguousContainer< T > BufferObject< target, usage >::BufferData (const T & *pArr*) `[inline]`

Here is the call graph for this function:



Here is the caller graph for this function:

template<BufferType target, BufferUsage usage> void BufferObject< target, usage >::BufferData (GLsizeiptr *pSize*, const GLvoid * *pSource* = `nullptr`) `[inline]`

template<BufferType target, BufferUsage usage> void BufferObject< target, usage >::BufferSubData (GLintptr *pOffset*, GLsizeiptr *pSize*, const GLvoid * *pSource* = `nullptr`) `[inline]`

template<BufferType target, BufferUsage usage> void BufferObject< target, usage >::Clean `[inline]`

template<BufferType target, BufferUsage usage> template<typename T , size_t N> BufferObject< target, usage >::operator std::array< T, N > `[inline]`

template<BufferType target, BufferUsage usage> template<typename T > BufferObject< target, usage >::operator std::vector< T > `[inline]`

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> BufferObject< target, usage >::operator unsigned int () const `[inline]`

template<BufferType target, BufferUsage usage> BufferObject< target, usage > & BufferObject< target, usage >::operator= (BufferObject< target, usage > && *rhs*) `[inline]`

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> BufferObject & BufferObject< target, usage >::operator= (const BufferObject< target, usage > &) `[delete]`

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> template<typename T > BufferObject & BufferObject< target, usage >::operator= (const T & *pArr*)

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> template<typename T > BufferObject< target, usage > & BufferObject< target, usage >::operator= (const T & *pArr*) `[inline]`

Here is the call graph for this function:

Field Documentation

template<BufferType target, BufferUsage usage> GLuint BufferObject< target, usage >::g_lastBound = 0 `[static], [private]`

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> GLuint BufferObject< target, usage >::m_id {} `[private]`

template<BufferType target = BufferType::Array, BufferUsage usage = BufferUsage::StreamDraw> GLsizeiptr BufferObject< target, usage >::m_sizeInBytes {} `[private]`

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

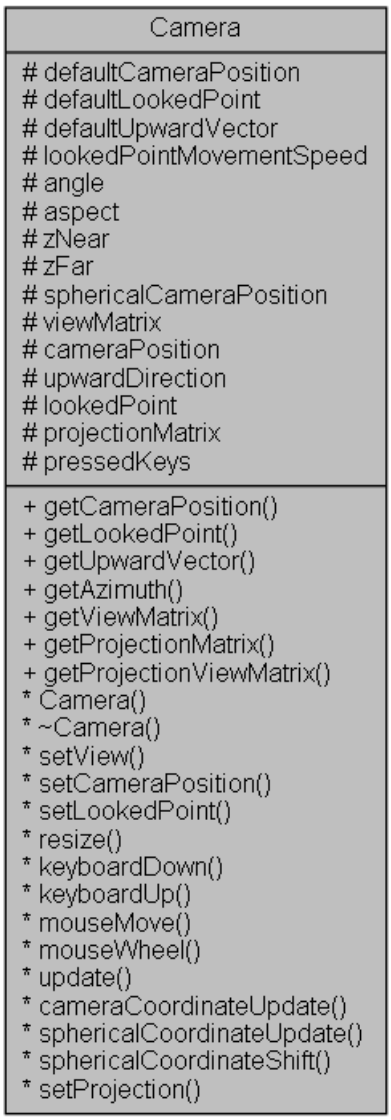
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**BufferObject.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**BufferObject.inl**

Camera Class Reference

The own OpenGL camera class.

```
#include <Camera.h>
```

Collaboration diagram for Camera:



Public Member Functions

- glm::vec3 **getCameraPosition** ()
  Getter for the camera position.

- glm::vec3 **getLookedPoint** ()
  Getter for the cameras looked point.

- glm::vec3 **getUpwardVector** ()
  Getter for the upward direction vector.

- float **getAzimuth** ()

Getter for the forward vector.

- glm::mat4 **getViewMatrix** ()
  Getter for the view matrix.

- glm::mat4 **getProjectionMatrix** ()
  Getter for the projection matrix.

- glm::mat4 **getProjectionViewMatrix** ()
  *The View * Projection matrix used for the camera transformation in the render.*

Protected Attributes

- const glm::vec3 **defaultCameraPosition** = glm::vec3(45.0f, 45.0f, 45.0f)
  The start camera position.

- const glm::vec3 **defaultLookedPoint** = glm::vec3(0.0f)
  The start looked point.

- const glm::vec3 **defaultUpwardVector** = glm::vec3(0.0f, 10.0f, 0.0f)
  The default upward vector.

- float **lookedPointMovementSpeed** = 1.0f
  The looked point movement speed.

- const float **angle** = 45.0f
  The default angle.

- float **aspect** = 640 / 480.0f
  The default angle.

- const float **zNear** = 0.8f
  The default near cutting line.

- const float **zFar** = 350.0f
  The default far cutting line.

- glm::vec3 **sphericalCameraPosition**
  The camera position in spherical coordinate.

- glm::mat4 **viewMatrix**
  The view matrix of the camera.

- glm::vec3 **cameraPosition**
  The camera position.

- glm::vec3 **upwardDirection**

The vector pointing upwards.

- glm::vec3 **lookedPoint**
  The camera look at point.

- glm::mat4 **projectionMatrix**
  The projection matrix.

- std::set< int > **pressedKeys**
  This set store the current pressed keys.

Traffic Simulation.

- **Camera** (void)
  *The **Camera** class constructor.*

- **~Camera** (void)
  *The **Camera** class destructor.*

- void **setView** (glm::vec3 **cameraPosition**, glm::vec3 **lookedPoint**)
  Set the camera position and the looked point.

- void **setCameraPosition** (glm::vec3 **cameraPosition**)
  Adapter function for the SetView.

- void **setLookedPoint** (glm::vec3 **lookedPoint**)
  Adapter function for the SetView.

- void **resize** (int width, int height)
  ***Camera** projectionMatrix update function for the screen resize.*

- void **keyboardDown** (SDL_KeyboardEvent &key)
  Event handler function for the key down event.

- void **keyboardUp** (SDL_KeyboardEvent &key)
  Event handler function for the key up event.

- void **mouseMove** (SDL_MouseMotionEvent &mouse)
  Event handler function for the mouse move event.

- void **mouseWheel** (SDL_MouseWheelEvent &wheel)
  Event handler function for the mouse wheel event.

- void **update** ()
  Update viewMatrix from cameraPosition, lookedPoint and upwardDirection for apply changes.

- void **cameraCoordinateUpdate** ()

Calculate camera coordinates from current spherical coordinates.

Nem jelenítheti meg a képet kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt, hogy a szándéka szerinti fájlra és helyen van.

Nem jelenítheti meg a képet kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt, hogy a szándéka szerinti fájlra és helyen van.

Nem jelenítheti meg a képet kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt, hogy a szándéka szerinti fájlra és helyen van.

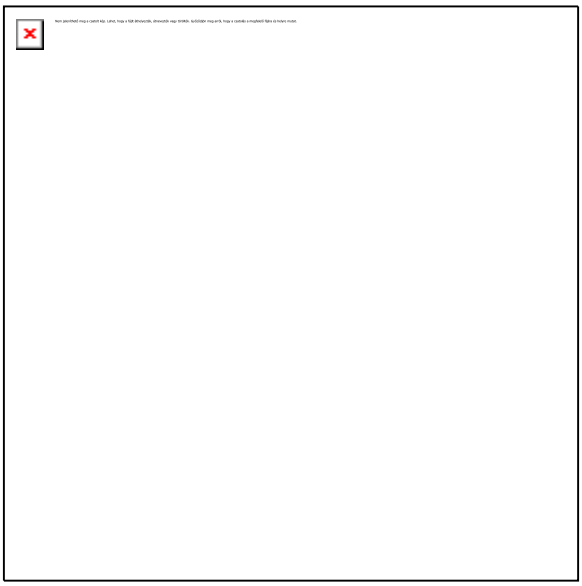
Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

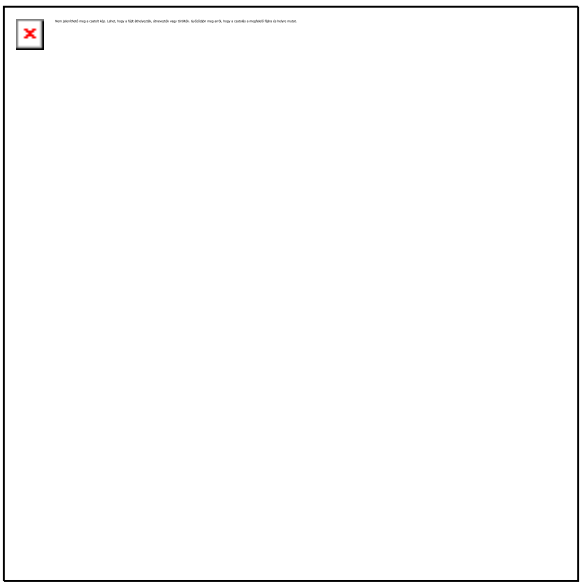
Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

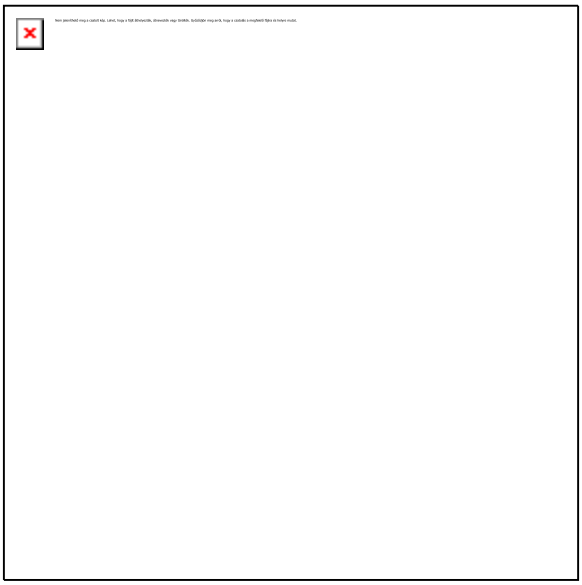
Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

.

- void **sphericalCoordinateUpdate** ()
  Calculate spherical coordinates from current camera coordinates.







.

- void **sphericalCoordinateShift** (float radius, float azimuth, float inclination)
  Shift the spherical coordinates.

- void **setProjection** (float **angle**, float **aspect**, float **zNear**, float **zFar**)
  Calculate the projection matrix for the camera transformation.

Detailed Description

The own OpenGL camera class.

Camera implementation header with spherical coordinates. Contact `sandorbalazs9402@gmail.com`.

Camera implementation file with spherical coordinates. Contact: `sandorbalazs9402@gmail.com`.

The camera movable in a sphere with the mouse and in the scene with the keyboard.

Author

Sándor Balázs - AZA6NL

Date

2021.05.02.

Author

Sándor Balázs - AZA6NL

Date

2021.05.01.

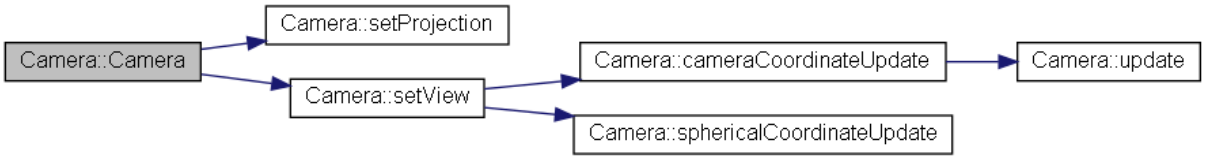
Constructor & Destructor Documentation

Camera::Camera (void)

The **Camera** class constructor.

Initialize the variables, set the view and the projection matrix.

Here is the call graph for this function:



Camera::~Camera (void)

The **Camera** class destructor.

Don't have function at the moment.

Member Function Documentation

void Camera::cameraCoordinateUpdate () [protected]

Calculate camera coordinates from current spherical coordinates.

Nem található meg a csatolt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a csatolás a megfelelő fájlra és helyre mutat.

Nem található meg a csatolt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a csatolás a megfelelő fájlra és helyre mutat.

Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

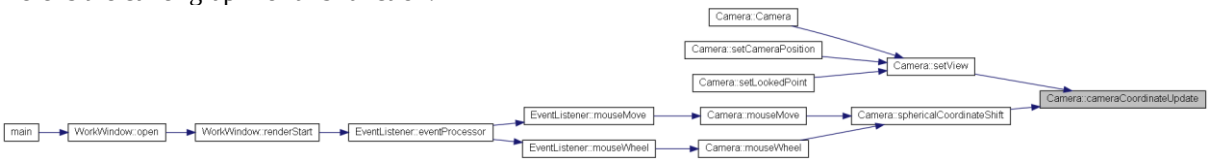
.

Attention

The camera z axis show upward!

Here is the call graph for this function:

Here is the caller graph for this function:



float Camera::getAzimuth () `[inline]`

Getter for the forward vector.

(**Camera** from looked point)

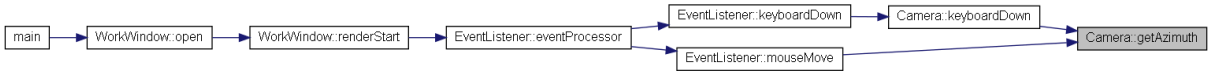
Returns

The forward vector.

glm::vec3 getForwardVector() { return lookedPoint-cameraPosition; }

Returns

Here is the caller graph for this function:



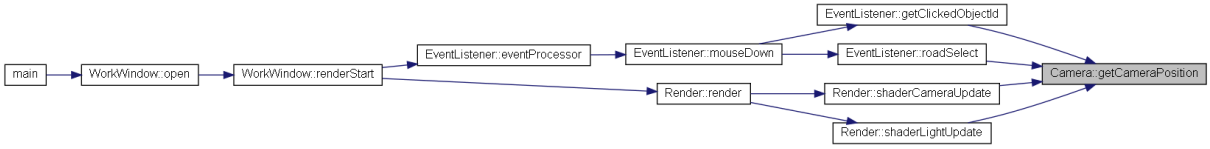
glm::vec3 Camera::getCameraPosition () `[inline]`

Getter for the camera position.

Returns

The current camera position

Here is the caller graph for this function:



glm::vec3 Camera::getLookedPoint () `[inline]`

Getter for the cameras looked point.

Returns

The current looked point.

glm::mat4 Camera::getProjectionMatrix () `[inline]`

Getter for the projection matrix.

Returns

The current projection matrix.

Here is the caller graph for this function:

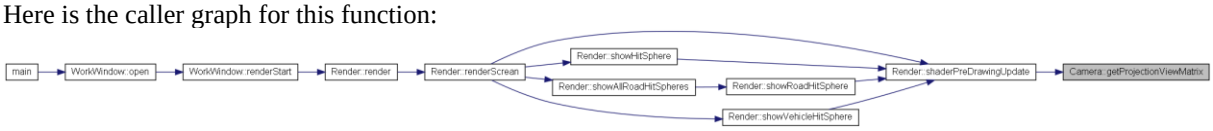
glm::mat4 Camera::getProjectionViewMatrix () `[inline]`

The View * Projection matrix used for the camera transformation in the render.

Returns

The product of the view and the projection matrix.

Here is the caller graph for this function:



glm::vec3 Camera::getUpwardVector () `[inline]`

Getter for the upward direction vector.

Returns

The current upward vector.

glm::mat4 Camera::getViewMatrix () `[inline]`

Getter for the view matrix.

Returns

The current view matrix.

Here is the caller graph for this function:

void Camera::keyboardDown (SDL_KeyboardEvent & *key*)

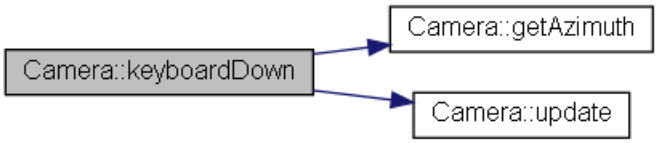
Event handler function for the key down event.

Store the pressed keys in the pressedKeys set. (The keyboard move direction is set by camera direction.)

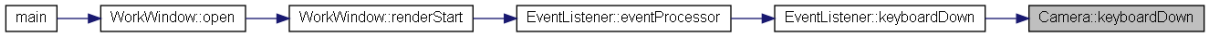
Parameters

key	keyboard down event.

Here is the call graph for this function:



Here is the caller graph for this function:



void Camera::keyboardUp (SDL_KeyboardEvent & *key*)

Event handler function for the key up event.

Erase the key from the pressedKeys set.

Parameters

key	keyboard up event.

Here is the caller graph for this function:

void Camera::mouseMove (SDL_MouseMotionEvent & *mouse*)

Event handler function for the mouse move event.

Parameters

key	mouse move event.

Here is the call graph for this function:

Here is the caller graph for this function:

void Camera::mouseWheel (SDL_MouseWheelEvent & *wheel*)

Event handler function for the mouse wheel event.

Parameters

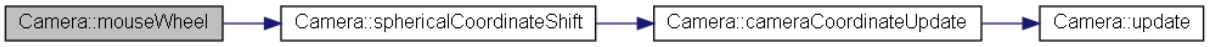
key	mouse wheel event.

(Logarithmic mouse wheeling)

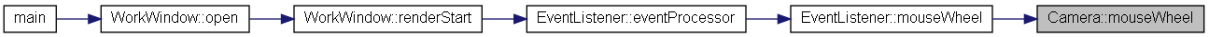
Parameters

key	mouse wheel event.

Here is the call graph for this function:



Here is the caller graph for this function:



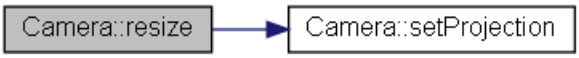
void Camera::resize (int *width*, int *height*)

Camera projectionMatrix update function for the screen resize.

Parameters

width	The OpenGL screen width.
height	The OpenGL screen height.

Here is the call graph for this function:



Here is the caller graph for this function:

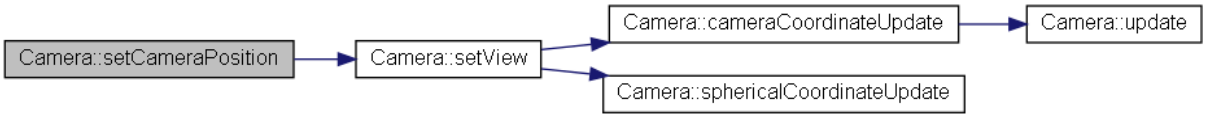
void Camera::setCameraPosition (glm::vec3 *camPos*)

Adapter function for the SetView.

Parameters

cameraPosition	The settable camera position.

Here is the call graph for this function:



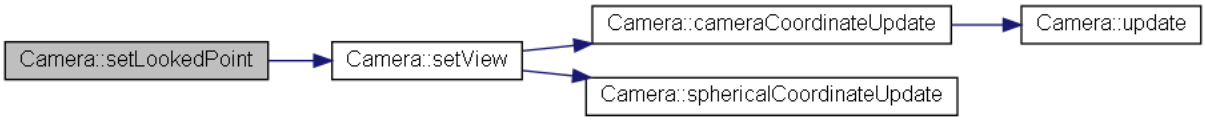
void Camera::setLookedPoint (glm::vec3 *lookPoint*)

Adapter function for the SetView.

Parameters

lookedPoint	The settable looked point.

Here is the call graph for this function:



void Camera::setProjection (float *angle*, float *aspect*, float *zNear*, float *zFar*) `[protected]`

Calculate the projection matrix for the camera transformation.

Parameters

angle	The view pyramid angle.
aspect	Te view aspect.
zNear	Near cutting plane.

zFar	Far cutting plane.

Here is the caller graph for this function:

void Camera::setView (glm::vec3 *camPos*, glm::vec3 *lookPoint*)

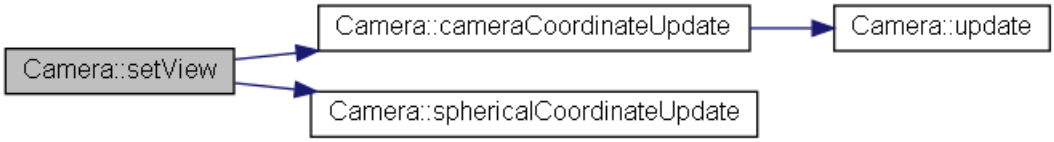
Set the camera position and the looked point.

Generate the spherical coordinates from the sphere position and update the camera position.

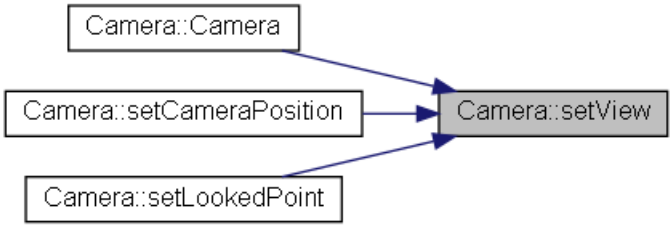
Parameters

cameraPosition	The settable camera position.
lookedPoint	The settable looked point.

Here is the call graph for this function:



Here is the caller graph for this function:



void Camera::sphericalCoordinateShift (float *radius*, float *azimuth*, float *inclination*) `[protected]`

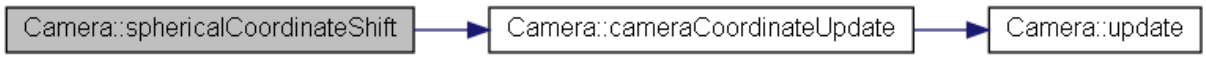
Shift the spherical coordinates.

(with limitations)

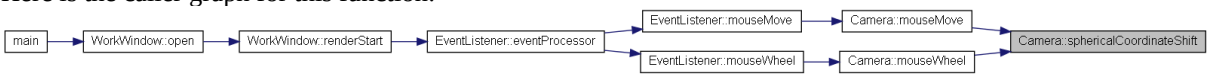
Parameters

radius	The spherical coordinates radius;
azimuth	The spherical coordinates azimuth;
inclination	The spherical coordinates inclination;

Here is the call graph for this function:



Here is the caller graph for this function:



void Camera::sphericalCoordinateUpdate () `[protected]`

Calculate spherical coordinates from current camera coordinates.

Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

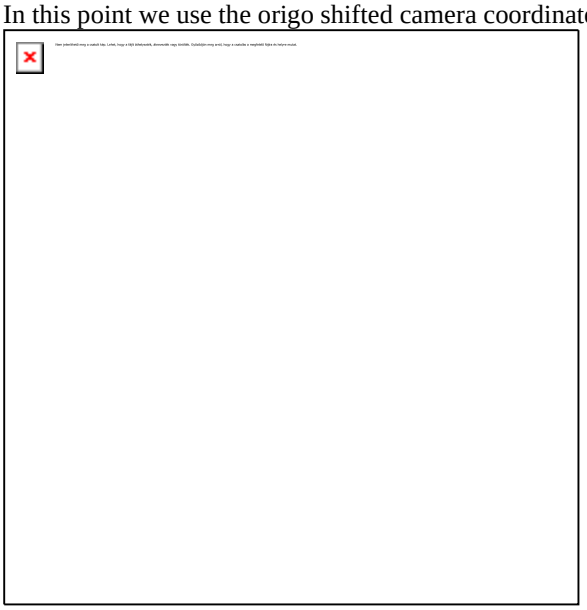
Nem jeleníthető meg a vaskelt kép. Lehet, hogy a fájl áthelyezték, átnevezték vagy törölték. Ellenőrizze meg azt is, hogy a vaskelés a megfelelő fájlra és helyre mutat.

.

Attention

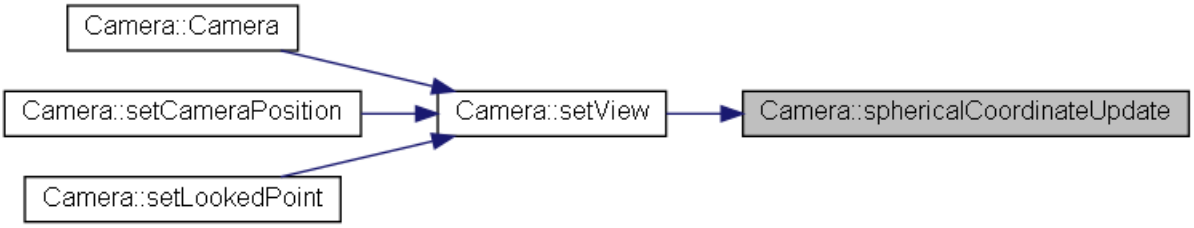
The camera z axis show upward!

In this point we use the origo shifted camera coordinates



Built in zero division prevention.

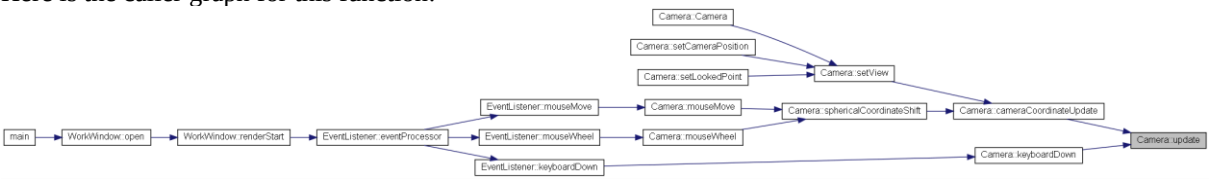
Here is the caller graph for this function:



void Camera::update () [protected]

Update viewMatrix from cameraPosition, lookedPoint and upwardDirection for apply changes.

Here is the caller graph for this function:



Field Documentation

const float Camera::angle = 45.0f [protected]

The default angle.

float Camera::aspect = 640 / 480.0f [protected]

The default angle.

glm::vec3 Camera::cameraPosition `[protected]`

The camera position.

const glm::vec3 Camera::defaultCameraPosition = glm::vec3(45.0f, 45.0f, 45.0f) `[protected]`

The start camera position.

const glm::vec3 Camera::defaultLookedPoint = glm::vec3(0.0f) `[protected]`

The start looked point.

const glm::vec3 Camera::defaultUpwardVector = glm::vec3(0.0f, 10.0f, 0.0f) `[protected]`

The default upward vector.

glm::vec3 Camera::lookedPoint `[protected]`

The camera look at point.

float Camera::lookedPointMovementSpeed = 1.0f `[protected]`

The looked point movement speed.

std::set<int> Camera::pressedKeys `[protected]`

This set store the current pressed keys.

glm::mat4 Camera::projectionMatrix `[protected]`

The projection matrix.

glm::vec3 Camera::sphericalCameraPosition `[protected]`

The camera position in spherical coordinate.

Ilyen jelenítheti meg a szabó kép. Lehet, hogy a fájlt áttekintenék, átnevezték vagy törölték. Ellenőrizze meg azt, hogy a szabály a megfelelő fájlra és helyre mutat.

Ilyen jelenítheti meg a szabó kép. Lehet, hogy a fájlt áttekintenék, átnevezték vagy törölték. Ellenőrizze meg azt, hogy a szabály a megfelelő fájlra és helyre mutat.

Ilyen jelenítheti meg a szabó kép. Lehet, hogy a fájlt áttekintenék, átnevezték vagy törölték. Ellenőrizze meg azt, hogy a szabály a megfelelő fájlra és helyre mutat.

glm::vec3 Camera::upwardDirection `[protected]`

The vector pointing upwards.

glm::mat4 Camera::viewMatrix `[protected]`

The view matrix of the camera.

const float Camera::zFar = 350.0f `[protected]`

The default far cutting line.

const float Camera::zNear = 0.8f `[protected]`

The default near cutting line.

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

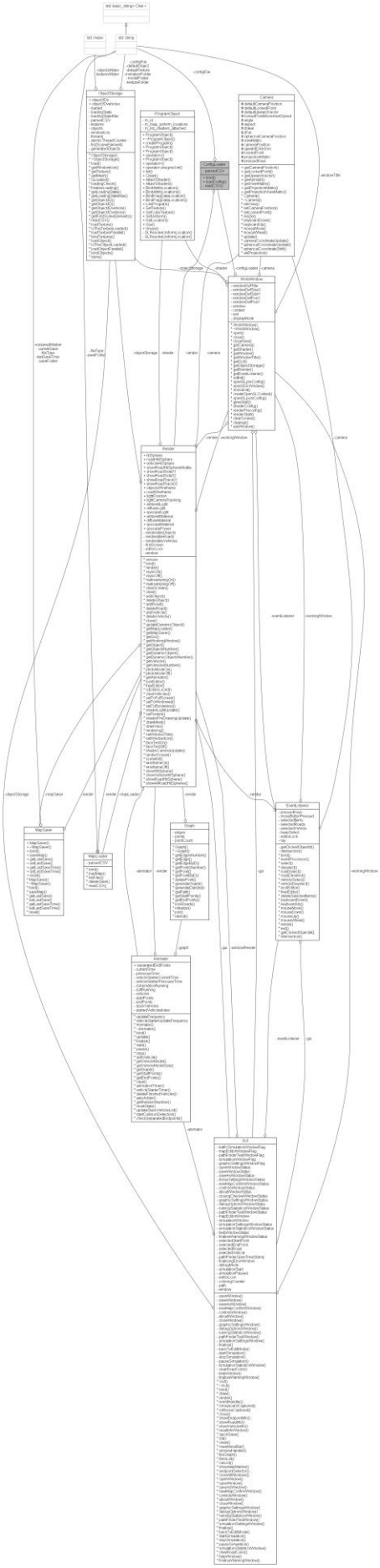
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**Camera.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**Camera.cpp**

ConfigLoader Class Reference

The configuration loader class.

```
#include <ConfigLoader.h>
```

Collaboration diagram for ConfigLoader:



Public Member Functions

- void **bind** (**Render** ***render**)
  Bind the configuration logger for the render.

- void **loadConfig** ()
  Load and apply the configurations.

Private Member Functions

- void **readCSV** (std::string fileName)
  Read and parse the configuration CSV file.

Private Attributes

- std::string **configFile** = "config.csv"
  The config file location.

- **Render** * **render** = NULL
  The render pointer.

- std::vector< std::vector< std::string > > **parsedCSV**
  The parsed CSV configuration file in the memory.

Detailed Description

The configuration loader class.

The configuration loader class. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Member Function Documentation

void ConfigLoader::bind (Render * *render*)

Bind the configuration logger for the render.

Parameters

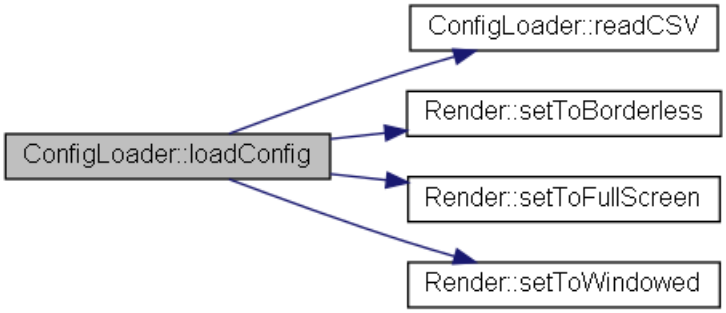
render	The current render pointer.

Here is the caller graph for this function:

void ConfigLoader::loadConfig ()

Load and apply the configurations.

Here is the call graph for this function:



Here is the caller graph for this function:

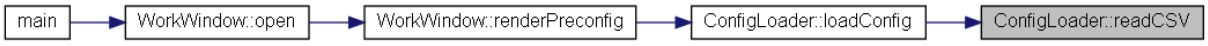
void ConfigLoader::readCSV (std::string *fileName*) `[private]`

Read and parse the configuration CSV file.

Parameters

fileName	The configuration file name.

Here is the caller graph for this function:



Field Documentation

std::string ConfigLoader::configFile = "config.csv" `[private]`

The config file location.

std::vector<std::vector<std::string> > ConfigLoader::parsedCSV `[private]`

The parsed CSV configuration file in the memory.

Render* ConfigLoader::render = NULL `[private]`

The render pointer.

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

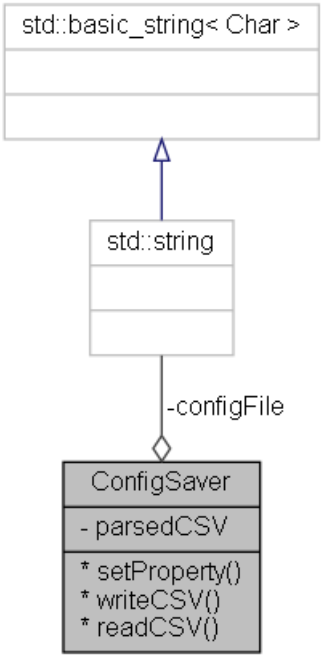
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**ConfigLoader.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**ConfigLoader.cpp**

ConfigSaver Class Reference

Configuration saver class.

```
#include <ConfigSaver.h>
```

Collaboration diagram for ConfigSaver:



Private Attributes

- std::string **configFile** = "config.csv"
  The configuration file path and name.

- std::vector< std::vector< std::string > > **parsedCSV**
  The parsed CSV file in the memory.

Traffic Simulation

- void **setProperty** (std::string propertyName, std::string value)
  Set and save a configuration property.

- void **writeCSV** (std::string fileName)
  Write out the configuration data.

- void **readCSV** (std::string fileName)
  Read in the configuration data.

Detailed Description

Configuration saver class.

Configuration Saver class Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Member Function Documentation

void ConfigSaver::readCSV (std::string *fileName*) `[private]`

Read in the configuration data.

Parameters

fileName	The configuration file name.

Here is the caller graph for this function:

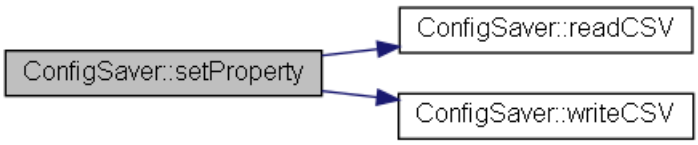
void ConfigSaver::setProperty (std::string *propertyName*, std::string *value*)

Set and save a configuration property.

Parameters

propertyName	The settable property name.
value	The settable property value.

Here is the call graph for this function:



Here is the caller graph for this function:

void ConfigSaver::writeCSV (std::string *fileName*) `[private]`

Write out the configuration data.

Parameters

fileName	The configuration file name.

Here is the caller graph for this function:

Field Documentation

std::string ConfigSaver::configFile = "config.csv" `[private]`

The configuration file path and name.

std::vector<std::vector<std::string> > ConfigSaver::parsedCSV `[private]`

The parsed CSV file in the memory.

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

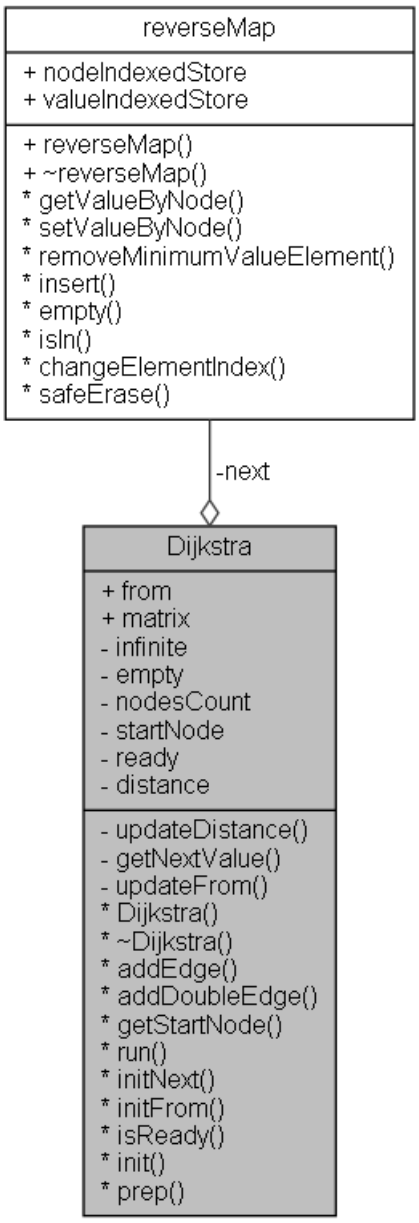
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**ConfigSaver.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**ConfigSaver.cpp**

Dijkstra Class Reference

Dijkstra algorithm implementation.

```
#include <Dijkstra.h>
```

Collaboration diagram for Dijkstra:



Data Fields

- std::vector< size_t > **from**
  The algorithm result matrix.

- std::vector< std::vector< int > > **matrix**
  The graph representation matrix.

Private Member Functions

- void **updateDistance** (size_t node)
  Update the distance.

- size_t **getNextValue** (size_t node)
  Getter for the next value.

- void **updateFrom** (size_t node)
  Update from.

Private Attributes

- const int **infinite** = INT_MAX - 1
  Infinite mark.

- const int **empty** = -1
  Empty mark.

- size_t **nodesCount**
  Node count.

- size_t **startNode**
  Start node.

- std::set< size_t > **ready**
  Ready nodes list.

- std::vector< int > **distance**
  Distances list.

- **reverseMap next**
  The next reverse map.

Traffic Simulation

- **Dijkstra** (size_t **nodesCount**, size_t **startNode**)
  Dijkstra *constructor.*

- **~Dijkstra** ()
  Dijkstra *destructor.*

- void **addEdge** (size_t fromNode, size_t toNode, int weight)
  Adding a new edge to the matrix.

- void **addDoubleEdge** (size_t node1, size_t node2, int weight)
  Adding a new double edge to the matrix.

- size_t **getStartNode** ()
  Getter for the start node.

- void **run** ()
  Run the algorithm and generate the result vector.

- void **initNext** (size_t **startNode**)
  Initialize the next node.

- void **initFrom** (size_t **startNode**)
  Initialize from.

- bool **isReady** (size_t node)
  Getter for the ready state.

- void **init** ()
  Initialize all node and data structures.

- void **prep** ()
  Preparation.

Detailed Description

Dijkstra algorithm implementation.

Dijkstra algorithm implementation. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

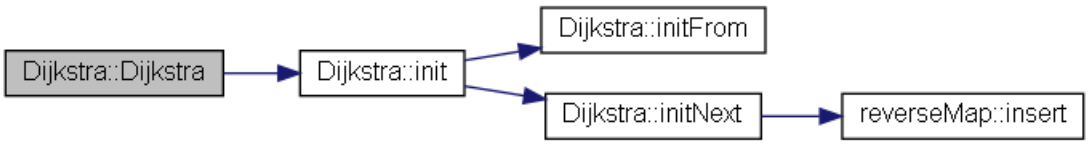
Dijkstra::Dijkstra (size_t *nodesCount*, size_t *startNode*)

Dijkstra constructor.

Parameters

nodesCount	The graph node count.
startNode	The graph start node.

Here is the call graph for this function:



Dijkstra::~Dijkstra ()

Dijkstra destructor.

Member Function Documentation

void Dijkstra::addDoubleEdge (size_t *node1*, size_t *node2*, int *weight*)

Adding a new double edge to the matrix.

Parameters

fromNode	The new edge start node.
toNode	The new edge end node.
weight	The new edge weight.

Here is the caller graph for this function:

void Dijkstra::addEdge (size_t *fromNode*, size_t *toNode*, int *weight*)

Adding a new edge to the matrix.

Parameters

fromNode	The new edge start node.
toNode	The new edge end node.
weight	The new edge weight.

size_t Dijkstra::getNextValue (size_t *node*) `[private]`

Getter for the next value.

Parameters

node	The node.

Returns

The next value.

size_t Dijkstra::getStartNode ()

Getter for the start node.

Returns

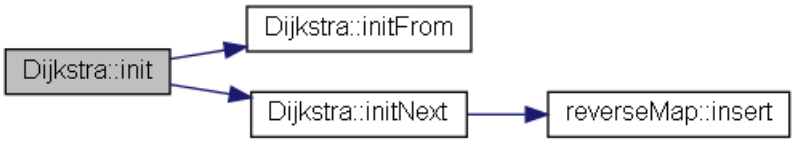
The start node id.

Here is the caller graph for this function:

void Dijkstra::init () `[private]`

Initialize all node and data structures.

Here is the call graph for this function:



Here is the caller graph for this function:

void Dijkstra::initFrom (size_t *startNode*) `[private]`

Initialize from.

Parameters

startNode	Initializeable node.

Here is the caller graph for this function:

void Dijkstra::initNext (size_t *startNode*) `[private]`

Initialize the next node.

Parameters

startNode	The Initializeable node.

Here is the call graph for this function:

Here is the caller graph for this function:

bool Dijkstra::isReady (size_t *node*) `[private]`

Getter for the ready state.

Parameters

node	The node.

Returns

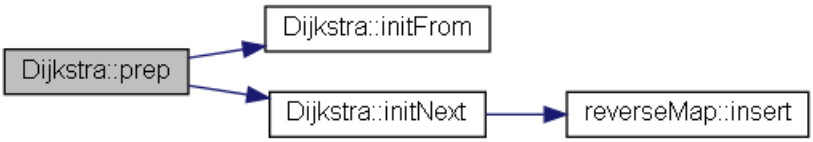
Is the node ready?

Here is the caller graph for this function:

void Dijkstra::prep () `[private]`

Preparation.

Here is the call graph for this function:

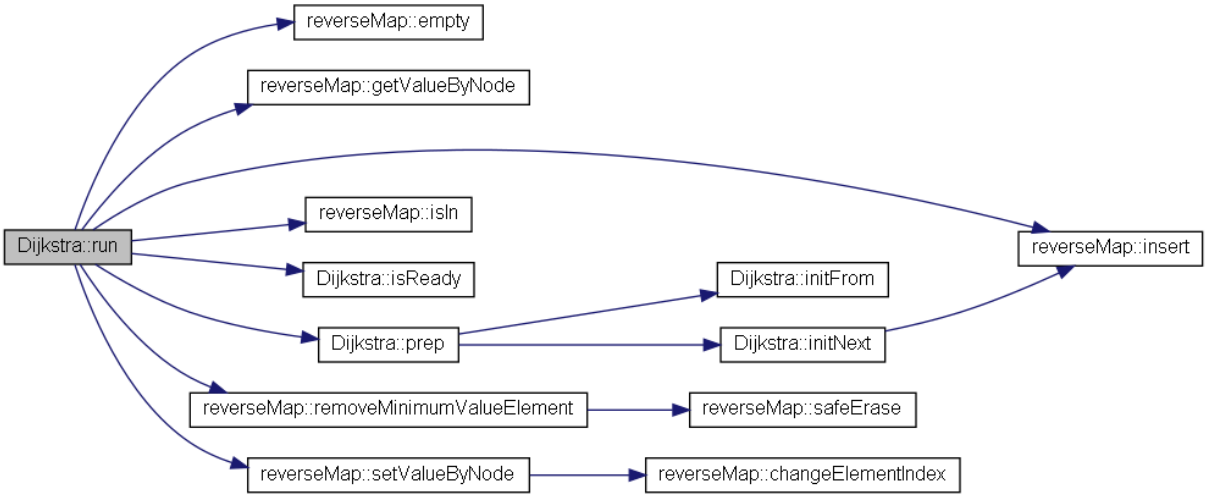


Here is the caller graph for this function:

void Dijkstra::run ()

Run the algorithm and generate the result vector.

Here is the call graph for this function:



Here is the caller graph for this function:

void Dijkstra::updateDistance (size_t *node*) `[private]`

Update the distance.

Parameters

node	The updatble node.

void Dijkstra::updateFrom (size_t *node*) `[private]`

Update from.

Parameters

node	The updatable node.

Field Documentation

std::vector<int> Dijkstra::distance [private]

Distances list.

const int Dijkstra::empty = -1 [private]

Empty mark.

std::vector<size_t> Dijkstra::from

The algorithm result matrix.

const int Dijkstra::infinite = INT_MAX - 1 [private]

Infinite mark.

std::vector<std::vector<int> > Dijkstra::matrix

The graph representation matrix.

reverseMap Dijkstra::next [private]

The next reverse map.

size_t Dijkstra::nodesCount [private]

Node count.

std::set<size_t> Dijkstra::ready [private]

Ready nodes list.

size_t Dijkstra::startNode [private]

Start node.

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

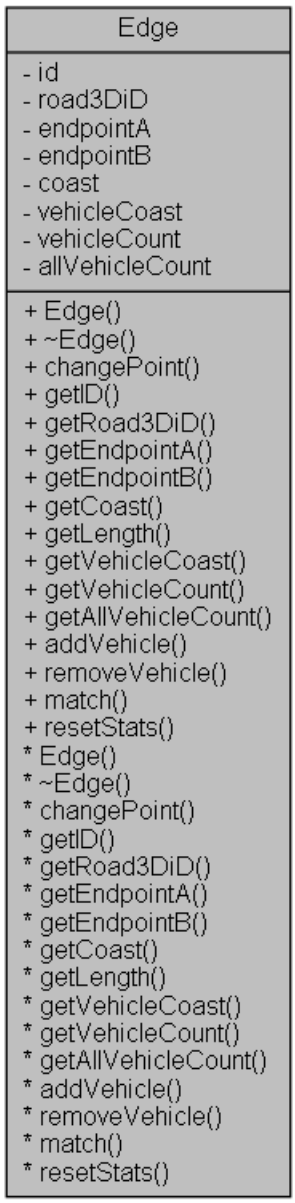
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Dijkstra.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Dijkstra.cpp**

Edge Class Reference

The abstract **Edge** implementation.

```
#include <Edge.h>
```

Collaboration diagram for Edge:



Public Member Functions

- Traffic Simulation**Edge** (size_t **id**, size_t **road3DiD**, size_t **endpointA**, size_t **endpointB**, int **coast**)
  Edge *constructor.*

- **~Edge** ()
  Edge *destructor.*

- void **changePoint** (size_t originalPoint, size_t newPoint)

Change point.

- size_t **getID** ()
  Getter for the vector ID.

- size_t **getRoad3DiD** ()
  Getter for the render ID.

- size_t **getEndpointA** ()
  Getter for the Endpoint A vector ID.

- size_t **getEndpointB** ()
  Getter for the Endpoint B vector ID.

- int **getCoast** ()
  Getter for the cost.

- int **getLength** ()
  Getter for the length.

- int **getVehicleCoast** ()
  Getter for the vehicle cost.

- size_t **getVehicleCount** ()
  Getter for the vehicle count.

- size_t **getAllVehicleCount** ()
  Getter for the all vehicle count.

- void **addVehicle** (int c, bool repath)
  Add vehicle to the edge statistic.

- void **removeVehicle** (int c, bool repath)
  Remove vehicle to the edge statistic.

- bool **match** (size_t **endpointA**, size_t **endpointB**)
  This edges are parallel?

- void **resetStats** ()
  Resetting the edge statistic.

Private Attributes

- size_t **id** = -1
  The vector ID.

- size_t **road3DiD** = -1

The render ID.

- size_t **endpointA** = -1
  The edge endpoint A vector ID.

- size_t **endpointB** = -1
  The edge endpoint B vector ID.

- int **coast** = 0
  The cost.

- int **vehicleCoast** = 0
  The current vehicle cost.

- size_t **vehicleCount** = 0
  The current vehicle cost.

- size_t **allVehicleCount** = 0
  The all vehicle cost.

Detailed Description

The abstract **Edge** implementation.

The abstract graph **Edge** class. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.20.

Constructor & Destructor Documentation

Edge::Edge (size_t *id*, size_t *road3DiD*, size_t *endpointA*, size_t *endpointB*, int *coast*)

Edge constructor.

Parameters

id	The vector id.
road3DiD	The render ID.
endpointA	The endpoint A ID.
endpointB	The endpoint B ID.
coast	The road cost.

Edge::~Edge ()

Edge destructor.

Member Function Documentation

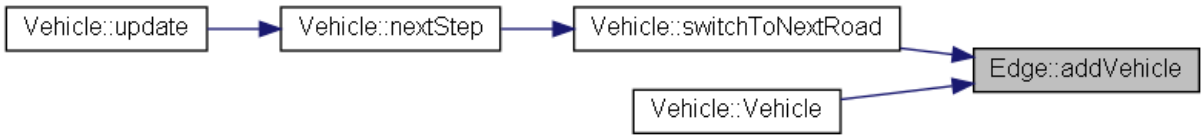
void Edge::addVehicle (int *c*, bool *repath*)

Add vehicle to the edge statistic.

Parameters

The	cost.
repath	Repath is working?

Here is the caller graph for this function:



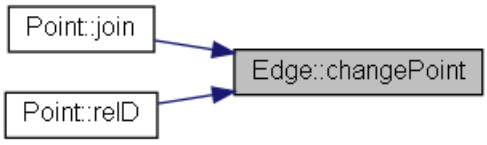
void Edge::changePoint (size_t *originalPoint*, size_t *newPoint*)

Change point.

Parameters

originalPoint	Original point.
newPoint	New point.

Here is the caller graph for this function:



size_t Edge::getAllVehicleCount ()

Getter for the all vehicle count.

Returns

The all vehicle count.

Here is the caller graph for this function:

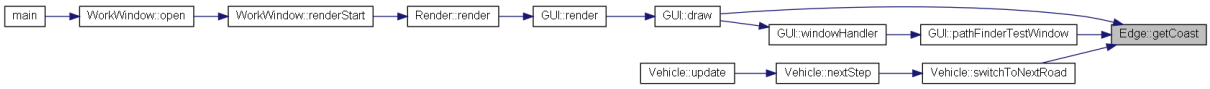
int Edge::getCoast ()

Getter for the cost.

Returns

The cost.

Here is the caller graph for this function:



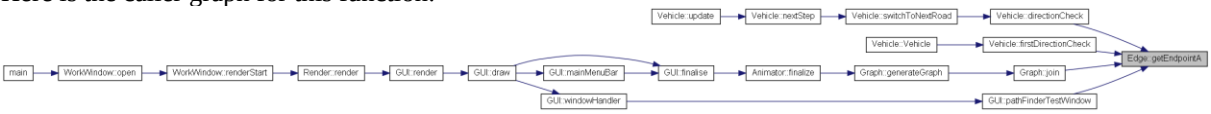
size_t Edge::getEndpointA ()

Getter for the Endpoint A vector ID.

Returns

The Endpoint A vector ID.

Here is the caller graph for this function:



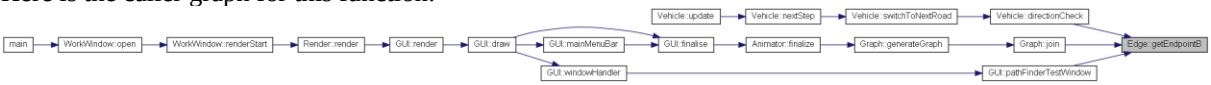
size_t Edge::getEndpointB ()

Getter for the Endpoint B vector ID.

Returns

The Endpoint B vector ID.

Here is the caller graph for this function:



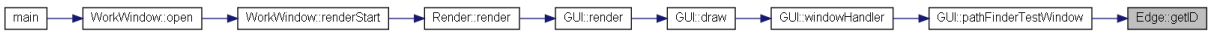
size_t Edge::getID ()

Getter for the vector ID.

Returns

The vector ID.

Here is the caller graph for this function:



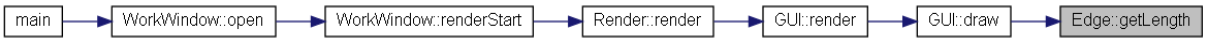
int Edge::getLength ()

Getter for the length.

Returns

The current legth.

Here is the caller graph for this function:



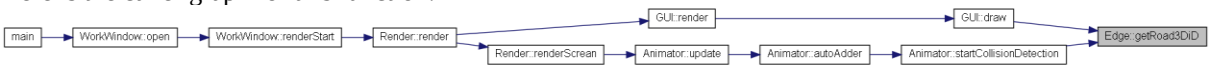
size_t Edge::getRoad3DiD ()

Getter for the render ID.

Returns

The render ID.

Here is the caller graph for this function:



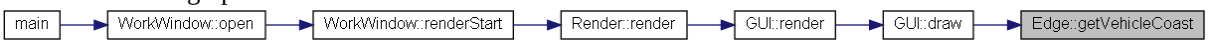
int Edge::getVehicleCoast ()

Getter for the vehicle cost.

Returns

The vehicle cost.

Here is the caller graph for this function:



size_t Edge::getVehicleCount ()

Getter for the vehicle count.

Returns

The vehicle count.

Here is the caller graph for this function:

bool Edge::match (size_t *endpointA*, size_t *endpointB*)

This edges are parallel?

Parameters

endpointA	Endpoint A.
endpointB	Endpoint B

Returns

True if the two edge is parallel.

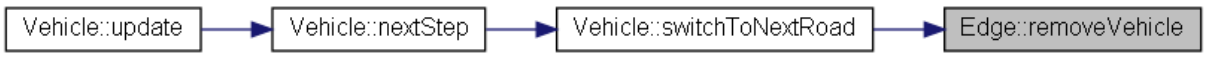
void Edge::removeVehicle (int *c*, bool *repath*)

Remove vehicle to the edge statistic.

Parameters

The	cost.
repath	Repath is working?

Here is the caller graph for this function:



void Edge::resetStats ()

Resetting the edge statistic.

Here is the caller graph for this function:

Field Documentation

size_t Edge::allVehicleCount = 0 [private]

The all vehicle cost.

int Edge::coast = 0 [private]

The cost.

size_t Edge::endpointA = -1 [private]

The edge endpoint A vector ID.

size_t Edge::endpointB = -1 [private]

The edge endpoint B vector ID.

size_t Edge::id = -1 [private]

The vector ID.

size_t Edge::road3DiD = -1 [private]

The render ID.

int Edge::vehicleCoast = 0 [private]

The current vehicle cost.

size_t Edge::vehicleCount = 0 [private]

The current vehicle cost.

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

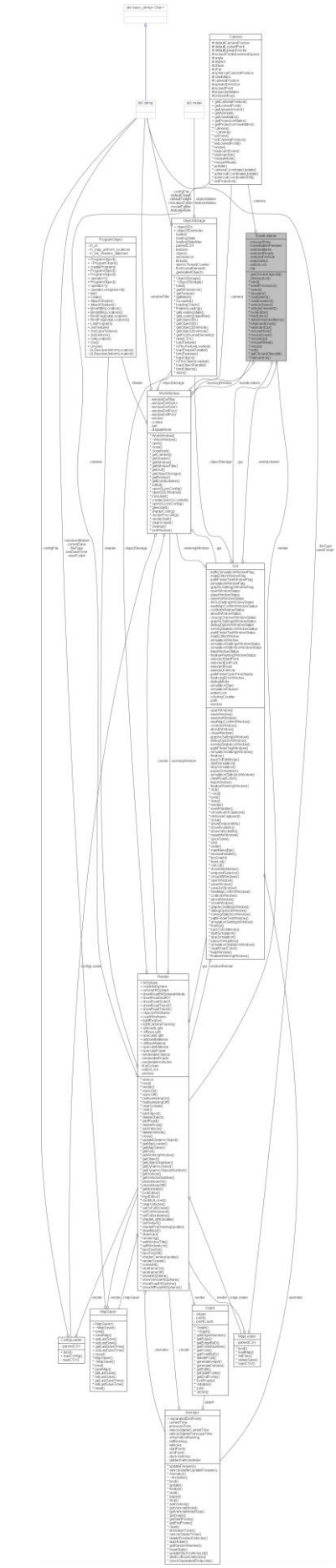
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Edge.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Edge.cpp**

EventListener Class Reference

Event listener implementation. Contact: `sandorbalazs9402@gmail.com`.

```
#include <EventListener.h>
```

Collaboration diagram for EventListener:



Private Member Functions

- Traffic Simulationint **getClickedObjectId** (SDL_MouseButtonEvent &mouse, bool vehicleCheck=false)
  Ray tracing mouse picking implementation.

- float **intersection** (glm::vec3 **ray**, glm::vec3 cameraPosition, glm::vec3 hitSphereCenter, float hitSphereRadius)
  Ray and sphere intersection.

Private Attributes

- **WorkWindow * workingWindow**
  The binded pointer to the current working window.

- **Camera * camera**
  The binded pointer to the current camera.

- **GUI * gui**
  *The binded pointer to the current **GUI**.*

- **Render * render**
  The binded pointer to the current render.

- std::set< int > **pressedKeys**
  The once pressed key collection.

- std::set< int > **moseButtonPressed**
  The once pressed mouse buttons collection.

- std::vector< size_t > **selectedItems**
  The list of the selected objects.

- std::vector< size_t > **selectedRoads**
  The list of the selected roads.

- size_t **selectedVehicle** = -1
  The selected vehicle.

- bool **keepSelect** = false
  Keep select flag.

- bool **editorLock** = false
  Editor lock flag.

- glm::vec3 **ray**
  The current mouse ray for ray picking calculations.

Traffic Simulation

- void **bind** (**WorkWindow** *currentWindow)
  Initialize and bind the event listener.

- void **eventProcessor** ()
  The main event processor loop.

- void **select** (size_t objectID)
  Object selector.

- void **deselect** ()
  Object deselect.

- void **roadSelect** ()
  Road selector from mouse clicking.

- void **roadDeselect** ()
  Road deselect.

- void **vehicleSelect** (size_t vehicleID)
  ***Vehicle** selector.*

- void **vehicleDeselect** ()
  ***Vehicle** deselect.*

- void **lockEditor** ()
  Locking the editor events.

- void **freeEditor** ()
  Free the editor events.

- void **deleteSelectedItems** ()
  Delete selected items.

- void **keyboardDown** (SDL_KeyboardEvent &)
  Keyboard down event processor.

- void **keyboardUp** (SDL_KeyboardEvent &)
  Keyboard up event processor.

- void **mouseMove** (SDL_MouseMotionEvent &)
  The mouse move event processor.

- void **mouseDown** (SDL_MouseButtonEvent &)
  The mouse Down event processor.

- void **mouseUp** (SDL_MouseButtonEvent &)
  The mouse up event processor.

- void **mouseWheel** (SDL_MouseWheelEvent &)
  The mouse wheel event processor.

- void **resize** (SDL_WindowEvent &window)
  The resize event processor.

- void **exit** ()
  Exit event handler.

Detailed Description

Event listener implementation. Contact: `sandorbalazs9402@gmail.com`.

Ray mousepicking implementation. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.12.14.

Author

Sándor Balázs - AZA6NL

Date

2021.11.09.

Member Function Documentation

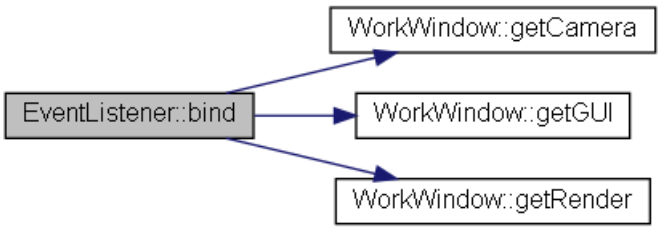
void EventListener::bind (WorkWindow * *currentWindow*)

Initialize and bind the event listener.

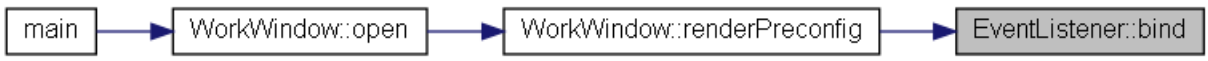
Parameters

currentWindow	The current working window for the event listener.

Here is the call graph for this function:



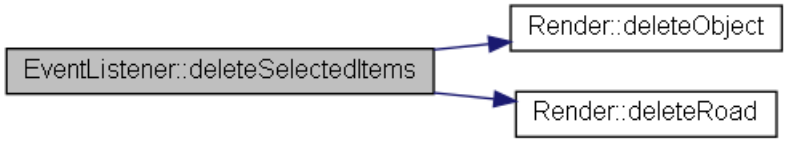
Here is the call graph for this function:



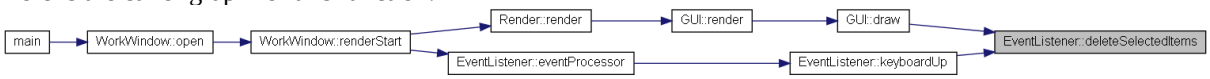
void EventListener::deleteSelectedItems ()

Delete selected items.

Here is the call graph for this function:



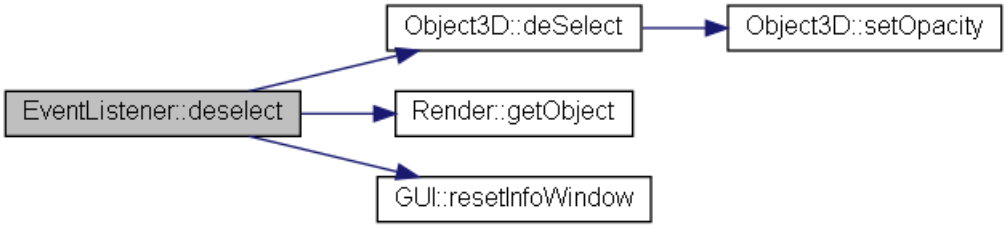
Here is the caller graph for this function:



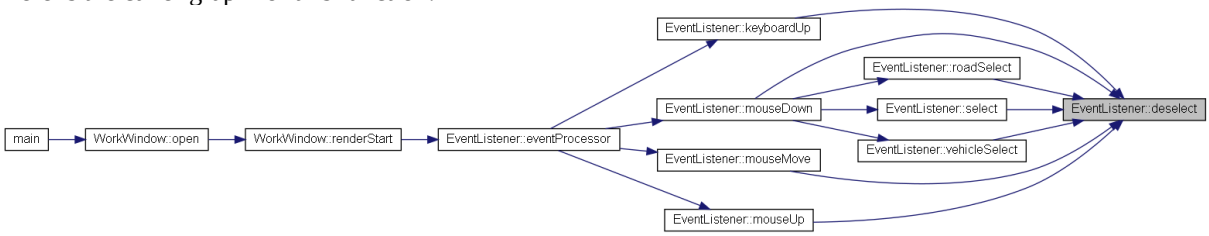
void EventListener::deselect ()

Object deselect.

Here is the call graph for this function:



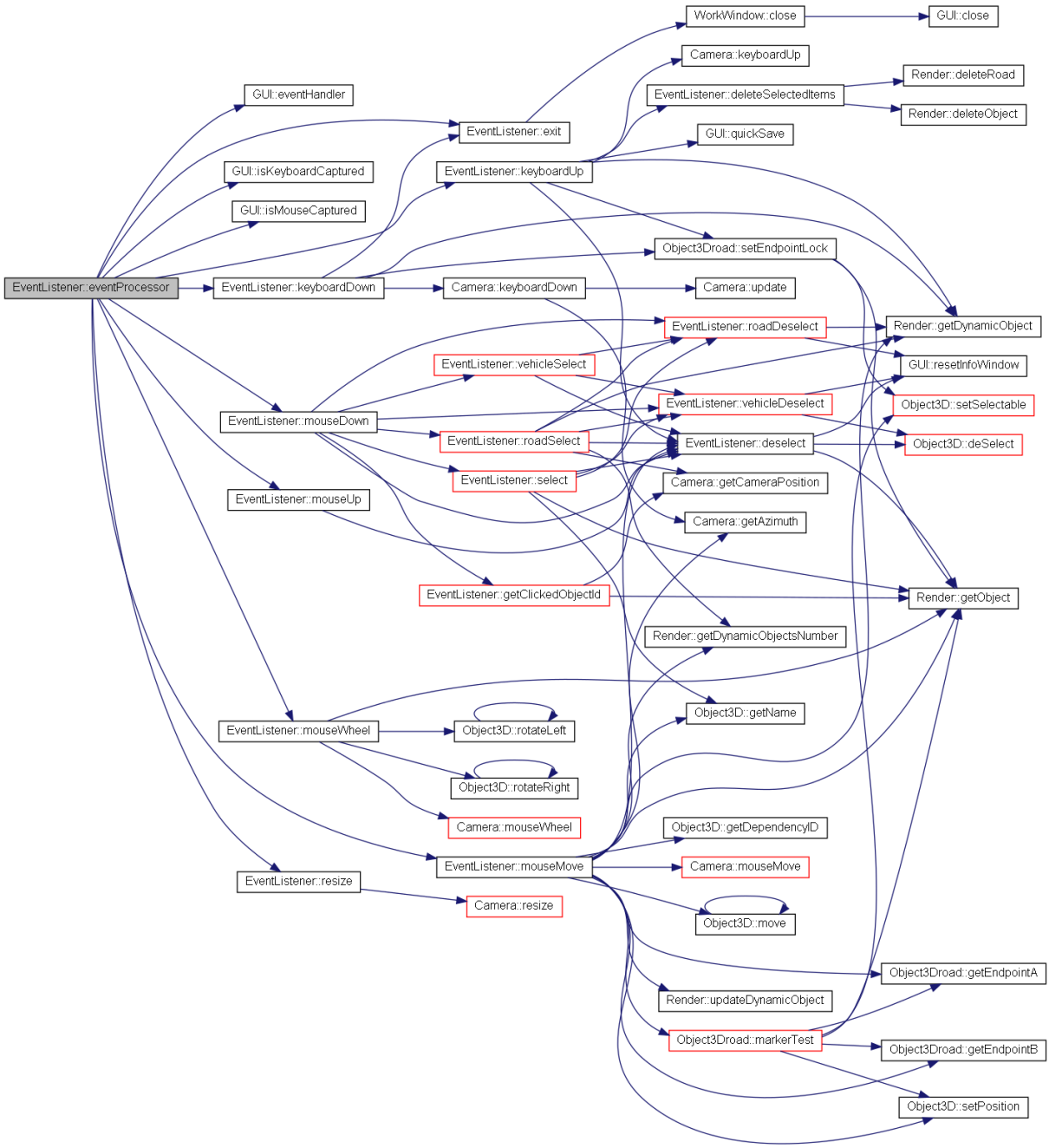
Here is the caller graph for this function:



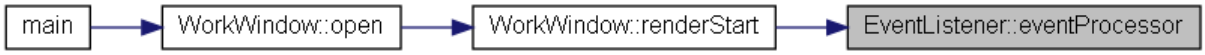
void EventListener::eventProcessor ()

The main event processor loop.

Here is the call graph for this function:



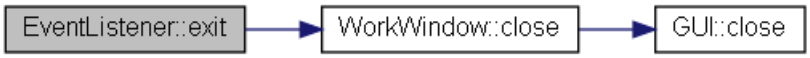
Here is the caller graph for this function:



void EventListener::exit () [private]

Exit event handler.

Here is the call graph for this function:



Here is the caller graph for this function:

void EventListener::freeEditor ()

Free the editor events.

Here is the caller graph for this function:

int EventListener::getClickedObjectId (SDL_MouseButtonEvent & *mouse*, bool *vehicleCheck* = `false`) `[private]`

Ray tracing mouse picking implementation.

Calculating a ray true the camera and the mouse picked point and test the intersections for all the hit spheres.

The vector generation is a revers incremental image synthesis.

When have multiple hit sphere in one line we select the one closest to the camera.

The function's main step's:

Step 0: Get the screen size.

Step 1: Get the 3d Normalized mouse coordinates in Device Coordinates.

Step 2: Convert the device mouse vector to 4d Homogeneous Clip Coordinates.

Step 5: Normalize the world mouse vector.

Step 6: Intersect the world mouse vector with all the renderable object's hit spheres.

Step 7: Depth test.

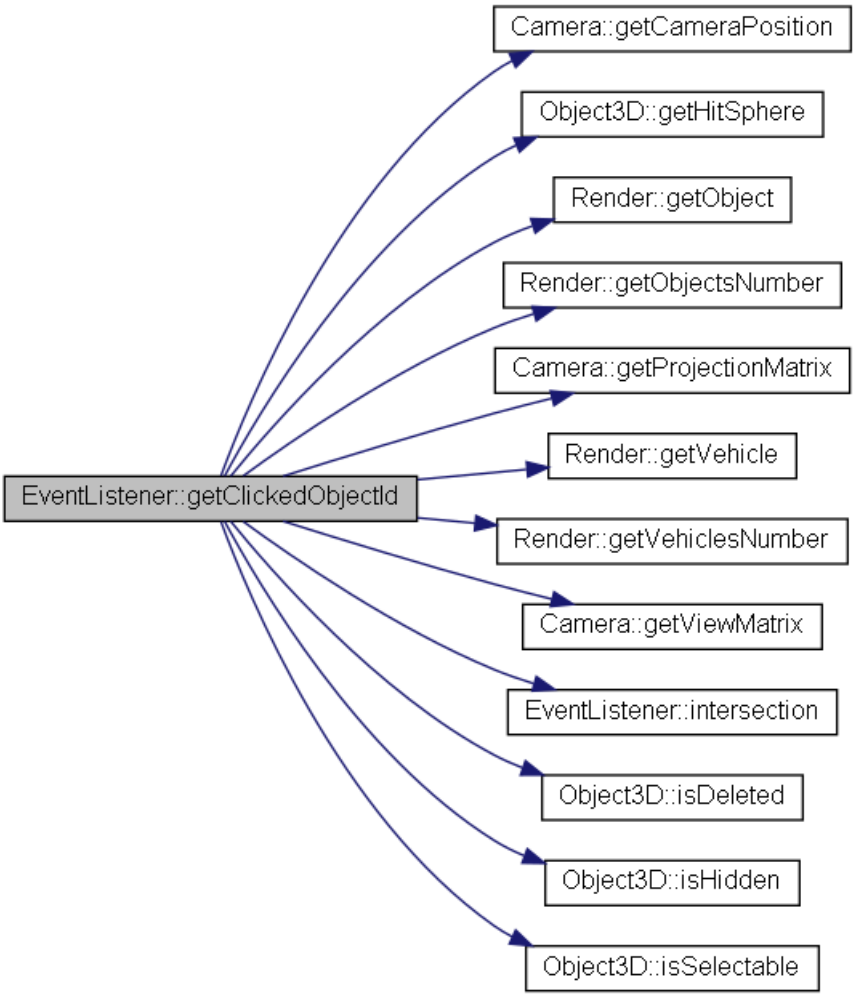
Parameters

mouse	The current mouse click event.

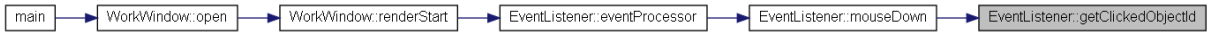
Returns

The selected item render id.

Here is the call graph for this function:



Here is the caller graph for this function:



float EventListener::intersection (glm::vec3 *ray*, glm::vec3 *cameraPosition*, glm::vec3 *hitSphereCenter*, float *hitSphereRadius*) `[private]`

Ray and sphere intersection.

Parameters

ray	The ray for the intersection.
cameraPosition	The camera position for the depth test.
hitSphereCenter	The sphere center.
hitSphereRadius	The sphere radius.

Returns

The sphere and the ray distance.

Here is the caller graph for this function:

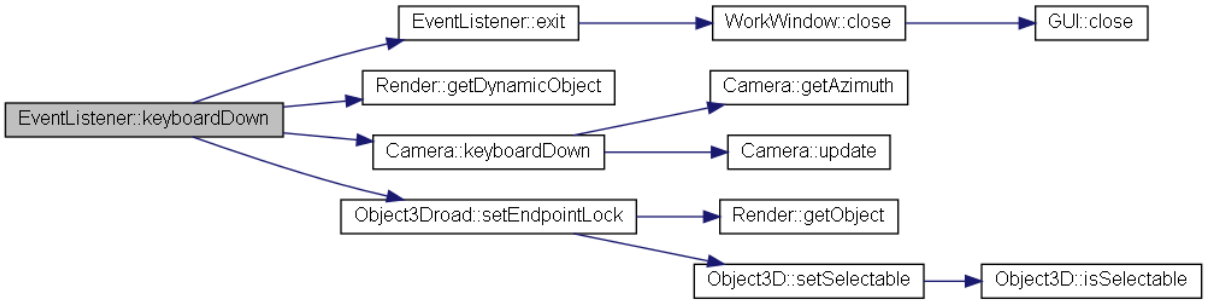
void EventListener::keyboardDown (SDL_KeyboardEvent & *key*) `[private]`

Keyboard down event processor.

Parameters

key	The current key event.

Here is the call graph for this function:



Here is the caller graph for this function:

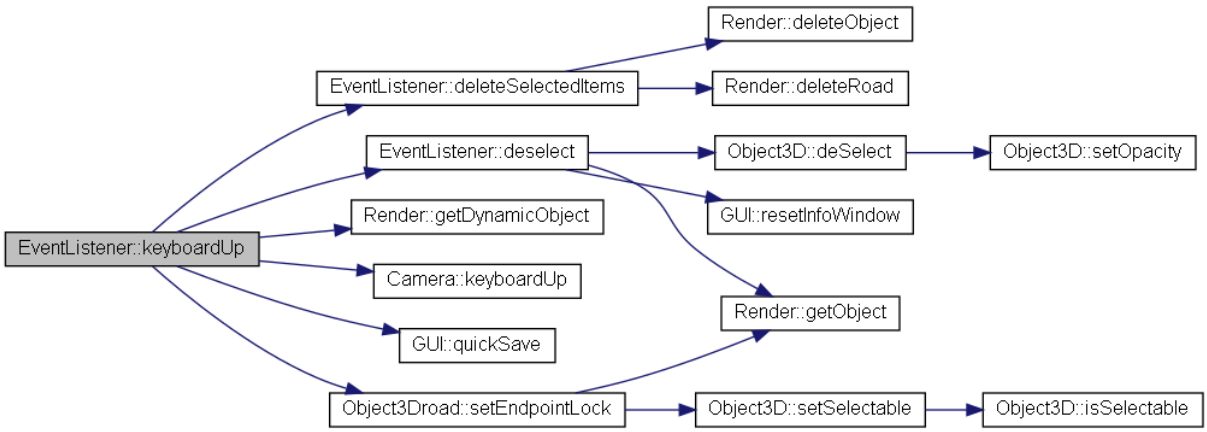
void EventListener::keyboardUp (SDL_KeyboardEvent & *key*) `[private]`

Keyboard up event processor.

Parameters

key	The current key event.

Here is the call graph for this function:



Here is the caller graph for this function:

void EventListener::lockEditor ()

Locking the editor events.

Here is the caller graph for this function:

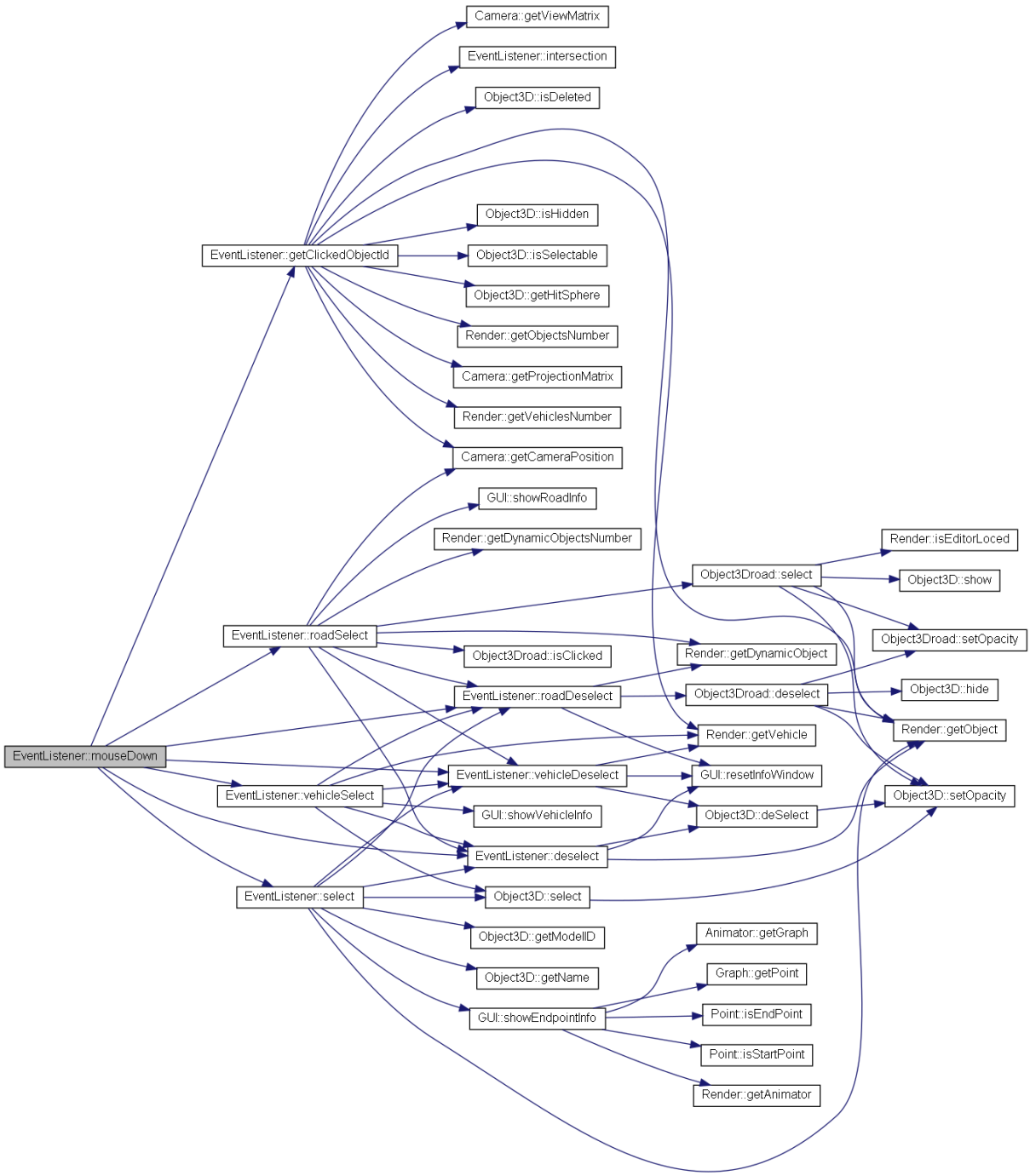
void EventListener::mouseDown (SDL_MouseButtonEvent & *mouse*) `[private]`

The mouse Down event processor.

Parameters

mouse	The current mouse event.

Here is the call graph for this function:



Here is the caller graph for this function:

void EventListener::mouseMove (SDL_MouseMotionEvent & *mouse*) `[private]`

The mouse move event processor.

(The keyboard move direction is set by camera direction.)

Parameters

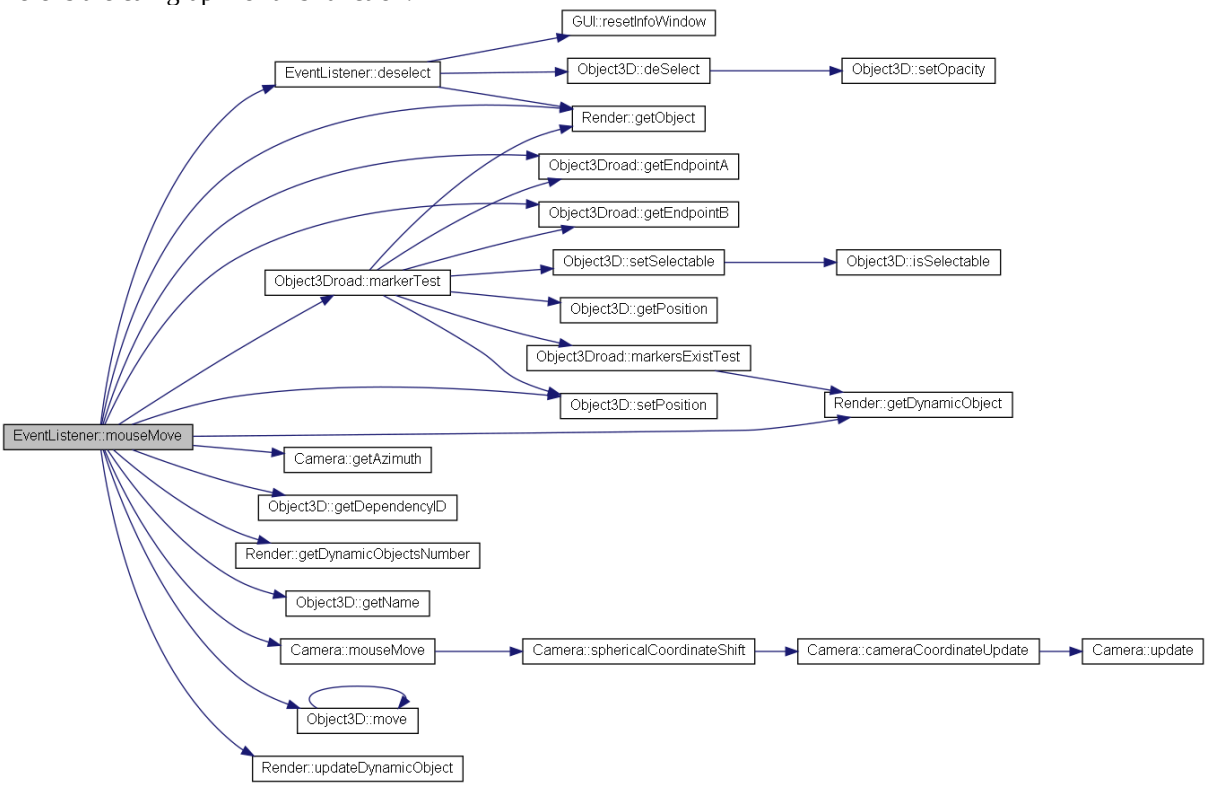
mouse	The current mouse event.

(The mouse move direction is set by camera direction.)

Parameters

mouse	The current mouse event.

Here is the call graph for this function:



Here is the caller graph for this function:

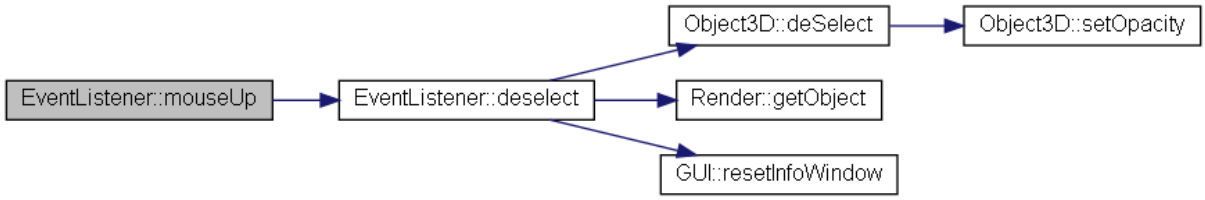
void EventListener::mouseUp (SDL_MouseButtonEvent & *mouse*) `[private]`

The mouse up event processor.

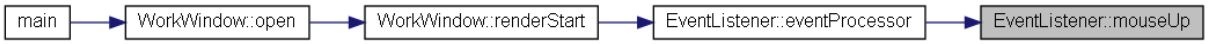
Parameters

mouse	The current mouse event.

Here is the call graph for this function:



Here is the caller graph for this function:



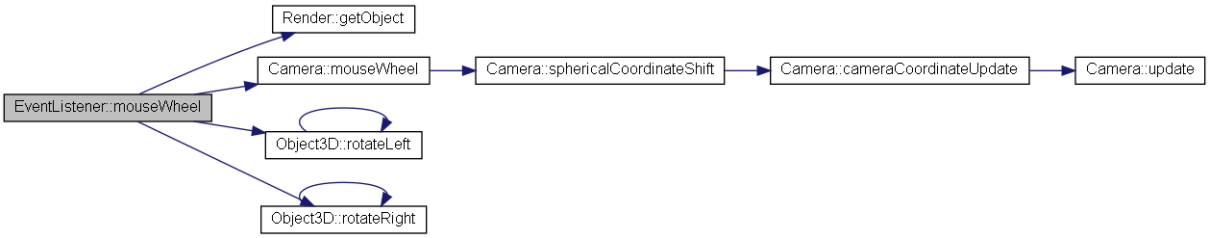
void EventListener::mouseWheel (SDL_MouseWheelEvent & *wheel*) `[private]`

The mouse wheel event processor.

Parameters

mouse	The current mouse wheel event.

Here is the call graph for this function:



Here is the caller graph for this function:

void EventListener::resize (SDL_WindowEvent & *window*) `[private]`

The resize event processor.

Parameters

mouse	The current window event.

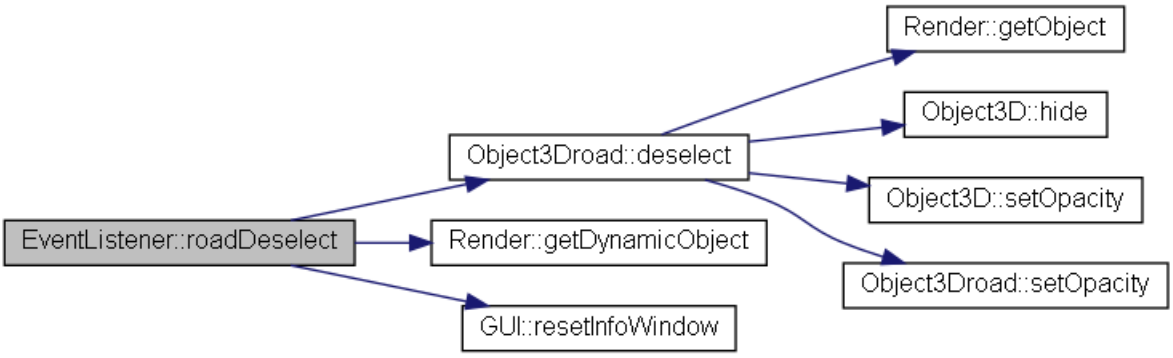
Here is the call graph for this function:

Here is the caller graph for this function:

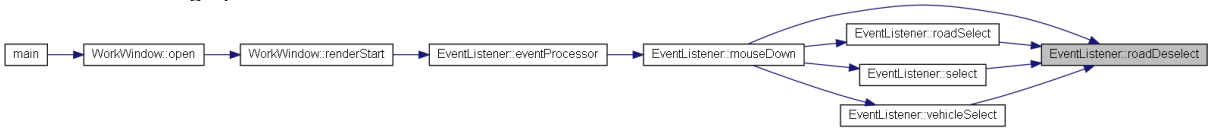
void EventListener::roadDeselect ()

Road deselect.

Here is the call graph for this function:



Here is the caller graph for this function:

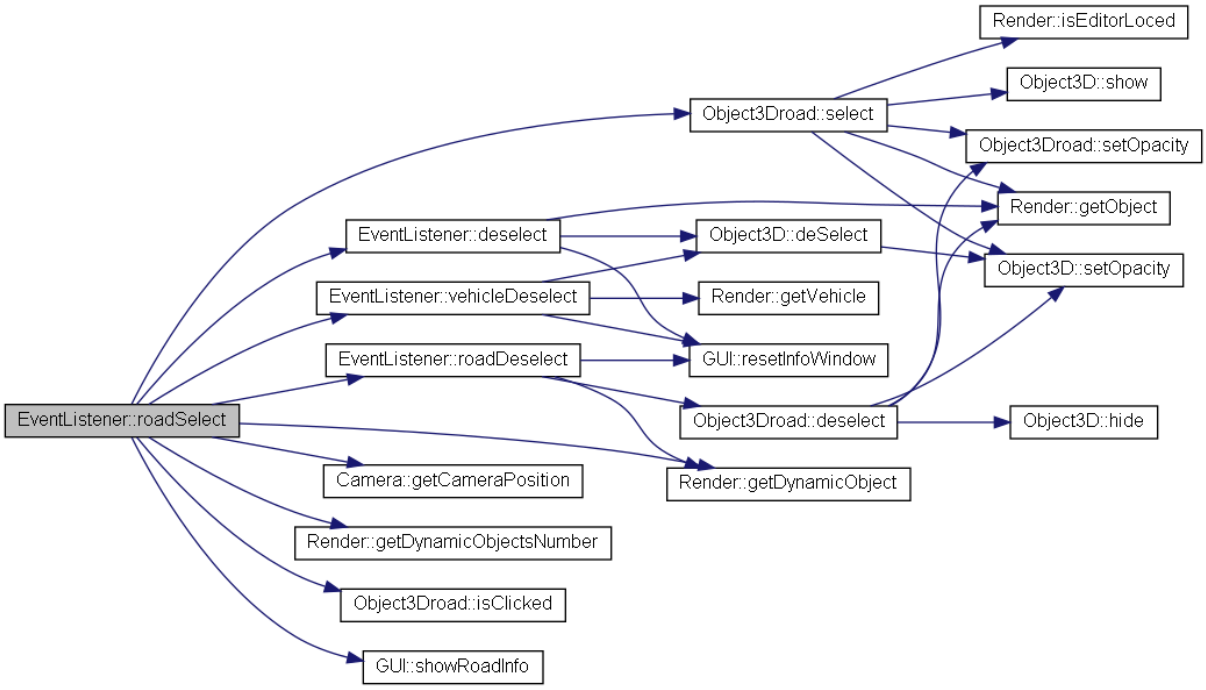


void EventListener::roadSelect ()

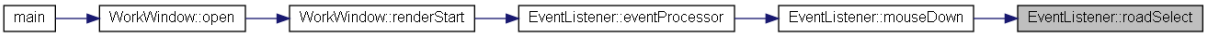
Road selector from mouse clicking.

(Ray picking for road included.)

Here is the call graph for this function:



Here is the caller graph for this function:



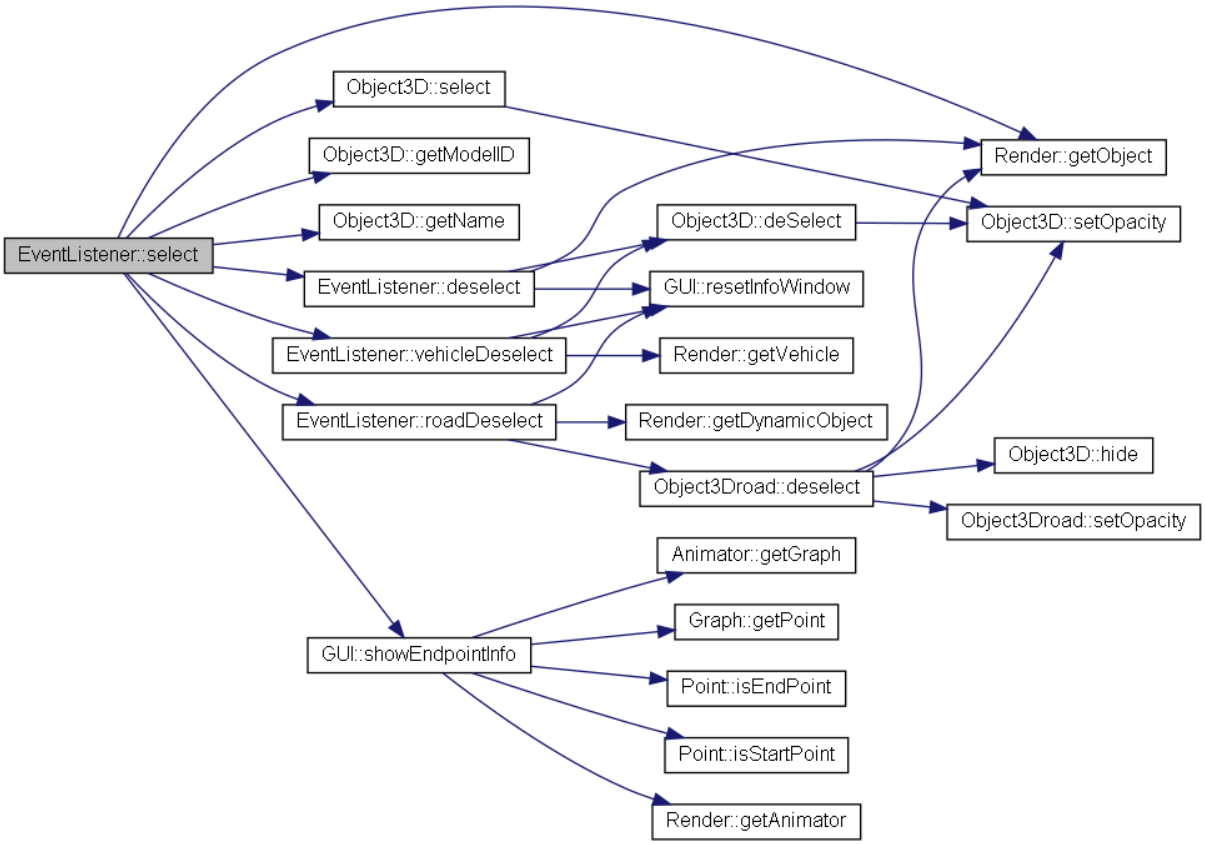
void EventListener::select (size_t *objectID*)

Object selector.

Parameters

objectID	The selected object ID from the mouse ray picker.

Here is the call graph for this function:

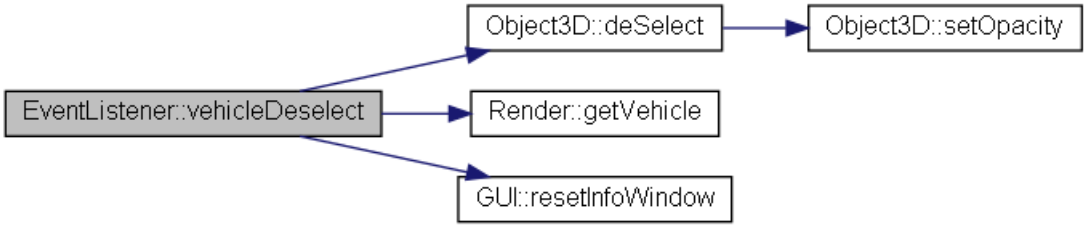


Here is the caller graph for this function:

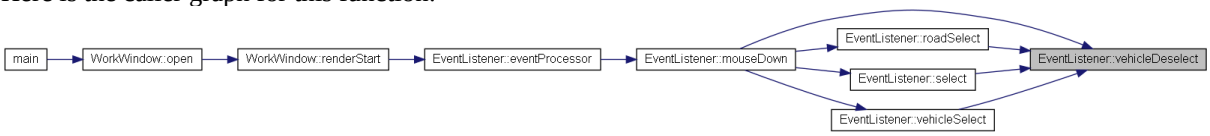
void EventListener::vehicleDeselect ()

Vehicle deselect.

Here is the call graph for this function:



Here is the caller graph for this function:



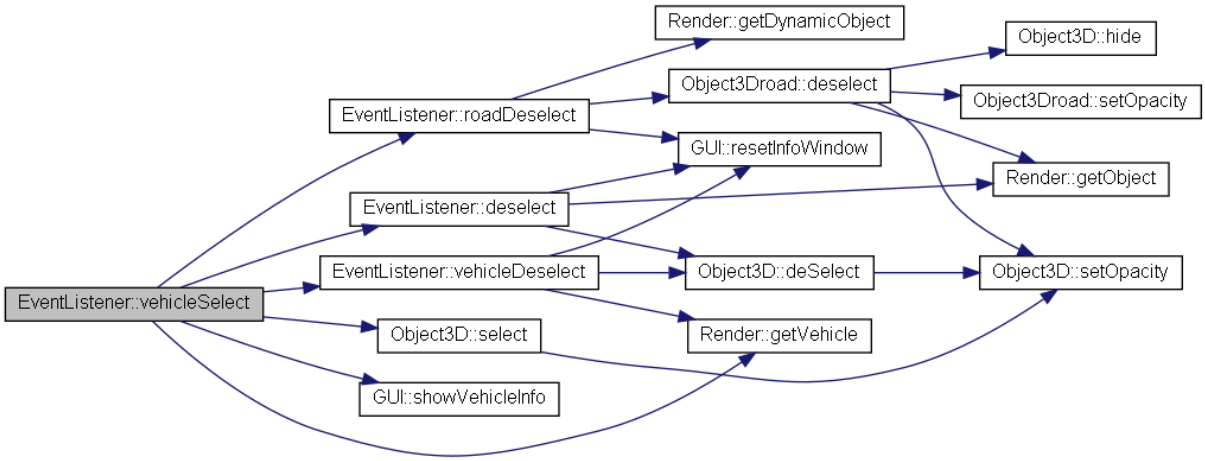
void EventListener::vehicleSelect (size_t *vehicleID*)

Vehicle selector.

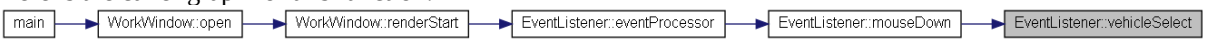
Parameters

vehicleID	The selected vehicle ID from the mouse ray picker.

Here is the call graph for this function:



Here is the caller graph for this function:



Field Documentation

Camera* EventListener::camera [private]

The binded pointer to the current camera.

bool EventListener::editorLock = false [private]

Editor lock flag.

GUI* EventListener::gui [private]

The binded pointer to the current **GUI**.

bool EventListener::keepSelect = false [private]

Keep select flag.

std::set<int> EventListener::moseButtonPressed [private]

The once pressed mouse buttons collection.

std::set<int> EventListener::pressedKeys [private]

The once pressed key collection.

glm::vec3 EventListener::ray [private]

The current mouse ray for ray picking calculations.

Render* EventListener::render [private]

The binded pointer to the current render.

std::vector<size_t> EventListener::selectedItems [private]

The list of the selected objects.

std::vector<size_t> EventListener::selectedRoads [private]

The list of the selected roads.

size_t EventListener::selectedVehicle = -1 [private]

The selected vehicle.

WorkWindow* EventListener::workingWindow [private]

The binded pointer to the current working window.

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

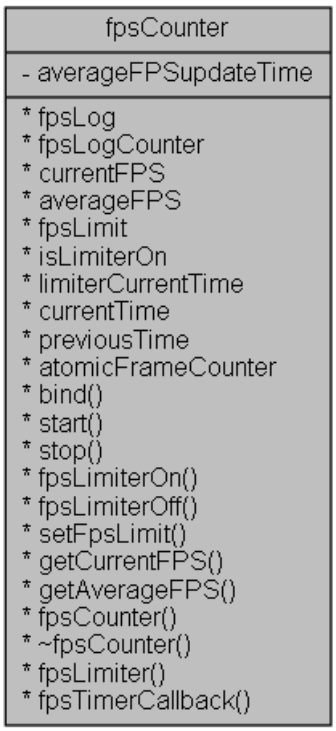
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**EventListener.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**EventListener.cpp**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**RayPicking.cpp**

fpsCounter Class Reference

FPS counter, logger and limiter static class.

```
#include <fpsCounter.h>
```

Collaboration diagram for fpsCounter:



Static Private Attributes

- static const int **averageFPSupdateTime** = 500
  Parallel FPS counter update interval.

Traffic Simulation

- static float **fpsLog** [100] = { 0.0f }
  The FPS Log array.

- static int **fpsLogCounter** = 0
  The FPS log index.

- static float **currentFPS**
  The current FPS.

- static float **averageFPS**
  The average FPS.

- static float **fpsLimit** = 60.0f
  The current FPS limit.

- static bool **isLimiterOn** = false
  The FPS limiter switch.

- static Uint32 **limiterCurrentTime**
  The current time for the limiter delay.

- static Uint32 **currentTime**
  The current time.

- static Uint32 **previousTime**
  The previous saved time.

- static SDL_atomic_t **atomicFrameCounter**
  The atomic frame counter.

- static void **bind** ()
  Initialize the class.

- static void **start** ()
  Start FPS counting.

- static void **stop** ()
  Stop the counter.

- static void **fpsLimiterOn** ()
  FPS Limiter ON.

- static void **fpsLimiterOff** ()
  FPS Limiter OFF.

- static void **setFpsLimit** (float limit)
  Set FPS limiter.

- static float **getCurrentFPS** ()
  Getter for current FPS.

- static float **getAverageFPS** ()
  Getter for average FPS.

- **fpsCounter** ()
  Private constructor for a static class.

- **~fpsCounter** ()
  Private destructor for a static class.

- static void **fpsLimiter** (float limit)
  FPS Limiter setter.

- static Uint32 **fpsTimerCallback** (Uint32 interval, void *data)
  The atomic frame counter callback function.

Detailed Description

FPS counter, logger and limiter static class.

FPS counter, logger and limiter. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

fpsCounter::fpsCounter () `[private]`

Private constructor for a static class.

fpsCounter::~fpsCounter () `[private]`

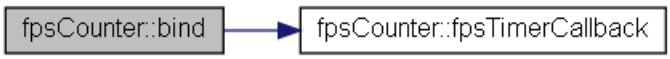
Private destructor for a static class.

Member Function Documentation

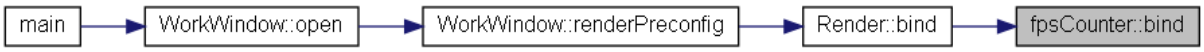
void fpsCounter::bind () `[static]`

Initialize the class.

Here is the call graph for this function:



Here is the caller graph for this function:



void fpsCounter::fpsLimiter (float *limit*) `[static], [private]`

FPS Limiter setter.

Parameters

limit	The new FPS limit.

Here is the caller graph for this function:

void fpsCounter::fpsLimiterOff () `[static]`

FPS Limiter OFF.

Here is the caller graph for this function:

void fpsCounter::fpsLimiterOn () `[static]`

FPS Limiter ON.

Here is the caller graph for this function:

Uint32 fpsCounter::fpsTimerCallback (Uint32 *interval*, void * *data*) `[static]`, `[private]`

The atomic frame counter callback function.

Parameters

interval	The SDL timer interval.
data	The SDL Timer data.

Returns

The SDL Timer return.

Here is the caller graph for this function:

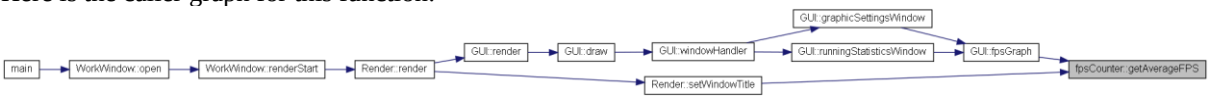
float fpsCounter::getAverageFPS () `[static]`

Getter for average FPS.

Returns

average FPS.

Here is the caller graph for this function:



float fpsCounter::getCurrentFPS () `[static]`

Getter for current FPS.

Returns

current FPS.

void fpsCounter::setFpsLimit (float *limit*) `[static]`

Set FPS limiter.

Parameters

limit	New FPS limit.

Here is the caller graph for this function:

void fpsCounter::start () `[static]`

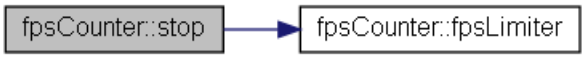
Start FPS counting.

Here is the caller graph for this function:

void fpsCounter::stop () `[static]`

Stop the counter.

Here is the call graph for this function:



Here is the caller graph for this function:

Field Documentation

SDL_atomic_t fpsCounter::atomicFrameCounter `[static]`, `[private]`

The atomic frame counter.

float fpsCounter::averageFPS `[static]`, `[private]`

The average FPS.

const int fpsCounter::averageFPSupdateTime = 500 `[static]`, `[private]`

Parallel FPS counter update interval.

float fpsCounter::currentFPS `[static]`, `[private]`

The current FPS.

Uint32 fpsCounter::currentTime `[static]`, `[private]`

The current time.

float fpsCounter::fpsLimit = 60.0f `[static]`, `[private]`

The current FPS limit.

float fpsCounter::fpsLog = { 0.0f } `[static]`

The FPS Log array.

int fpsCounter::fpsLogCounter = 0 `[static], [private]`

The FPS log index.

bool fpsCounter::isLimiterOn = false `[static], [private]`

The FPS limiter switch.

Uint32 fpsCounter::limiterCurrentTime `[static], [private]`

The current time for the limiter delay.

Uint32 fpsCounter::previousTime `[static], [private]`

The previous saved time.

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

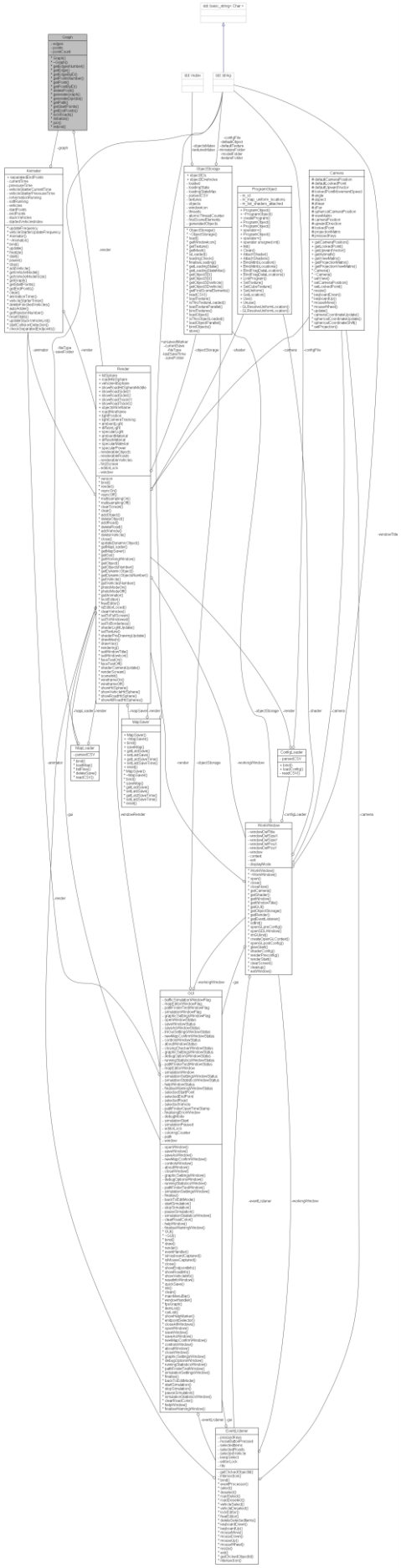
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**fpsCounter.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**fpsCounter.cpp**

Graph Class Reference

The abstract **Edge** implementation.

```
#include <Graph.h>
```

Collaboration diagram for Graph:


std::basic_string< Char >
std::hidden
std::string
graph
Animator
ObjectStorage
ProgramObject
Camera
Render
MapLoader
MapSaver
MissLoader
ConfigLoader
RealWindow
GUI
OverlayLoader
-objectMaker
-textureMaker
-configFile
-defaultObject
-defaultTexture
-mouseCursor
-keyboardCursor
-graph
-animator
-fileType
-viewFolder
-render
-objectStorage
-shader
-camera
-configFile
-windowTitle
-mapLoader
-mapSaver
-window
-workingWindow
-windowRender
-eventListener

Private Attributes

- **Render** * **render**
  The render pointer.

- std::vector< **Edge** * > **edges**
  The list of the edges in the graph.

- std::vector< **Point** * > **points**
  The list of the points in the graph.

- size_t **pointCount** = 0
  The real point count.

Traffic Simulation

- **Graph** (**Render** ***render**)
  ***Graph** constructor.*

- **~Graph** ()
  ***Graph** destructor.*

- size_t **getEdgesNumber** ()
  Getter for the edge count.

- **Edge** * **getEdge** (size_t edge)
  Getter for the edges by vector id.

- **Edge** * **getEdgeByID** (size_t edge)
  Getter for the edges by real ID.

- size_t **getPointsNumber** ()
  Getter for the points count.

- **Point** * **getPoint** (size_t point)
  Getter for the points by vector id.

- **Point** * **getPointByID** (size_t point)
  Getter for the points by real id.

- void **deletePoint** (size_t point)
  Delete a point.

- void **generateGraph** ()
  Generate the graph by the renderable roads.

- **Dijkstra** * **generateDijkstra** (size_t startPoint)
  *Generate Adjacency matrix from the graph and the **Dijkstra** object.*

- std::vector< size_t > **getPath** (**Dijkstra** *dijkstra, size_t target)
  *Generate path matrix from the **Dijkstra** object results.*

- std::vector< size_t > **getStartPoints** ()
  Getter for the start point list.

- std::vector< size_t > **getEndPoints** ()
  Getter for the end point list.

- void **lockRoads** ()
  Lock the roads.

- void **initialize** ()
  initialize the graph.

- void **join** ()
  Join the multiple points.

- void **rebind** ()
  Rebind the real ID-s.

Detailed Description

The abstract **Edge** implementation.

The abstract graph class. Contact: `sandorbalazs9402@gmail.com`.

The abstract graph class. Contact: `sandorbalazs9402@gmail.com` Spirit In The Sky.

Author

Sándor Balázs - AZA6NL

Date

2021.11.20.

Constructor & Destructor Documentation

Graph::Graph (Render * *render*)

Graph constructor.

Parameters

render	Pointer to the current render.

Graph::~Graph ()

Graph destructor.

Member Function Documentation

void Graph::deletePoint (size_t *point*)

Delete a point.

Parameters

point	The deletable point.

Here is the caller graph for this function:

Dijkstra * Graph::generateDijkstra (size_t *startPoint*)

Generate Adjacency matrix from the graph and the **Dijkstra** object.

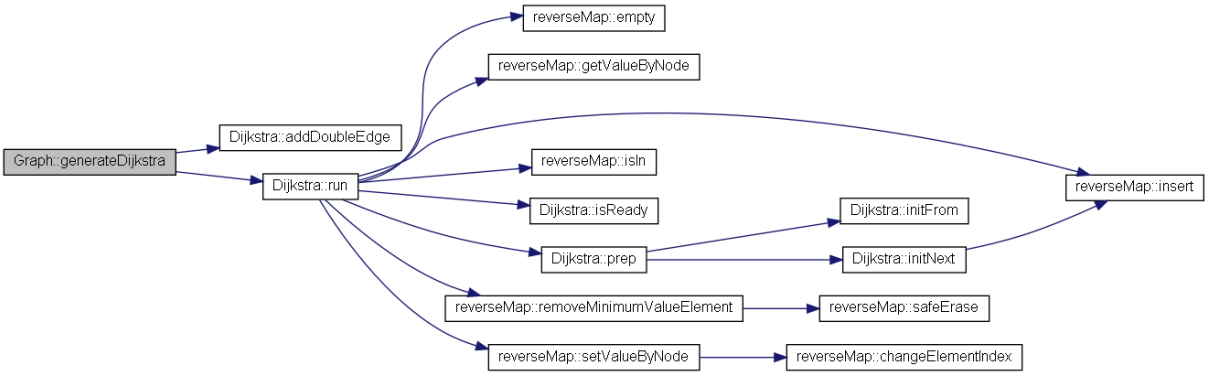
Parameters

startPoint	The start point in the graph.

Returns

The generated **Dijkstra** object.

Here is the call graph for this function:

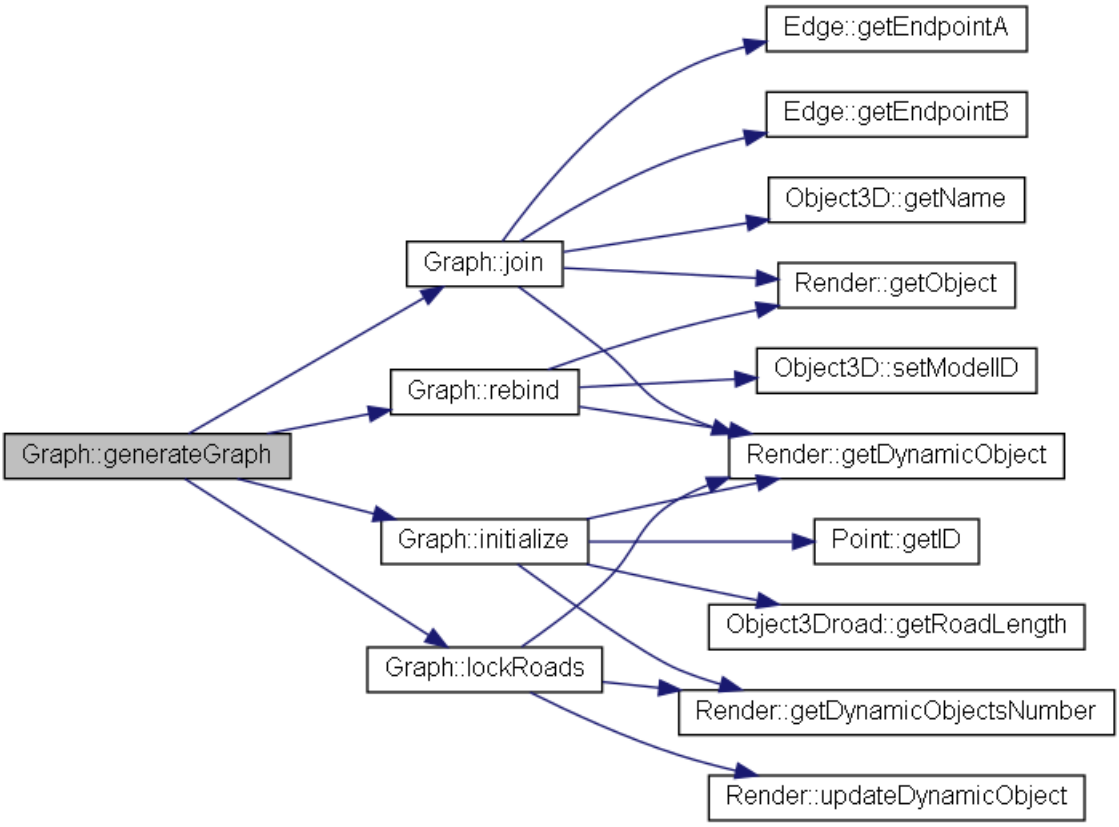


Here is the caller graph for this function:

void Graph::generateGraph ()

Generate the graph by the renderable roads.

Here is the call graph for this function:



Here is the caller graph for this function:

Edge * Graph::getEdge (size_t *edge*)

Getter for the edges by vector id.

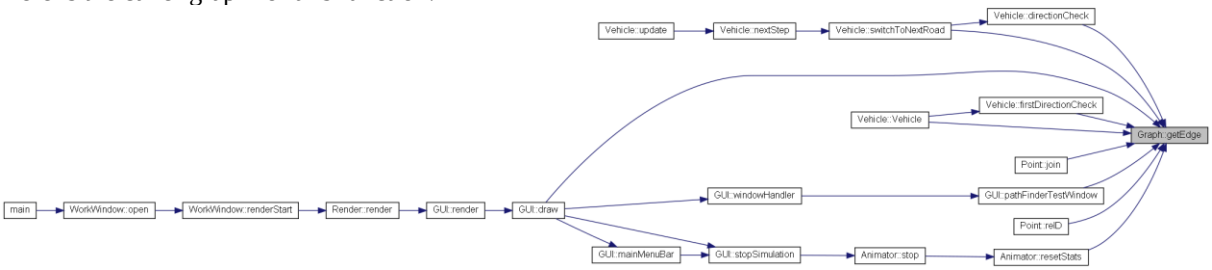
Parameters

edge	The needed edge vector ID.

Returns

The needed edge.

Here is the caller graph for this function:



Edge * Graph::getEdgeByID (size_t *edge*)

Getter for the edges by real ID.

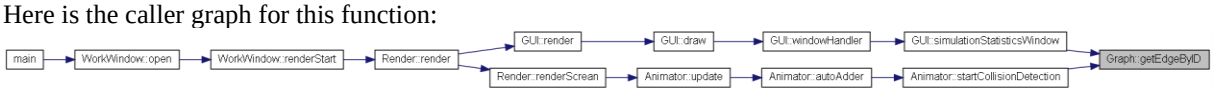
Parameters

edge	The needed edge real ID.

Returns

The needed edge.

Here is the caller graph for this function:



size_t Graph::getEdgesNumber ()

Getter for the edge count.

Returns

The edge count.

Here is the caller graph for this function:

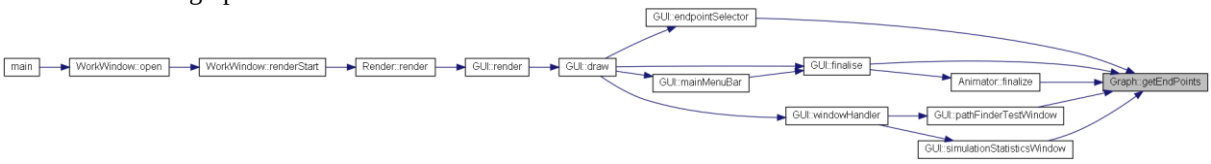
std::vector< size_t > Graph::getEndPoints ()

Getter for the end point list.

Returns

The endpoint list.

Here is the caller graph for this function:



std::vector< size_t > Graph::getPath (Dijkstra * *dijkstra*, size_t *target*)

Generate path matrix from the **Dijkstra** object results.

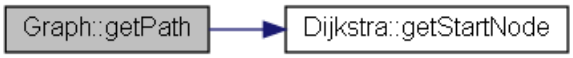
Parameters

Dijkstra	The **Dijkstra** object.
target	The target point.

Returns

The path.

Here is the call graph for this function:



Here is the caller graph for this function:

Point * Graph::getPoint (size_t *point*)

Getter for the points by vector id.

Parameters

edge	The needed point vector ID.

Returns

The needed point.

Here is the caller graph for this function:

Point * Graph::getPointByID (size_t *point*)

Getter for the points by real id.

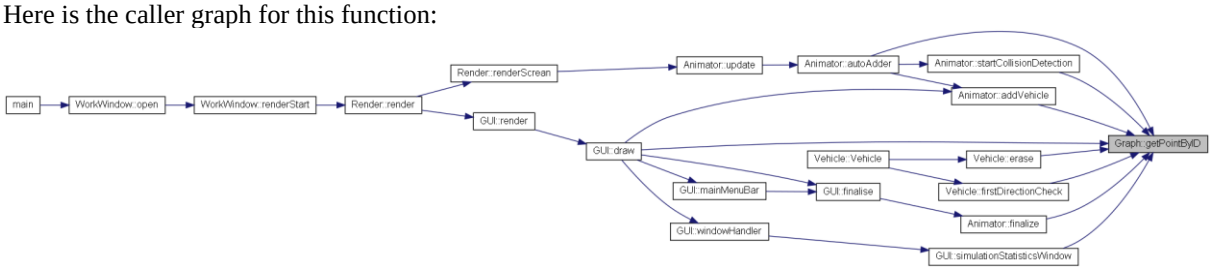
Parameters

edge	The needed point real ID.

Returns

The needed point.

Here is the caller graph for this function:



size_t Graph::getPointsNumber ()

Getter for the points count.

Returns

The points count.

Here is the caller graph for this function:

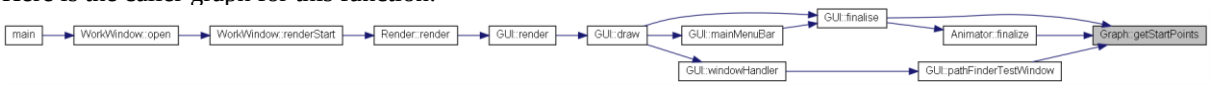
std::vector< size_t > Graph::getStartPoints ()

Getter for the start point list.

Returns

The startpoint list.

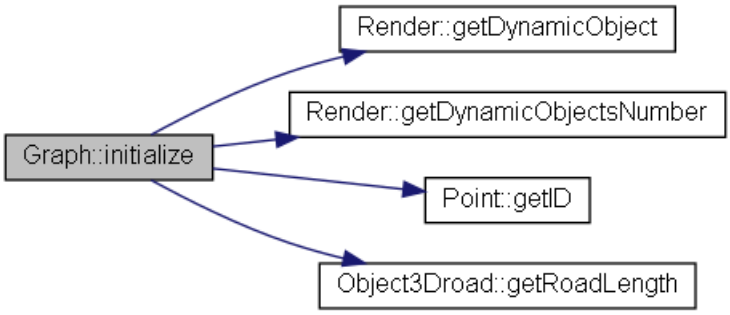
Here is the caller graph for this function:



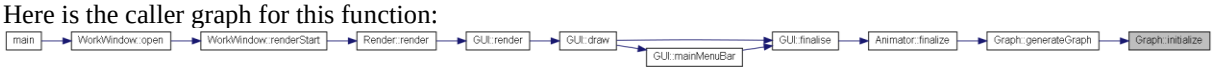
void Graph::initialize () [private]

initialize the graph.

Here is the call graph for this function:



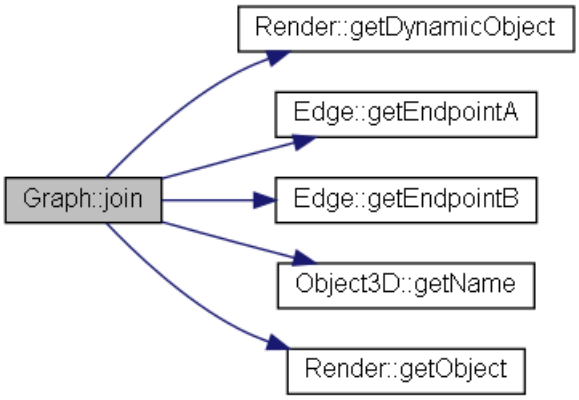
Here is the caller graph for this function:



void Graph::join () [private]

Join the multiple points.

Here is the call graph for this function:

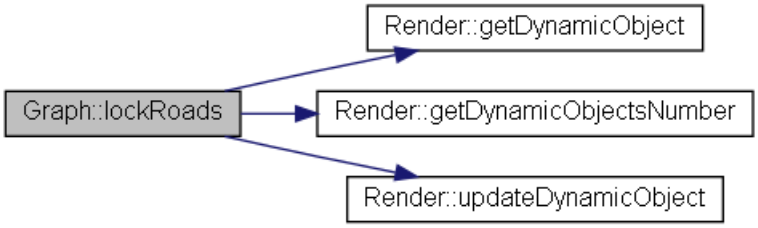


Here is the caller graph for this function:

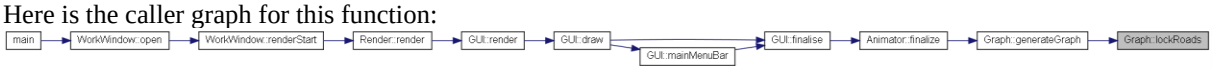
void Graph::lockRoads () [private]

Lock the roads.

Here is the call graph for this function:



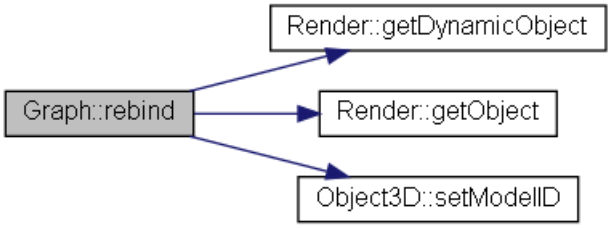
Here is the caller graph for this function:



void Graph::rebind () [private]

Rebind the real ID-s.

Here is the call graph for this function:



Here is the caller graph for this function:

Field Documentation

std::vector<Edge*> Graph::edges `[private]`

The list of the edges in the graph.

size_t Graph::pointCount = 0 `[private]`

The real point count.

(Without deleted points.)

std::vector<Point*> Graph::points `[private]`

The list of the points in the graph.

Render* Graph::render `[private]`

The render pointer.

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

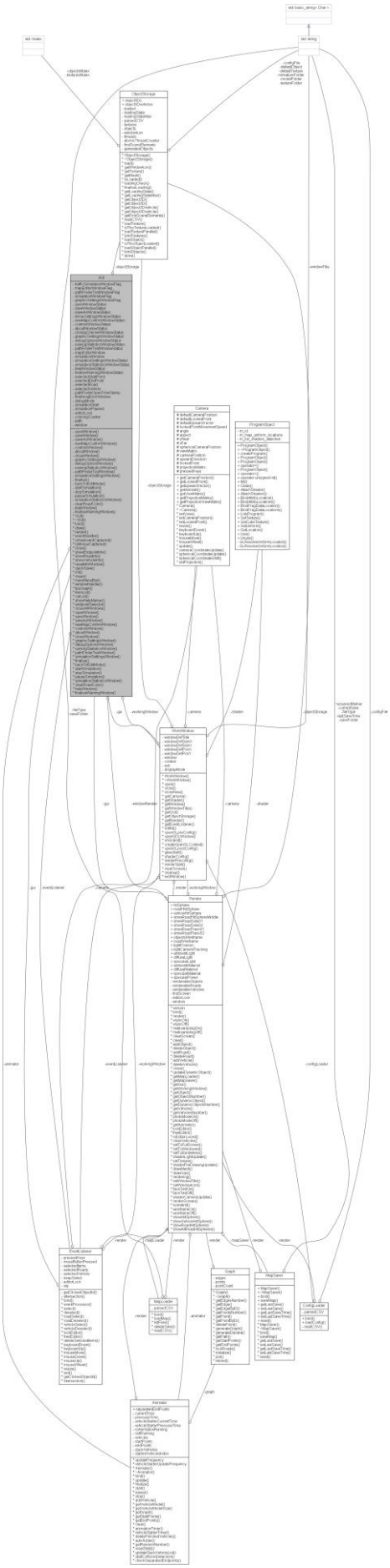
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Graph.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Graph.cpp**

GUI Class Reference

GUI Descriptor class. Contact: `sandorbalazs9402@gmail.com`.

```
#include <GUI.h>
```

Collaboration diagram for GUI:



Private Member Functions

- Traffic Simulationvoid **openWindow** ()
  Description of the opening window.

- void **saveWindow** ()
  the save window descriptor.

- void **saveAsWindow** ()
  the save as window descriptor.

- void **newMapConfirmWindow** ()
  Window for the new map confirmation.

- void **controlsWindow** ()
  Window for the controls tutorial.

- void **aboutWindow** ()
  Window for the about tab.

- void **closeWindow** ()
  Application closing checker window.

- void **graphicSettingsWindow** ()
  Window descriptor for the graphic settings window.

- void **debugOptionsWindow** ()
  Window for the debug options.

- void **runningStatisticsWindow** ()
  Window for the running statistics.

- void **pathFinderTestWindow** ()
  Pathfinder test window.

- void **simulationSettingsWindow** ()
  The window for the simulation settings.

- void **finalise** ()
  Finalize the map.

- void **backToEditMode** ()
  Back to the edit mode.

- void **startSimulation** ()
  Start simulation.

- void **stopSimulation** ()
  Stop running simulation.

- void **pauseSimulation** ()
  Pause running simulation.

- void **simulationStatisticsWindow** ()
  Window for SimulationStatistics.

- void **clearRoadColor** ()
  Clear the road colors.

- void **helpWindow** ()
  Show help window.

- void **finaliseWarningWindow** ()
  Show the finalizing warning window.

Private Attributes

- ImGuiWindowFlags **trafficSimulationWindowFlag** = 0
- ImGuiWindowFlags **mapEditorWindowFlag** = 0
- ImGuiWindowFlags **pathFinderTestWindowFlag** = 0
- ImGuiWindowFlags **simulationWindowFlag** = 0
- ImGuiWindowFlags **graphicSettingsWindowFlag** = 0
- bool **openWindowStatus** = false
- bool **saveWindowStatus** = false
- bool **saveAsWindowStatus** = false
- bool **ImGuiSettingsWindowStatus** = false
- bool **newMapConfirmWindowStatus** = false
- bool **controlsWindowStatus** = false
- bool **aboutWindowStatus** = false
- bool **closingCheckerWindowStatus** = false
- bool **graphicSettingsWindowStatus** = false
- bool **debugOptionsWindowStatus** = false
- bool **runningStatisticsWindowStatus** = false
- bool **pathFinderTestWindowStatus** = false
- bool **mapEditorWindow** = true
- bool **simulationWindow** = false
- bool **simulationSettingsWindowStatus** = false
- bool **simulationStatisticsWindowStatus** = false
- bool **helpWindowStatus** = false
- bool **finaliseWarningWindowStatus** = false
- size_t **selectedStartPoint** = -1
- size_t **selectedEndPoint** = -1
- size_t **selectedRoad** = -1
- size_t **selectedVehicle** = -1
- Uint32 **pathFinderOpenTimeStamp**
- bool **finalisingErrorWindow** = false
- const bool **debugMode** = false
- bool **simulationStart** = false

Marker for simulation starting.

- bool **simulationPaused** = false
  Marker for paused simulations.

- bool **editorLock** = false
  The marker for the editor locking.

- size_t **coloringCounter** = 0
  The road path coloring counter.

- std::vector< size_t > **path**
  The current path for pathfinder debugger window.

- **WorkWindow** * **workingWindow**
  The current binded working window.

- SDL_Window * **window**
  The current binded OpenGL window scene.

- **Render** * **windowRender**
  The current binded render.

- **ObjectStorage** * **objectStorage**
  The main preloaded object storage.

- **EventListener** * **eventListener**
  The current working event listener.

- **Animator** * **animator**
  The actual animator class.

Traffic Simulation

- **GUI** (void)
  *The **GUI** constructor.*

- **~GUI** (void)
  *The **GUI** destructor.*

- void **bind** (**WorkWindow** *currentWindow)
  *Bind the **GUI** system to the current working window.*

- void **draw** ()
  *The main **GUI** function.*

- void **render** ()

***Render** the **GUI** graphics in the render cycle.*

- void **eventHandler** (SDL_Event *event)
  ***GUI** event listener.*

- bool **isKeyboardCaptured** ()
  *The **GUI** want to use the keyboard events?*

- bool **isMouseCaptured** ()
  *The **GUI** want to use the mouse events?*

- void **close** ()
  Application closing checker window.

- void **showEndpointInfo** (size_t pointModelID)
  Show the selected endpoint info.

- void **showRoadInfo** (size_t edgeModelID)
  Show the selected road info.

- void **showVehicleInfo** (size_t vehicleModelID)
  Show the selected vehicle data.

- void **resetInfoWindow** ()
  Reset the selected item info window content.

- void **quickSave** ()
  QuickSave function.

- static bool **init** (SDL_Window ***window**)
  IMgui initialization and bind to the current OpenGL window.

- static void **clean** ()
  Stop the IMgui system.

- void **mainMenuBar** ()
  The main menu bar description.

- void **windowHandler** ()
  The widows opening and closing system.

- void **fpsGraph** ()
  The FPS graph.

- void **itemList** ()
  Item list.

- void **carList** (**Point** *point)
  Car list.

- void **showHelpMarker** (const char *desc)
  Show a help when hovered.

- void **endpointSelector** (**Point** *point)
  Endpoint selector generator for start point properties.

- void **closeAllWindows** (std::string exeption)
  Close all windows.

Detailed Description

GUI Descriptor class. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

GUI::GUI (void)

The **GUI** constructor.

GUI::~GUI (void)

The **GUI** destructor.

Member Function Documentation

void GUI::aboutWindow () `[private]`

Window for the about tab.

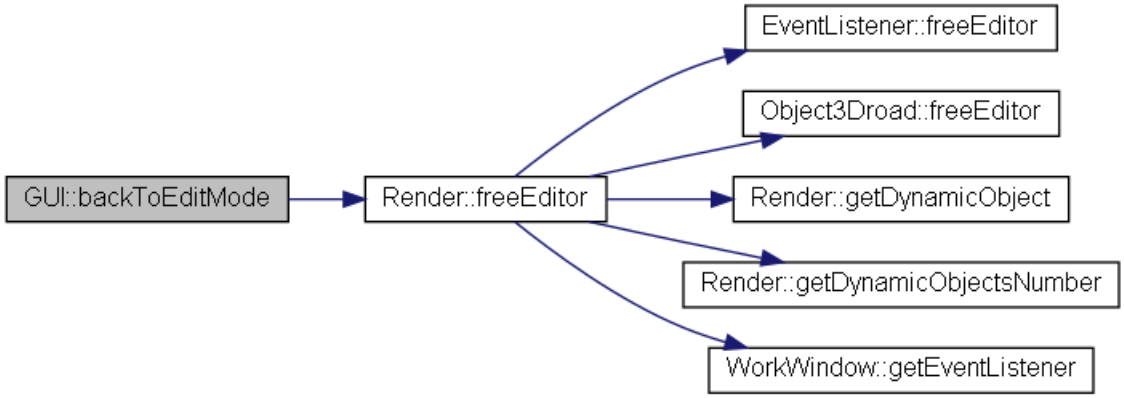
Here is the call graph for this function:

Here is the caller graph for this function:

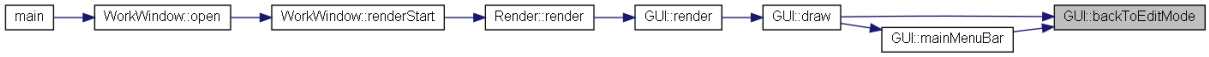
void GUI::backToEditMode () `[private]`

Back to the edit mode.

Here is the call graph for this function:



Here is the caller graph for this function:



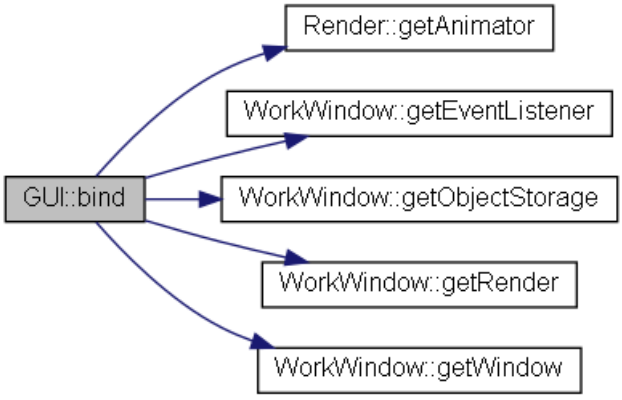
void GUI::bind (WorkWindow * *currentWindow*)

Bind the **GUI** system to the current working window.

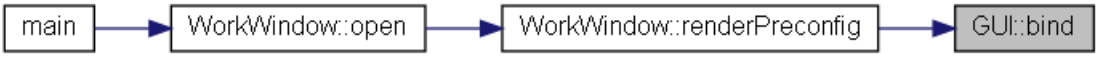
Parameters

currentWindow	The current working window.

Here is the call graph for this function:



Here is the caller graph for this function:



void GUI::carList (Point * *point*) `[private]`

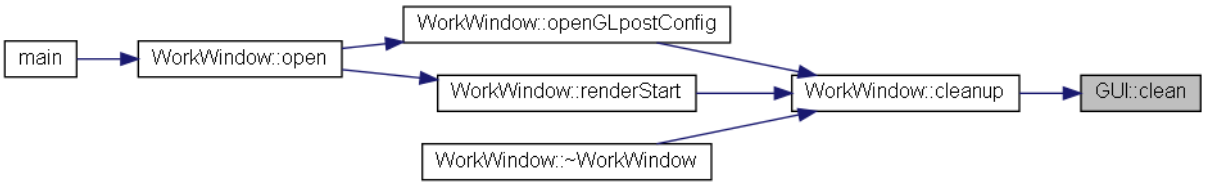
Car list.

Here is the caller graph for this function:

void GUI::clean () `[static]`

Stop the IMgui system.

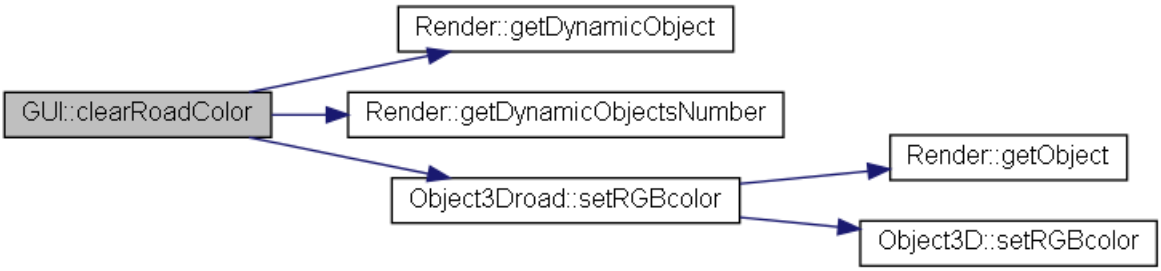
Here is the caller graph for this function:



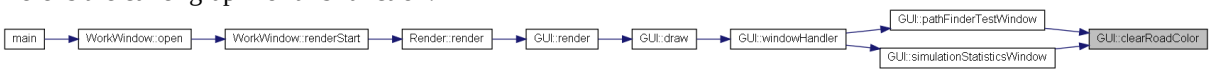
void GUI::clearRoadColor () `[private]`

Clear the road colors.

Here is the call graph for this function:



Here is the caller graph for this function:



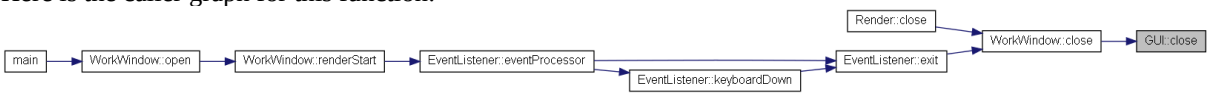
void GUI::close ()

Application closing checker window.

Returns

The user want to close?

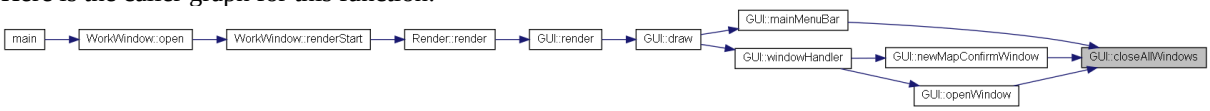
Here is the caller graph for this function:



void GUI::closeAllWindows (std::string *exeption*) `[private]`

Close all windows.

Here is the caller graph for this function:



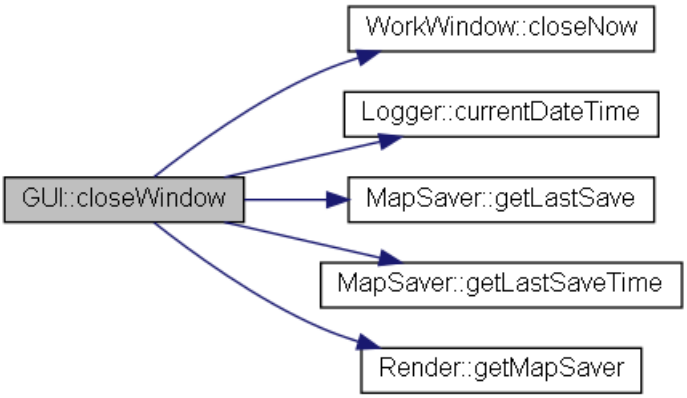
void GUI::closeWindow () `[private]`

Application closing checker window.

Returns

The user want to close?

Here is the call graph for this function:



Here is the caller graph for this function:

void GUI::controlsWindow () [private]

Window for the controls tutorial.

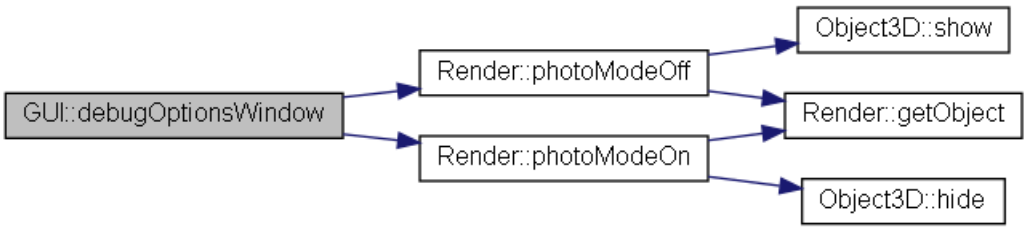
Here is the call graph for this function:

Here is the caller graph for this function:

void GUI::debugOptionsWindow () [private]

Window for the debug options.

Here is the call graph for this function:



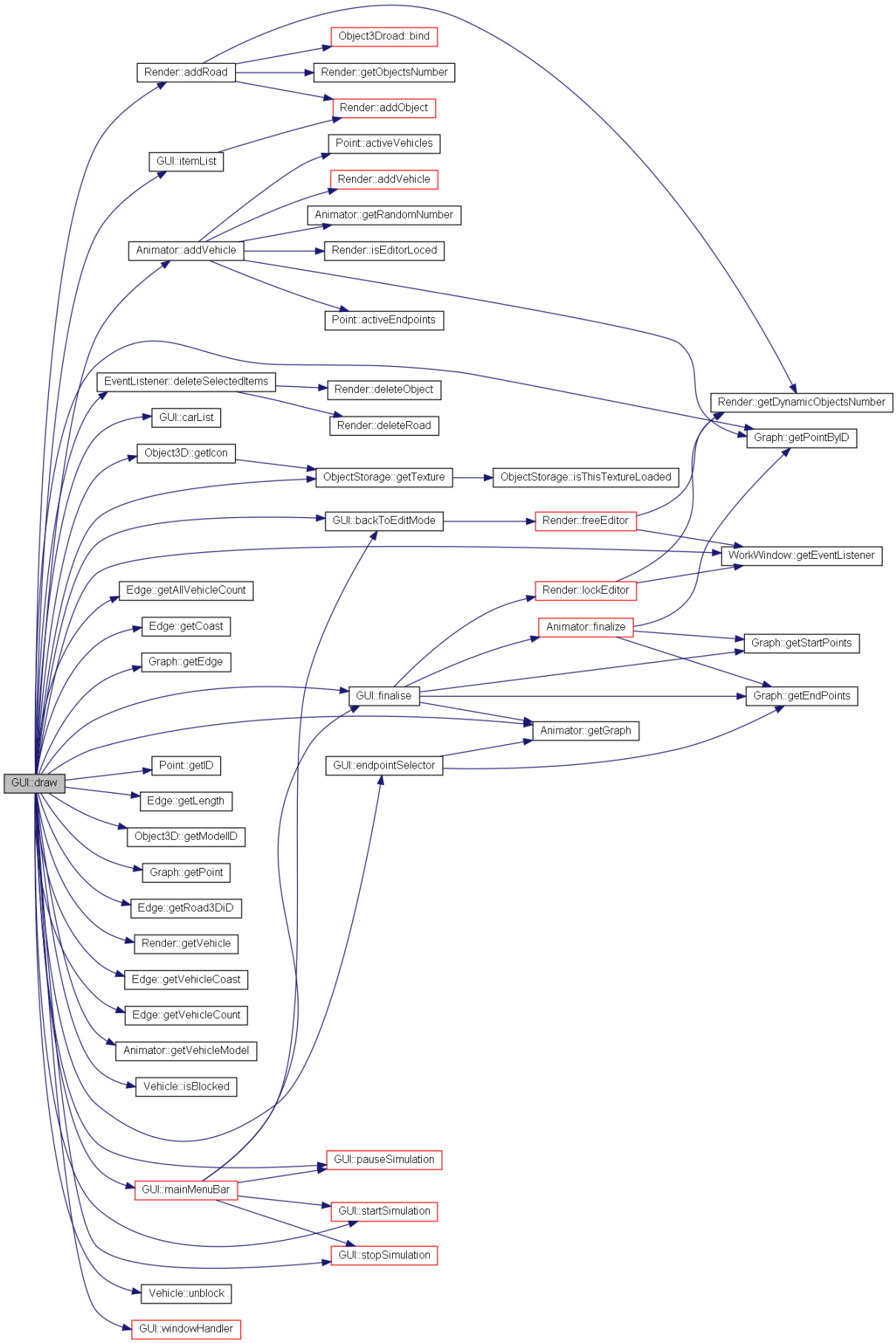
Here is the caller graph for this function:

void GUI::draw ()

The main **GUI** function.

Set the **GUI** structure.

Here is the call graph for this function:



Here is the caller graph for this function:

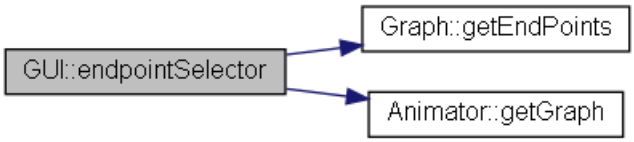
void GUI::endpointSelector (Point * *point*) `[private]`

Endpoint selector generator for start point properties.

Parameters

startPointID	The current start point.

Here is the call graph for this function:



Here is the caller graph for this function:

void GUI::eventHandler (SDL_Event * *event*)

GUI event listener.

Parameters

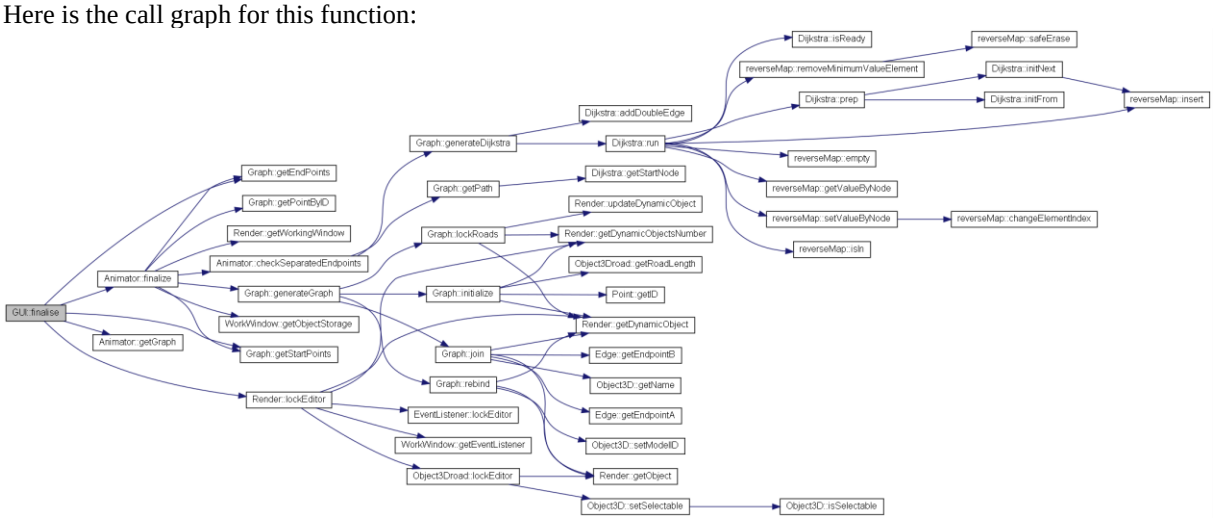
event	SDL Window events.

Here is the caller graph for this function:

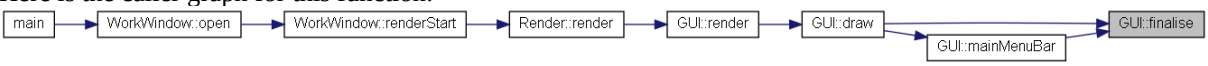
void GUI::finalise () `[private]`

Finalize the map.

Here is the call graph for this function:



Here is the caller graph for this function:



void GUI::finaliseWarningWindow () `[private]`

Show the finalizing warning window.

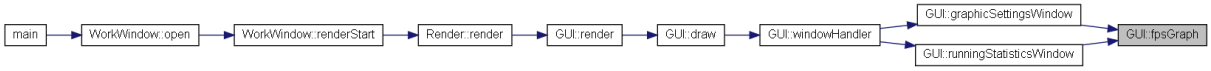
Here is the caller graph for this function:

void GUI::fpsGraph () [private]

The FPS graph.

Here is the call graph for this function:

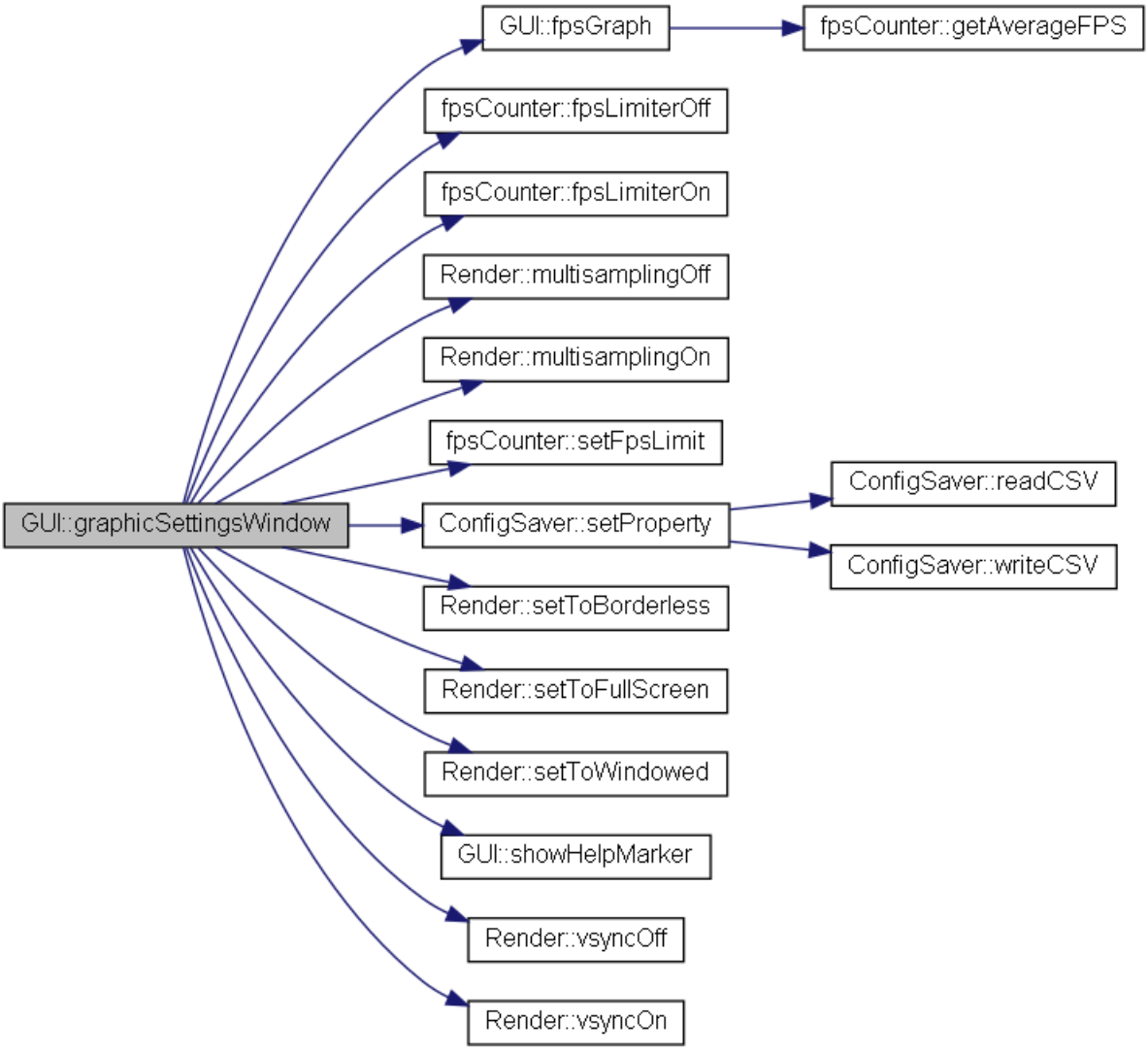
Here is the caller graph for this function:



void GUI::graphicSettingsWindow () [private]

Window descriptor for the graphic settings window.

Here is the call graph for this function:



Here is the caller graph for this function:

void GUI::helpWindow () [private]

Show help window.

Here is the call graph for this function:

Here is the caller graph for this function:

bool GUI::init (SDL_Window * *window*) [static]

IMgui initialization and bind to the current OpenGL window.

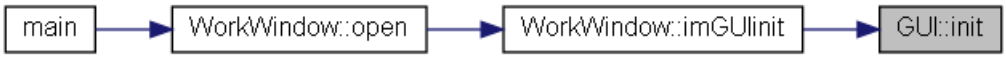
Parameters

window	The current OpenGL window for the **GUI**.

Returns

The success or the error codes.

Here is the caller graph for this function:



bool GUI::isKeyboardCaptured ()

The **GUI** want to use the keyboard events?

Returns

True if the **GUI** want to use the keyboard events.

Here is the caller graph for this function:

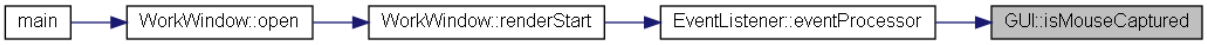
bool GUI::isMouseCaptured ()

The **GUI** want to use the mouse events?

Returns

True if the **GUI** want to use the mouse events.

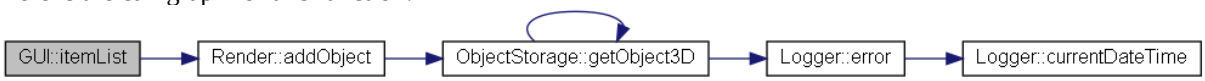
Here is the caller graph for this function:



void GUI::itemList () [private]

Item list.

Here is the call graph for this function:

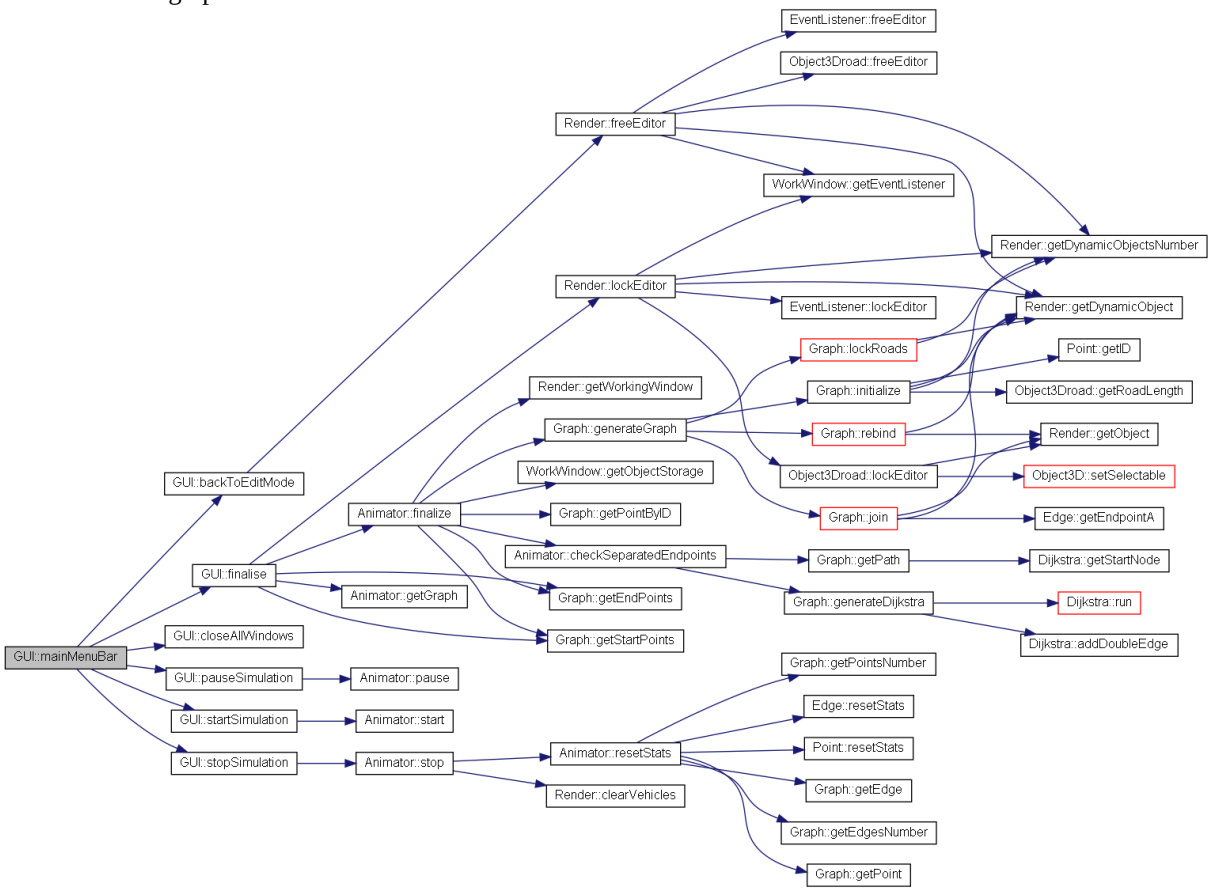


Here is the caller graph for this function:

void GUI::mainMenuBar () [private]

The main menu bar description.

Here is the call graph for this function:

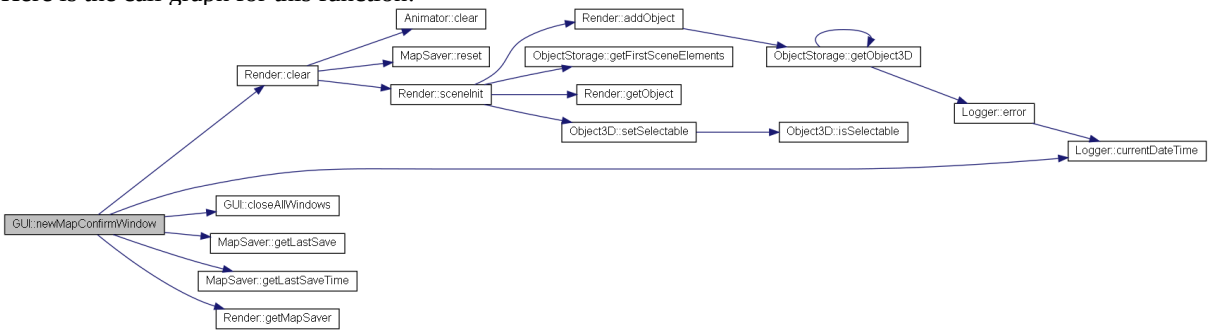


Here is the caller graph for this function:

void GUI::newMapConfirmWindow () [private]

Window for the new map confirmation.

Here is the call graph for this function:

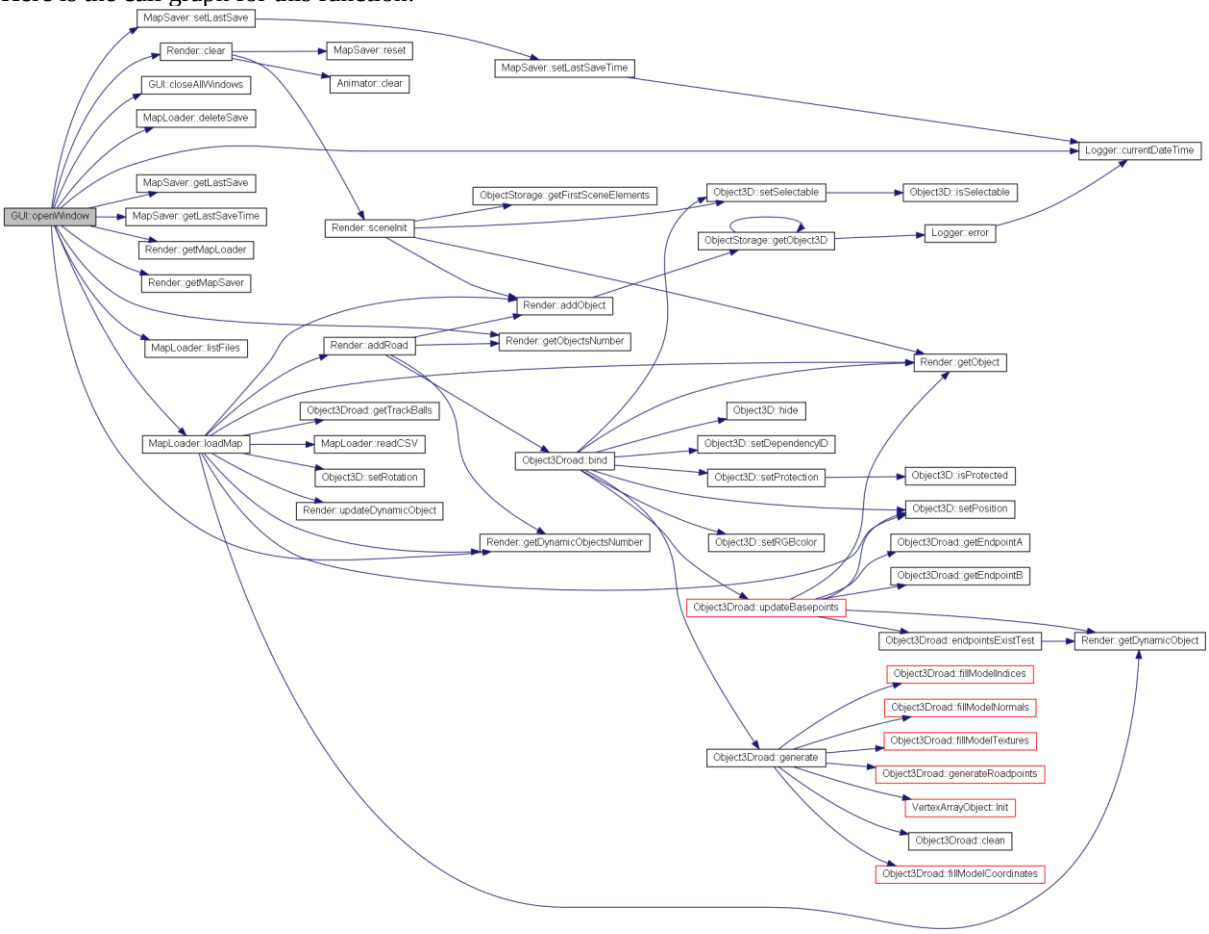


Here is the caller graph for this function:

void GUI::openWindow () [private]

Description of the opening window.

Here is the call graph for this function:

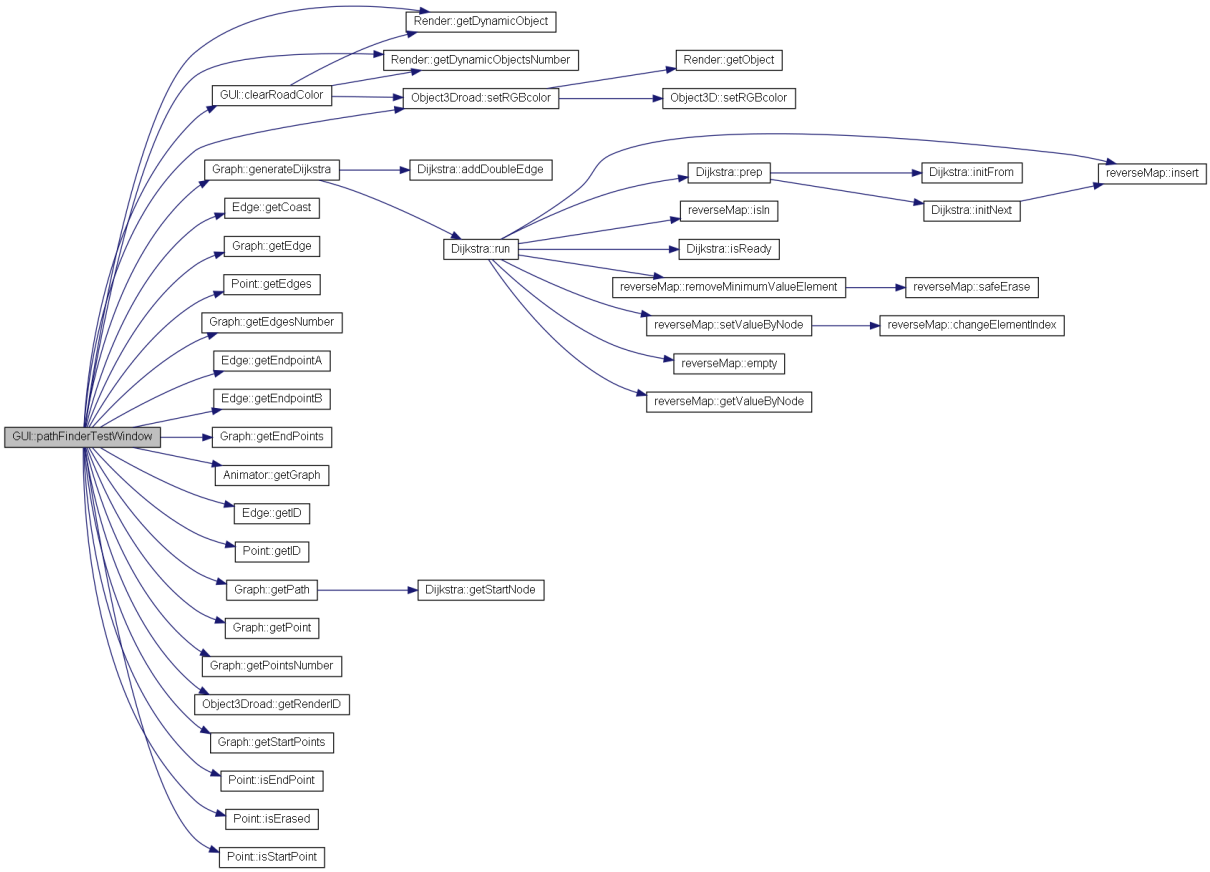


Here is the caller graph for this function:

void GUI::pathFinderTestWindow () [private]

Pathfinder test window.

Here is the call graph for this function:



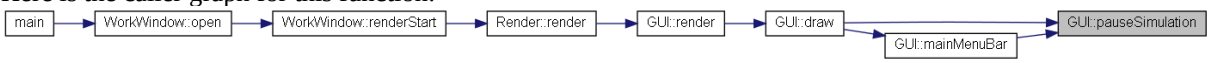
Here is the caller graph for this function:

void GUI::pauseSimulation () `[private]`

Pause running simulation.

Here is the call graph for this function:

Here is the caller graph for this function:



void GUI::quickSave ()

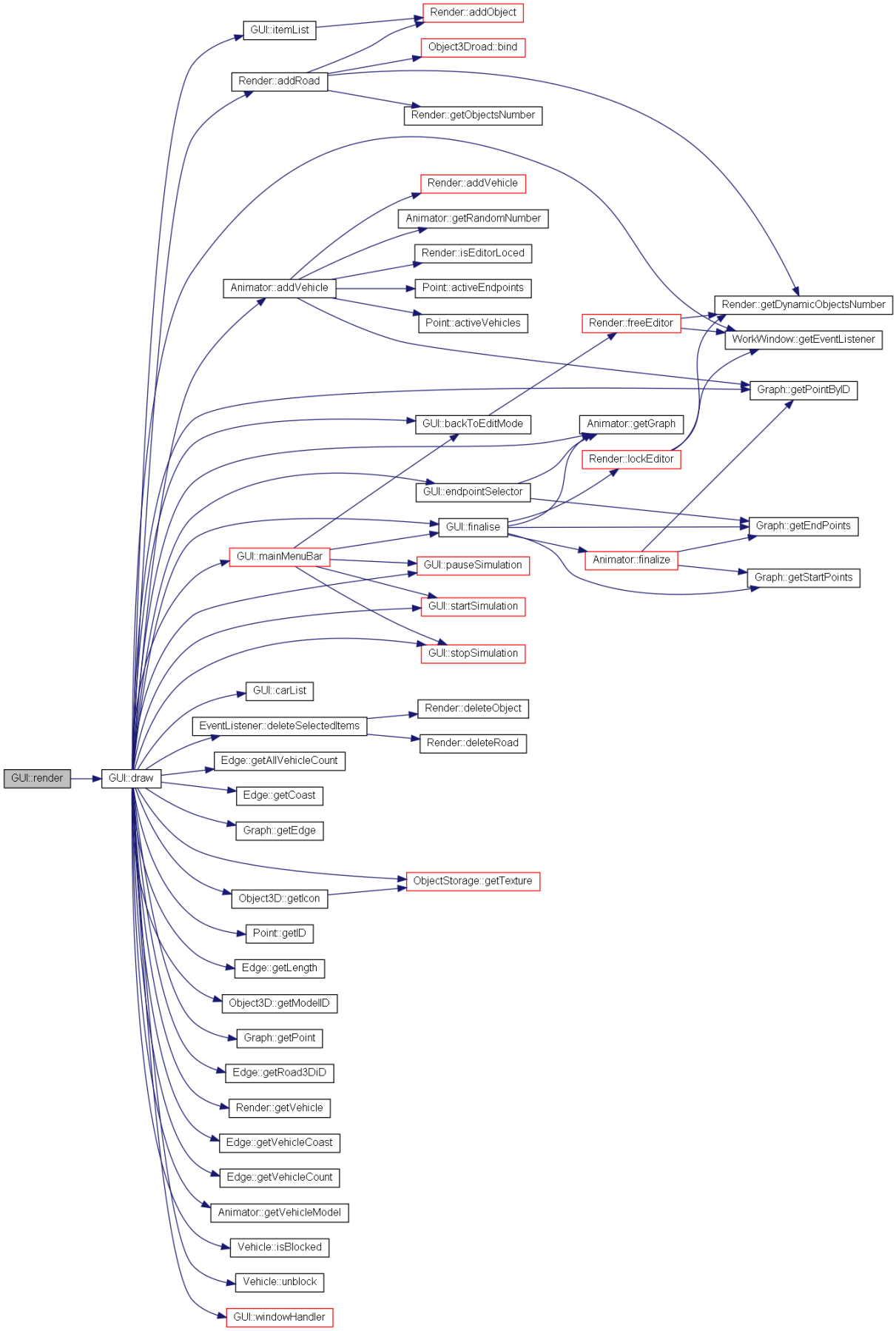
QuickSave function.

Here is the caller graph for this function:

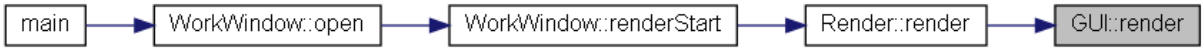
void GUI::render ()

Render the **GUI** graphics in the render cycle.

Here is the call graph for this function:



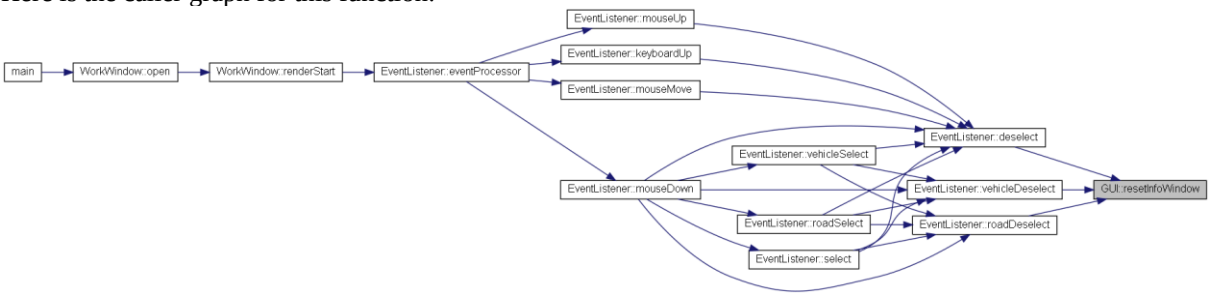
Here is the caller graph for this function:



void GUI::resetInfoWindow ()

Reset the selected item info window content.

Here is the caller graph for this function:

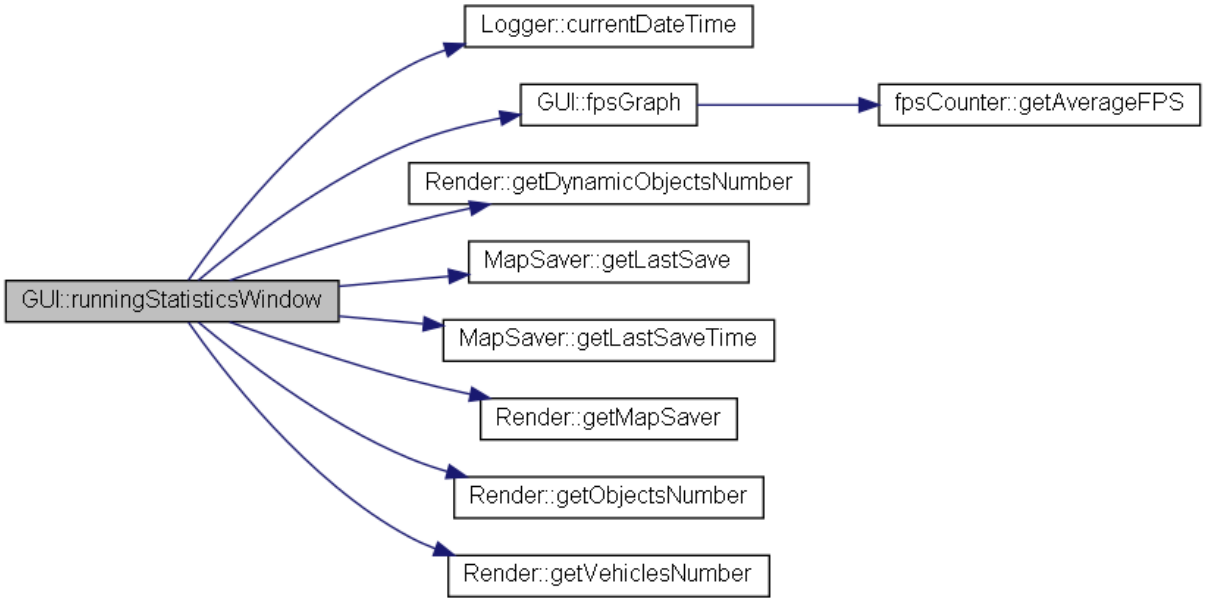


void GUI::runningStatisticsWindow () [private]

Window for the running statistics.

Window descriptor for the graphic settings window.

Here is the call graph for this function:

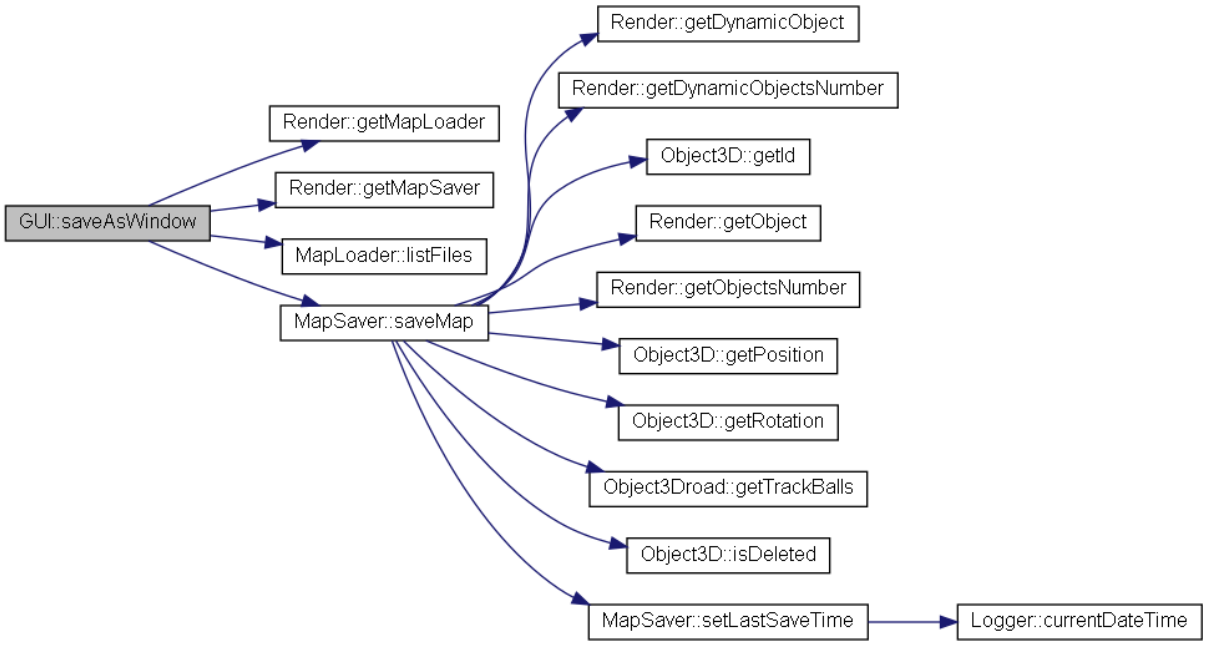


Here is the caller graph for this function:

void GUI::saveAsWindow () [private]

the save as window descriptor.

Here is the call graph for this function:

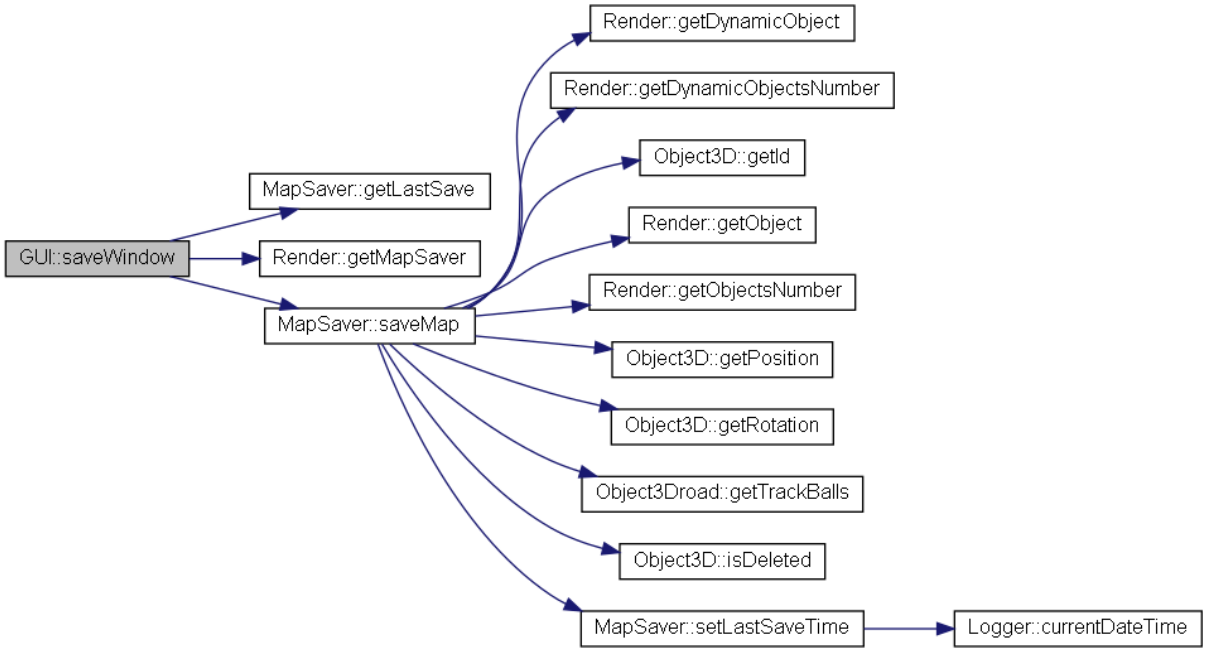


Here is the caller graph for this function:

void GUI::saveWindow () `[private]`

the save window descriptor.

Here is the call graph for this function:



Here is the caller graph for this function:

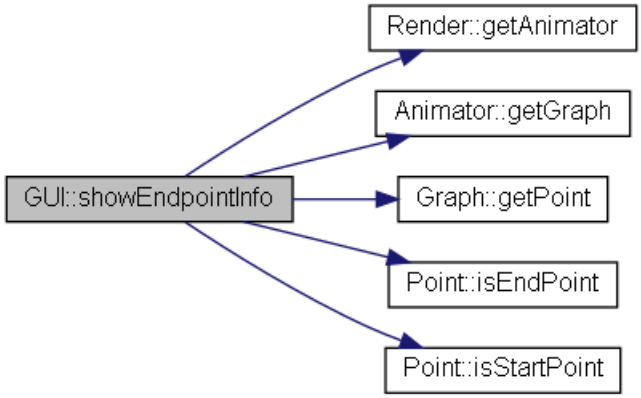
void GUI::showEndpointInfo (size_t *pointModelID*)

Show the selected endpoint info.

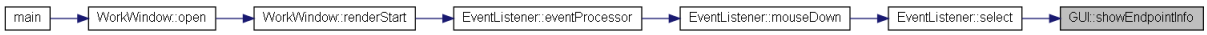
Parameters

pointModelID	The selected point. (Model object!)

Here is the call graph for this function:



Here is the caller graph for this function:



void GUI::showHelpMarker (const char * *desc*) `[private]`

Show a help when hovered.

Parameters

desc	The help text.

Here is the caller graph for this function:

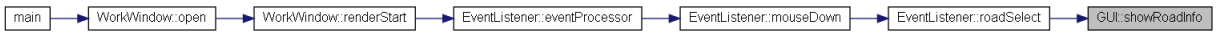
void GUI::showRoadInfo (size_t *edgeModelID*)

Show the selected road info.

Parameters

edgeModelID	The selected edge. (Model object!)

Here is the caller graph for this function:



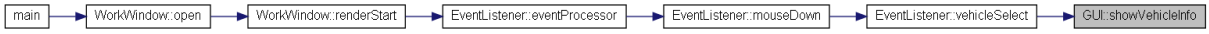
void GUI::showVehicleInfo (size_t *vehicleModelID*)

Show the selected vehicle data.

Parameters

vehicleModelID	The selected vehicle. (Model object!)

Here is the caller graph for this function:



void GUI::simulationSettingsWindow () `[private]`

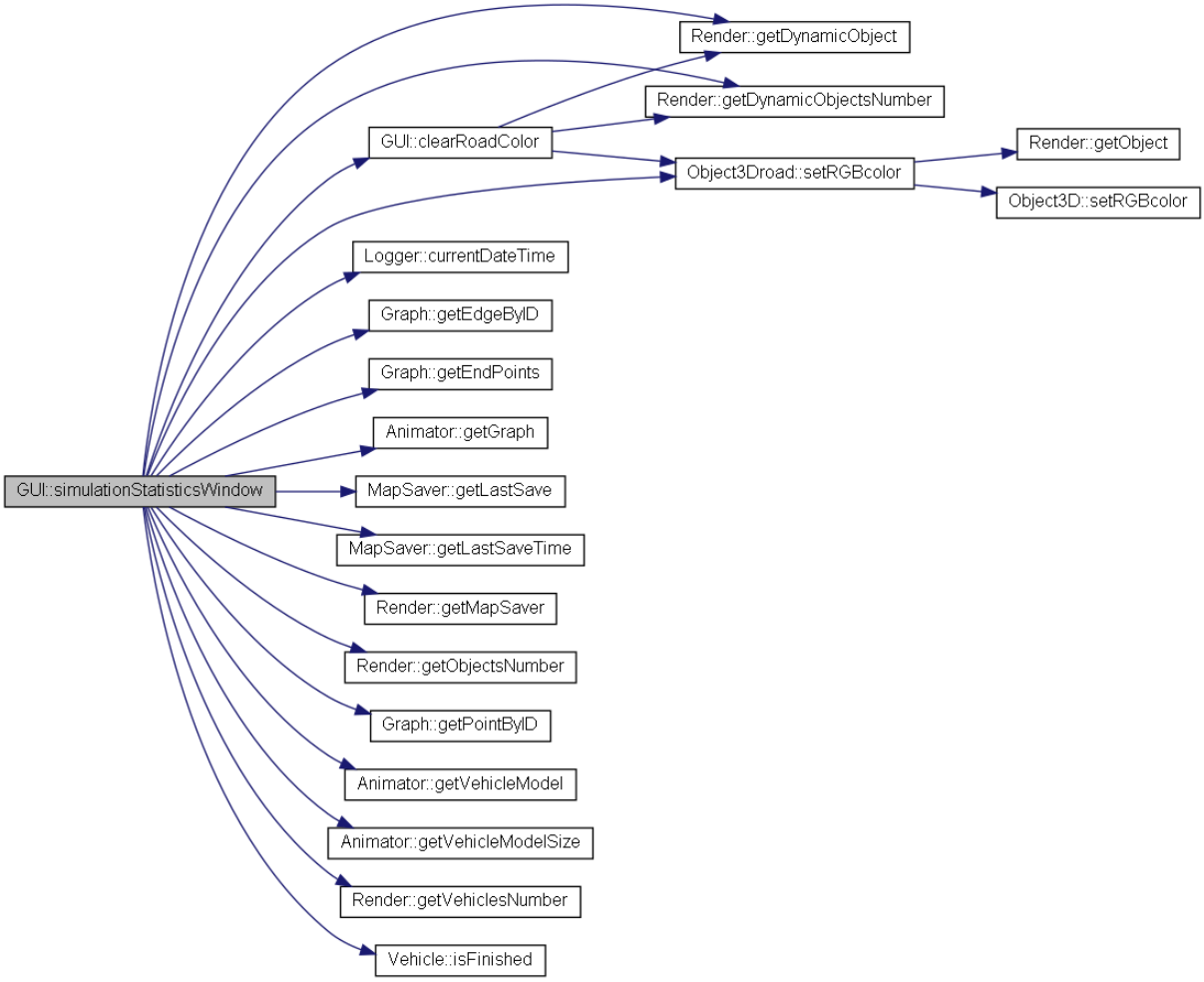
The window for the simulation settings.

Here is the caller graph for this function:

void GUI::simulationStatisticsWindow () [private]

Window for SimulationStatistics.

Here is the call graph for this function:



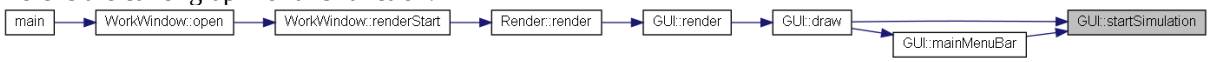
Here is the caller graph for this function:

void GUI::startSimulation () [private]

Start simulation.

Here is the call graph for this function:

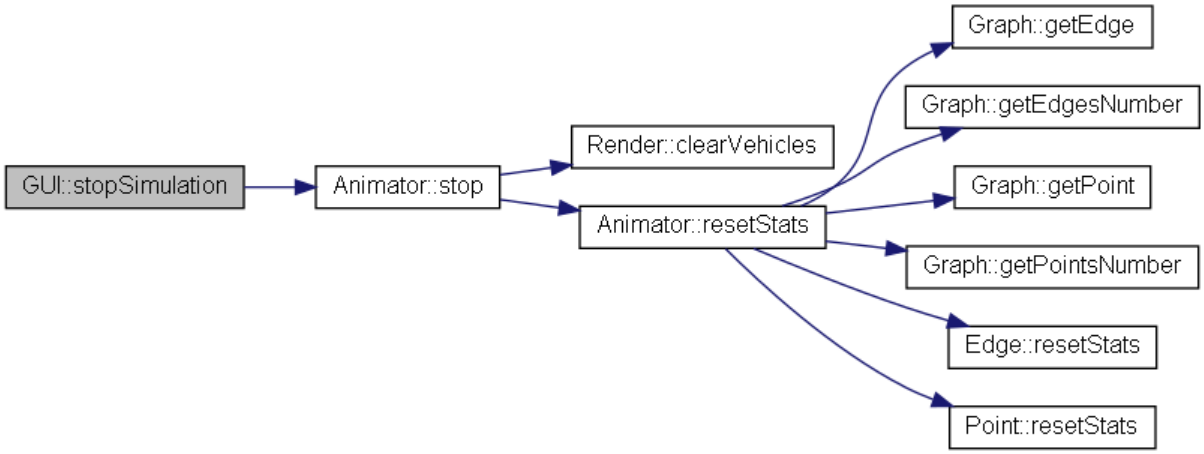
Here is the caller graph for this function:



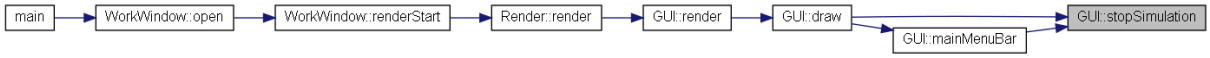
void GUI::stopSimulation () [private]

Stop running simulation.

Here is the call graph for this function:



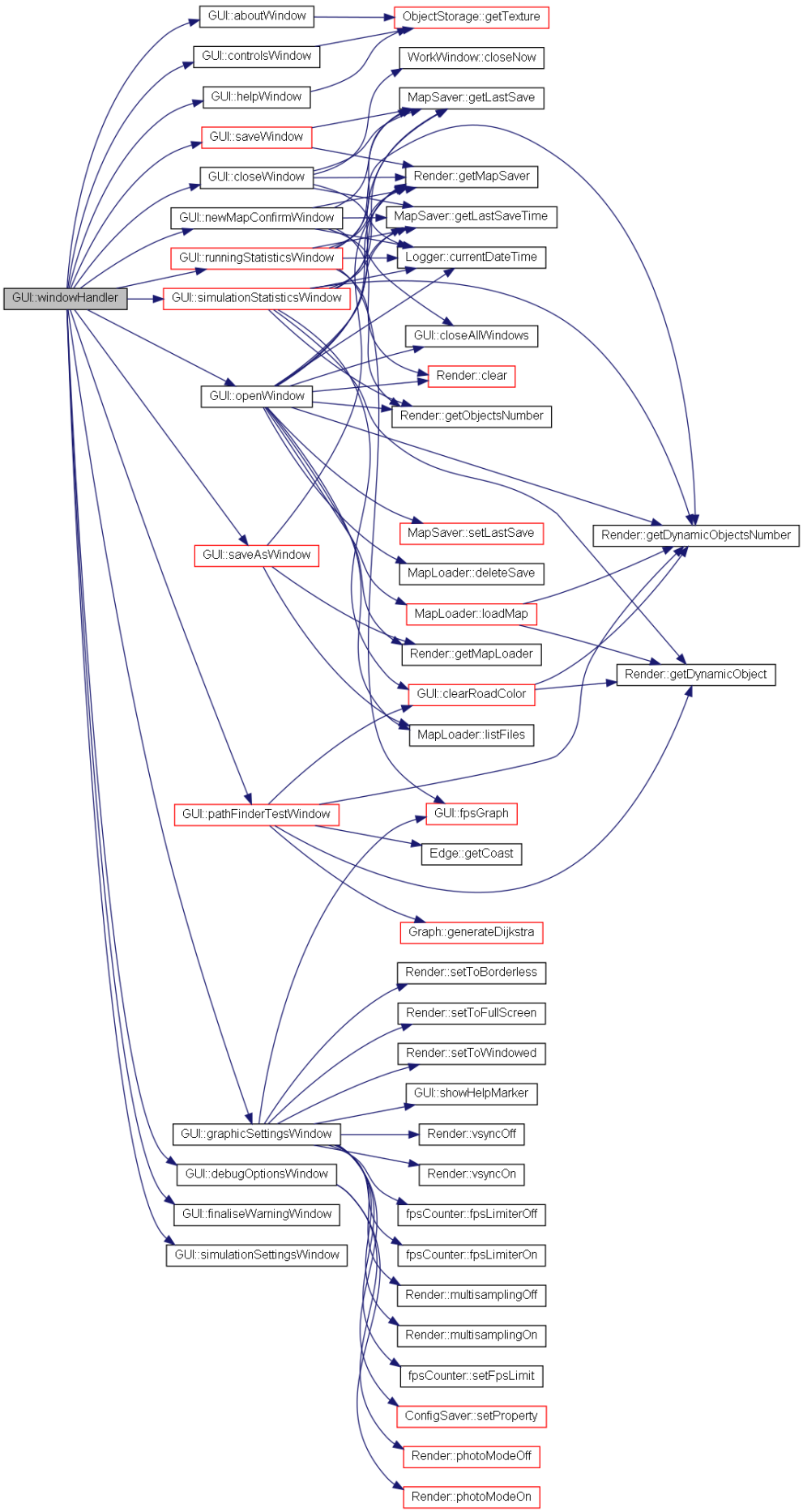
Here is the caller graph for this function:



void GUI::windowHandler () [private]

The widows opening and closing system.

Here is the call graph for this function:


GUI::windowHandler
GUI::aboutWindow
GUI::controlsWindow
GUI::helpWindow
GUI::saveWindow
GUI::closeWindow
GUI::newMapConfirmWindow
GUI::runningStatisticsWindow
GUI::simulationStatisticsWindow
GUI::openWindow
GUI::saveAsWindow
GUI::pathFinderTestWindow
GUI::graphicSettingsWindow
GUI::debugOptionsWindow
GUI::finaliseWarningWindow
GUI::simulationSettingsWindow
ObjectStorage::getTexture
WorkWindow::closeNow
MapSaver::getLastSave
Render::getMapSaver
MapSaver::getLastSaveTime
Logger::currentDateTime
GUI::closeAllWindows
Render::clear
Render::getObjectsNumber
MapSaver::setLastSave
MapLoader::deleteSave
MapLoader::loadMap
Render::getMapLoader
GUI::clearRoadColor
MapLoader::listFiles
Render::getDynamicObjectsNumber
Render::getDynamicObject
GUI::fpsGraph
Edge::getCoast
Graph::generateDijkstra
Render::setToBorderless
Render::setToFullScreen
Render::setToWindowed
GUI::showHelpMarker
Render::vsyncOff
Render::vsyncOn
fpsCounter::fpsLimiterOff
fpsCounter::fpsLimiterOn
Render::multisamplingOff
Render::multisamplingOn
fpsCounter::setFpsLimit
ConfigSaver::setProperty
Render::photoModeOff
Render::photoModeOn

Here is the caller graph for this function:

Field Documentation

bool GUI::aboutWindowStatus = false [private]

Animator* GUI::animator [private]

The actual animator class.

bool GUI::closingCheckerWindowStatus = false [private]

size_t GUI::coloringCounter = 0 [private]

The road path coloring counter.

For pathfinder debugger window.

bool GUI::controlsWindowStatus = false [private]

const bool GUI::debugMode = false [private]

bool GUI::debugOptionsWindowStatus = false [private]

bool GUI::editorLock = false [private]

The marker for the editor locking.

EventListener* GUI::eventListener [private]

The current working event listener.

bool GUI::finaliseWarningWindowStatus = false `[private]`

bool GUI::finalisingErrorWindow = false `[private]`

ImGuiWindowFlags GUI::graphicSettingsWindowFlag = 0 `[private]`

bool GUI::graphicSettingsWindowStatus = false `[private]`

bool GUI::helpWindowStatus = false `[private]`

bool GUI::ImGuiSettingsWindowStatus = false `[private]`

bool GUI::mapEditorWindow = true `[private]`

ImGuiWindowFlags GUI::mapEditorWindowFlag = 0 `[private]`

bool GUI::newMapConfirmWindowStatus = false `[private]`

ObjectStorage* GUI::objectStorage `[private]`

The main preloaded object storage.

bool GUI::openWindowStatus = false `[private]`

std::vector<size_t> GUI::path `[private]`

The current path for pathfinder debugger window.

Uint32 GUI::pathFinderOpenTimeStamp `[private]`

ImGuiWindowFlags GUI::pathFinderTestWindowFlag = 0 `[private]`

bool GUI::pathFinderTestWindowStatus = false `[private]`

bool GUI::runningStatisticsWindowStatus = false `[private]`

bool GUI::saveAsWindowStatus = false `[private]`

bool GUI::saveWindowStatus = false `[private]`

size_t GUI::selectedEndPoint = -1 `[private]`

size_t GUI::selectedRoad = -1 `[private]`

size_t GUI::selectedStartPoint = -1 `[private]`

size_t GUI::selectedVehicle = -1 `[private]`

bool GUI::simulationPaused = false `[private]`

Marker for paused simulations.

bool GUI::simulationSettingsWindowStatus = false `[private]`

bool GUI::simulationStart = false `[private]`

Marker for simulation starting.

bool GUI::simulationStatisticsWindowStatus = false `[private]`

bool GUI::simulationWindow = false `[private]`

ImGuiWindowFlags GUI::simulationWindowFlag = 0 `[private]`

ImGuiWindowFlags GUI::trafficSimulationWindowFlag = 0 `[private]`

SDL_Window* GUI::window `[private]`

The current binded OpenGL window scene.

Render* GUI::windowRender `[private]`

The current binded render.

WorkWindow* GUI::workingWindow `[private]`

The current binded working window.

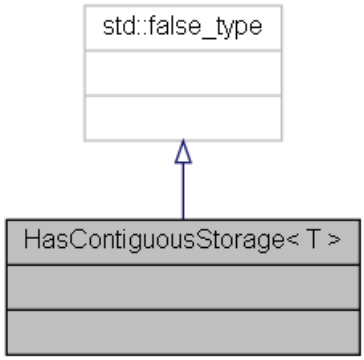
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**GUI.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**GUI.cpp**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**GUIWindows.cpp**

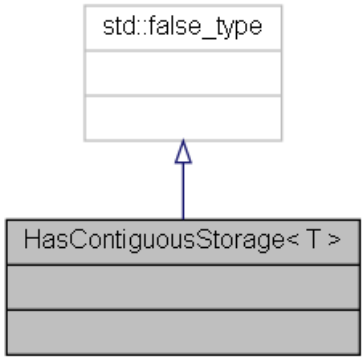
HasContiguousStorage< T > Struct Template Reference

```
#include <GLconversions.hpp>
```

Inheritance diagram for HasContiguousStorage< T >:



Collaboration diagram for HasContiguousStorage< T >:



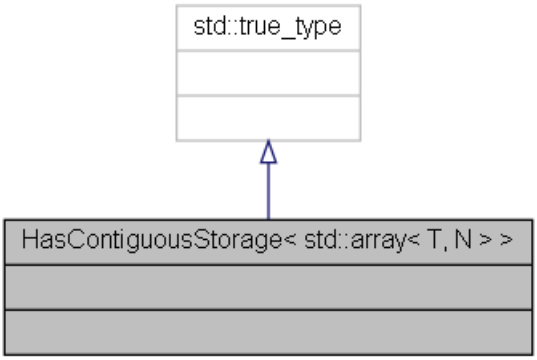
The documentation for this struct was generated from the following file:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**GLconversions.hpp**

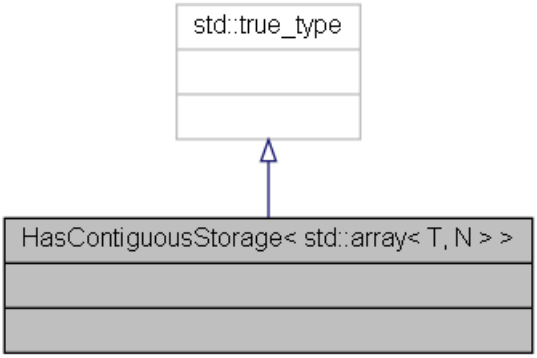
HasContiguousStorage< std::array< T, N > > Struct Template Reference

```
#include <GLconversions.hpp>
```

Inheritance diagram for HasContiguousStorage< std::array< T, N > >:



Collaboration diagram for HasContiguousStorage< std::array< T, N > >:



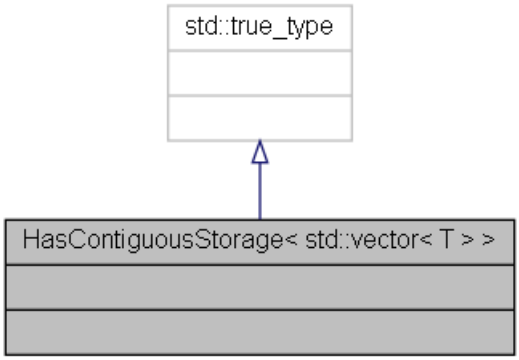
The documentation for this struct was generated from the following file:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**GLconversions.hpp**

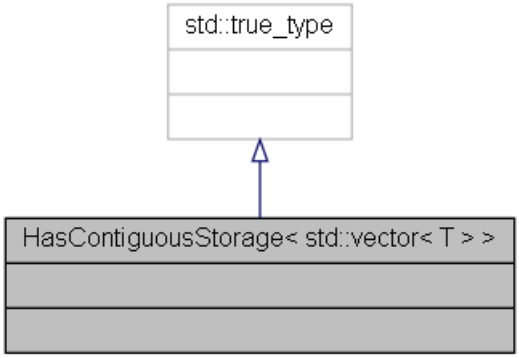
HasContiguousStorage< std::vector< T > > Struct Template Reference

```
#include <GLconversions.hpp>
```

Inheritance diagram for HasContiguousStorage< std::vector< T > >:



Collaboration diagram for HasContiguousStorage< std::vector< T > >:



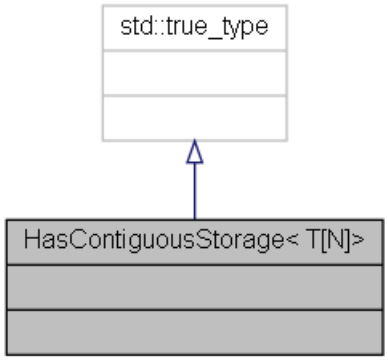
The documentation for this struct was generated from the following file:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**GLconversions.hpp**

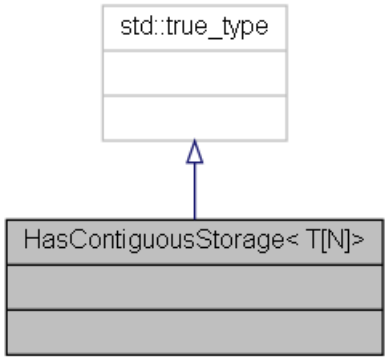
HasContiguousStorage< T[N]> Struct Template Reference

```
#include <GLconversions.hpp>
```

Inheritance diagram for HasContiguousStorage< T[N]>:



Collaboration diagram for HasContiguousStorage< T[N]>:

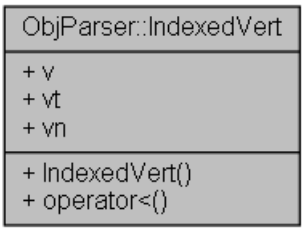


The documentation for this struct was generated from the following file:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**GLconversions.hpp**

ObjParser::IndexedVert Struct Reference

Collaboration diagram for ObjParser::IndexedVert:



Public Member Functions

- **IndexedVert** (int _v, int _vt, int _vn)
- bool **operator<** (const **IndexedVert** &rhs) const

Data Fields

- int **v**
- int **vt**
- int **vn**

Constructor & Destructor Documentation

ObjParser::IndexedVert::IndexedVert (int _*v*, int _*vt*, int _*vn*) `[inline]`

Member Function Documentation

bool ObjParser::IndexedVert::operator< (const IndexedVert & *rhs*) const `[inline]`

Field Documentation

int ObjParser::IndexedVert::v

int ObjParser::IndexedVert::vn

int ObjParser::IndexedVert::vt

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

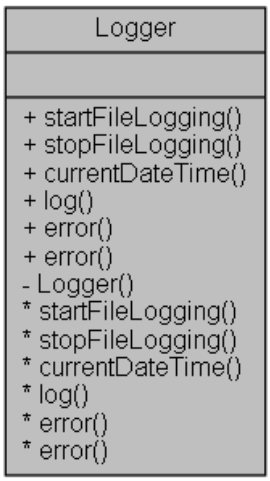
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ObjParser_OGL3.h**

Logger Class Reference

Logging system implementation.

```
#include <Logger.h>
```

Collaboration diagram for Logger:



Static Public Member Functions

- Traffic Simulationstatic void **startFileLogging** ()
  Start and initialize the file logging and open the log files.

- static void **stopFileLogging** ()
  Stop the file logging and close the log files.

- static std::string **currentDateTime** ()
  Get the current date and time in String.

- static void **log** (std::string message)
  Log a message.

- static void **error** (std::string message)
  Log an error.

- static void **error** (std::string message, int errorCode)
  Log an error and error code.

Private Member Functions

- **Logger** ()
  The constructor is private and empty because is a static class.

Detailed Description

Logging system implementation.

Logging system implementation. (Static class.) Contact: `sandorbalazs9402@gmail.com`.

(Static class.)

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

Logger::Logger () `[inline], [private]`

The constructor is private and empty because is a static class.

Member Function Documentation

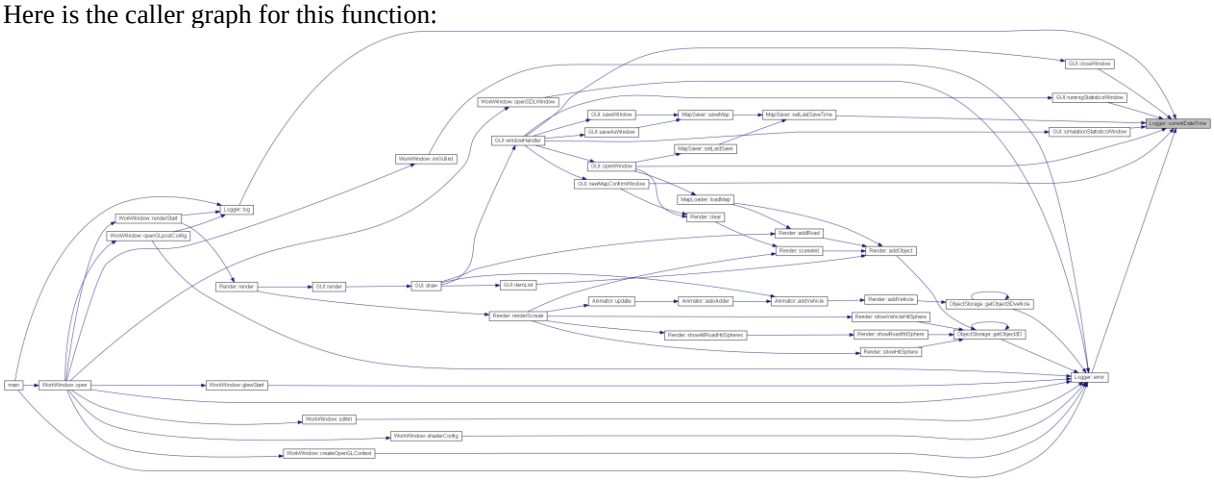
std::string Logger::currentDateTime () `[static]`

Get the current date and time in String.

Returns

The current date and time.

Here is the caller graph for this function:



void Logger::error (std::string *message*) `[static]`

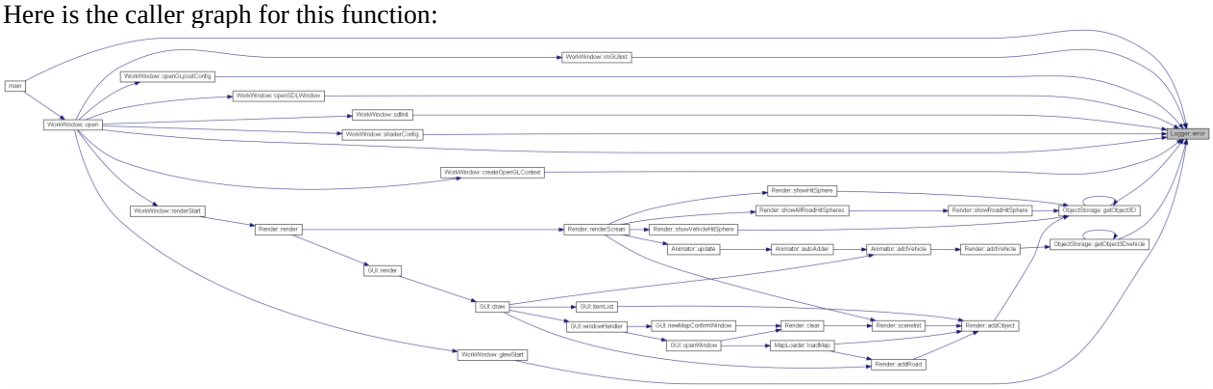
Log an error.

Parameters

message	The error message.

Here is the call graph for this function:

Here is the caller graph for this function:



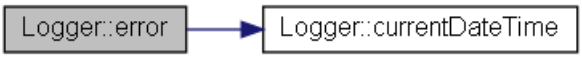
void Logger::error (std::string *message*, int *errorCode*) `[static]`

Log an error and error code.

Parameters

message	The error message.
errorCode	The error code.

Here is the call graph for this function:



void Logger::log (std::string *message*) `[static]`

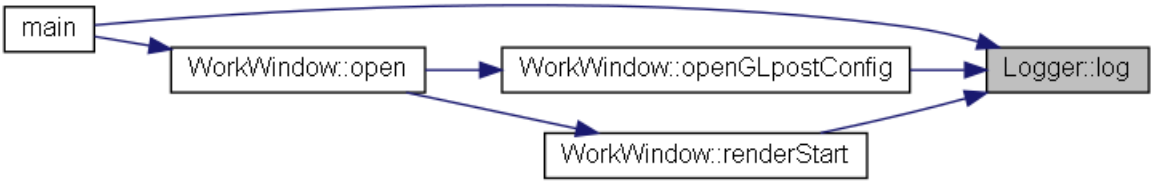
Log a message.

Parameters

message	The log message.

Here is the call graph for this function:

Here is the caller graph for this function:



void Logger::startFileLogging () `[static]`

Start and initialize the file logging and open the log files.

Here is the caller graph for this function:

void Logger::stopFileLogging () `[static]`

Stop the file logging and close the log files.

Here is the caller graph for this function:

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

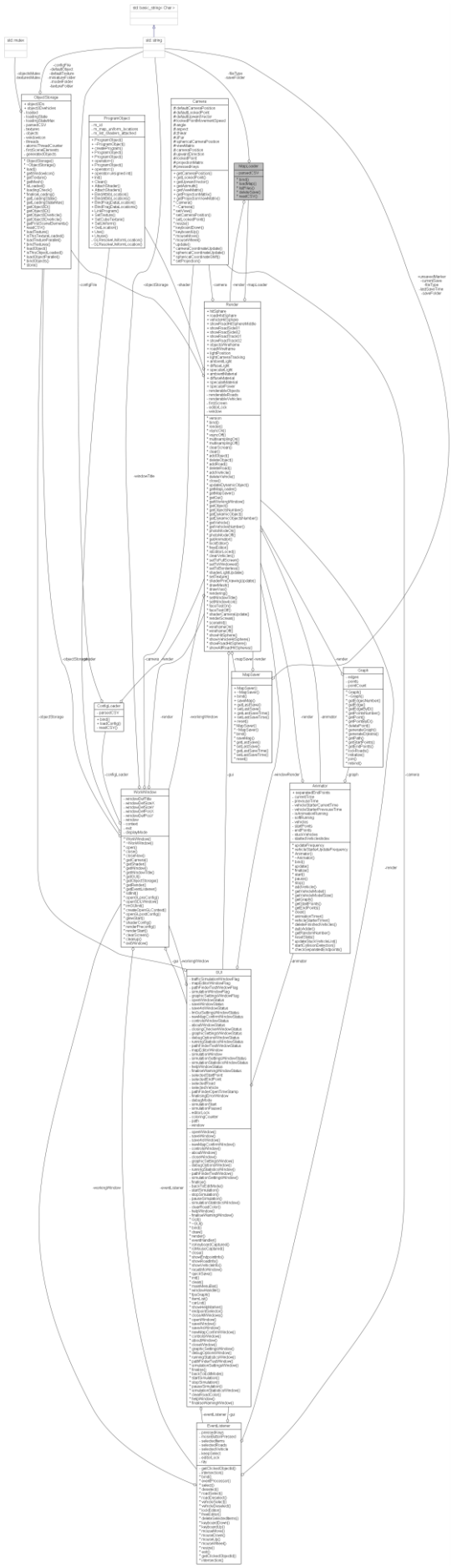
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**Logger.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**Logger.cpp**

MapLoader Class Reference

Map loader class.

```
#include <MapLoader.h>
```

Collaboration diagram for MapLoader:


std::basic_string< Char >
std::string
std::mutex
ObjectStorage
ProgramSpace
Camera
MapLoader
Render
Graph
ConfigLoader
WorkWindow
Animator
GUI
EventListener

Private Attributes

- const std::string **fileType** = ".csv"
  The file type of the save files.

- std::string **saveFolder** = "Saves/"
  The save folder.

- **Render** * **render** = NULL
  The render pointer.

- std::vector< std::vector< std::string > > **parsedCSV**
  The parsed CSV save.

Traffic Simulation

- void **bind** (**Render** ***render**)
  Bind map loader to the render.

- void **loadMap** (std::string fileName)
  Load a map from CSV file.

- std::vector< std::string > **listFiles** ()
  List of the exist maps in the saves folder.

- int **deleteSave** (std::string fileName)
  Delete a saved map.

- void **readCSV** (std::string fileName)
  Read the CSV file to the memory.

Detailed Description

Map loader class.

Map loader implementation. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Member Function Documentation

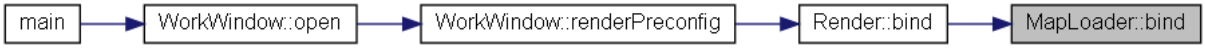
void MapLoader::bind (Render * *render*)

Bind map loader to the render.

Parameters

render	Pointer for the current render.

Here is the caller graph for this function:



int MapLoader::deleteSave (std::string *fileName*)

Delete a saved map.

Parameters

fileName	The deletable map.

Returns

The success code of the deleting.

Here is the caller graph for this function:

std::vector< std::string > MapLoader::listFiles ()

List of the exist maps in the saves folder.

Returns

List of the exist maps.

Here is the caller graph for this function:

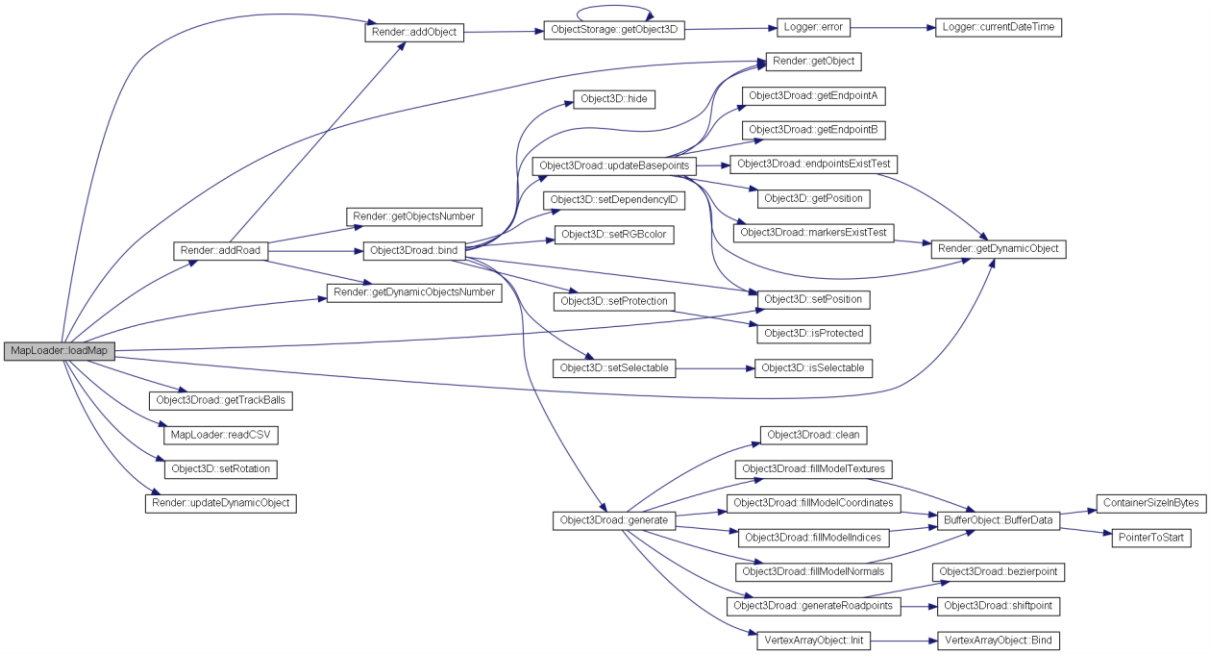
void MapLoader::loadMap (std::string *fileName*)

Load a map from CSV file.

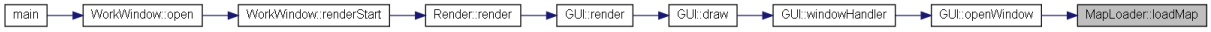
Parameters

fileName	The loadable map name.

Here is the call graph for this function:



Here is the caller graph for this function:



void MapLoader::readCSV (std::string *fileName*) `[private]`

Read the CSV file to the memory.

Parameters

fileName	The readable file name.

Here is the caller graph for this function:

Field Documentation

const std::string MapLoader::fileType = ".csv" `[private]`

The file type of the save files.

std::vector<std::vector<std::string> > MapLoader::parsedCSV `[private]`

The parsed CSV save.

Render* MapLoader::render = NULL `[private]`

The render pointer.

std::string MapLoader::saveFolder = "Saves/" `[private]`

The save folder.

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

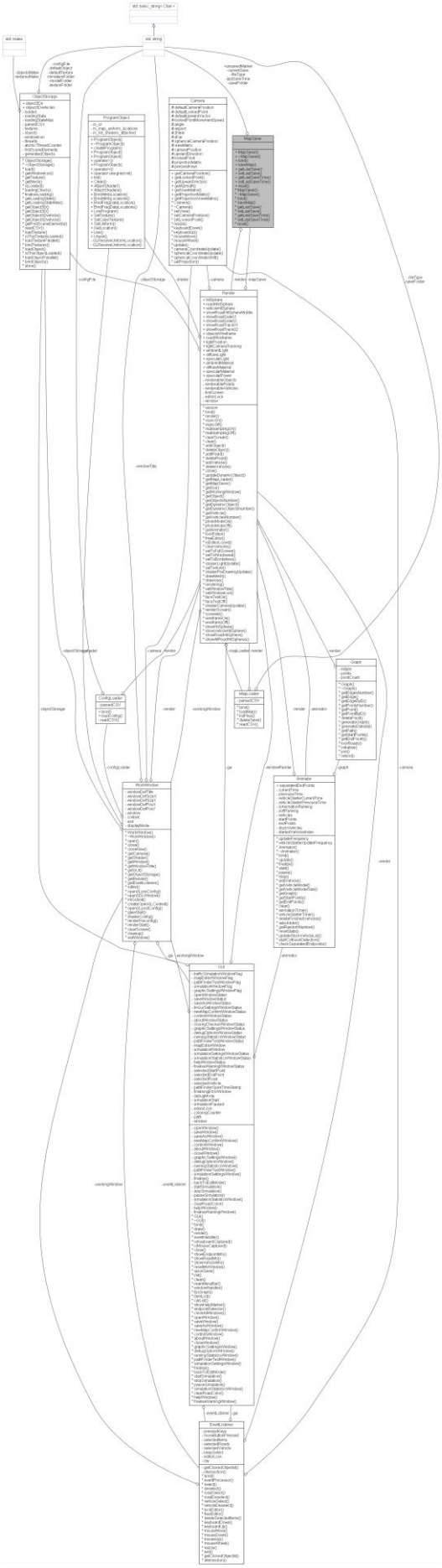
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**MapLoader.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**MapLoader.cpp**

MapSaver Class Reference

Map saver mechanism implementation.

```
#include <MapSaver.h>
```

Collaboration diagram for MapSaver:



Public Member Functions

- Traffic Simulation**MapSaver** (void)
  Map saver constructor.

- **~MapSaver** (void)
  Map saver destructor.

- void **bind** (**Render** ***render**)
  Bind the map saver to the render.

- void **saveMap** (std::string fileName)
  Save the map to the file.

- std::string **getLastSave** ()
  Get the last save name.

- void **setLastSave** (std::string loadedMap)
  Set the last save name.

- std::string **getLastSaveTime** ()
  Get the last save time.

- void **setLastSaveTime** ()
  Set the last save time.

- void **reset** ()
  Reset.

Data Fields

- const std::string **unsavedMarker** = "Unsaved"
  The unsaved map name.

Private Attributes

- const std::string **fileType** = ".csv"
  The file type.

- std::string **currentSave** = **unsavedMarker**
  The current save name.

- const std::string **saveFolder** = "Saves\\"
  The save folder.

- std::string **lastSaveTime** = **unsavedMarker**

The last save time in string.

- **Render * render**
  The current render pointer.

Detailed Description

Map saver mechanism implementation.

Map saver mechanism. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

MapSaver::MapSaver (void)

Map saver constructor.

Parameters

Empty.	

MapSaver::~MapSaver (void)

Map saver destructor.

Parameters

Empty.	

Member Function Documentation

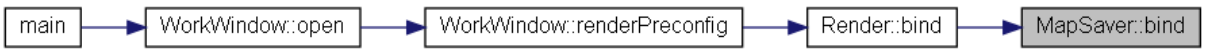
void MapSaver::bind (Render * *render*)

Bind the map saver to the render.

Parameters

render	Pointer to the current render.

Here is the caller graph for this function:



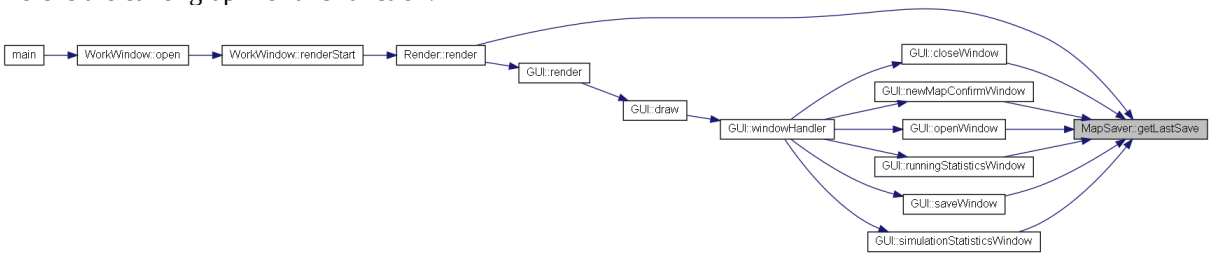
std::string MapSaver::getLastSave ()

Get the last save name.

Returns

The last save name.

Here is the caller graph for this function:



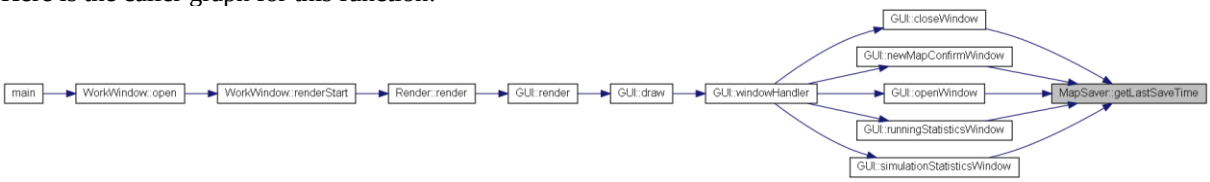
std::string MapSaver::getLastSaveTime ()

Get the last save time.

Returns

The last save time.

Here is the caller graph for this function:



void MapSaver::reset ()

Reset.

Here is the caller graph for this function:

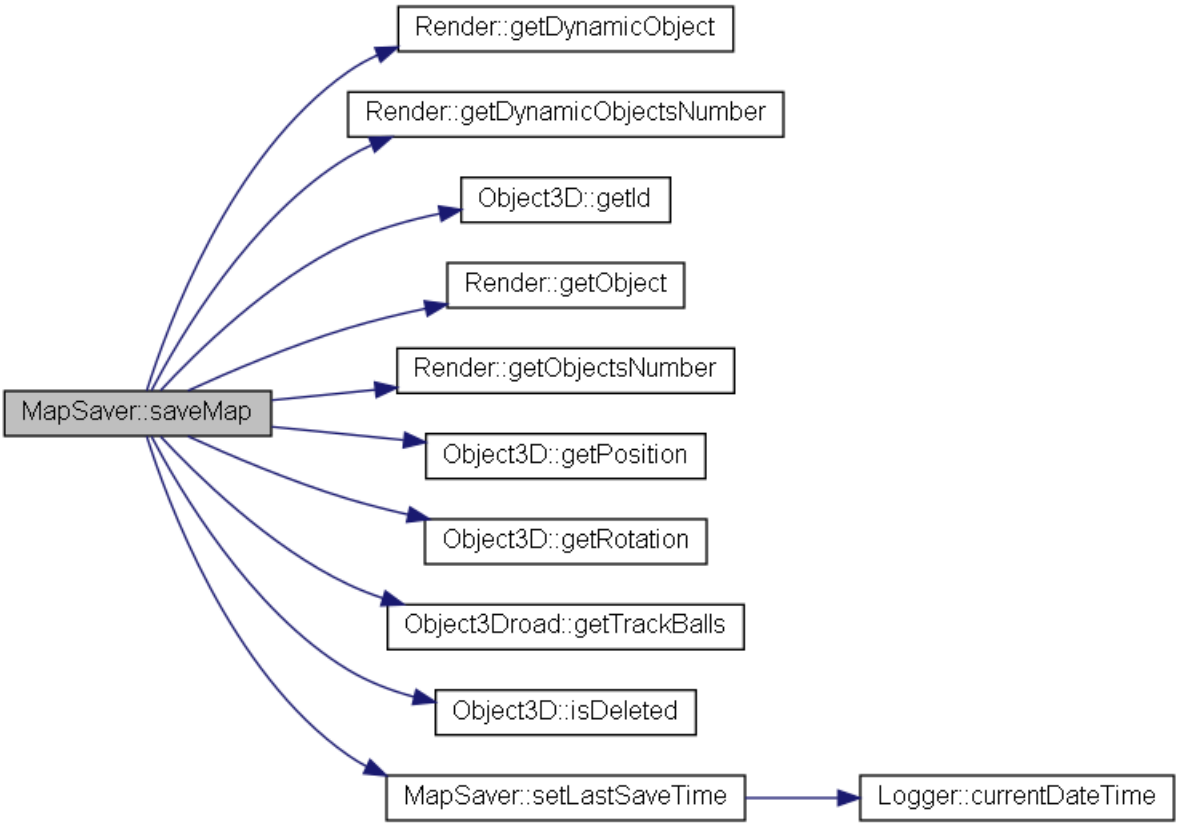
void MapSaver::saveMap (std::string *fileName*)

Save the map to the file.

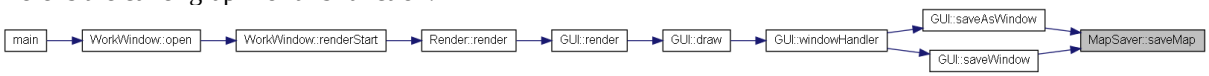
Parameters

fileName	The save file name.

Here is the call graph for this function:



Here is the caller graph for this function:



void MapSaver::setLastSave (std::string *loadedMap*)

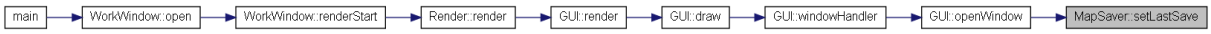
Set the last save name.

Parameters

The	last save name.

Here is the call graph for this function:

Here is the caller graph for this function:



void MapSaver::setLastSaveTime ()

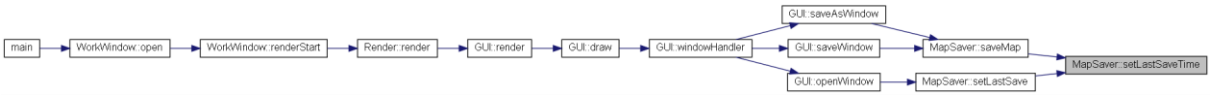
Set the last save time.

Parameters

The	last save time.

Here is the call graph for this function:

Here is the caller graph for this function:



Field Documentation

std::string MapSaver::currentSave = unsavedMarker [private]

The current save name.

const std::string MapSaver::fileType = ".csv" [private]

The file type.

std::string MapSaver::lastSaveTime = unsavedMarker [private]

The last save time in string.

Render* MapSaver::render [private]

The current render pointer.

const std::string MapSaver::saveFolder = "Saves\\" [private]

The save folder.

const std::string MapSaver::unsavedMarker = "Unsaved"

The unsaved map name.

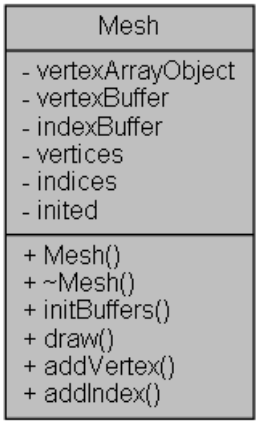
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**MapSaver.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/**MapSaver.cpp**

Mesh Class Reference

`#include <Mesh_OGL3.h>`

Collaboration diagram for Mesh:



Data Structures

- struct **Vertex**

Public Member Functions

- **Mesh** (void)
- **~Mesh** (void)
- void **initBuffers** ()
- void **draw** ()
- void **addVertex** (const **Vertex** &vertex)
- void **addIndex** (unsigned int index)

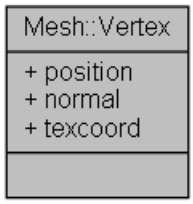
Private Attributes

- GLuint **vertexArrayObject**
- GLuint **vertexBuffer**
- GLuint **indexBuffer**
- std::vector< **Vertex** > **vertices**
- std::vector< unsigned int > **indices**
- bool **inited** = false

Data Structure Documentation

struct Mesh::Vertex

Collaboration diagram for Mesh::Vertex:



Data Fields:

vec3	normal	
vec3	position	
vec2	texcoord	

Constructor & Destructor Documentation

Mesh::Mesh (void)

Mesh::~Mesh (void)

Member Function Documentation

void Mesh::addIndex (unsigned int *index*) `[inline]`

Here is the caller graph for this function:

void Mesh::addVertex (const Vertex & *vertex*) `[inline]`

Here is the caller graph for this function:

void Mesh::draw ()

void Mesh::initBuffers ()

Here is the caller graph for this function:

Field Documentation

GLuint Mesh::indexBuffer `[private]`

std::vector<unsigned int> Mesh::indices `[private]`

bool Mesh::inited = false `[private]`

GLuint Mesh::vertexArrayObject `[private]`

GLuint Mesh::vertexBuffer `[private]`

std::vector<Vertex> Mesh::vertices `[private]`

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

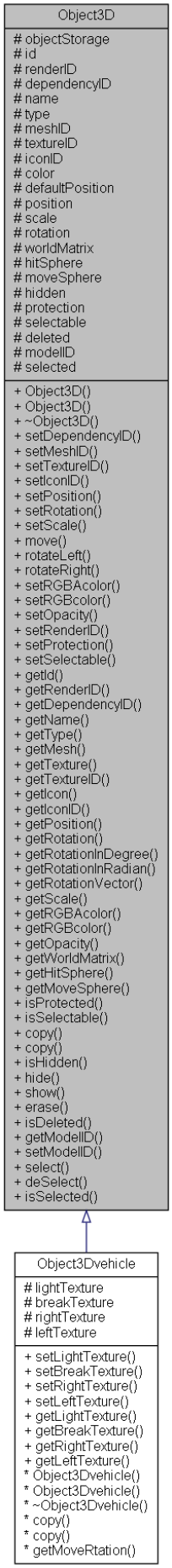
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**Mesh_OGL3.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**Mesh_OGL3.cpp**

Object3D Class Reference

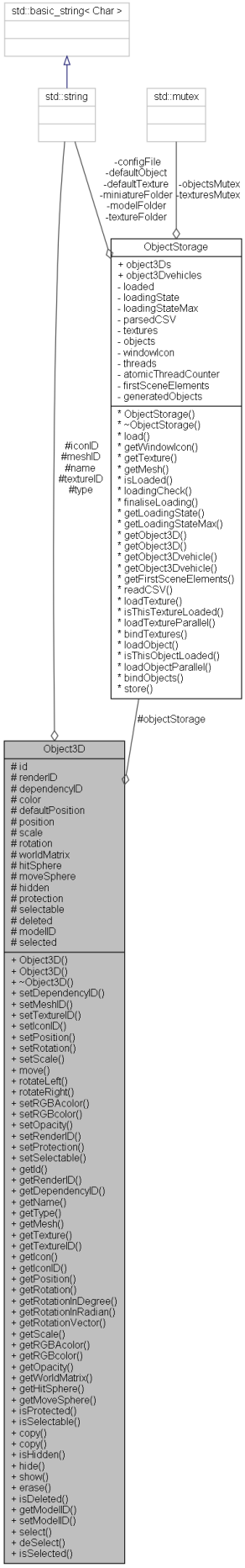
The 3D renderable object representation.

```
#include <Object3D.h>
```

Inheritance diagram for Object3D:


Object3D
objectStorage
id
renderID
dependencyID
name
type
meshID
textureID
iconID
color
defaultPosition
position
scale
rotation
worldMatrix
hitSphere
moveSphere
hidden
protection
selectable
deleted
modelID
selected
+ Object3D()
+ Object3D()
+ ~Object3D()
+ setDependencyID()
+ setMeshID()
+ setTextureID()
+ setIconID()
+ setPosition()
+ setRotation()
+ setScale()
+ move()
+ rotateLeft()
+ rotateRight()
+ setRGBAcolor()
+ setRGBcolor()
+ setOpacity()
+ setRenderID()
+ setProtection()
+ setSelectable()
+ getId()
+ getRenderID()
+ getDependencyID()
+ getName()
+ getType()
+ getMesh()
+ getTexture()
+ getTextureID()
+ getIcon()
+ getIconID()
+ getPosition()
+ getRotation()
+ getRotationInDegree()
+ getRotationInRadian()
+ getRotationVector()
+ getScale()
+ getRGBAcolor()
+ getRGBcolor()
+ getOpacity()
+ getWorldMatrix()
+ getHitSphere()
+ getMoveSphere()
+ isProtected()
+ isSelectable()
+ copy()
+ copy()
+ isHidden()
+ hide()
+ show()
+ erase()
+ isDeleted()
+ getModelID()
+ setModelID()
+ select()
+ deSelect()
+ isSelected()
Object3Dvehicle
lightTexture
breakTexture
rightTexture
leftTexture
+ setLightTexture()
+ setBreakTexture()
+ setRightTexture()
+ setLeftTexture()
+ getLightTexture()
+ getBreakTexture()
+ getRightTexture()
+ getLeftTexture()
* Object3Dvehicle()
* Object3Dvehicle()
* ~Object3Dvehicle()
* copy()
* copy()
* getMoveRtation()

Collaboration diagram for Object3D:


std::basic_string< Char >
std::string
std::mutex
-configFile
-defaultObject
-defaultTexture
-miniatureFolder
-modelFolder
-textureFolder
-objectsMutex
-texturesMutex
ObjectStorage
+ object3Ds
+ object3Dvehicles
- loaded
- loadingState
- loadingStateMax
- parsedCSV
- textures
- objects
- windowIcon
- threads
- atomicThreadCounter
- firstSceneElements
- generatedObjects
* ObjectStorage()
* ~ObjectStorage()
* load()
* getWindowIcon()
* getTexture()
* getMesh()
* isLoaded()
* loadingCheck()
* finaliseLoading()
* getLoadingState()
* getLoadingStateMax()
* getObject3D()
* getObject3D()
* getObject3Dvehicle()
* getObject3Dvehicle()
* getFirstSceneElements()
* readCSV()
* loadTexture()
* isThisTextureLoaded()
* loadTextureParallel()
* bindTextures()
* loadObject()
* isThisObjectLoaded()
* loadObjectParallel()
* bindObjects()
* store()
#iconID
#meshID
#name
#textureID
#type
#objectStorage
Object3D
id
renderID
dependencyID
color
defaultPosition
position
scale
rotation
worldMatrix
hitSphere
moveSphere
hidden
protection
selectable
deleted
modelID
selected
+ Object3D()
+ Object3D()
+ ~Object3D()
+ setDependencyID()
+ setMeshID()
+ setTextureID()
+ setIconID()
+ setPosition()
+ setRotation()
+ setScale()
+ move()
+ rotateLeft()
+ rotateRight()
+ setRGBAcolor()
+ setRGBcolor()
+ setOpacity()
+ setRenderID()
+ setProtection()
+ setSelectable()
+ getId()
+ getRenderID()
+ getDependencyID()
+ getName()
+ getType()
+ getMesh()
+ getTexture()
+ getTextureID()
+ getIcon()
+ getIconID()
+ getPosition()
+ getRotation()
+ getRotationInDegree()
+ getRotationInRadian()
+ getRotationVector()
+ getScale()
+ getRGBAcolor()
+ getRGBcolor()
+ getOpacity()
+ getWorldMatrix()
+ getHitSphere()
+ getMoveSphere()
+ isProtected()
+ isSelectable()
+ copy()
+ copy()
+ isHidden()
+ hide()
+ show()
+ erase()
+ isDeleted()
+ getModelID()
+ setModelID()
+ select()
+ deSelect()
+ isSelected()

Public Member Functions

- **Object3D** (**ObjectStorage** ***objectStorage**)
  *Empty constructor for the **ObjectStorage**.*

- **Object3D** (int **id**, std::string **name**, std::string **type**, std::string **meshID**, std::string **textureID**, std::string **iconID**, glm::vec3 initPosition, glm::vec3 initScale, glm::vec4 initRotation, glm::vec4 **color**, glm::vec4 **hitSphere**, glm::vec4 **moveSphere**, **ObjectStorage** ***objectStorage**)
  *The full constructor for the **Object3D** class.*

- **~Object3D** (void)
  *The **Object3D** destructor.*

- void **setDependencyID** (int **dependencyID**)
  Setter for the dependencyID.

- void **setMeshID** (std::string **meshID**)
  Setter for the mesh.

- void **setTextureID** (std::string **textureID**)
  Setter for the Texture ID.

- void **setIconID** (std::string **iconID**)
  Setter for the Icon ID.

- void **setPosition** (glm::vec3 **position**)
  Setter for the position.

- void **setRotation** (glm::vec4 **rotation**)
  Setter for the new rotation.

- void **setScale** (glm::vec3 **scale**)
  Setter for the new scale.

- void **move** (glm::vec3 move)
  Shift the position.

- void **rotateLeft** (float rotateLeft)
  Rotate the rotation left.

- void **rotateRight** (float rotateRight)
  Rotate the rotation right.

- void **setRGBAcolor** (glm::vec4 RGBAcolor)
  Setter for the RGBA color.

- void **setRGBcolor** (glm::vec3 RGBcolor)

Setter for the RGB color.

- void **setOpacity** (float opacity)
  Setter for the opacity.

- void **setRenderID** (int newRenderID)
  The object current renderID from the render.

- void **setProtection** (bool **isProtected**)
  Set the object protection.

- void **setSelectable** (bool **isSelectable**)
  Set the selectable flag.

- int **getId** ()
  Getter for the ID.

- int **getRenderID** ()
  Getter for the renderID.

- int **getDependencyID** ()
  Getter for the dependency ID.

- std::string **getName** ()
  Getter for the name.

- std::string **getType** ()
  Getter for the Type.

- std::unique_ptr< **Mesh** > & **getMesh** ()
  *Getter for the **Mesh**.*

- **Texture2D** & **getTexture** ()
  Getter for the texture.

- std::string **getTextureID** ()
  Getter for the texture ID.

- **Texture2D** & **getIcon** ()
  Getter for the Icon.

- std::string **getIconID** ()
  Getter for the icon ID.

- glm::vec3 **getPosition** ()
  Getter for the position.

- glm::vec4 **getRotation** ()
  Getter for the rotation.

- float **getRotationInDegree** ()
  Getter for the rotation in degree.

- float **getRotationInRadian** ()
  Getter for the rotation in radian.

- glm::vec3 **getRotationVector** ()
  Getter for the rotation vector.

- glm::vec3 **getScale** ()
  Getter for the scale.

- glm::vec4 **getRGBAcolor** ()
  Getter for the RGBA color.

- glm::vec3 **getRGBcolor** ()
  Getter for the RGB color.

- float **getOpacity** ()
  Getter for the opacity.

- glm::mat4 **getWorldMatrix** ()
  Calculate the world matrix.

- glm::vec4 **getHitSphere** ()
  Getter for the hit sphere.

- glm::vec4 **getMoveSphere** ()
  Getter for the move sphere.

- bool **isProtected** ()
  Getter for the protection flag.

- bool **isSelectable** ()
  Getter for the selectable flag.

- **Object3D copy** ()
  Object copy method.

- **Object3D copy** (int **renderID**)
  Object copy method.

- bool **isHidden** ()
  Getter for the hidden flag.

- void **hide** ()
  Hide the object.

- void **show** ()
  Show the object.

- void **erase** ()
  Set the object deleted.

- bool **isDeleted** ()
  Getter for the deleted flag.

- size_t **getModelID** ()
  Getter for the Model ID.

- void **setModelID** (size_t **id**)
  Setter for the model ID.

- void **select** ()
  Set the object selected.

- void **deSelect** ()
  Set the object deselect.

- bool **isSelected** ()
  Getter for the selected flag.

Protected Attributes

- **ObjectStorage** * **objectStorage**
  Pointer for the objects storage.

- int **id** = -1
  The object type ID.

- int **renderID** = -1
  The current object render ID in the render.

- int **dependencyID** = -1
  The dependency id if it exist.

- std::string **name** = ""
  The object type name.

- std::string **type** = ""
  The object type.

- std::string **meshID** = ""
  The geometry ID.

- std::string **textureID** = "white.png"
  The texture ID.

- std::string **iconID** = "sphere_mark_mini.png"
  The Icon ID.

- glm::vec4 **color** = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f)
  The color.

- glm::vec3 **defaultPosition** = glm::vec3(0.0f, 0.0f, 0.0f)
  The default position.

- glm::vec3 **position** = glm::vec3(0.0f, 0.0f, 0.0f)
  The current position.

- glm::vec3 **scale** = glm::vec3(1.0f, 1.0f, 1.0f)
  The current scale.

- glm::vec4 **rotation** = glm::vec4(0.0f, 0.0f, 1.0f, 0.0f)
  The current rotation.

- glm::vec4 **worldMatrix** = glm::vec4(1.0f)
  The current calculated world matrix.

- glm::vec4 **hitSphere** = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f)
  The hit sphere for the ray picking.

- glm::vec4 **moveSphere** = glm::vec4(0.0f, 0.0f, 0.0f, 1.0f)
  The move sphere.

- bool **hidden** = false
  The hidden flag.

- bool **protection** = false
  The protection flag.

- bool **selectable** = true
  The selectable flag.

- bool **deleted** = false
  The deleted flag.

- size_t **modelID** = -1

The graph model id.

- bool **selected** = true
  The selected flag.

Detailed Description

The 3D renderable object representation.

The **Object3D** renderable object representation. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

Object3D::Object3D (ObjectStorage * *objectStorage*)

Empty constructor for the **ObjectStorage**.

Parameters

objectStorage	The copyable **ObjectStorage**.

Here is the caller graph for this function:

Object3D::Object3D (int *id*, std::string *name*, std::string *type*, std::string *meshID*, std::string *textureID*, std::string *iconID*, glm::vec3 *initPosition*, glm::vec3 *initScale*, glm::vec4 *initRotation*, glm::vec4 *color*, glm::vec4 *hitSphere*, glm::vec4 *moveSphere*, ObjectStorage * *objectStorage*)

The full constructor for the **Object3D** class.

Parameters

id	The object type ID.
name	The object type name.
type	The object category.
meshID	The geometry.
textureID	The Texture.
iconID	The icon texture.
initPosition	The initialization position.
initScale	The initialization scale.
initRotation	The initialization rotation.
color	The base color.
hitSphere	The hit sphere for the ray picking.
moveSphere	The move sphere.

objectStorage	The pointer for the object storage.

Object3D::~Object3D (void)

The **Object3D** destructor.

Parameters

objectStorage	

Member Function Documentation

Object3D Object3D::copy ()

Object copy method.

Returns

The copy of this object.

Here is the call graph for this function:

Here is the caller graph for this function:

Object3D Object3D::copy (int *renderID*)

Object copy method.

(And set render id too.)

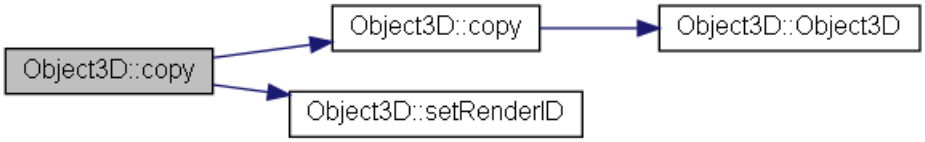
Parameters

renderID	The settable render ID.

Returns

The copy of this object.

Here is the call graph for this function:

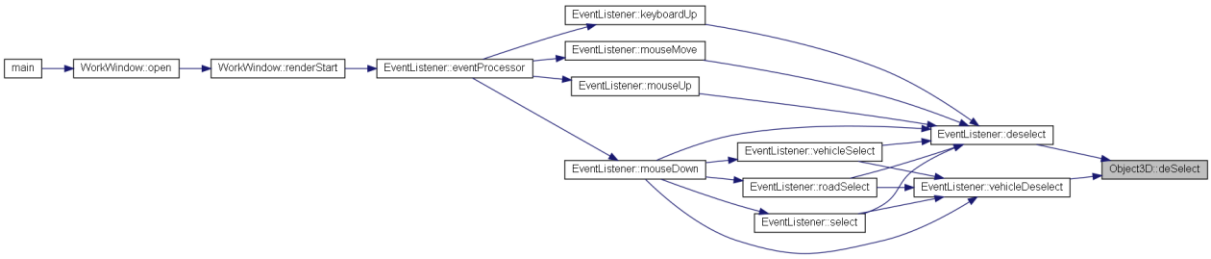


void Object3D::deSelect ()

Set the object deselect.

Here is the call graph for this function:

Here is the caller graph for this function:



void Object3D::erase ()

Set the object deleted.

int Object3D::getDependencyID ()

Getter for the dependency ID.

Returns

The dependency ID.

Here is the caller graph for this function:

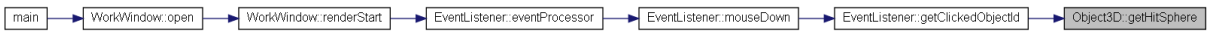
glm::vec4 Object3D::getHitSphere ()

Getter for the hit sphere.

Returns

The hit sphere.

Here is the caller graph for this function:



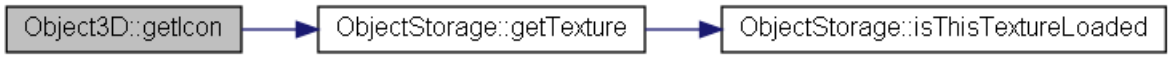
Texture2D & Object3D::getIcon ()

Getter for the Icon.

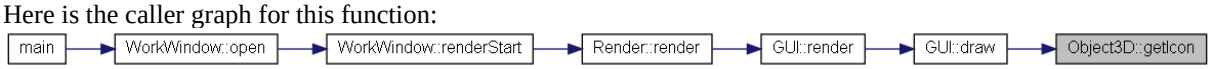
Returns

The icon.

Here is the call graph for this function:



Here is the caller graph for this function:



std::string Object3D::getIconID ()

Getter for the icon ID.

Returns

The icon ID.

int Object3D::getId ()

Getter for the ID.

Returns

The ID.

Here is the caller graph for this function:

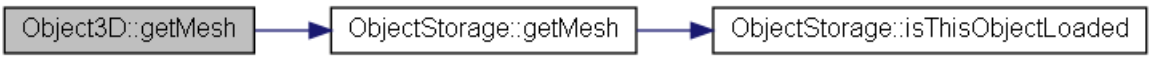
std::unique_ptr< Mesh > & Object3D::getMesh ()

Getter for the **Mesh**.

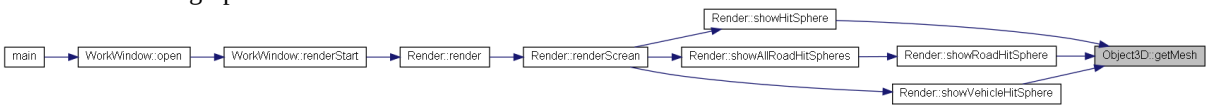
Returns

The mesh.

Here is the call graph for this function:



Here is the caller graph for this function:



size_t Object3D::getModelID ()

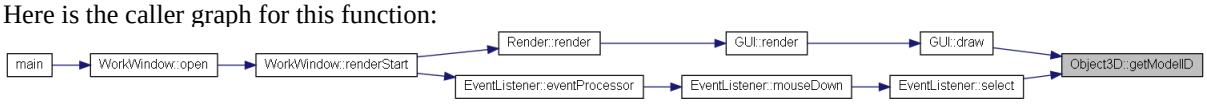
Getter for the Model ID.

(Link for the graph model.)

Returns

The Model ID.

Here is the caller graph for this function:



glm::vec4 Object3D::getMoveSphere ()

Getter for the move sphere.

Returns

The move sphere.

std::string Object3D::getName ()

Getter for the name.

Returns

The name.

Here is the caller graph for this function:

float Object3D::getOpacity ()

Getter for the opacity.

Returns

The opacity.

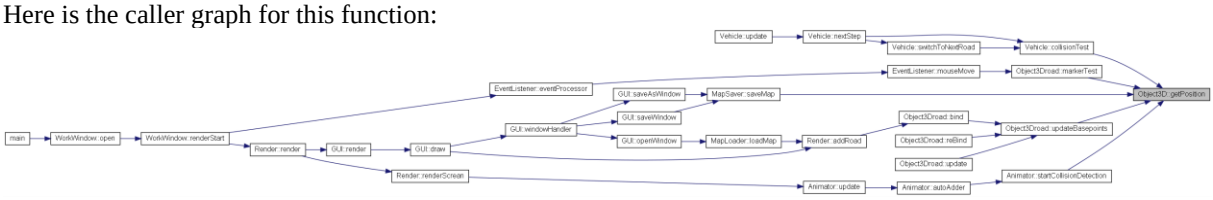
glm::vec3 Object3D::getPosition ()

Getter for the position.

Returns

The Position.

Here is the caller graph for this function:



int Object3D::getRenderID ()

Getter for the renderID.

Returns

The renderID.

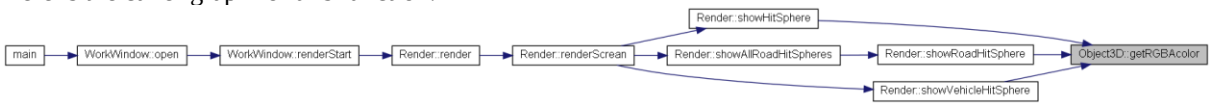
glm::vec4 Object3D::getRGBAcolor ()

Getter for the RGBA color.

Returns

The RGBA color.

Here is the caller graph for this function:



glm::vec3 Object3D::getRGBcolor ()

Getter for the RGB color.

Returns

RGB color.

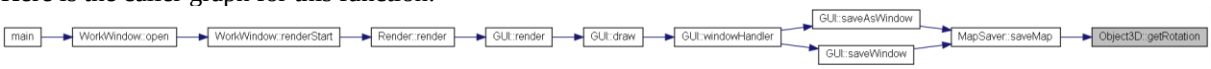
glm::vec4 Object3D::getRotation ()

Getter for the rotation.

Returns

The rotation.

Here is the caller graph for this function:



float Object3D::getRotationInDegree ()

Getter for the rotation in degree.

Returns

The rotation in degree.

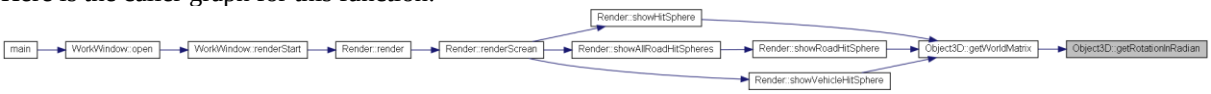
float Object3D::getRotationInRadian ()

Getter for the rotation in radian.

Returns

the rotation in radian.

Here is the caller graph for this function:



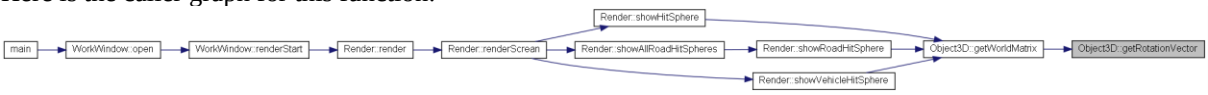
glm::vec3 Object3D::getRotationVector ()

Getter for the rotation vector.

Returns

The rotation vector.

Here is the caller graph for this function:



glm::vec3 Object3D::getScale ()

Getter for the scale.

Returns

The scale.

Texture2D & Object3D::getTexture ()

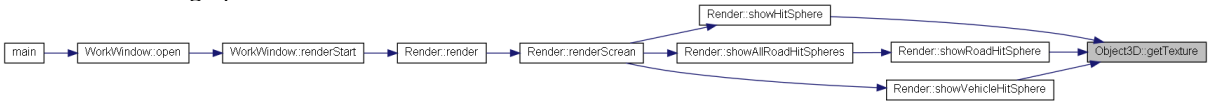
Getter for the texture.

Returns

The texture.

Here is the call graph for this function:

Here is the call graph for this function:



std::string Object3D::getTextureID ()

Getter for the texture ID.

Returns

The texture ID.

std::string Object3D::getType ()

Getter for the Type.

Returns

The type.

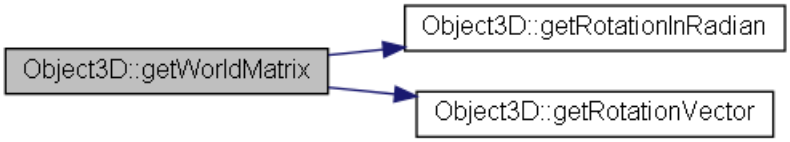
glm::mat4 Object3D::getWorldMatrix ()

Calculate the world matrix.

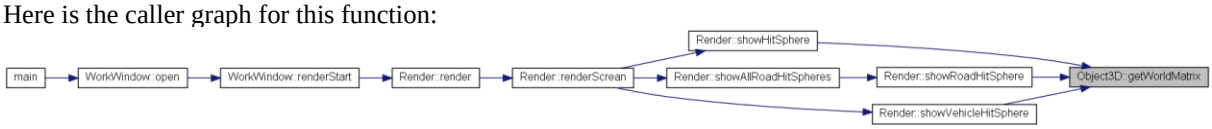
Returns

The calculated World Matrix.

Here is the call graph for this function:



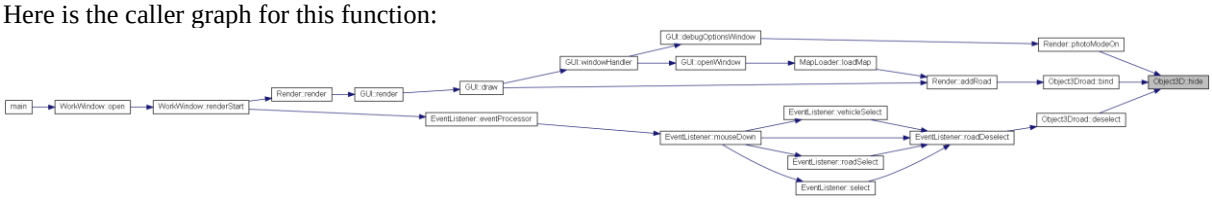
Here is the caller graph for this function:



void Object3D::hide ()

Hide the object.

Here is the caller graph for this function:



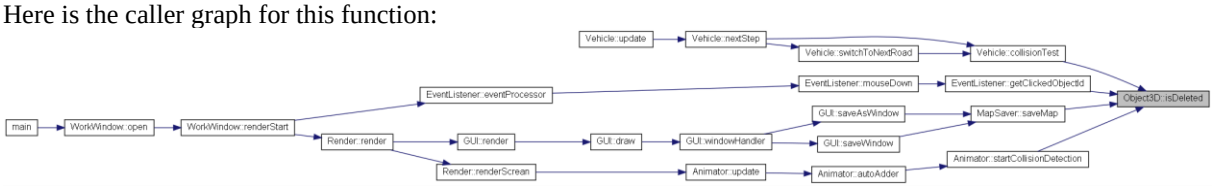
bool Object3D::isDeleted ()

Getter for the deleted flag.

Returns

The deleted flag.

Here is the caller graph for this function:



bool Object3D::isHidden ()

Getter for the hidden flag.

Returns

The hidden flag.

Here is the caller graph for this function:

bool Object3D::isProtected ()

Getter for the protection flag.

Returns

The protection flag.

Here is the caller graph for this function:

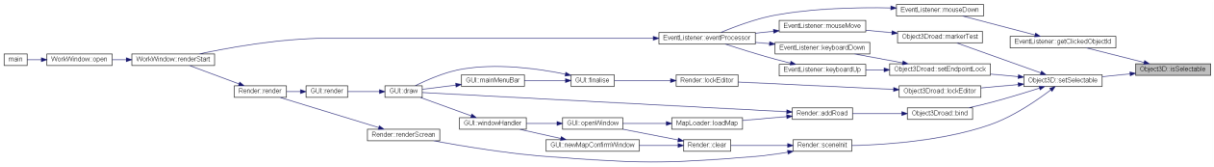
bool Object3D::isSelectable ()

Getter for the selectable flag.

Returns

The selectable flag.

Here is the caller graph for this function:



bool Object3D::isSelected ()

Getter for the selected flag.

Returns

The selected flag.

void Object3D::move (glm::vec3 *move*)

Shift the position.

Parameters

move	The shift vector.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3D::rotateLeft (float *rotateLeft*)

Rotate the rotation left.

Parameters

rotateLeft	The left rotation.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3D::rotateRight (float *rotateRight*)

Rotate the rotation right.

Parameters

rotateLeft	The left right.

Here is the call graph for this function:

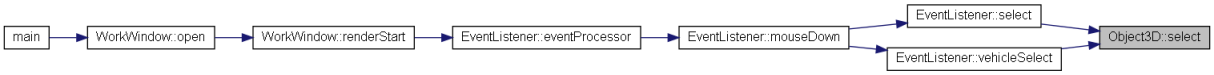
Here is the caller graph for this function:

void Object3D::select ()

Set the object selected.

Here is the call graph for this function:

Here is the caller graph for this function:



void Object3D::setDependencyID (int *dependencyID*)

Setter for the dependencyID.

(Used when the object has dependency.(road))

Parameters

dependencyID	The dependencyID

Here is the caller graph for this function:

void Object3D::setIconID (std::string *iconID*)

Setter for the Icon ID.

Parameters

iconID	The Icon ID.

void Object3D::setMeshID (std::string *meshID*)

Setter for the mesh.

Parameters

meshID	The mesh.

void Object3D::setModelID (size_t *id*)

Setter for the model ID.

(Link for the graph model.)

Parameters

id	The new model ID.

Here is the caller graph for this function:

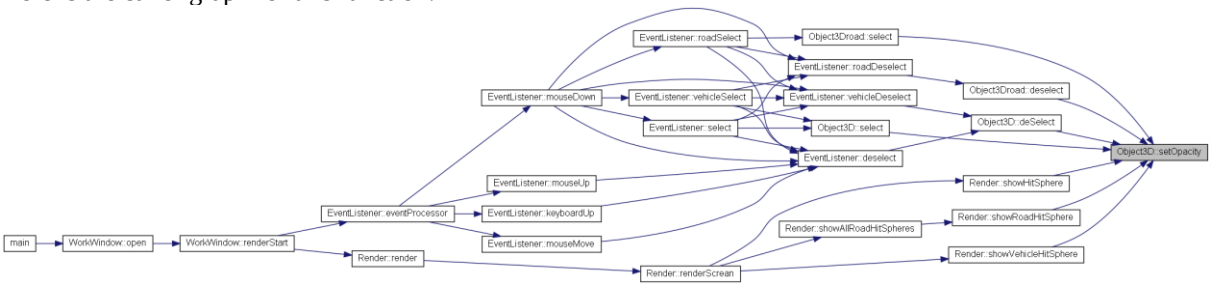
void Object3D::setOpacity (float *opacity*)

Setter for the opacity.

Parameters

opacity	The new opacity.

Here is the caller graph for this function:



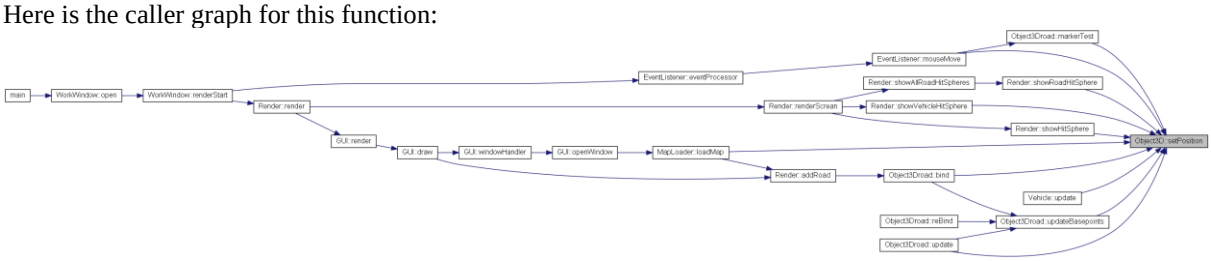
void Object3D::setPosition (glm::vec3 *position*)

Setter for the position.

Parameters

position	The new position.

Here is the caller graph for this function:



void Object3D::setProtection (bool *isProtected*)

Set the object protection.

Parameters

isProtected	The new protection state.

Here is the call graph for this function:

Here is the caller graph for this function:

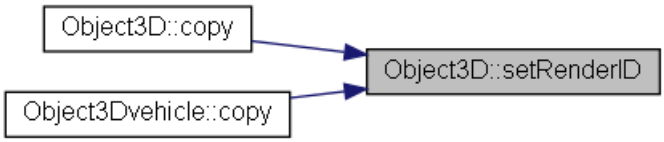
void Object3D::setRenderID (int *newRenderID*)

The object current renderID from the render.

Parameters

newRenderID	The new render ID.

Here is the caller graph for this function:



void Object3D::setRGBAcolor (glm::vec4 *RGBAcolor*)

Setter for the RGBA color.

Parameters

RGBAcolor	The new RGBA color.

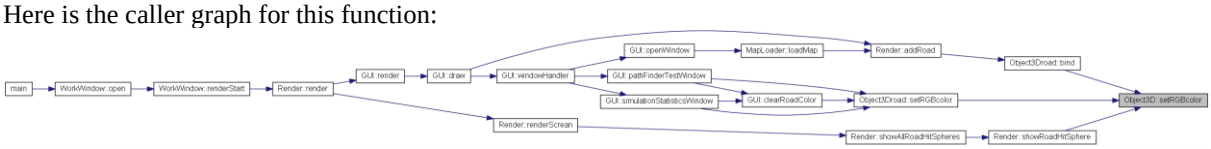
void Object3D::setRGBcolor (glm::vec3 *RGBcolor*)

Setter for the RGB color.

Parameters

RGBAcolor	The new RGB color.

Here is the caller graph for this function:



void Object3D::setRotation (glm::vec4 *rotation*)

Setter for the new rotation.

Parameters

rotation	The new rotation.

Here is the caller graph for this function:

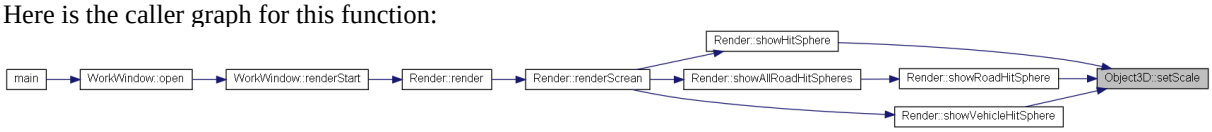
void Object3D::setScale (glm::vec3 *scale*)

Setter for the new scale.

Parameters

scale	The new scale.

Here is the caller graph for this function:



void Object3D::setSelectable (bool *isSelectable*)

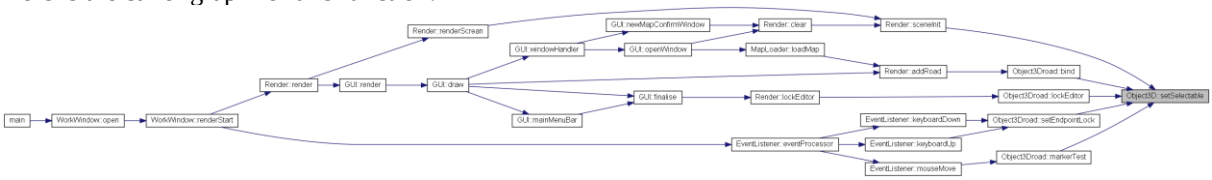
Set the selectable flag.

Parameters

isSelectable	The new state for the selectable flag.

Here is the call graph for this function:

Here is the caller graph for this function:



void Object3D::setTextureID (std::string *textureID*)

Setter for the Texture ID.

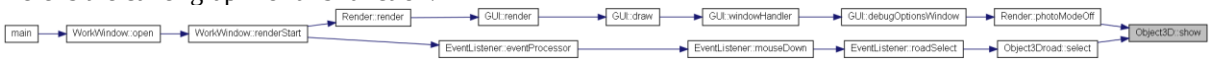
Parameters

textureID	the Texture ID,

void Object3D::show ()

Show the object.

Here is the caller graph for this function:



Field Documentation

glm::vec4 Object3D::color = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f) `[protected]`

The color.

glm::vec3 Object3D::defaultPosition = glm::vec3(0.0f, 0.0f, 0.0f) `[protected]`

The default position.

bool Object3D::deleted = false `[protected]`

The deleted flag.

int Object3D::dependencyID = -1 `[protected]`

The dependency id if it exist.

bool Object3D::hidden = false `[protected]`

The hidden flag.

glm::vec4 Object3D::hitSphere = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f) `[protected]`

The hit sphere for the ray picking.

std::string Object3D::iconID = "sphere_mark_mini.png" `[protected]`

The Icon ID.

int Object3D::id = -1 `[protected]`

The object type ID.

std::string Object3D::meshID = "" `[protected]`

The geometry ID.

size_t Object3D::modelID = -1 `[protected]`

The graph model id.

glm::vec4 Object3D::moveSphere = glm::vec4(0.0f, 0.0f, 0.0f, 1.0f) `[protected]`

The move sphere.

std::string Object3D::name = "" `[protected]`

The object type name.

ObjectStorage* Object3D::objectStorage `[protected]`

Pointer for the objects storage.

glm::vec3 Object3D::position = glm::vec3(0.0f, 0.0f, 0.0f) `[protected]`

The current position.

bool Object3D::protection = false `[protected]`

The protection flag.

int Object3D::renderID = -1 `[protected]`

The current object render ID in the render.

glm::vec4 Object3D::rotation = glm::vec4(0.0f, 0.0f, 1.0f, 0.0f) `[protected]`

The current rotation.

glm::vec3 Object3D::scale = glm::vec3(1.0f, 1.0f, 1.0f) `[protected]`

The current scale.

bool Object3D::selectable = true `[protected]`

The selectable flag.

bool Object3D::selected = true `[protected]`

The selected flag.

std::string Object3D::textureID = "white.png" `[protected]`

The texture ID.

std::string Object3D::type = "" `[protected]`

The object type.

glm::vec4 Object3D::worldMatrix = glm::vec4(1.0f) `[protected]`

The current calculated world matrix.

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

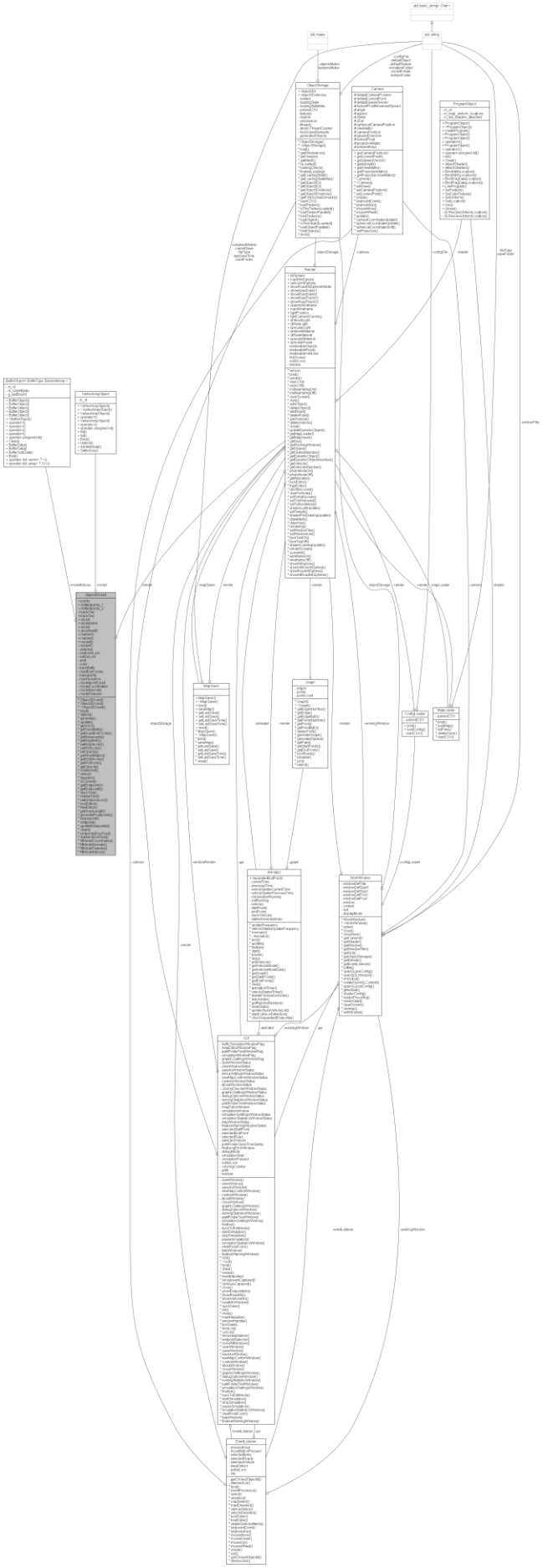
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**Object3D.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**Object3D.cpp**

Object3Droad Class Reference

The parametric generated road implementation.

```
#include <Object3Droad.h>
```

Collaboration diagram for Object3Droad:


std::basic_string< Char >
std::string
std::mutex
ObjectStorage
Camera
ProgramObject
Render
BufferObject< BufferType, ElementArray >
VertexArrayObject
ObjectLoad
MapLoader
Graph
ConfigLoader
MapLoader
Animator
WorldWindow
GUI
EventListener

Data Fields

- std::vector< glm::vec3 > **points**
  The Bézier generated center road line.

- std::vector< glm::vec3 > **shiftedpoints_1**
  The shifted side 1.

- std::vector< glm::vec3 > **shiftedpoints_2**
  The shifted side 2.

- std::vector< glm::vec3 > **trackOne**
  The shifted track middle line 1.

- std::vector< glm::vec3 > **trackTwo**
  The shifted track middle line 2.

- size_t **stickA** = -1
  The endpoint A sticked id.

- char **stickMarkA** = 'Q'
  To the endpoint A witch road endpoint joined? Q when not sticked.

- size_t **stickB** = -1
  The endpoint a sticked id.

- char **stickMarkB** = 'Q'
  To the endpoint B witch road endpoint joined? Q when not sticked.

- size_t **markerA** = -1
  The endpoint A sticked marker.

- size_t **markerB** = -1
  The endpoint B sticked marker.

- size_t **modelID** = -1
  The abstract model ID.

Private Attributes

- size_t **renderID** = -1
  The render ID.

- bool **selected** = true
  Is selected?

- bool **endpointLock** = true
  The endpoint is locked?

- bool **editorLock** = false
  The editor is locked?

- float **shift** = 2.0f
  The road with/2.

- glm::vec4 **color**
  The road render color.

- size_t **trackBalls** [4]
  The trackball ID-s array.

- size_t **roadEndCircles** [2]
  The end circle ID-s array.

- glm::vec3 **basepoints** [4]
  The base points coordinates array.

- float **roadYposition**
  The roads y height.

- size_t **modelpointCount**
  The model points count.

- **VertexArrayObject model**
  The model object.

- **ArrayBuffer modelCoordinates**
  The model coordinates.

- **IndexBuffer modelIndices**
  The vertexes.

- **ArrayBuffer modelNormals**
  The model normals.

- **ArrayBuffer modelTextures**
  The model texture coordinates.

- **Render** * **render**
  The current render pointer.

Traffic Simulation

- **Object3Droad** ()
  *The **Object3Droad** constructor.*

- **Object3Droad** (const **Object3Droad** &)
  ***Object3Droad** copy constructor.*

- **~Object3Droad** (void)
  Destructor.

- void **bind** (**Render** ***render**, size_t newRenderID, size_t trackBall_0, size_t trackBall_1, size_t trackBall_2, size_t trackBall_3, size_t roadEndCircle_1, size_t roadEndCircle_2)
  Bind this road to the render and initialize the road, generate the 3D model and initialize the connected objects.

- void **reBind** (size_t trackBall_0, size_t trackBall_1, size_t trackBall_2, size_t trackBall_3, size_t roadEndCircle_1, size_t roadEndCircle_2)
  Rebind the connected objects after changes.

- void **generate** ()
  Generate the current 3D Model.

- void **update** ()
  Update the road and recreate the model.

- **VertexArrayObject** & **getVAO** ()
  Getter the renderable road VAO.

- size_t * **getTrackBalls** ()
  Get the trackballs IDs.

- size_t * **getRoadEndCircles** ()
  Get the endpoint circles IDs.

- glm::vec3 * **getBasepoints** ()
  Get the base point coordinates.

- size_t **getRenderID** ()
  Getter for the road render it.

- void **setRGBAcolor** (glm::vec4 RGBAcolor)
  Setter for the road render color with opacity.

- void **setRGBcolor** (glm::vec3 RGBcolor)
  Setter for the road render color without opacity.

- void **setOpacity** (float opacity)
  Set road render opacity.

- glm::mat4 **getWorldMatrix** ()
  Calculate for the road world matrix for the road rendering.

- glm::vec4 **getRGBAcolor** ()
  Getter for the road render color with opacity.

- glm::vec3 **getRGBcolor** ()
  Getter for the road render color without opacity.

- float **getOpacity** ()
  Getter for the road opacity.

- bool **isSelected** ()
  Checker for the selected status flag.

- void **select** ()
  Set the road selected.

- void **deselect** ()
  Deselect the road.

- bool **isClicked** (glm::vec3 cameraPosition, glm::vec3 ray)
  The road click checker.

- glm::vec3 **getEndpointA** ()
  Getter for the road Endpoint A position.

- glm::vec3 **getEndpointB** ()
  Getter for the road Endpoint B position.

- void **stuckTest** (size_t road)
  Check the road endpoints sticking to an other road or endpoint marker.

- char **markerTest** (size_t marker)
  Testing the marker sticking to a road endpoint.

- void **setEndpointLock** (bool lock)
  Set the endpoint lockers.

- void **lockEditor** ()
  Lock the editable mode.

- void **freeEditor** ()
  Free the editable mode.

- float **getRoadLength** ()

Calculate the road exact length.

- void **generateRoadpoints** ()
  Generate the road one side points by a Bézier curve.

- glm::vec3 **bezierpoint** (float u)
  Generate a Bézier point bx u.

- glm::vec3 **shiftpoint** (glm::vec3 point1, glm::vec3 point2, float currentShift, float yPosition)
  Shift a point in Right angle to the point1 point2 section with currentShift in the yPosition level.

- void **updateBasepoints** ()
  Update the base points position for the Bézier curve and do the endpoint and marker tests.

- void **clean** ()
  Clean all the generated storages.

- void **endpointsExistTest** ()
  Test the endpoints validity.

- void **markersExistTest** ()
  Tester for the markers validity.

- void **fillModelCoordinates** ()
  Fill the model coordinate storage.

- void **fillModelNormals** ()
  Fill the model normals.

- void **fillModelTextures** ()
  Fill the texture coordinates.

- void **fillModelIndices** ()
  Generate the 3D Vertexes.

Detailed Description

The parametric generated road implementation.

The parametric generated road implementation. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.12.15.

Constructor & Destructor Documentation

Object3Droad::Object3Droad ()

The **Object3Droad** constructor.

Set the basic variables too.

Object3Droad::Object3Droad (const Object3Droad &)

Object3Droad copy constructor.

Parameters

The	copyable road.

Object3Droad::~Object3Droad (void)

Destructor.

Parameters

Empty.	

Member Function Documentation

glm::vec3 Object3Droad::bezierpoint (float *u*) `[private]`

Generate a Bézier point bx u.

Parameters

u	The parameter of the Bézier point.

Returns

The generated Bézier point.

Here is the caller graph for this function:

void Object3Droad::bind (Render * *render*, size_t *newRenderID*, size_t *trackBall_0*, size_t *trackBall_1*, size_t *trackBall_2*, size_t *trackBall_3*, size_t *roadEndCircle_1*, size_t *roadEndCircle_2*)

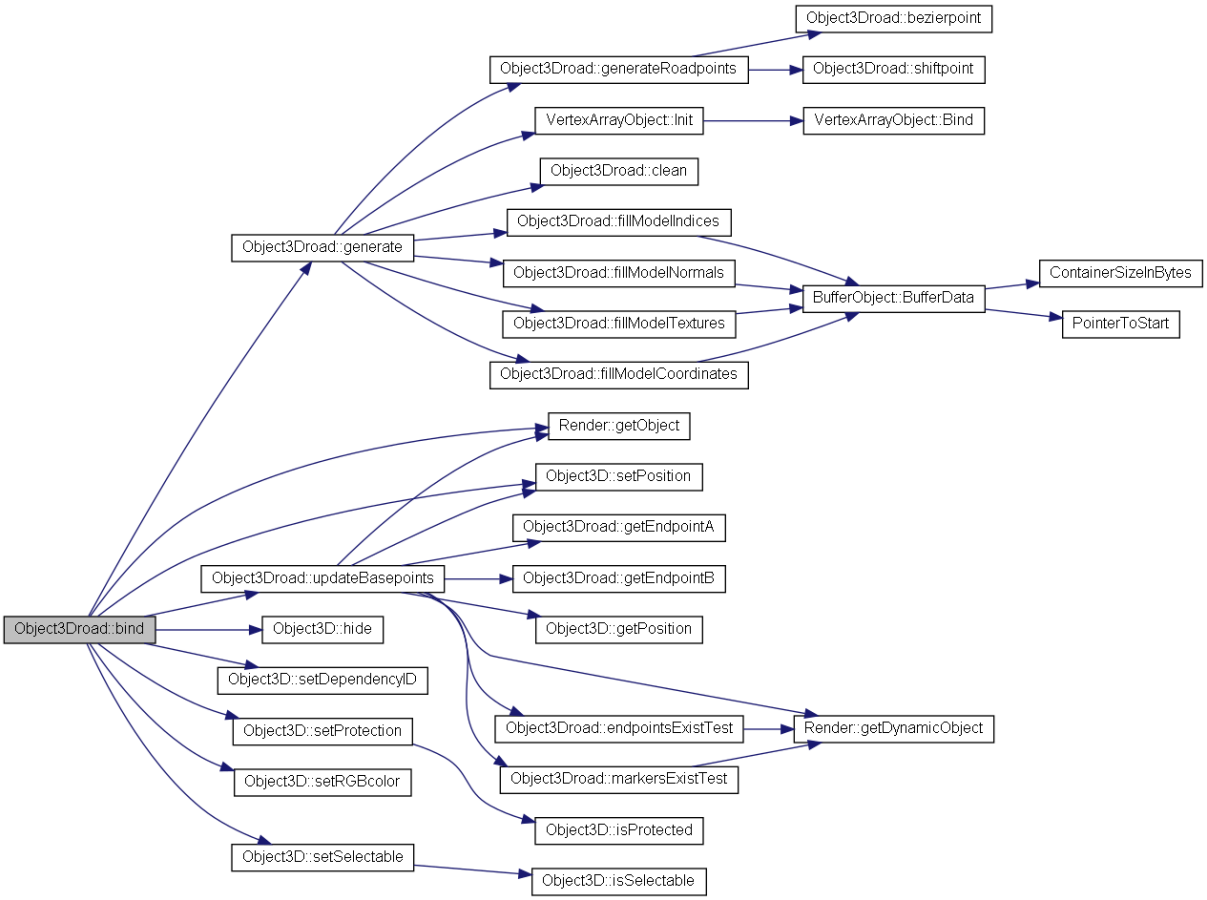
Bind this road to the render and initialize the road, generate the 3D model and initialize the connected objects.

Parameters

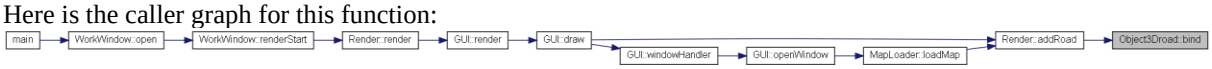
render	The current render pointer.
newRenderID	The current render ID.
trackBall_0	The trackBall_0 object ID.
trackBall_1	The trackBall_1 object ID.

trackBall_2	The trackBall_2 object ID.
trackBall_3	The trackBall_3 object ID.
roadEndCircle_1	The roadEndCircle_1 object ID.
roadEndCircle_2	The roadEndCircle_2 object ID.

Here is the call graph for this function:



Here is the caller graph for this function:



void Object3Droad::clean () [private]

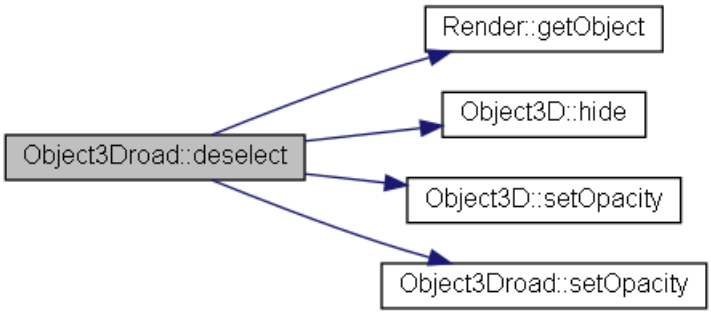
Clean all the generated storages.

Here is the caller graph for this function:

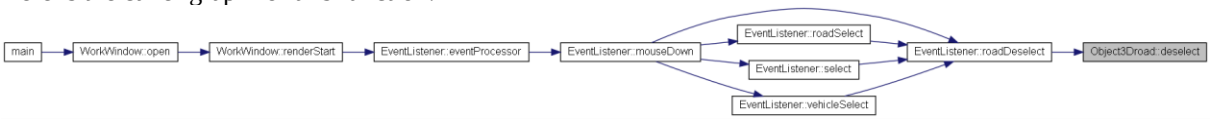
void Object3Droad::deselect ()

Deselect the road.

Here is the call graph for this function:



Here is the caller graph for this function:



void Object3Droad::endpointsExistTest () `[private]`

Test the endpoints validity.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3Droad::fillModelCoordinates () `[private]`

Fill the model coordinate storage.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3Droad::fillModelIndices () `[private]`

Generate the 3D Vertexes.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3Droad::fillModelNormals () `[private]`

Fill the model normals.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3Droad::fillModelTextures () `[private]`

Fill the texture coordinates.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3Droad::freeEditor ()

Free the editable mode.

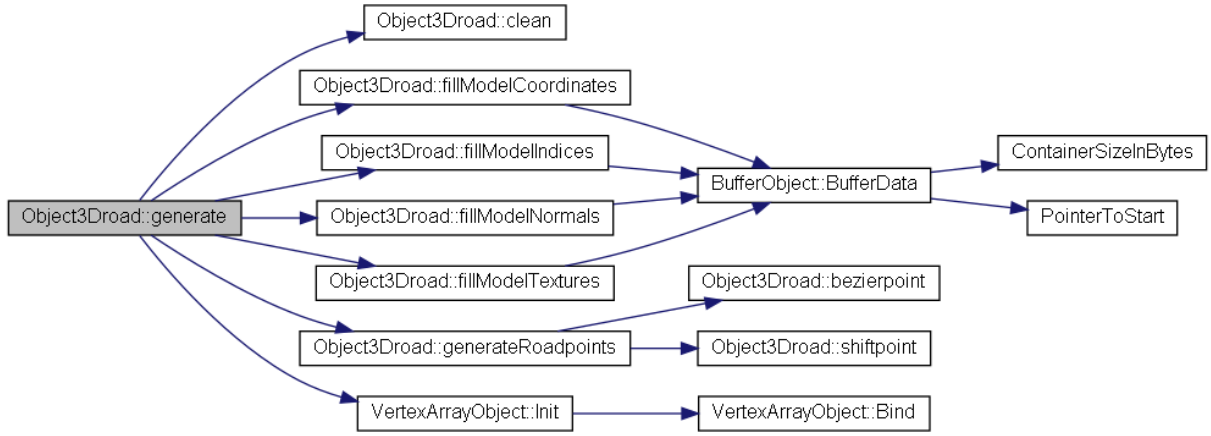
Here is the caller graph for this function:

void Object3Droad::generate ()

Generate the current 3D Model.

Generate the current 3D Mosel.

Here is the call graph for this function:

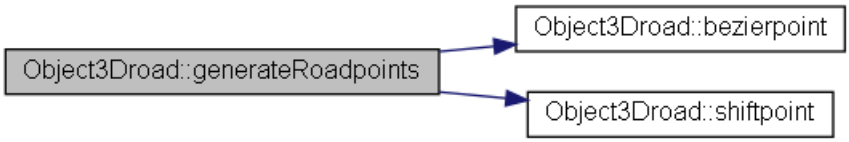


Here is the caller graph for this function:

void Object3Droad::generateRoadpoints () `[private]`

Generate the road one side points by a Bézier curve.

Here is the call graph for this function:



Here is the caller graph for this function:

glm::vec3 * Object3Droad::getBasepoints ()

Get the base point coordinates.

Returns

The base point coordinates in array.

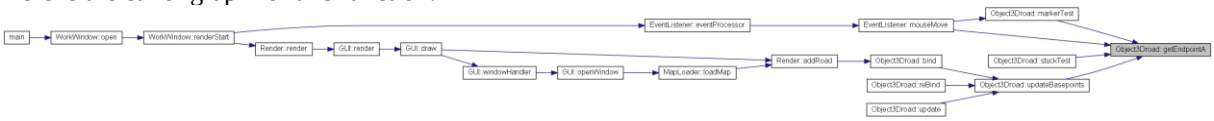
glm::vec3 Object3Droad::getEndpointA ()

Getter for the road Endpoint A position.

Returns

The road Endpoint A position.

Here is the caller graph for this function:



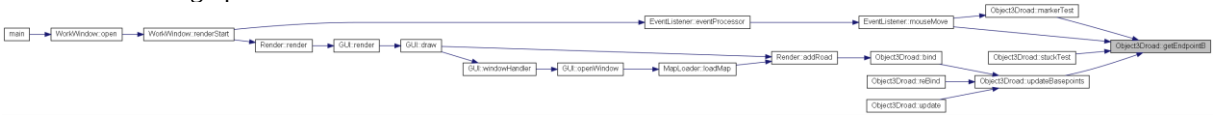
glm::vec3 Object3Droad::getEndpointB ()

Getter for the road Endpoint B position.

Returns

The road Endpoint B position.

Here is the caller graph for this function:



float Object3Droad::getOpacity ()

Getter for the road opacity.

Returns

The road opacity.

size_t Object3Droad::getRenderID ()

Getter for the road render it.

Returns

The road render it.

Here is the caller graph for this function:

glm::vec4 Object3Droad::getRGBAcolor ()

Getter for the road render color with opacity.

Returns

The road render color with opacity.

glm::vec3 Object3Droad::getRGBcolor ()

Getter for the road render color without opacity.

Returns

The road render color without opacity.

size_t * Object3Droad::getRoadEndCircles ()

Get the endpoint circles IDs.

Returns

The endpoint circles IDs.

float Object3Droad::getRoadLength ()

Calculate the road exact length.

Returns

The road exact length.

Here is the caller graph for this function:

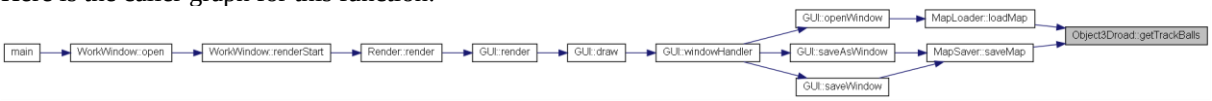
size_t * Object3Droad::getTrackBalls ()

Get the trackballs IDs.

Returns

The trackballs IDs array.

Here is the caller graph for this function:



VertexArrayObject & Object3Droad::getVAO ()

Getter the renderable road VAO.

Returns

The renderable road VAO.

glm::mat4 Object3Droad::getWorldMatrix ()

Calculate for the road world matrix for the road rendering.

Returns

The road world matrix for the road rendering.

bool Object3Droad::isClicked (glm::vec3 *cameraPosition*, glm::vec3 *ray*)

The road click checker.

Return true if a ray intersects the road hit one of the spheres.

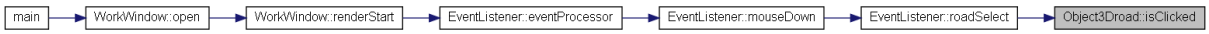
Parameters

cameraPosition	The current camera position.
ray	The ray for the intersection.

Returns

Return true if a ray intersects the road hit one of the spheres.

Here is the caller graph for this function:



bool Object3Droad::isSelected ()

Checker for the selected status flag.

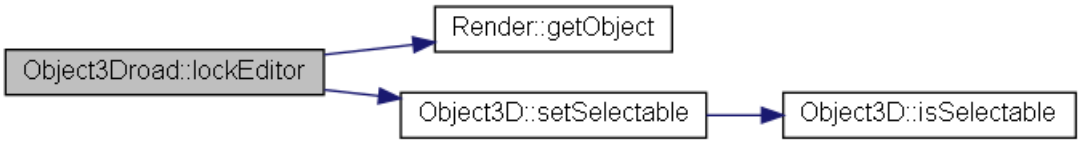
Returns

Return true if the road is in selected status.

void Object3Droad::lockEditor ()

Lock the editable mode.

Here is the call graph for this function:



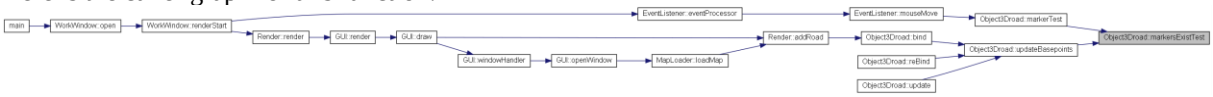
Here is the caller graph for this function:

void Object3Droad::markersExistTest () `[private]`

Tester for the markers validity.

Here is the call graph for this function:

Here is the caller graph for this function:



char Object3Droad::markerTest (size_t *marker*)

Testing the marker sticking to a road endpoint.

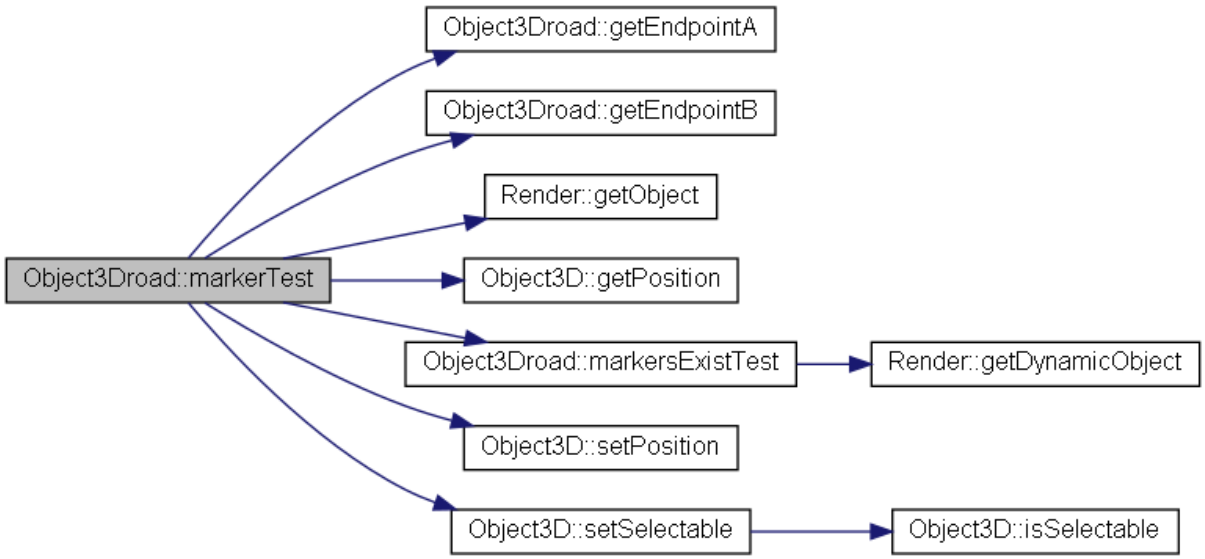
Parameters

marker	The testable marker ID.

Returns

Return the sticked endpoint name.

Here is the call graph for this function:



Here is the caller graph for this function:

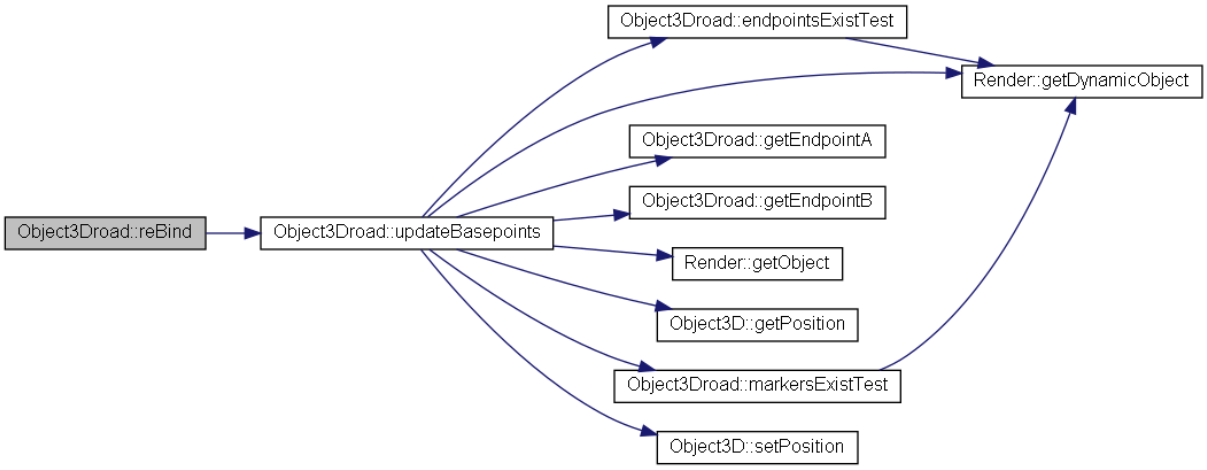
void Object3Droad::reBind (size_t *trackBall_0*, size_t *trackBall_1*, size_t *trackBall_2*, size_t *trackBall_3*, size_t *roadEndCircle_1*, size_t *roadEndCircle_2*)

Rebind the connected objects after changes.

Parameters

trackBall_0	The trackBall_0 object ID.
trackBall_1	The trackBall_1 object ID.
trackBall_2	The trackBall_2 object ID.
trackBall_3	The trackBall_3 object ID.
roadEndCircle_1	The roadEndCircle_1 object ID.
roadEndCircle_2	The roadEndCircle_2 object ID.

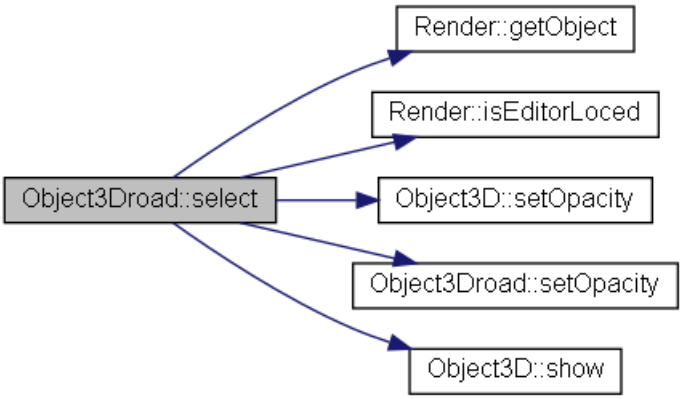
Here is the call graph for this function:



void Object3Droad::select ()

Set the road selected.

Here is the call graph for this function:



Here is the caller graph for this function:

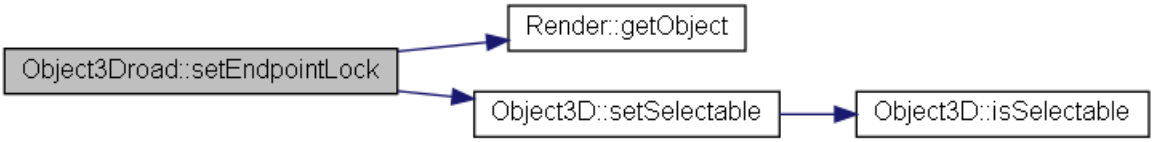
void Object3Droad::setEndpointLock (bool *lock*)

Set the endpoint lockers.

Parameters

lock	Lock or free the endpoints.

Here is the call graph for this function:



Here is the caller graph for this function:

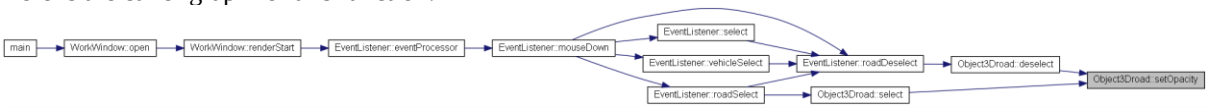
void Object3Droad::setOpacity (float *opacity*)

Set road render opacity.

Parameters

opacity	The road render opacity.

Here is the caller graph for this function:



void Object3Droad::setRGBAcolor (glm::vec4 *RGBAcolor*)

Setter for the road render color with opacity.

Parameters

RGBAcolor	The road render color with opacity.

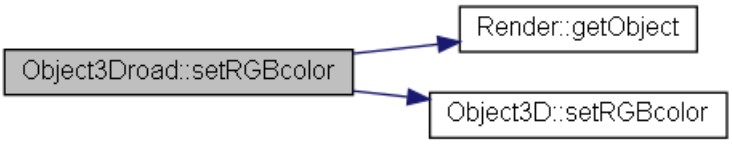
void Object3Droad::setRGBcolor (glm::vec3 *RGBcolor*)

Setter for the road render color without opacity.

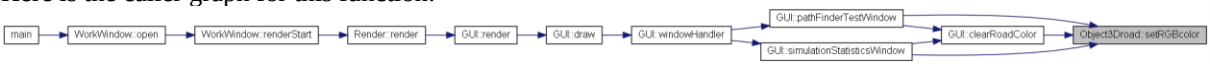
Parameters

RGBcolor	The road render color without opacity.

Here is the call graph for this function:



Here is the caller graph for this function:



glm::vec3 Object3Droad::shiftpoint (glm::vec3 *point1*, glm::vec3 *point2*, float *currentShift*, float *yPosition*) `[private]`

Shift a point in Right angle to the point1 point2 section with currentShift in the yPosition level.

Parameters

point1	The section start point.
point2	The section end point.
currentShift	The shift vector length.
yPosition	The Y level.

Returns

The shifted point.

Here is the caller graph for this function:

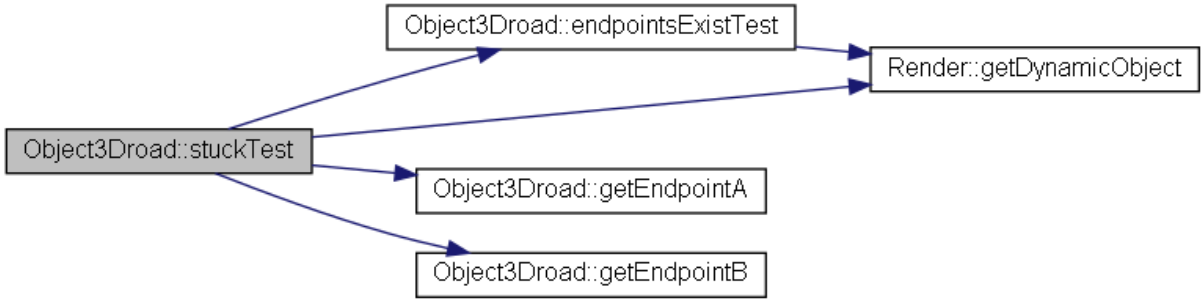
void Object3Droad::stuckTest (size_t *road*)

Check the road endpoints sticking to an other road or endpoint marker.

Parameters

road	

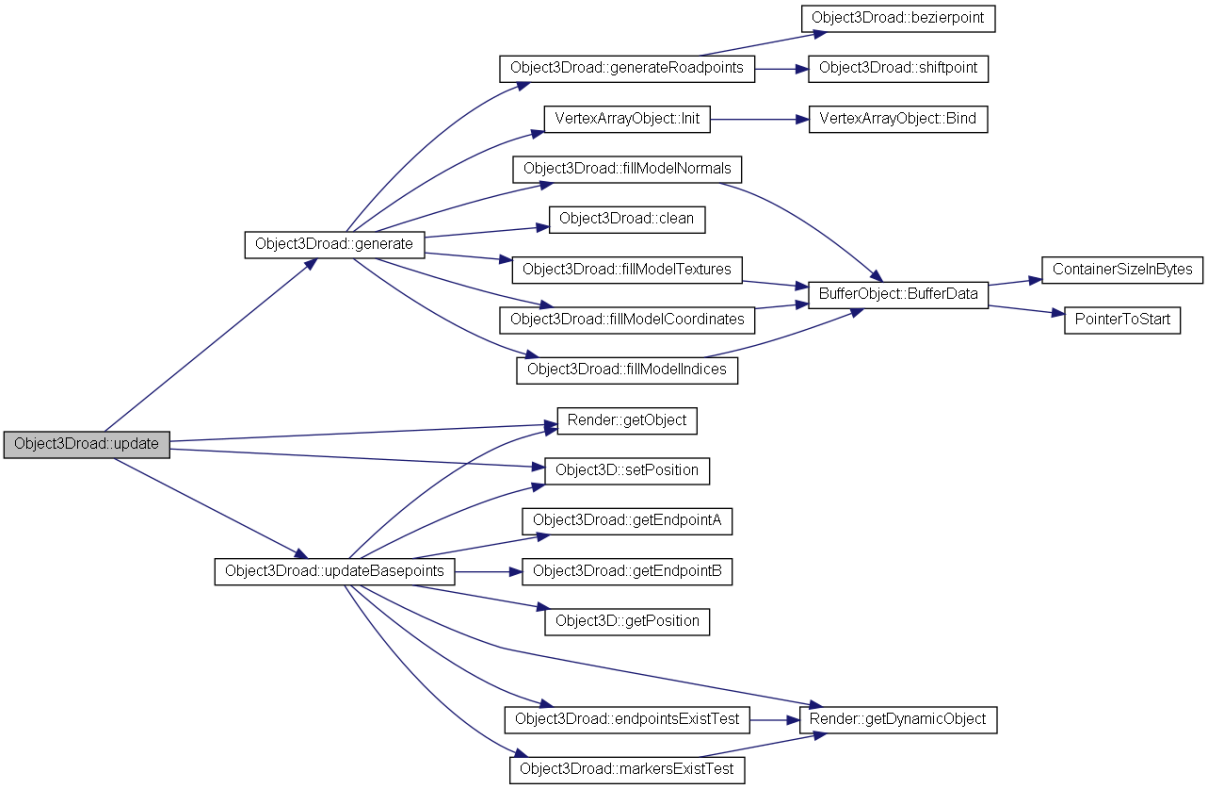
Here is the call graph for this function:



void Object3Droad::update ()

Update the road and recreate the model.

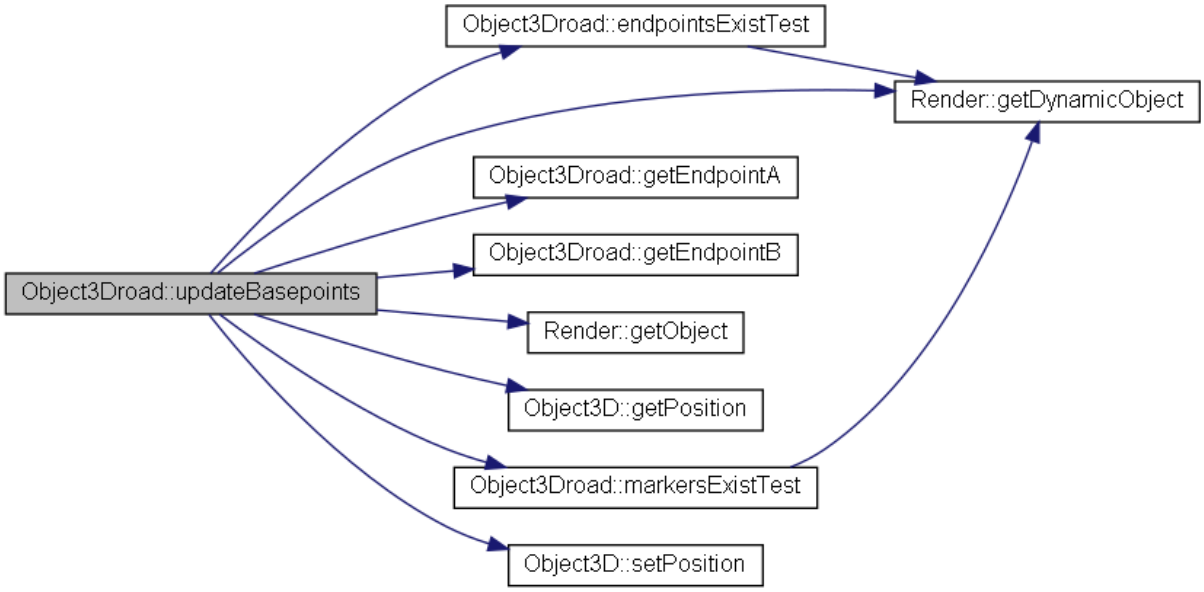
Here is the call graph for this function:



void Object3Droad::updateBasepoints () `[private]`

Update the base points position for the Bézier curve and do the endpoint and marker tests.

Here is the call graph for this function:



Here is the caller graph for this function:

Field Documentation

glm::vec3 Object3Droad::basepoints[4] `[private]`

The base points coordinates array.

glm::vec4 Object3Droad::color `[private]`

The road render color.

bool Object3Droad::editorLock = false `[private]`

The editor is locked?

bool Object3Droad::endpointLock = true `[private]`

The endpoint is locked?

size_t Object3Droad::markerA = -1

The endpoint A sticked marker.

-1 when not sticked.

size_t Object3Droad::markerB = -1

The endpoint B sticked marker.

-1 when not sticked.

VertexArrayObject Object3Droad::model `[private]`

The model object.

ArrayBuffer Object3Droad::modelCoordinates `[private]`

The model coordinates.

size_t Object3Droad::modelID = -1

The abstract model ID.

IndexBuffer Object3Droad::modelIndices `[private]`

The vertexes.

ArrayBuffer Object3Droad::modelNormals `[private]`

The model normals.

size_t Object3Droad::modelpointCount `[private]`

The model points count.

ArrayBuffer Object3Droad::modelTextures `[private]`

The model texture coordinates.

std::vector<glm::vec3> Object3Droad::points

The Bézier generated center road line.

Render* Object3Droad::render `[private]`

The current render pointer.

size_t Object3Droad::renderID = -1 `[private]`

The render ID.

size_t Object3Droad::roadEndCircles[2] `[private]`

The end circle ID-s array.

float Object3Droad::roadYposition `[private]`

The roads y height.

bool Object3Droad::selected = true `[private]`

Is selected?

float Object3Droad::shift = 2.0f `[private]`

The road with/2.
This is the road shift.

std::vector<glm::vec3> Object3Droad::shiftedpoints_1

The shifted side 1.

std::vector<glm::vec3> Object3Droad::shiftedpoints_2

The shifted side 2.

size_t Object3Droad::stickA = -1

The endpoint A sticked id.
-1 when not sticked.

size_t Object3Droad::stickB = -1

The endpoint a sticked id.
-1 when not sticked.

char Object3Droad::stickMarkA = 'Q'

To the endpoint A witch road endpoint joined? Q when not sticked.

char Object3Droad::stickMarkB = 'Q'

To the endpoint B witch road endpoint joined? Q when not sticked.

size_t Object3Droad::trackBalls[4] `[private]`

The trackball ID-s array.

std::vector<glm::vec3> Object3Droad::trackOne

The shifted track middle line 1.

std::vector<glm::vec3> Object3Droad::trackTwo

The shifted track middle line 2.

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

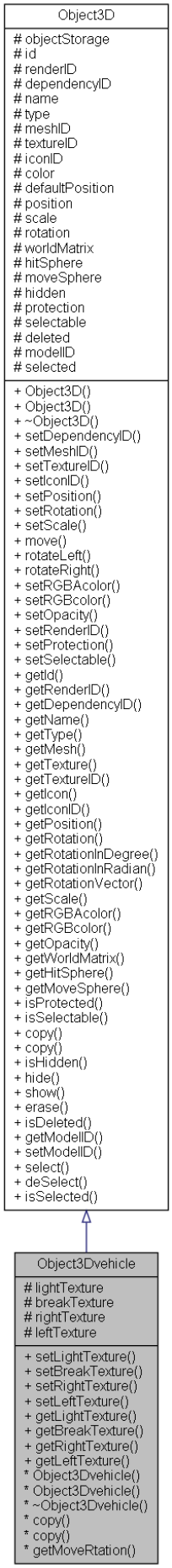
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**Object3Droad.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**Object3Droad.cpp**

Object3Dvehicle Class Reference

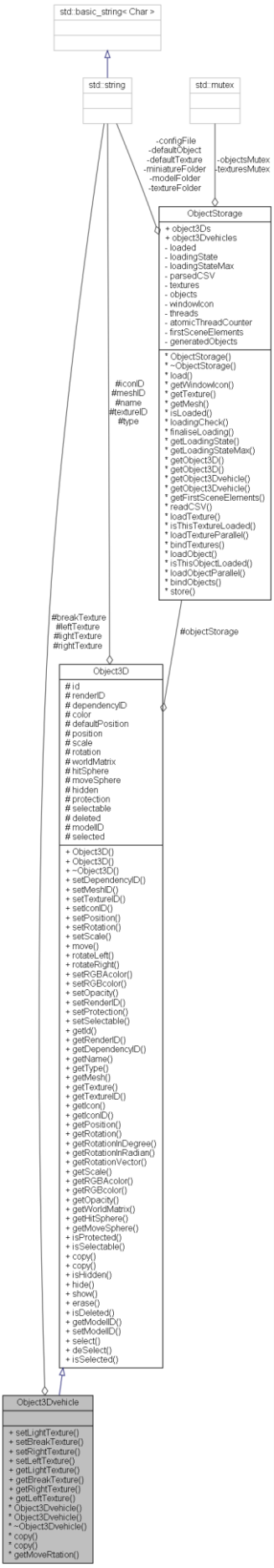
The renderable vehicle representation.

```
#include <Object3Dvehicle.h>
```

Inheritance diagram for Object3Dvehicle:


Object3D
objectStorage
id
renderID
dependencyID
name
type
meshID
textureID
iconID
color
defaultPosition
position
scale
rotation
worldMatrix
hitSphere
moveSphere
hidden
protection
selectable
deleted
modelID
selected
+ Object3D()
+ Object3D()
+ ~Object3D()
+ setDependencyID()
+ setMeshID()
+ setTextureID()
+ setIconID()
+ setPosition()
+ setRotation()
+ setScale()
+ move()
+ rotateLeft()
+ rotateRight()
+ setRGBAcolor()
+ setRGBcolor()
+ setOpacity()
+ setRenderID()
+ setProtection()
+ setSelectable()
+ getId()
+ getRenderID()
+ getDependencyID()
+ getName()
+ getType()
+ getMesh()
+ getTexture()
+ getTextureID()
+ getIcon()
+ getIconID()
+ getPosition()
+ getRotation()
+ getRotationInDegree()
+ getRotationInRadian()
+ getRotationVector()
+ getScale()
+ getRGBAcolor()
+ getRGBcolor()
+ getOpacity()
+ getWorldMatrix()
+ getHitSphere()
+ getMoveSphere()
+ isProtected()
+ isSelectable()
+ copy()
+ copy()
+ isHidden()
+ hide()
+ show()
+ erase()
+ isDeleted()
+ getModelID()
+ setModelID()
+ select()
+ deSelect()
+ isSelected()
Object3Dvehicle
lightTexture
breakTexture
rightTexture
leftTexture
+ setLightTexture()
+ setBreakTexture()
+ setRightTexture()
+ setLeftTexture()
+ getLightTexture()
+ getBreakTexture()
+ getRightTexture()
+ getLeftTexture()
* Object3Dvehicle()
* Object3Dvehicle()
* ~Object3Dvehicle()
* copy()
* copy()
* getMoveRtation()

Collaboration diagram for Object3Dvehicle:


std::basic_string< Char >
std::string
std::mutex
-configFile
-defaultObject
-defaultTexture
-miniatureFolder
-modelFolder
-textureFolder
-objectsMutex
-texturesMutex
ObjectStorage
+ object3Ds
+ object3Dvehicles
- loaded
- loadingState
- loadingStateMax
- parsedCSV
- textures
- objects
- windowIcon
- threads
- atomicThreadCounter
- firstSceneElements
- generatedObjects
* ObjectStorage()
* ~ObjectStorage()
* load()
* getWindowIcon()
* getTexture()
* getMesh()
* isLoaded()
* loadingCheck()
* finaliseLoading()
* getLoadingState()
* getLoadingStateMax()
* getObject3D()
* getObject3D()
* getObject3Dvehicle()
* getObject3Dvehicle()
* getFirstSceneElements()
* readCSV()
* loadTexture()
* isThisTextureLoaded()
* loadTextureParallel()
* bindTextures()
* loadObject()
* isThisObjectLoaded()
* loadObjectParallel()
* bindObjects()
* store()
#iconID
#meshID
#name
#textureID
#type
#objectStorage
#breakTexture
#leftTexture
#lightTexture
#rightTexture
Object3D
id
renderID
dependencyID
color
defaultPosition
position
scale
rotation
worldMatrix
hitSphere
moveSphere
hidden
protection
selectable
deleted
modelID
selected
+ Object3D()
+ Object3D()
+ ~Object3D()
+ setDependencyID()
+ setMeshID()
+ setTextureID()
+ setIconID()
+ setPosition()
+ setRotation()
+ setScale()
+ move()
+ rotateLeft()
+ rotateRight()
+ setRGBAcolor()
+ setRGBcolor()
+ setOpacity()
+ setRenderID()
+ setProtection()
+ setSelectable()
+ getId()
+ getRenderID()
+ getDependencyID()
+ getName()
+ getType()
+ getMesh()
+ getTexture()
+ getTextureID()
+ getIcon()
+ getIconID()
+ getPosition()
+ getRotation()
+ getRotationInDegree()
+ getRotationInRadian()
+ getRotationVector()
+ getScale()
+ getRGBAcolor()
+ getRGBcolor()
+ getOpacity()
+ getWorldMatrix()
+ getHitSphere()
+ getMoveSphere()
+ isProtected()
+ isSelectable()
+ copy()
+ copy()
+ isHidden()
+ hide()
+ show()
+ erase()
+ isDeleted()
+ getModelID()
+ setModelID()
+ select()
+ deSelect()
+ isSelected()
Object3Dvehicle
+ setLightTexture()
+ setBreakTexture()
+ setRightTexture()
+ setLeftTexture()
+ getLightTexture()
+ getBreakTexture()
+ getRightTexture()
+ getLeftTexture()
* Object3Dvehicle()
* Object3Dvehicle()
* ~Object3Dvehicle()
* copy()
* copy()
* getMoveRtation()

Public Member Functions

- void **setLightTexture** (int)
  Setter for special Texture setter (Not implemented yet.)

- void **setBreakTexture** (int)
  Setter for special Texture setter (Not implemented yet.)

- void **setRightTexture** (int)
  Setter for special Texture setter (Not implemented yet.)

- void **setLeftTexture** (int)
  Setter for special Texture setter (Not implemented yet.)

- **Texture2D getLightTexture** ()
  Getter for special texture.

- **Texture2D getBreakTexture** ()
  Getter for special texture.

- **Texture2D getRightTexture** ()
  Getter for special texture.

- **Texture2D getLeftTexture** ()
  Getter for special texture.

- void **setDependencyID** (int **dependencyID**)
  Setter for the dependencyID.

- void **setMeshID** (std::string **meshID**)
  Setter for the mesh.

- void **setTextureID** (std::string **textureID**)
  Setter for the Texture ID.

- void **setIconID** (std::string **iconID**)
  Setter for the Icon ID.

- void **setPosition** (glm::vec3 **position**)
  Setter for the position.

- void **setRotation** (glm::vec4 **rotation**)
  Setter for the new rotation.

- void **setScale** (glm::vec3 **scale**)
  Setter for the new scale.

- void **move** (glm::vec3 move)
  Shift the position.

- void **rotateLeft** (float rotateLeft)
  Rotate the rotation left.

- void **rotateRight** (float rotateRight)
  Rotate the rotation right.

- void **setRGBAcolor** (glm::vec4 RGBAcolor)
  Setter for the RGBA color.

- void **setRGBcolor** (glm::vec3 RGBcolor)
  Setter for the RGB color.

- void **setOpacity** (float opacity)
  Setter for the opacity.

- void **setRenderID** (int newRenderID)
  The object current renderID from the render.

- void **setProtection** (bool **isProtected**)
  Set the object protection.

- void **setSelectable** (bool **isSelectable**)
  Set the selectable flag.

- int **getId** ()
  Getter for the ID.

- int **getRenderID** ()
  Getter for the renderID.

- int **getDependencyID** ()
  Getter for the dependency ID.

- std::string **getName** ()
  Getter for the name.

- std::string **getType** ()
  Getter for the Type.

- std::unique_ptr< **Mesh** > & **getMesh** ()
  *Getter for the **Mesh**.*

- **Texture2D** & **getTexture** ()

Getter for the texture.

- std::string **getTextureID** ()
  Getter for the texture ID.

- **Texture2D** & **getIcon** ()
  Getter for the Icon.

- std::string **getIconID** ()
  Getter for the icon ID.

- glm::vec3 **getPosition** ()
  Getter for the position.

- glm::vec4 **getRotation** ()
  Getter for the rotation.

- float **getRotationInDegree** ()
  Getter for the rotation in degree.

- float **getRotationInRadian** ()
  Getter for the rotation in radian.

- glm::vec3 **getRotationVector** ()
  Getter for the rotation vector.

- glm::vec3 **getScale** ()
  Getter for the scale.

- glm::vec4 **getRGBAcolor** ()
  Getter for the RGBA color.

- glm::vec3 **getRGBcolor** ()
  Getter for the RGB color.

- float **getOpacity** ()
  Getter for the opacity.

- glm::mat4 **getWorldMatrix** ()
  Calculate the world matrix.

- glm::vec4 **getHitSphere** ()
  Getter for the hit sphere.

- glm::vec4 **getMoveSphere** ()
  Getter for the move sphere.

- bool **isProtected** ()
  Getter for the protection flag.

- bool **isSelectable** ()
  Getter for the selectable flag.

- bool **isHidden** ()
  Getter for the hidden flag.

- void **hide** ()
  Hide the object.

- void **show** ()
  Show the object.

- void **erase** ()
  Set the object deleted.

- bool **isDeleted** ()
  Getter for the deleted flag.

- size_t **getModelID** ()
  Getter for the Model ID.

- void **setModelID** (size_t **id**)
  Setter for the model ID.

- void **select** ()
  Set the object selected.

- void **deSelect** ()
  Set the object deselect.

- bool **isSelected** ()
  Getter for the selected flag.

Protected Attributes

- std::string **lightTexture**
  Light texture.

- std::string **breakTexture**
  break light texture,

- std::string **rightTexture**
  right marker texture.

- std::string **leftTexture**
  Left marker texture.

- **ObjectStorage** * **objectStorage**
  Pointer for the objects storage.

- int **id** = -1
  The object type ID.

- int **renderID** = -1
  The current object render ID in the render.

- int **dependencyID** = -1
  The dependency id if it exist.

- std::string **name** = ""
  The object type name.

- std::string **type** = ""
  The object type.

- std::string **meshID** = ""
  The geometry ID.

- std::string **textureID** = "white.png"
  The texture ID.

- std::string **iconID** = "sphere_mark_mini.png"
  The Icon ID.

- glm::vec4 **color** = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f)
  The color.

- glm::vec3 **defaultPosition** = glm::vec3(0.0f, 0.0f, 0.0f)
  The default position.

- glm::vec3 **position** = glm::vec3(0.0f, 0.0f, 0.0f)
  The current position.

- glm::vec3 **scale** = glm::vec3(1.0f, 1.0f, 1.0f)
  The current scale.

- glm::vec4 **rotation** = glm::vec4(0.0f, 0.0f, 1.0f, 0.0f)
  The current rotation.

- glm::vec4 **worldMatrix** = glm::vec4(1.0f)
  The current calculated world matrix.

- glm::vec4 **hitSphere** = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f)
  The hit sphere for the ray picking.

- glm::vec4 **moveSphere** = glm::vec4(0.0f, 0.0f, 0.0f, 1.0f)
  The move sphere.

- bool **hidden** = false
  The hidden flag.

- bool **protection** = false
  The protection flag.

- bool **selectable** = true
  The selectable flag.

- bool **deleted** = false
  The deleted flag.

- size_t **modelID** = -1
  The graph model id.

- bool **selected** = true
  The selected flag.

Traffic Simulation

- **Object3Dvehicle** (**ObjectStorage** ***objectStorage**)
  *Constructor for the empty **Vehicle** object.*

- **Object3Dvehicle** (int **id**, std::string **name**, std::string **type**, std::string **meshID**, std::string **textureID**, std::string **lightTexture**, std::string **breakTexture**, std::string **rightTexture**, std::string **leftTexture**, std::string **iconID**, glm::vec3 initPosition, glm::vec3 initScale, glm::vec4 initRotation, glm::vec4 **color**, glm::vec4 **hitSphere**, glm::vec4 **moveSphere**, **ObjectStorage** ***objectStorage**)
  Constructor for copy and initialization.

- **~Object3Dvehicle** (void)
  The vehicle destructor.

- **Object3Dvehicle copy** ()
  Copy this object to an other and return it.

- **Object3Dvehicle copy** (int **renderID**)
  Copy this object to an other and return it and set the render ID.

- static float **getMoveRtation** (glm::vec3 pointA, glm::vec3 pointB)
  Calculate the vehicle rotation from the road current and next point.

Detailed Description

The renderable vehicle representation.

The renderable vehicle representation. Contact: `sandorbalazs9402@gmail.com`.

Store all special data for the vehicles.

Author

Sándor Balázs - AZA6NL

Date

2021.12.15.

Constructor & Destructor Documentation

Object3Dvehicle::Object3Dvehicle (ObjectStorage * *objectStorage*)

Constructor for the empty **Vehicle** object.

Parameters

objectStorage	The object storage pointer.

Here is the caller graph for this function:

Object3Dvehicle::Object3Dvehicle (int *id*, std::string *name*, std::string *type*, std::string *meshID*, std::string *textureID*, std::string *lightTexture*, std::string *breakTexture*, std::string *rightTexture*, std::string *leftTexture*, std::string *iconID*, glm::vec3 *initPosition*, glm::vec3 *initScale*, glm::vec4 *initRotation*, glm::vec4 *color*, glm::vec4 *hitSphere*, glm::vec4 *moveSphere*, ObjectStorage * *objectStorage*)

Constructor for copy and initialization.

The data source is the configuration CSV.

Parameters

id	The vehicle ID.
name	The vehicle name.
type	The vehicle type.
meshID	The vehicle mesh ID.
textureID	The vehicle texture ID.
lightTexture	The vehicle lightTexture.
breakTexture	The vehicle breakTexture.
rightTexture	The vehicle rightTexture.
leftTexture	The vehicle leftTexture.
iconID	The vehicle iconID.
initPosition	The vehicle initPosition.
initScale	The vehicle initScale.
initRotation	The vehicle initRotation.
color	The vehicle color.
hitSphere	The vehicle hitSphere.
moveSphere	The vehicle moveSphere.

objectStorage	The vehicle objectStorage.

Object3Dvehicle::~Object3Dvehicle (void)

The vehicle destructor.

Parameters

Empty.	

Member Function Documentation

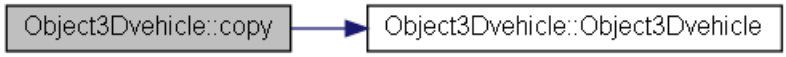
Object3Dvehicle Object3Dvehicle::copy ()

Copy this object to an other and return it.

Returns

The copy from this object.

Here is the call graph for this function:



Here is the caller graph for this function:

Object3Dvehicle Object3Dvehicle::copy (int *renderID*)

Copy this object to an other and return it and set the render ID.

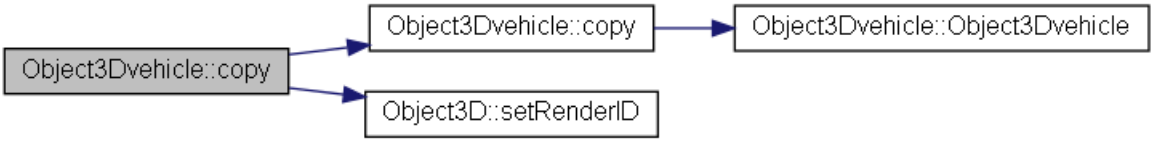
Parameters

renderID	The new render id.

Returns

The copy from this object.

Here is the call graph for this function:

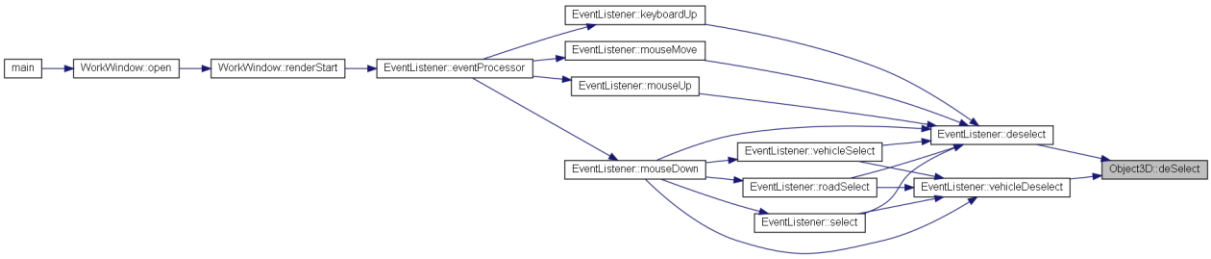


void Object3D::deSelect () `[inherited]`

Set the object deselect.

Here is the call graph for this function:

Here is the caller graph for this function:



void Object3D::erase () [inherited]

Set the object deleted.

Texture2D Object3Dvehicle::getBreakTexture ()

Getter for special texture.

(Not implemented yet.)

Returns

The needed special texture.

int Object3D::getDependencyID () [inherited]

Getter for the dependency ID.

Returns

The dependency ID.

Here is the caller graph for this function:

glm::vec4 Object3D::getHitSphere () [inherited]

Getter for the hit sphere.

Returns

The hit sphere.

Here is the caller graph for this function:

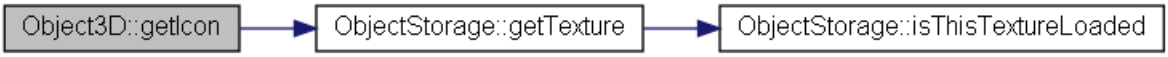
Texture2D & Object3D::getIcon () [inherited]

Getter for the Icon.

Returns

The icon.

Here is the call graph for this function:



Here is the caller graph for this function:

std::string Object3D::getIconID () [inherited]

Getter for the icon ID.

Returns

The icon ID.

int Object3D::getId () [inherited]

Getter for the ID.

Returns

The ID.

Here is the caller graph for this function:

Texture2D Object3Dvehicle::getLeftTexture ()

Getter for special texture.

(Not implemented yet.)

Returns

The needed special texture.

Texture2D Object3Dvehicle::getLightTexture ()

Getter for special texture.

(Not implemented yet.)

Returns

The needed special texture.

std::unique_ptr< Mesh > & Object3D::getMesh () [inherited]

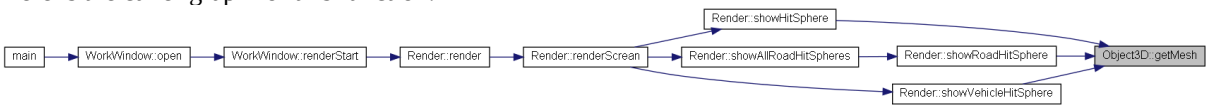
Getter for the **Mesh**.

Returns

The mesh.

Here is the call graph for this function:

Here is the caller graph for this function:



size_t Object3D::getModelID () [inherited]

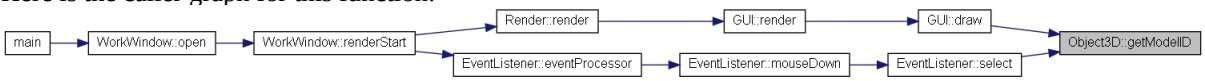
Getter for the Model ID.

(Link for the graph model.)

Returns

The Model ID.

Here is the caller graph for this function:



float Object3Dvehicle::getMoveRtation (glm::vec3 *pointA*, glm::vec3 *pointB*) [static]

Calculate the vehicle rotation from the road current and next point.

Parameters

pointA	The current point of the road.
pointB	The next point of the road.

Returns

The calculated direction.

Here is the caller graph for this function:

glm::vec4 Object3D::getMoveSphere () [inherited]

Getter for the move sphere.

Returns

The move sphere.

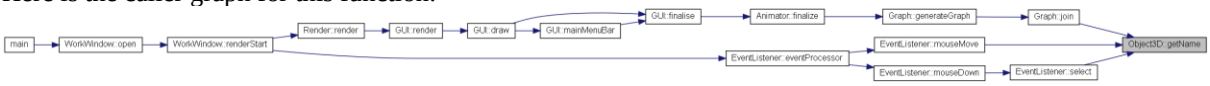
std::string Object3D::getName () [inherited]

Getter for the name.

Returns

The name.

Here is the caller graph for this function:



float Object3D::getOpacity () [inherited]

Getter for the opacity.

Returns

The opacity.

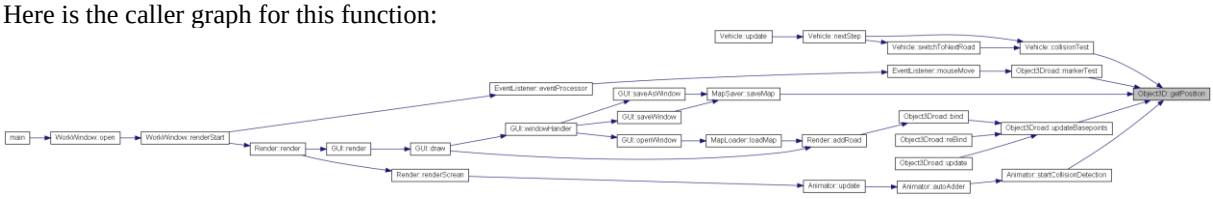
glm::vec3 Object3D::getPosition () `[inherited]`

Getter for the position.

Returns

The Position.

Here is the caller graph for this function:



int Object3D::getRenderID () `[inherited]`

Getter for the renderID.

Returns

The renderID.

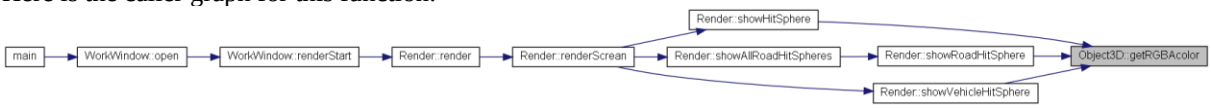
glm::vec4 Object3D::getRGBAcolor () `[inherited]`

Getter for the RGBA color.

Returns

The RGBA color.

Here is the caller graph for this function:



glm::vec3 Object3D::getRGBcolor () `[inherited]`

Getter for the RGB color.

Returns

RGB color.

Texture2D Object3Dvehicle::getRightTexture ()

Getter for special texture.

(Not implemented yet.)

Returns

The needed special texture.

glm::vec4 Object3D::getRotation () [inherited]

Getter for the rotation.

Returns

The rotation.

Here is the caller graph for this function:

float Object3D::getRotationInDegree () [inherited]

Getter for the rotation in degree.

Returns

The rotation in degree.

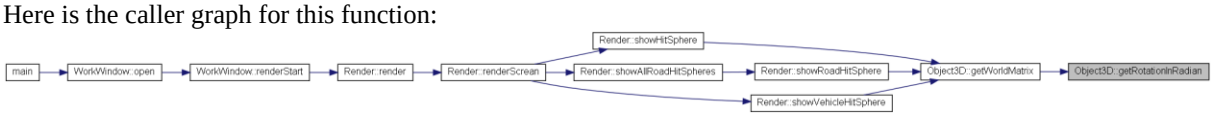
float Object3D::getRotationInRadian () [inherited]

Getter for the rotation in radian.

Returns

the rotation in radian.

Here is the caller graph for this function:



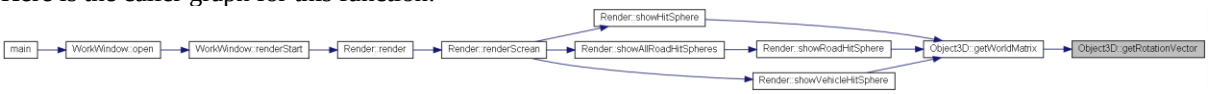
glm::vec3 Object3D::getRotationVector () [inherited]

Getter for the rotation vector.

Returns

The rotation vector.

Here is the caller graph for this function:



glm::vec3 Object3D::getScale () [inherited]

Getter for the scale.

Returns

The scale.

Texture2D & Object3D::getTexture () [inherited]

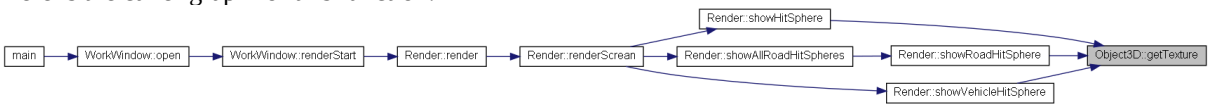
Getter for the texture.

Returns

The texture.

Here is the call graph for this function:

Here is the call graph for this function:



std::string Object3D::getTextureID () [inherited]

Getter for the texture ID.

Returns

The texture ID.

std::string Object3D::getType () [inherited]

Getter for the Type.

Returns

The type.

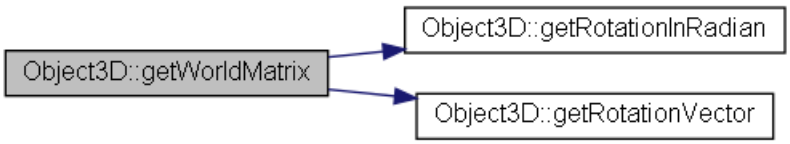
glm::mat4 Object3D::getWorldMatrix () [inherited]

Calculate the world matrix.

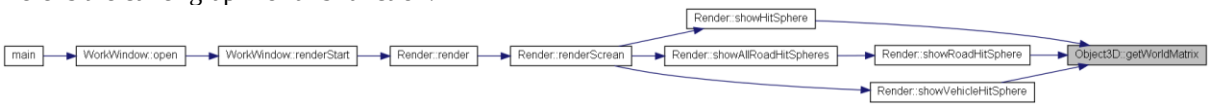
Returns

The calculated World Matrix.

Here is the call graph for this function:



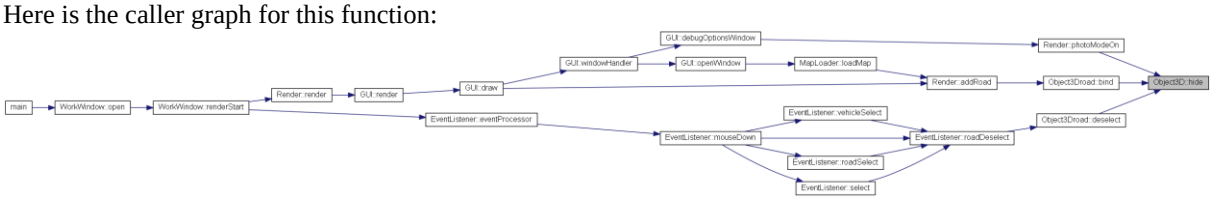
Here is the caller graph for this function:



void Object3D::hide () [inherited]

Hide the object.

Here is the caller graph for this function:



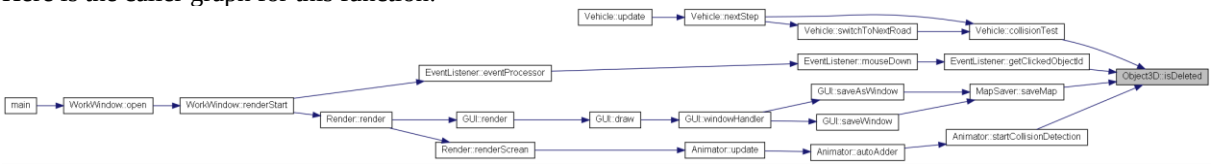
bool Object3D::isDeleted () [inherited]

Getter for the deleted flag.

Returns

The deleted flag.

Here is the caller graph for this function:



bool Object3D::isHidden () [inherited]

Getter for the hidden flag.

Returns

The hidden flag.

Here is the caller graph for this function:

bool Object3D::isProtected () [inherited]

Getter for the protection flag.

Returns

The protection flag.

Here is the caller graph for this function:

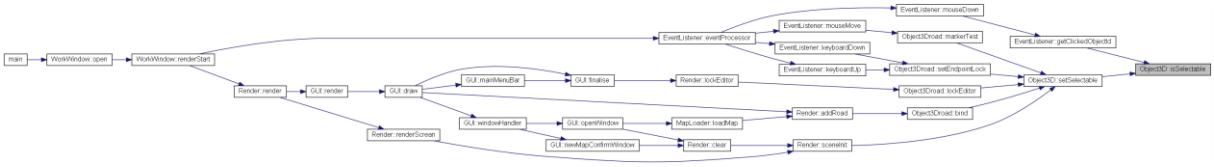
bool Object3D::isSelectable () [inherited]

Getter for the selectable flag.

Returns

The selectable flag.

Here is the caller graph for this function:



bool Object3D::isSelected () `[inherited]`

Getter for the selected flag.

Returns

The selected flag.

void Object3D::move (glm::vec3 *move*) `[inherited]`

Shift the position.

Parameters

move	The shift vector.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3D::rotateLeft (float *rotateLeft*) `[inherited]`

Rotate the rotation left.

Parameters

rotateLeft	The left rotation.

Here is the call graph for this function:

Here is the caller graph for this function:

void Object3D::rotateRight (float *rotateRight*) `[inherited]`

Rotate the rotation right.

Parameters

rotateLeft	The left right.

Here is the call graph for this function:

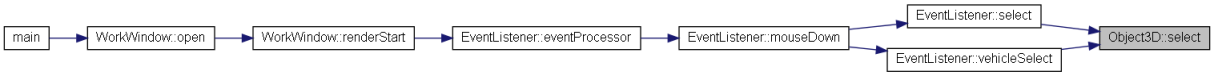
Here is the caller graph for this function:

void Object3D::select () `[inherited]`

Set the object selected.

Here is the call graph for this function:

Here is the caller graph for this function:



void Object3Dvehicle::setBreakTexture (int)

Setter for special Texture setter (Not implemented yet.)

Parameters

The	new texture ID

void Object3D::setDependencyID (int *dependencyID*) `[inherited]`

Setter for the dependencyID.

(Used when the object has dependency.(road))

Parameters

dependencyID	The dependencyID

Here is the caller graph for this function:

void Object3D::setIconID (std::string *iconID*) `[inherited]`

Setter for the Icon ID.

Parameters

iconID	The Icon ID.

void Object3Dvehicle::setLeftTexture (int)

Setter for special Texture setter (Not implemented yet.)

Parameters

The	new texture ID

void Object3Dvehicle::setLightTexture (int)

Setter for special Texture setter (Not implemented yet.)

Parameters

The	new texture ID

void Object3D::setMeshID (std::string *meshID*) `[inherited]`

Setter for the mesh.

Parameters

meshID	The mesh.

void Object3D::setModelID (size_t *id*) `[inherited]`

Setter for the model ID.

(Link for the graph model.)

Parameters

id	The new model ID.

Here is the caller graph for this function:

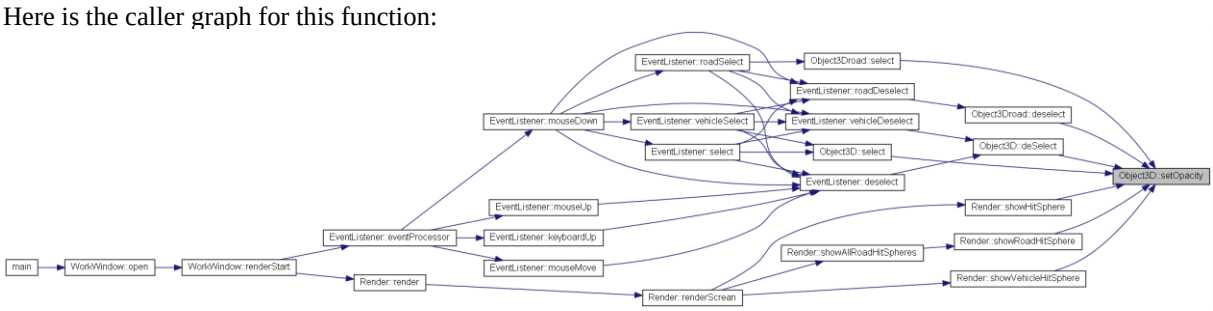
void Object3D::setOpacity (float *opacity*) `[inherited]`

Setter for the opacity.

Parameters

opacity	The new opacity.

Here is the caller graph for this function:



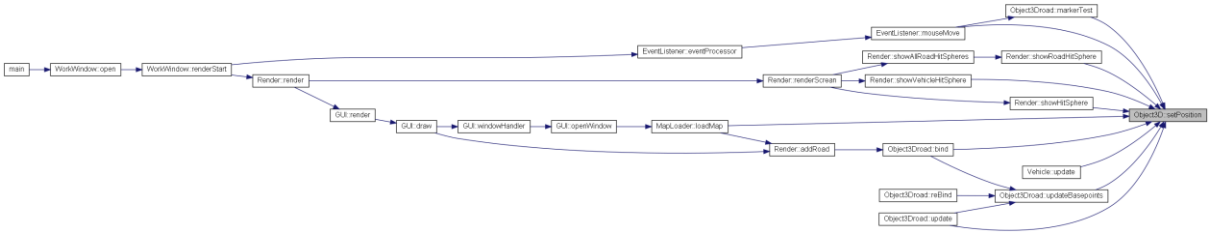
void Object3D::setPosition (glm::vec3 *position*) `[inherited]`

Setter for the position.

Parameters

position	The new position.

Here is the caller graph for this function:



void Object3D::setProtection (bool *isProtected*) `[inherited]`

Set the object protection.

Parameters

isProtected	The new protection state.

Here is the call graph for this function:

Here is the caller graph for this function:

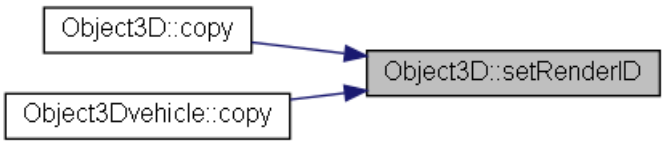
void Object3D::setRenderID (int *newRenderID*) `[inherited]`

The object current renderID from the render.

Parameters

newRenderID	The new render ID.

Here is the caller graph for this function:



void Object3D::setRGBAcolor (glm::vec4 *RGBAcolor*) `[inherited]`

Setter for the RGBA color.

Parameters

RGBAcolor	The new RGBA color.

void Object3D::setRGBcolor (glm::vec3 *RGBcolor*) `[inherited]`

Setter for the RGB color.

Parameters

RGBAcolor	The new RGB color.

Here is the caller graph for this function:

void Object3Dvehicle::setRightTexture (int)

Setter for special Texture setter (Not implemented yet.)

Parameters

The	new texture ID

void Object3D::setRotation (glm::vec4 *rotation*) `[inherited]`

Setter for the new rotation.

Parameters

rotation	The new rotation.

Here is the caller graph for this function:

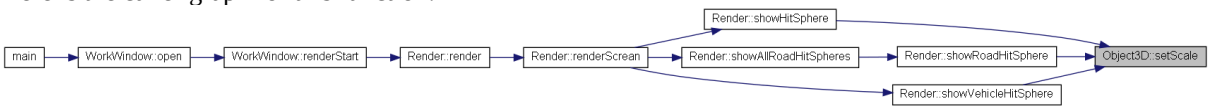
void Object3D::setScale (glm::vec3 *scale*) `[inherited]`

Setter for the new scale.

Parameters

scale	The new scale.

Here is the caller graph for this function:



void Object3D::setSelectable (bool *isSelectable*) `[inherited]`

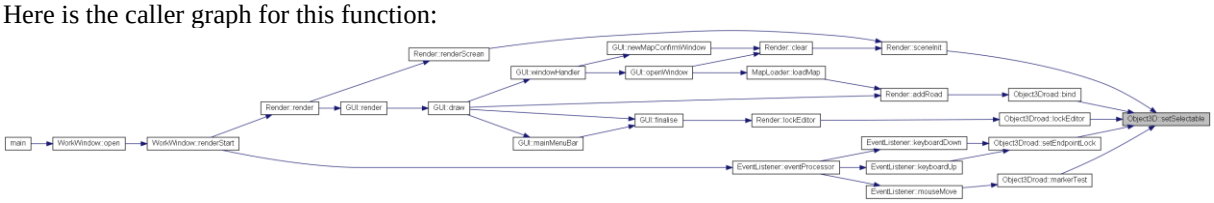
Set the selectable flag.

Parameters

isSelectable	The new state for the selectable flag.

Here is the call graph for this function:

Here is the caller graph for this function:



void Object3D::setTextureID (std::string *textureID*) `[inherited]`

Setter for the Texture ID.

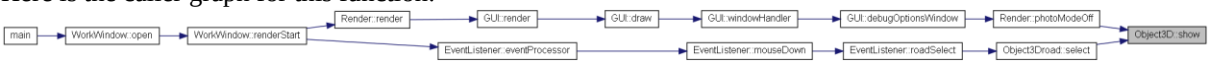
Parameters

textureID	the Texture ID,

void Object3D::show () [inherited]

Show the object.

Here is the caller graph for this function:



Field Documentation

std::string Object3Dvehicle::breakTexture [protected]

break light texture,

glm::vec4 Object3D::color = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f) [protected], [inherited]

The color.

glm::vec3 Object3D::defaultPosition = glm::vec3(0.0f, 0.0f, 0.0f) [protected], [inherited]

The default position.

bool Object3D::deleted = false [protected], [inherited]

The deleted flag.

int Object3D::dependencyID = -1 [protected], [inherited]

The dependency id if it exist.

bool Object3D::hidden = false [protected], [inherited]

The hidden flag.

glm::vec4 Object3D::hitSphere = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f) [protected], [inherited]

The hit sphere for the ray picking.

std::string Object3D::iconID = "sphere_mark_mini.png" [protected], [inherited]

The Icon ID.

int Object3D::id = -1 [protected], [inherited]

The object type ID.

std::string Object3Dvehicle::leftTexture `[protected]`

Left marker texture.

std::string Object3Dvehicle::lightTexture `[protected]`

Light texture.

std::string Object3D::meshID = "" `[protected], [inherited]`

The geometry ID.

size_t Object3D::modelID = -1 `[protected], [inherited]`

The graph model id.

glm::vec4 Object3D::moveSphere = glm::vec4(0.0f, 0.0f, 0.0f, 1.0f) `[protected], [inherited]`

The move sphere.

std::string Object3D::name = "" `[protected], [inherited]`

The object type name.

ObjectStorage* Object3D::objectStorage `[protected], [inherited]`

Pointer for the objects storage.

glm::vec3 Object3D::position = glm::vec3(0.0f, 0.0f, 0.0f) `[protected], [inherited]`

The current position.

bool Object3D::protection = false `[protected], [inherited]`

The protection flag.

int Object3D::renderID = -1 `[protected], [inherited]`

The current object render ID in the render.

std::string Object3Dvehicle::rightTexture `[protected]`

right marker texture.

glm::vec4 Object3D::rotation = glm::vec4(0.0f, 0.0f, 1.0f, 0.0f) `[protected], [inherited]`

The current rotation.

glm::vec3 Object3D::scale = glm::vec3(1.0f, 1.0f, 1.0f) `[protected], [inherited]`

The current scale.

bool Object3D::selectable = true `[protected], [inherited]`

The selectable flag.

bool Object3D::selected = true `[protected], [inherited]`

The selected flag.

std::string Object3D::textureID = "white.png" `[protected], [inherited]`

The texture ID.

std::string Object3D::type = "" `[protected], [inherited]`

The object type.

glm::vec4 Object3D::worldMatrix = glm::vec4(1.0f) `[protected], [inherited]`

The current calculated world matrix.

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

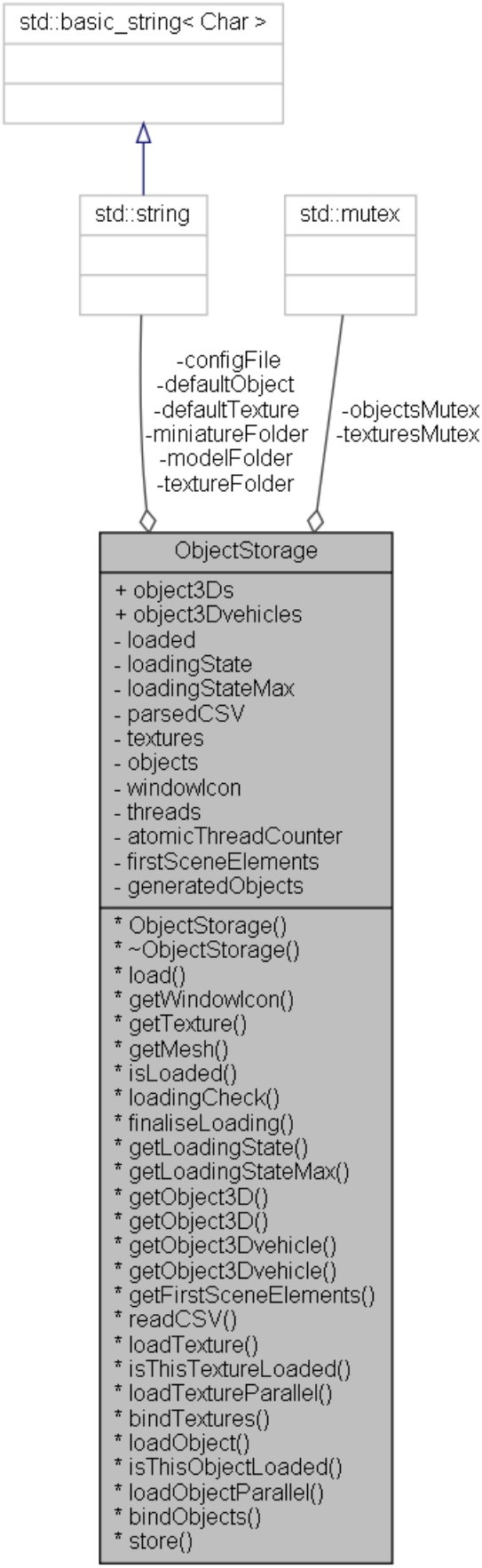
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**Object3Dvehicle.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**Object3Dvehicle.cpp**

ObjectStorage Class Reference

Load and store the 3D Models and Textures and create the render objects.

```
#include <ObjectStorage.h>
```

Collaboration diagram for ObjectStorage:


std::basic_string< Char >
std::string
std::mutex
-configFile
-defaultObject
-defaultTexture
-miniatureFolder
-modelFolder
-textureFolder
-objectsMutex
-texturesMutex
ObjectStorage
+ object3Ds
+ object3Dvehicles
- loaded
- loadingState
- loadingStateMax
- parsedCSV
- textures
- objects
- windowIcon
- threads
- atomicThreadCounter
- firstSceneElements
- generatedObjects
* ObjectStorage()
* ~ObjectStorage()
* load()
* getWindowIcon()
* getTexture()
* getMesh()
* isLoaded()
* loadingCheck()
* finaliseLoading()
* getLoadingState()
* getLoadingStateMax()
* getObject3D()
* getObject3D()
* getObject3Dvehicle()
* getObject3Dvehicle()
* getFirstSceneElements()
* readCSV()
* loadTexture()
* isThisTextureLoaded()
* loadTextureParallel()
* bindTextures()
* loadObject()
* isThisObjectLoaded()
* loadObjectParallel()
* bindObjects()
* store()

Data Fields

- std::map< int, **Object3D** > **object3Ds**
  The original 3D objects storage with the original configurations.

- std::map< int, **Object3Dvehicle** > **object3Dvehicles**
  The original 3D vehicle objects storage with the original configurations.

Private Attributes

- const std::string **defaultTexture** = "default.png"
  The default texture.

- const std::string **defaultObject** = "sphere_mark.obj"
  The default 3d Object.

- const std::string **configFile** = "3Dobjects/3d_object_library.csv"
  The configuration CSV file path.

- const std::string **textureFolder** = "3Dobjects/**textures**/"
  The texture storage folder path.

- const std::string **miniatureFolder** = "miniatures/"
  The miniatures storage folder path.

- const std::string **modelFolder** = "3Dobjects/models/"
  The 3D Objects storage folder path.

- bool **loaded** = false
  Loading finish flag.

- int **loadingState** = 0
  The current loading state.

- int **loadingStateMax** = 0
  The maximum of the loading state.

- std::vector< std::vector< std::string > > **parsedCSV**
  The loaded, parsed configuration CSV file.

- std::mutex **texturesMutex**
  Mutex for the parallel texture loading thread safety.

- std::mutex **objectsMutex**
  Mutex for the parallel 3D Object loading thread safety.

- std::map< std::string, **Texture2D** > **textures**
  The texture objects storage.

- std::map< std::string, std::unique_ptr< **Mesh** > > **objects**
  The 3D Objects storage.

- SDL_Surface * **windowIcon**
  The current used window icon.

- std::vector< std::thread > **threads**
  The loading threads storage.

- SDL_atomic_t **atomicThreadCounter**
  The thread safe thread counter.

- std::vector< int > **firstSceneElements**
  List of the first scene elements.

- std::map< std::string, **VertexArrayObject** > **generatedObjects**
  The storage of the dynamically generated objects.

Traffic Simulation

- **ObjectStorage** (void)
  Constructor, empty.

- **~ObjectStorage** (void)
  Destructor, empty.

- void **load** ()
  Load the CSV configuration file and load the Objects and the textures parallel by the CSV configuration file.

- SDL_Surface * **getWindowIcon** ()
  Getter for the window icon.

- **Texture2D** & **getTexture** (std::string textureName)
  Getter for the textures.

- std::unique_ptr< **Mesh** > & **getMesh** (std::string meshName)
  Getter for the Meshes.

- bool **isLoaded** ()
  Loading finish checker.

- bool **loadingCheck** ()
  Check all the loader threads.

- void **finaliseLoading** ()

Join all loading threads, bind the objects and textures.

- int **getLoadingState** ()
  Getter for the current state of the loading procedure.

- int **getLoadingStateMax** ()
  Getter for the maximum of the loading state for detect the loading finish.

- **Object3D getObject3D** (int object3Did)
  Create a new copy of the needed 3D object.

- **Object3D getObject3D** (int object3Did, int renderID)
  Create a new copy of the needed 3D object and set its render id too.

- **Object3Dvehicle getObject3Dvehicle** (int object3Did)
  Create a new copy of the needed 3D vehicle object.

- **Object3Dvehicle getObject3Dvehicle** (int object3Did, int renderID)
  Create a new copy of the needed 3D vehicle object and set its render id too.

- std::vector< int > & **getFirstSceneElements** ()
  Get the first scene elements list.

- void **readCSV** ()
  Read, process and store the configuration CSV file.

- void **loadTexture** (std::string fileName)
  Start load a texture in parallel thread safe.

- bool **isThisTextureLoaded** (std::string textureName)
  Thread loading checker.

- std::thread **loadTextureParallel** (std::string fileName)
  Load a texture in new thread.

- void **bindTextures** ()
  Bind loaded textures.

- void **loadObject** (std::string fileName)
  Start load a 3D Object in parallel thread safe.

- bool **isThisObjectLoaded** (std::string objectName)
  Thread loading checker.

- std::thread **loadObjectParallel** (std::string fileName)
  Load a 3D Object in a new thread.

- void **bindObjects** ()
  Bind a loaded object.

- void **store** (int csvID)
  Create and store the renderable 3D object data.

Detailed Description

Load and store the 3D Models and Textures and create the render objects.

Load and store the 3D Models and Textures and create the render objects. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.12.15.

Constructor & Destructor Documentation

ObjectStorage::ObjectStorage (void)

Constructor, empty.

Parameters

Empty.	

ObjectStorage::~ObjectStorage (void)

Destructor, empty.

Parameters

Empty.	

Member Function Documentation

void ObjectStorage::bindObjects () `[private]`

Bind a loaded object.

Here is the caller graph for this function:

void ObjectStorage::bindTextures () `[private]`

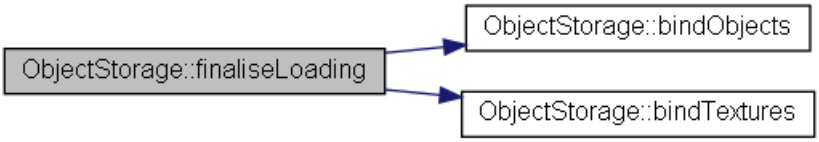
Bind loaded textures.

Here is the caller graph for this function:

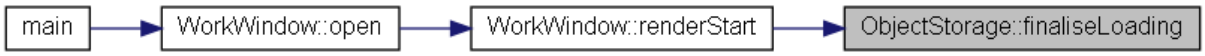
void ObjectStorage::finaliseLoading ()

Join all loading threads, bind the objects and textures.

Here is the call graph for this function:



Here is the caller graph for this function:



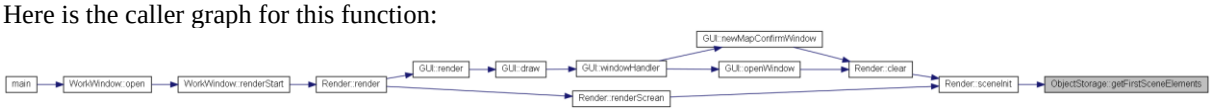
std::vector< int > & ObjectStorage::getFirstSceneElements ()

Get the first scene elements list.

Returns

The first scene elements list.

Here is the caller graph for this function:



int ObjectStorage::getLoadingState ()

Getter for the current state of the loading procedure.

Returns

The loading state of the loading procedure.

int ObjectStorage::getLoadingStateMax ()

Getter for the maximum of the loading state for detect the loading finish.

Returns

The maximum of the loading state.

std::unique_ptr< Mesh > & ObjectStorage::getMesh (std::string *meshName*)

Getter for the Meshes.

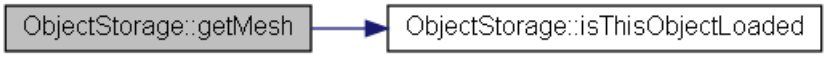
Parameters

textureName	The needed **Mesh** name.

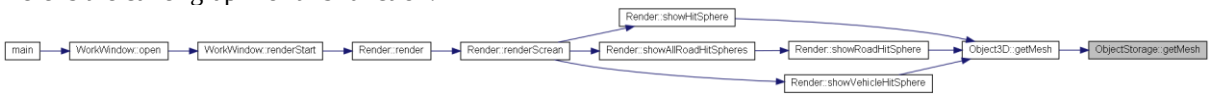
Returns

The needed **Mesh**.

Here is the call graph for this function:



Here is the caller graph for this function:



Object3D ObjectStorage::getObject3D (int *object3Did*)

Create a new copy of the needed 3D object.

If it not found give the first.

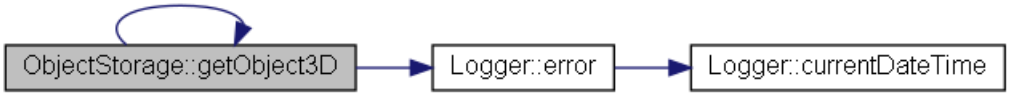
Parameters

object3Did	The needed 3D Object ID.

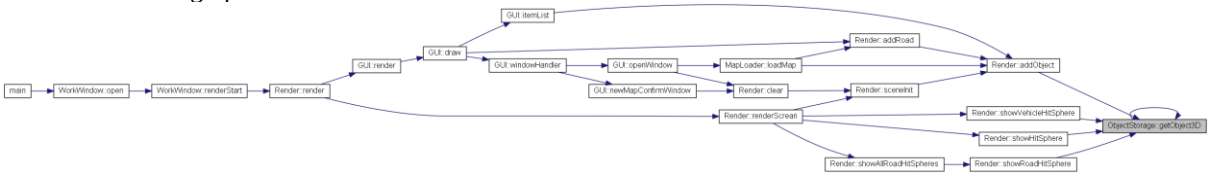
Returns

The copy of the needed 3D Object.

Here is the call graph for this function:



Here is the caller graph for this function:



Object3D ObjectStorage::getObject3D (int *object3Did*, int *renderID*)

Create a new copy of the needed 3D object and set its render id too.

If it not found give the first.

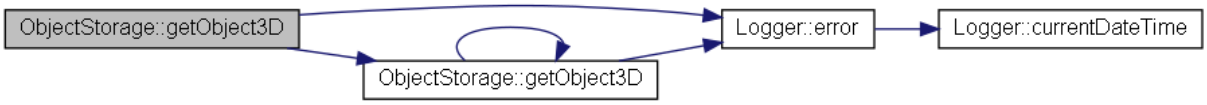
Parameters

object3Did	The needed 3D Object ID.
renderID	The needed 3D copy new render ID.

Returns

The copy of the needed 3D Object.

Here is the call graph for this function:



Object3Dvehicle ObjectStorage::getObject3Dvehicle (int *object3Did*)

Create a new copy of the needed 3D vehicle object.

If it not found give the first (5).

Parameters

object3Did	The needed 3D vehicle Object ID.

Returns

The copy of the needed 3D vehicle Object.

Here is the call graph for this function:

Here is the caller graph for this function:

Object3Dvehicle ObjectStorage::getObject3Dvehicle (int *object3Did*, int *renderID*)

Create a new copy of the needed 3D vehicle object and set its render id too.

If it not found give the first (5).

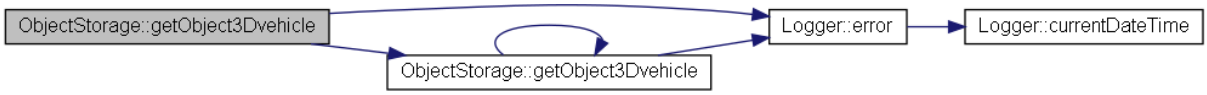
Parameters

object3Did	The needed 3D vehicle Object ID.
renderID	The needed 3D copy new render ID.

Returns

The copy of the needed 3D vehicle Object.

Here is the call graph for this function:



Texture2D & ObjectStorage::getTexture (std::string *textureName*)

Getter for the textures.

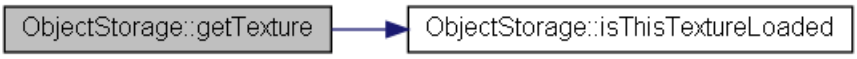
Parameters

textureName	The needed texture name.

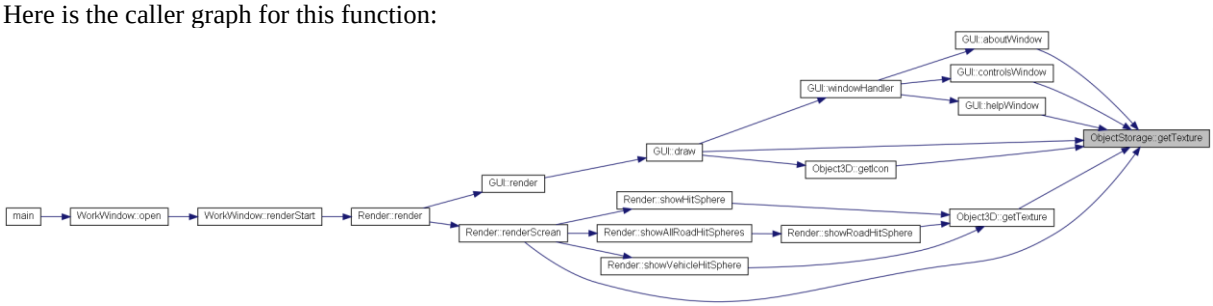
Returns

The needed texture.

Here is the call graph for this function:



Here is the caller graph for this function:



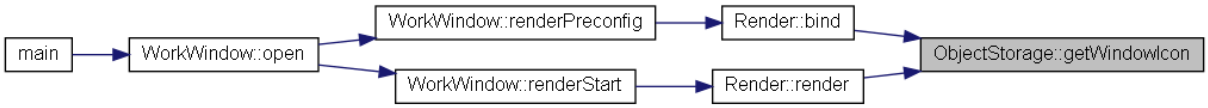
SDL_Surface * ObjectStorage::getWindowIcon ()

Getter for the window icon.

Returns

The current window icon.

Here is the caller graph for this function:



bool ObjectStorage::isLoaded ()

Loading finish checker.

Returns

Return true if the loading is finished.

Here is the caller graph for this function:

bool ObjectStorage::isThisObjectLoaded (std::string *objectName*) `[private]`

Thread loading checker.

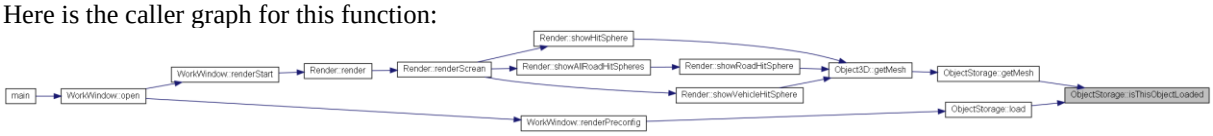
Parameters

textureName	The loadable 3D Object.

Returns

True if the loadable 3D Object is loaded.

Here is the caller graph for this function:



bool ObjectStorage::isThisTextureLoaded (std::string *textureName*) `[private]`

Thread loading checker.

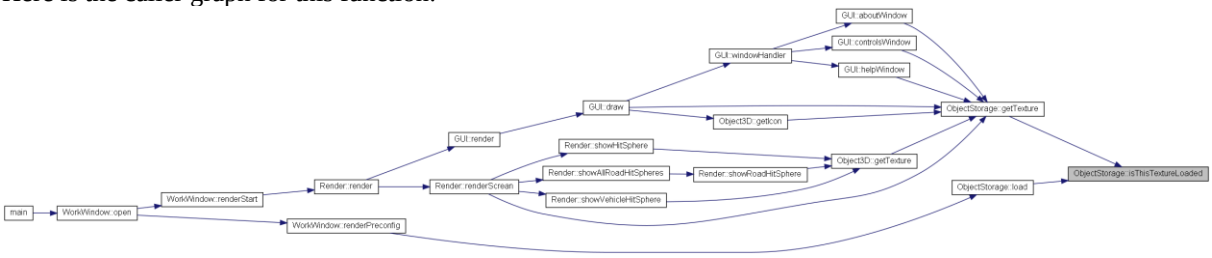
Parameters

textureName	The loadable texture.

Returns

True if the loadable texture is loaded.

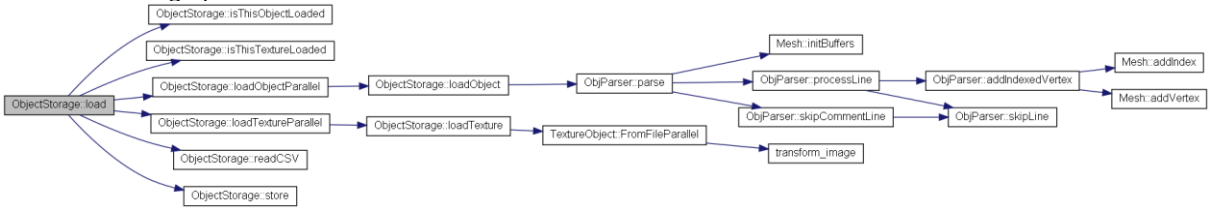
Here is the caller graph for this function:



void ObjectStorage::load ()

Load the CSV configuration file and load the Objects and the textures parallel by the CSV configuration file.

Here is the call graph for this function:



Here is the caller graph for this function:

bool ObjectStorage::loadingCheck ()

Check all the loader threads.

Return true if all loader threads finished.

Returns

True if all loader threads finished.

Here is the caller graph for this function:

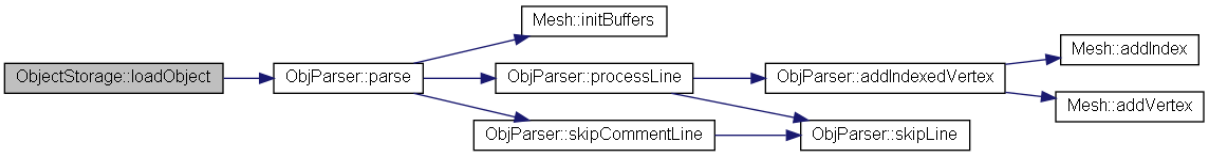
void ObjectStorage::loadObject (std::string *fileName*) `[private]`

Start load a 3D Object in parallel thread safe.

Parameters

fileName	The loadable 3D Object name.

Here is the call graph for this function:



Here is the caller graph for this function:

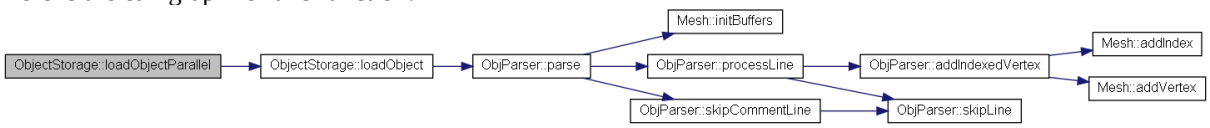
std::thread ObjectStorage::loadObjectParallel (std::string *fileName*) `[private]`

Load a 3D Object in a new thread.

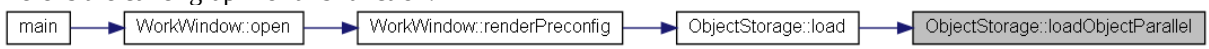
Parameters

fileName	The loadable 3D Object name.

Here is the call graph for this function:



Here is the caller graph for this function:



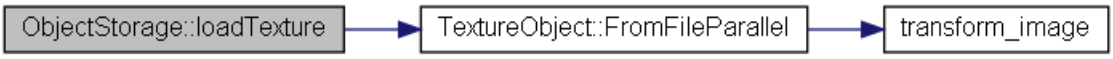
void ObjectStorage::loadTexture (std::string *fileName*) [private]

Start load a texture in parallel thread safe.

Parameters

fileName	The loadable texture name.

Here is the call graph for this function:



Here is the caller graph for this function:

std::thread ObjectStorage::loadTextureParallel (std::string *fileName*) [private]

Load a texture in new thread.

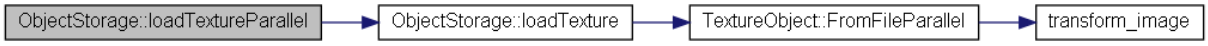
Parameters

fileName	The loadable texture name.

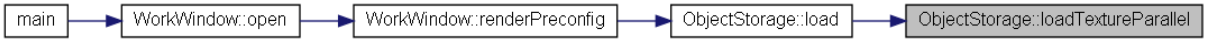
Returns

The new thread.

Here is the call graph for this function:



Here is the caller graph for this function:



void ObjectStorage::readCSV () [private]

Read, process and store the configuration CSV file.

Here is the caller graph for this function:

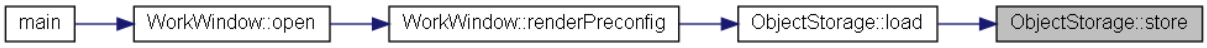
void ObjectStorage::store (int *csvID*) [private]

Create and store the renderable 3D object data.

Parameters

csvID	The processable row ID of the CSV configuration file.

Here is the caller graph for this function:



Field Documentation

SDL_atomic_t ObjectStorage::atomicThreadCounter [private]

The thread safe thread counter.

const std::string ObjectStorage::configFile = "3Dobjects/3d_object_library.csv" `[private]`

The configuration CSV file path.

const std::string ObjectStorage::defaultObject = "sphere_mark.obj" `[private]`

The default 3d Object.

(Use if the needed is not available because some reason.)

const std::string ObjectStorage::defaultTexture = "default.png" `[private]`

The default texture.

(Use if the needed is not available because some reason.)

std::vector<int> ObjectStorage::firstSceneElements `[private]`

List of the first scene elements.

std::map<std::string, VertexArrayObject> ObjectStorage::generatedObjects `[private]`

The storage of the dynamically generated objects.

NOT USED.

bool ObjectStorage::loaded = false `[private]`

Loading finish flag.

int ObjectStorage::loadingState = 0 `[private]`

The current loading state.

int ObjectStorage::loadingStateMax = 0 `[private]`

The maximum of the loading state.

const std::string ObjectStorage::miniatureFolder = "miniatures/" `[private]`

The miniatures storage folder path.

const std::string ObjectStorage::modelFolder = "3Dobjects/models/" `[private]`

The 3D Objects storage folder path.

std::map<int, Object3D> ObjectStorage::object3Ds

The original 3D objects storage with the original configurations.

std::map<int, Object3Dvehicle> ObjectStorage::object3Dvehicles

The original 3D vehicle objects storage with the original configurations.

std::map<std::string, std::unique_ptr<Mesh> > ObjectStorage::objects `[private]`

The 3D Objects storage.

All 3D Objet is loaded to the memory only once and all object use these.

std::mutex ObjectStorage::objectsMutex `[private]`

Mutex for the parallel 3D Object loading thread safety.

std::vector<std::vector<std::string> > ObjectStorage::parsedCSV `[private]`

The loaded, parsed configuration CSV file.

const std::string ObjectStorage::textureFolder = "3Dobjects/textures/" `[private]`

The texture storage folder path.

std::map<std::string, Texture2D> ObjectStorage::textures `[private]`

The texture objects storage.

All texture is loaded to the memory only once and all object use these.

std::mutex ObjectStorage::texturesMutex `[private]`

Mutex for the parallel texture loading thread safety.

std::vector<std::thread> ObjectStorage::threads `[private]`

The loading threads storage.

SDL_Surface* ObjectStorage::windowIcon `[private]`

The current used window icon.

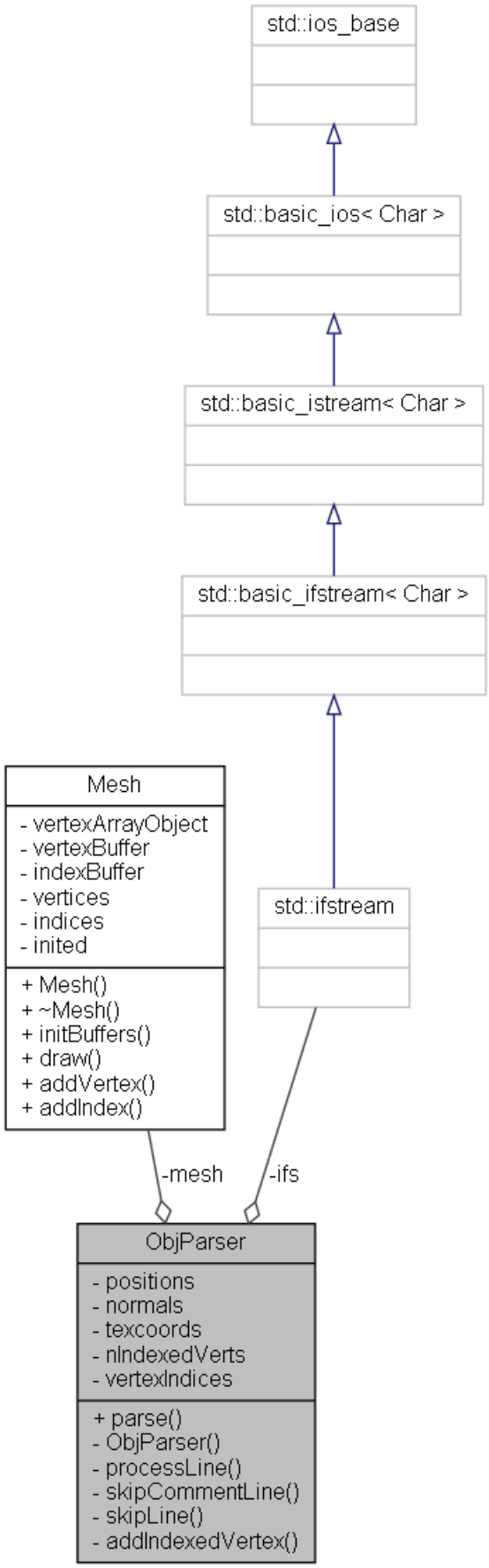
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**ObjectStorage.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/**ObjectStorage.cpp**

ObjParser Class Reference

```
#include <ObjParser_OGL3.h>
```

Collaboration diagram for ObjParser:


std::ios_base
std::basic_ios< Char >
std::basic_istream< Char >
std::basic_ifstream< Char >
Mesh
- vertexArrayObject
- vertexBuffer
- indexBuffer
- vertices
- indices
- inited
+ Mesh()
+ ~Mesh()
+ initBuffers()
+ draw()
+ addVertex()
+ addIndex()
std::ifstream
-mesh
-ifs
ObjParser
- positions
- normals
- texcoords
- nIndexedVerts
- vertexIndices
+ parse()
- ObjParser()
- processLine()
- skipCommentLine()
- skipLine()
- addIndexedVertex()

Data Structures

- struct **IndexedVert**

Public Types

- enum **Exception** { **EXC_FILENOTFOUND** }

Static Public Member Functions

- static std::unique_ptr< **Mesh** > **parse** (const char *fileName, bool parallel=false)

Private Member Functions

- **ObjParser** (void)
- bool **processLine** ()
- bool **skipCommentLine** ()
- void **skipLine** ()
- void **addIndexedVertex** (const **IndexedVert** &vertex)

Private Attributes

- **Mesh** * **mesh**
- std::ifstream **ifs**
- std::vector< glm::vec3 > **positions**
- std::vector< glm::vec3 > **normals**
- std::vector< glm::vec2 > **texcoords**
- unsigned int **nIndexedVerts**
- std::map< **IndexedVert**, unsigned int > **vertexIndices**

Member Enumeration Documentation

enum ObjParser::Exception

Enumerator:

EXC_FILENOTFOUND	

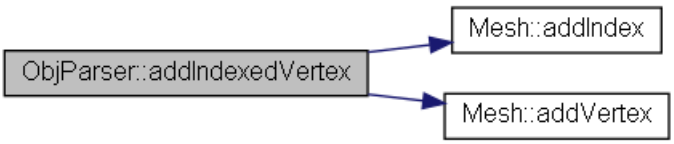
Constructor & Destructor Documentation

ObjParser::ObjParser (void) `[inline], [private]`

Member Function Documentation

void ObjParser::addIndexedVertex (const IndexedVert & *vertex*) `[private]`

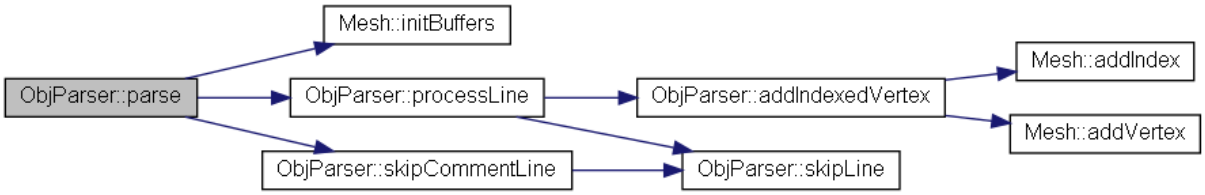
Here is the call graph for this function:



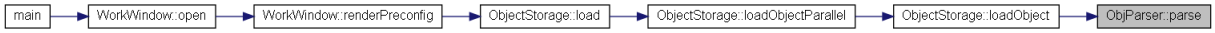
Here is the caller graph for this function:

std::unique_ptr< Mesh > ObjParser::parse (const char * *fileName*, bool *parallel* = `false`) `[static]`

Here is the call graph for this function:

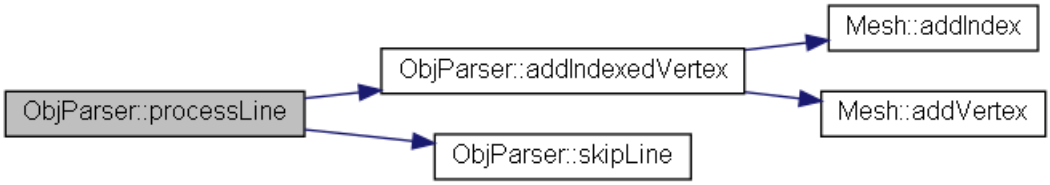


Here is the caller graph for this function:



bool ObjParser::processLine () `[private]`

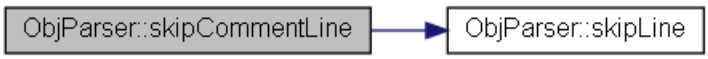
Here is the call graph for this function:



Here is the caller graph for this function:

bool ObjParser::skipCommentLine () `[private]`

Here is the call graph for this function:



Here is the caller graph for this function:

void ObjParser::skipLine () `[private]`

Here is the caller graph for this function:

Field Documentation

std::ifstream ObjParser::ifs `[private]`

Mesh* ObjParser::mesh `[private]`

unsigned int ObjParser::nIndexedVerts `[private]`

std::vector<glm::vec3> ObjParser::normals `[private]`

std::vector<glm::vec3> ObjParser::positions `[private]`

std::vector<glm::vec2> ObjParser::texcoords `[private]`

std::map<IndexedVert, unsigned int> ObjParser::vertexIndices `[private]`

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

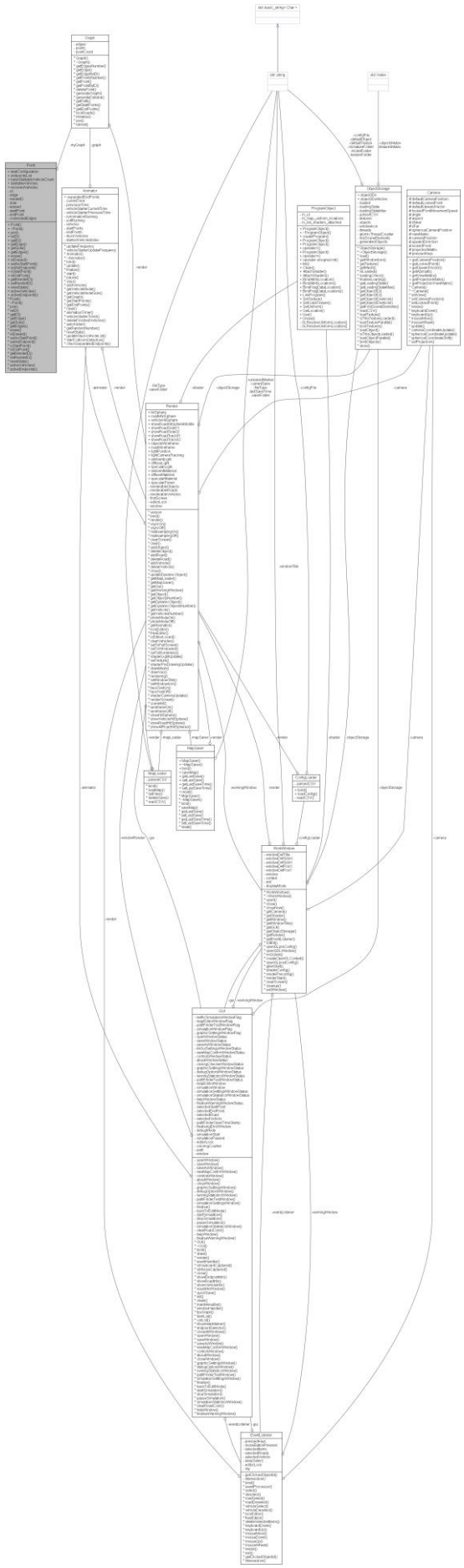
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ObjParser_OGL3.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ObjParser_OGL3.cpp**

Point Class Reference

The abstract point implementation.

```
#include <Point.h>
```

Collaboration diagram for Point:



Public Member Functions

- Traffic Simulation**Point** (size_t **id**, size_t **edge**, **Graph** ***myGraph**, char **side**)
  *The **Point** constructor.*

- **~Point** (void)
- void **join** (**Point** *point)
  Join this point to an other.

- void **reID** (size_t **id**)
  Change the point iD and rebind the roads too.

- size_t **getID** ()
  Getter for the point id.

- size_t **getEdge** ()
  Getter for the point belongs to this edge.

- char **getSide** ()
  Getter for the side of the edge.

- std::set< size_t > **getEdges** ()
  geter for the connected edges.

- void **erase** ()
  Logical deleting.

- bool **isErased** ()
  Is deleted?

- void **setAsStartPoint** ()
  Set the point as a start point.

- void **setAsEndpoint** ()
  Set the point as an endpoint.

- bool **isStartPoint** ()
  Is a start point?

- bool **isEndPoint** ()
  Is an endpoint?

- size_t **getRenderID** ()
  Getter for the renderID.

- void **setRenderID** (size_t **id**)

Setter for the renderID.

- void **resetStats** ()
  Resetting the running statistics.

- std::vector< size_t > **activeVehicles** ()
  Return active vehicles.

- std::vector< size_t > **activeEndpoints** ()
  Return active endpoints.

Data Fields

- std::vector< size_t > **startConfiguration**
  Startable cars if this is a startpoint.

- std::vector< bool > **endpointsList**
  Target endpoints if this is a startpoint.

- const int **baseStartableVehicleCount** = 50
  Base startable vehicle count.

- int **startableVehicles** = **baseStartableVehicleCount**
  The startable cars if this is a startpoint.

- int **receivedVehicles** = 0
  The received cars if this is a endpoint.

Private Attributes

- size_t **id** = -1
  The point id.

- size_t **edge** = -1
  This point belongs to this edge.

- size_t **renderID** = -1
  The renderable object ID if it exist.

- char **side** = 'Q'
  This point belongs to this side of the edge.

- bool **erased** = false
  Marker for logic delete.

- bool **startPoint** = false
  Mark for start points.

- bool **endPoint** = false
  mark for endpoints.

- **Graph * myGraph**
  My graph object.

- std::set< size_t > **connectedEdges**
  Set of connected edges.

Detailed Description

The abstract point implementation.

The abstract graph **Point** class. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.20.

Constructor & Destructor Documentation

Point::Point (size_t *id*, size_t *edge*, Graph * *myGraph*, char *side*)

The **Point** constructor.

Parameters

id	The unique ID of the point.
myGraph	the graph witch contains the element.
edge	This point belongs to this abstract edge.
side	This point belongs to this side of the road.
id	The unique ID of the point.
edge	This point belongs to this abstract edge.
side	This point belongs to this side of the road.

Point::~Point (void)

Member Function Documentation

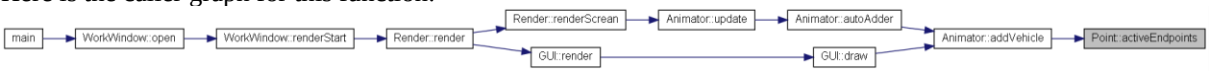
std::vector< size_t > Point::activeEndpoints ()

Return active endpoints.

Returns

Return active endpoints.

Here is the caller graph for this function:



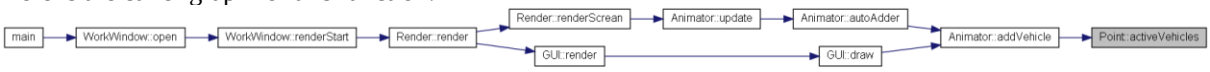
std::vector< size_t > Point::activeVehicles ()

Return active vehicles.

Returns

Return active vehicles.

Here is the caller graph for this function:



void Point::erase ()

Logical deleting.

size_t Point::getEdge ()

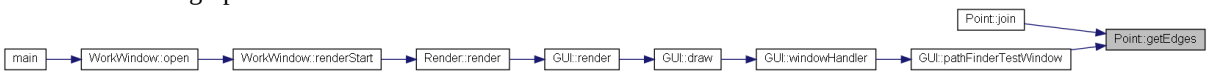
Getter for the point belongs to this edge.

Here is the caller graph for this function:

std::set< size_t > Point::getEdges ()

geter for the connected edges.

Here is the caller graph for this function:

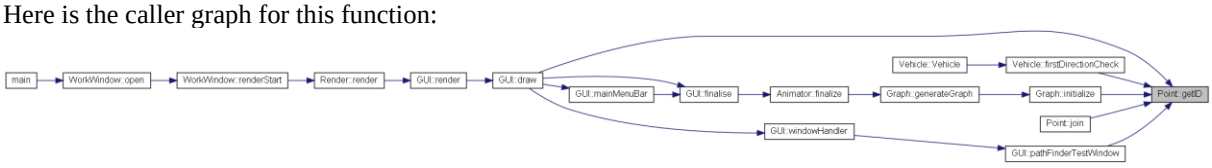


size_t Point::getID ()

Getter for the point id.

This is an unique ID.

Here is the caller graph for this function:



size_t Point::getRenderID ()

Getter for the renderID.

If it not have it is -1.

Returns

The renderID. If it not have it is -1.

char Point::getSide ()

Getter for the side of the edge.

Here is the caller graph for this function:

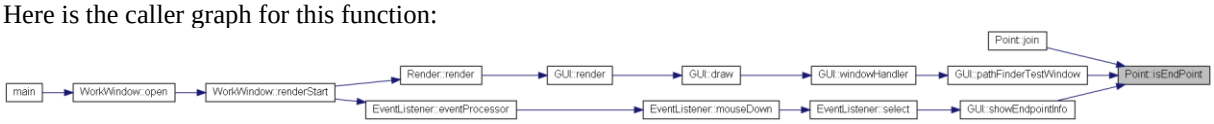
bool Point::isEndPoint ()

Is an endpoint?

Returns

True id this is an endpoint.

Here is the caller graph for this function:



bool Point::isErased ()

Is deleted?

Returns

Return true if it is deleted.

Here is the caller graph for this function:

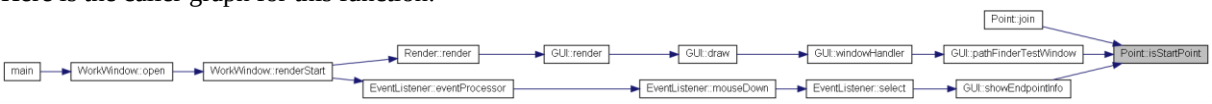
bool Point::isStartPoint ()

Is a start point?

Returns

True is this is a start point.

Here is the caller graph for this function:



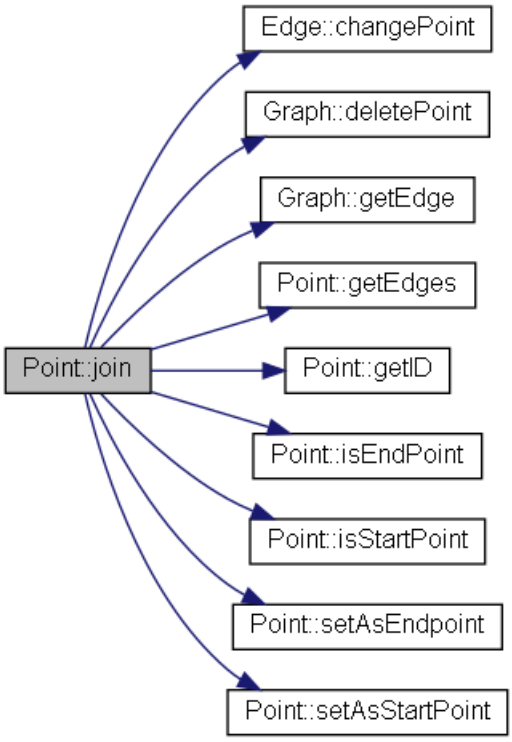
void Point::join (Point * *point*)

Join this point to an other.

Parameters

The	joinable point.

Here is the call graph for this function:



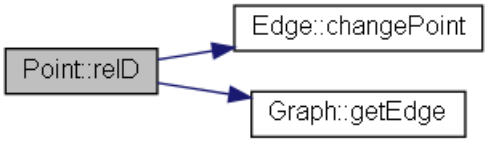
void Point::reID (size_t *id*)

Change the point iD and rebind the roads too.

Parameters

id	the new id.

Here is the call graph for this function:



void Point::resetStats ()

Resetting the running statistics.

Here is the caller graph for this function:

void Point::setAsEndpoint ()

Set the point as an endpoint.

Here is the caller graph for this function:

void Point::setAsStartPoint ()

Set the point as a start point.

Here is the caller graph for this function:

void Point::setRenderID (size_t *id*)

Setter for the renderID.

If it not have it is -1. This is the default.

Field Documentation

const int Point::baseStartableVehicleCount = 50

Base startable vehicle count.

std::set<size_t> Point::connectedEdges `[private]`

Set of connected edges.

(Store edgeID-s) (Contain own edge too)

size_t Point::edge = -1 `[private]`

This point belongs to this edge.

bool Point::endPoint = false `[private]`

mark for endpoints.

std::vector<bool> Point::endpointsList

Target endpoints if this is a startpoint.

bool Point::erased = false `[private]`

Marker for logic delete.

size_t Point::id = -1 `[private]`

The point id.

This is an unique ID.

Graph* Point::myGraph `[private]`

My graph object.

int Point::receivedVehicles = 0

The received cars if this is a endpoint.

size_t Point::renderID = -1 `[private]`

The renderable object ID if it exist.
If not exist -1.

char Point::side = 'Q' `[private]`

This point belongs to this side of the edge.

int Point::startableVehicles = baseStartableVehicleCount

The startable cars if this is a startpoint.

std::vector<size_t> Point::startConfiguration

Startable cars if this is a startpoint.

bool Point::startPoint = false `[private]`

Mark for start points.

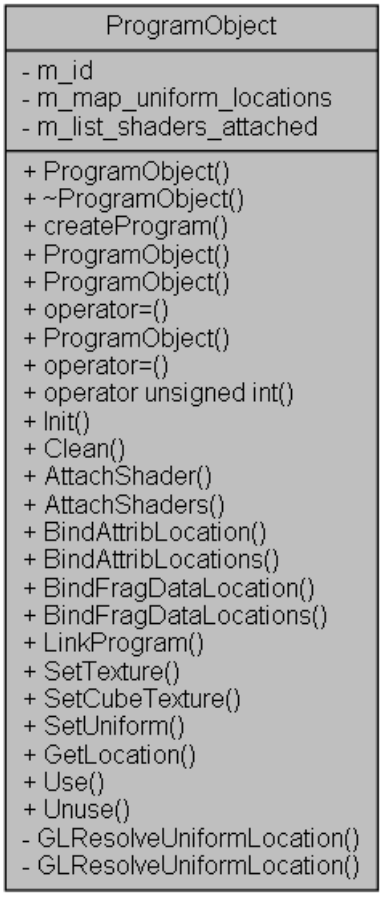
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Point.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Point.cpp**

ProgramObject Class Reference

`#include <ProgramObject.h>`

Collaboration diagram for ProgramObject:



Public Types

- using **Binding** = std::pair< int, const char * >

Public Member Functions

- **ProgramObject** ()
- **~ProgramObject** ()
- void **createProgram** ()
- **ProgramObject** (std::initializer_list< **ShaderObject** >, std::initializer_list< **Binding** >={}, std::initializer_list< **Binding** >={})
- **ProgramObject** (const **ProgramObject** &)=delete
- **ProgramObject** & **operator=** (const **ProgramObject** &)=delete
- **ProgramObject** (**ProgramObject** &&rhs)
- **ProgramObject** & **operator=** (**ProgramObject** &&rhs)
- **operator unsigned int** () const
- void **Init** (std::initializer_list< **ShaderObject** >, std::initializer_list< **Binding** >={}, std::initializer_list< **Binding** >={})
- void **Clean** ()
- **ProgramObject** & **AttachShader** (const **ShaderObject** &)
- **ProgramObject** & **AttachShaders** (std::initializer_list< **ShaderObject** >)
- void **BindAttribLocation** (int, const char *) const

- void **BindAttribLocations** (std::initializer_list< **Binding** >) const
- void **BindFragDataLocation** (int, const char *) const
- void **BindFragDataLocations** (std::initializer_list< **Binding** >) const
- bool **LinkProgram** ()
- void **SetTexture** (const char *_uniform, int _sampler, GLuint _textureID)
- void **SetCubeTexture** (const char *_uniform, int _sampler, GLuint _textureID)
- template<typename U , typename T > void **SetUniform** (U _uniform, const T &pArr)
- GLint **GetLocation** (const char *_uniform)
- void **Use** () const
- void **Unuse** () const

Private Member Functions

- GLint **GLResolveUniformLocation** (GLint _uniform)
- GLint **GLResolveUniformLocation** (const char *_uniform)

Private Attributes

- GLuint **m_id**
- std::unordered_map< std::string, GLint > **m_map_uniform_locations**
- std::vector< GLuint > **m_list_shaders_attached**

Member Typedef Documentation

using ProgramObject::Binding = std::pair<int, const char*>

Constructor & Destructor Documentation

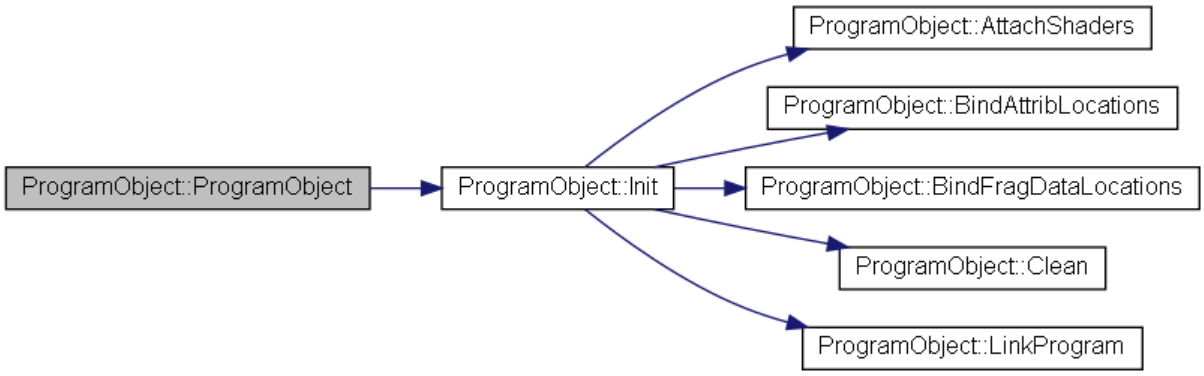
ProgramObject::ProgramObject ()

ProgramObject::~ProgramObject ()

Here is the call graph for this function:

ProgramObject::ProgramObject (std::initializer_list< ShaderObject > *shaderList*, std::initializer_list< Binding > *attribLocationBindingList* = { }, std::initializer_list< Binding > *fragDataBindingList* = { })

Here is the call graph for this function:



ProgramObject::ProgramObject (const ProgramObject &) `[delete]`

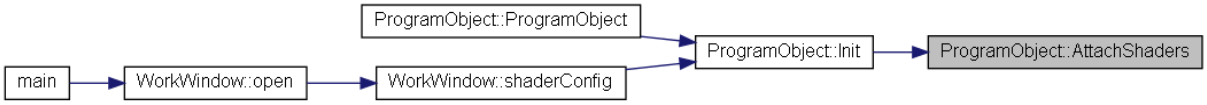
ProgramObject::ProgramObject (ProgramObject && *rhs*)

Member Function Documentation

ProgramObject & ProgramObject::AttachShader (const ShaderObject & *shader*)

ProgramObject & ProgramObject::AttachShaders (std::initializer_list< ShaderObject > *shaderList*)

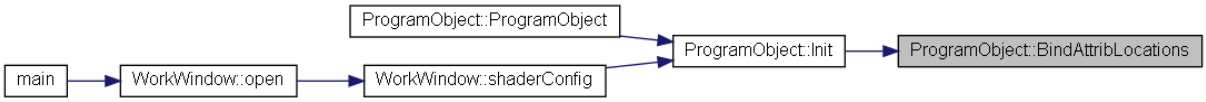
Here is the caller graph for this function:



void ProgramObject::BindAttribLocation (int *_index*, const char * *_inVariableName*) const

void ProgramObject::BindAttribLocations (std::initializer_list< Binding >) const

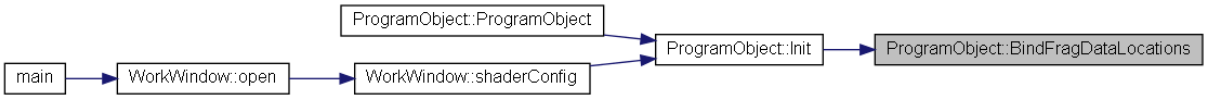
Here is the caller graph for this function:



void ProgramObject::BindFragDataLocation (int *_index*, const char * *_outVariableName*) const

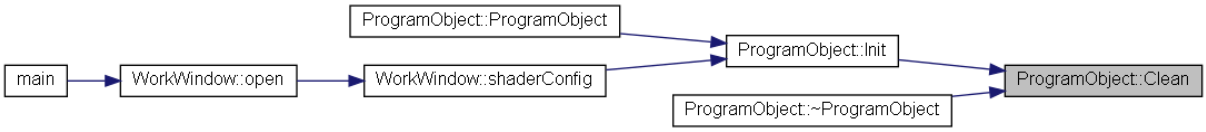
void ProgramObject::BindFragDataLocations (std::initializer_list< Binding > *fragDataBindingList*) const

Here is the caller graph for this function:



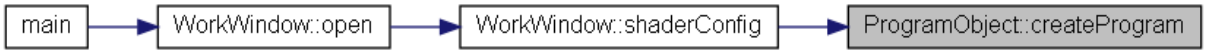
void ProgramObject::Clean ()

Here is the caller graph for this function:



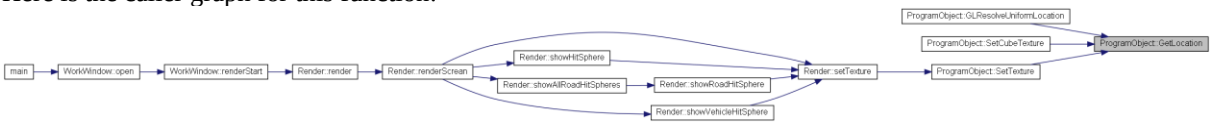
void ProgramObject::createProgram ()

Here is the caller graph for this function:



GLint ProgramObject::GetLocation (const char * *_uniform*)

Here is the caller graph for this function:

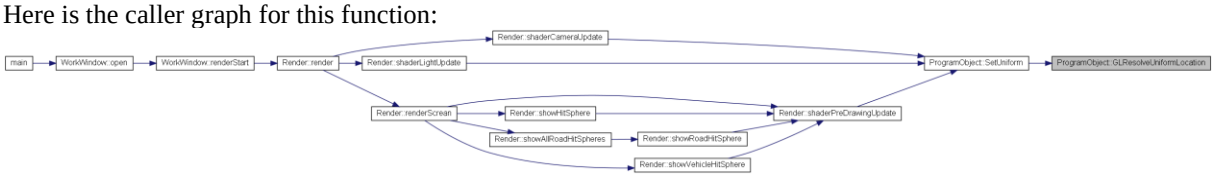


GLint ProgramObject::GLResolveUniformLocation (const char * _*uniform*) `[private]`

Here is the call graph for this function:

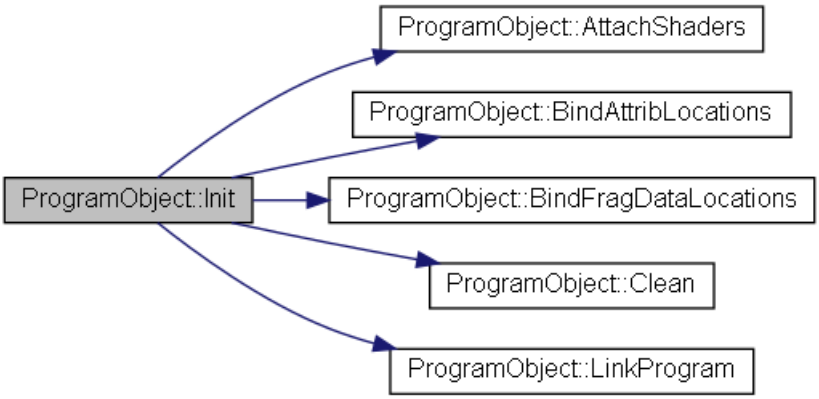
GLint ProgramObject::GLResolveUniformLocation (GLint _*uniform*) `[private]`

Here is the caller graph for this function:

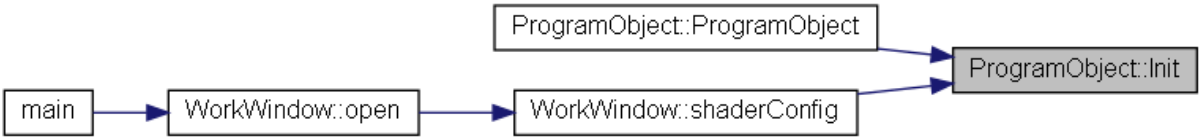


void ProgramObject::Init (std::initializer_list< ShaderObject > *shaderList*, std::initializer_list< Binding > *attribLocationBindingList* = `{}`, std::initializer_list< Binding > *fragDataBindingList* = `{}`)

Here is the call graph for this function:

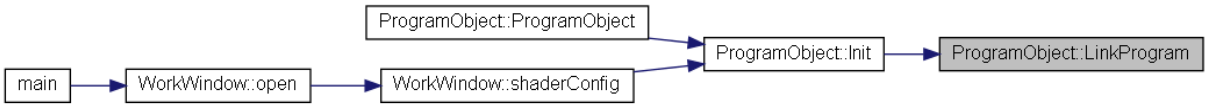


Here is the caller graph for this function:



bool ProgramObject::LinkProgram ()

Here is the caller graph for this function:



ProgramObject::operator unsigned int () const `[inline]`

ProgramObject & ProgramObject::operator= (const ProgramObject &) `[delete]`

ProgramObject & ProgramObject::operator= (ProgramObject && *rhs*)

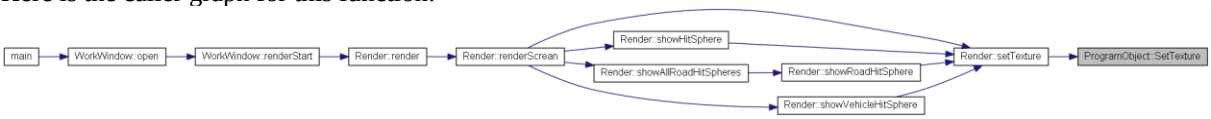
void ProgramObject::SetCubeTexture (const char * _*uniform*, int _*sampler*, GLuint _*textureID*)

Here is the call graph for this function:

void ProgramObject::SetTexture (const char * _*uniform*, int _*sampler*, GLuint _*textureID*)

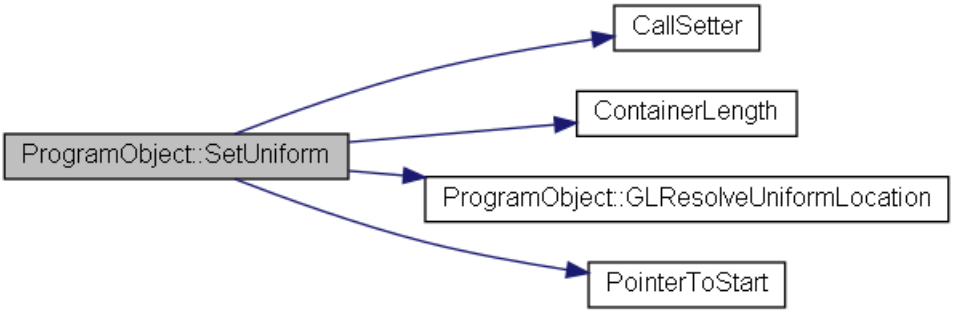
Here is the call graph for this function:

Here is the caller graph for this function:

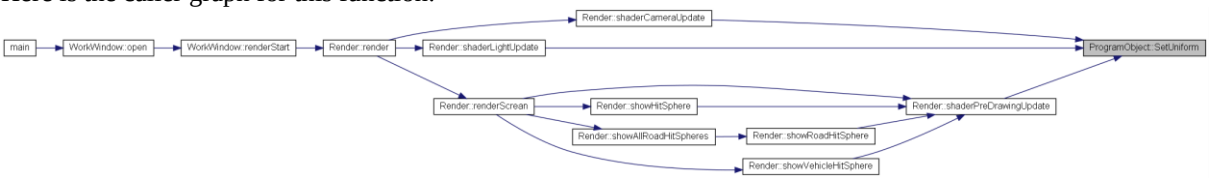


template<typename U , typename T > void ProgramObject::SetUniform (U _*uniform*, const T & *pArr*)

Here is the call graph for this function:

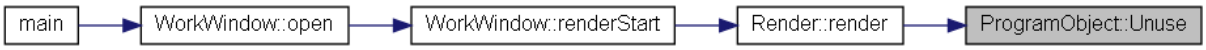


Here is the caller graph for this function:



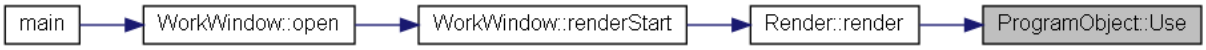
void ProgramObject::Unuse () const

Here is the caller graph for this function:



void ProgramObject::Use () const

Here is the caller graph for this function:



Field Documentation

GLuint ProgramObject::m_id `[private]`

std::vector< GLuint > ProgramObject::m_list_shaders_attached `[private]`

std::unordered_map< std::string, GLint > ProgramObject::m_map_uniform_locations `[private]`

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

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ProgramObject.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ProgramObject.cpp**

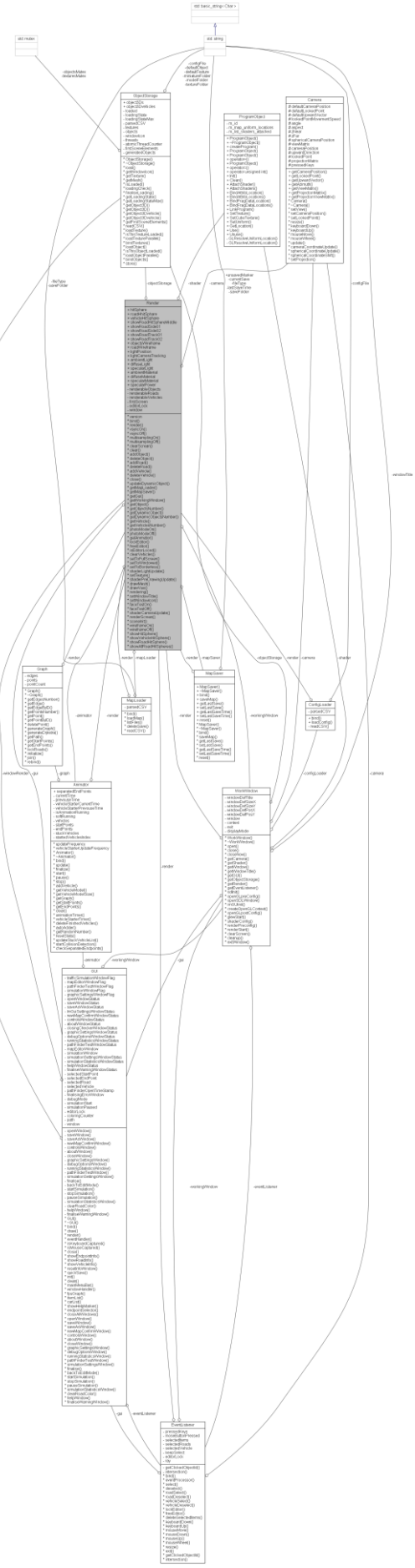
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ProgramObject.inl**

Render Class Reference

The 3D Scene render implementation.

```
#include <Render.h>
```

Collaboration diagram for Render:


std::basic_string< Char >
std::mutex
std::string
ObjectStorage
ProgramObject
Camera
Render
Graph
MapLoader
MapSaver
ConfigLoader
Animator
WorldWindow
UI
EventListener

Data Fields

- bool **hitSphare** = false
  Active the hit sphere drawing.

- bool **roadHhitSphare** = false
  Active the road hit sphere drawing.

- bool **vehicleHitSphare** = false
  Active the vehicle hit sphere drawing.

- bool **showRoadHitSphereMiddle** = true
  Show the road hit spheres.

- bool **showRoadSide01** = false
  Show road side 1 markers.

- bool **showRoadSide02** = false
  Show road side 2 markers.

- bool **showRoadTrack01** = false
  Show road track 1 markers.

- bool **showRoadTrack02** = false
  Show road track 2 markers.

- bool **objectsWireframe** = false
  Active the wire frame function for the objects.

- bool **roadWireframe** = false
  Active the wire frame function for the roads.

- glm::vec3 **lightPosition** = glm::vec3(55, 55, 55)
  The current light position, if the light position is fixed.

- bool **lightCameraTracking** = true
  *If it is true the light position is same with the **Camera** position.*

- glm::vec4 **ambientLight** = glm::vec4(0.1f, 0.1f, 0.1f, 1.0f)
  Ambient light.

- glm::vec4 **diffuseLight** = glm::vec4(0.75f, 0.75f, 0.75f, 1.0f)
  Diffuse light.

- glm::vec4 **specularLight** = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f)
  Specular light.

- glm::vec4 **ambientMaterial** = glm::vec4(1.0f, 1.0f, 1.0f, 0.0f)
  Ambient material properties.

- glm::vec4 **diffuseMaterial** = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f)
  Diffuse material.

- glm::vec4 **specularMaterial** = glm::vec4(1.0f, 1.0f, 1.0f, 0.0f)
  Specular material.

- float **specularPower** = 32.0f
  Specular light power.

Private Attributes

- std::vector< **Object3D** > **renderableObjects**
  List of the renderable scene objects.

- std::vector< **Object3Droad** * > **renderableRoads**
  List of the renderable scene roads.

- std::vector< **Object3Dvehicle** > **renderableVehicles**
  The lost of renderable vehicle.

- bool **firstScreen** = true
  Flag variable for the first scene configurations.

- bool **editorLock** = false
  Flag for editor locking.

- **WorkWindow** * **workingWindow**
  The access pointer for the current working window object.

- SDL_Window * **window**
  The access pointer for the current window object.

- **Camera** * **camera**
  The access pointer for the current camera object.

- **ProgramObject** * **shader**
  The access pointer for the current shader program.

- **GUI** * **gui**
  *The access pointer for the current **GUI** object.*

- **ObjectStorage** * **objectStorage**

The access pointer for the current object storage (This store the preloaded meshes and textures, allways in one instance.).

- **MapLoader mapLoader**
  The map loader class.

- **MapSaver mapSaver**
  The map saver class.

- **Animator animator**
  The render's animator instance.

Traffic Simulation

- static const char * **version** = "v1.3"
  The current version number.

- void **bind** (**WorkWindow *window**)
  Binding the render to the window.

- void **render** ()
  ***Render** the next frame.*

- void **vsyncOn** ()
  ***Render** mode configurator - Vsync on.*

- void **vsyncOff** ()
  ***Render** mode configurator - Vsync off.*

- void **multisamplingOn** ()
  ***Render** mode configurator - Multisampling on.*

- void **multisamplingOff** ()
  ***Render** mode configurator - Multisampling off.*

- void **clearScrean** ()
  Clearing the scene.

- void **clear** ()
  Clean and renew the map editor.

- int **addObject** (int objectStorageID)
  Add new renderable object from the preloaded object storage to the scene.

- void **deleteObject** (int renderID)
  Delete a scene object by renderID.

- int **addRoad** ()
  Add dynamically generated road to the scene.

- void **deleteRoad** (int dynamicRenderID)
  Delete a scene road object by dynamicRenderID.

- size_t **addVehicle** (int objectStorageID, size_t modelID)
  Add a new vehicle to the scene, from the objectStorage.

- void **deleteVehicle** (size_t renderableVehicleID)
  Delete a vehicle by renderID.

- void **close** ()
  Exit program.

- void **updateDynamicObject** (size_t dynamicRenderID)
  Update a dynamically generated object status by dynamicRenderID.

- **MapLoader** * **getMapLoader** ()
  Getter for the map loader program.

- **MapSaver** * **getMapSaver** ()
  Getter for the map saver program.

- **GUI** * **getGui** ()
  *Getter for the **GUI**.*

- **WorkWindow** * **getWorkingWindow** ()
  Getter for the working window pointer.

- **Object3D** * **getObject** (size_t renderID)
  Get scene preloaded object from the scene render storage.

- size_t **getObjectsNumber** ()
  Getter for the RenderableRoads size.

- **Object3Droad** * **getDynamicObject** (size_t dynamicRenderID)
  Getter for the RenderableRoads.

- size_t **getDynamicObjectsNumber** ()
  Getter for the renderableRoads size.

- **Object3Dvehicle** * **getVehicle** (size_t renderableVehicleID)
  Getter for the renderableVehicles;.

- size_t **getVehiclesNumber** ()
  Getter for the renderableVehicles size.

- void **photoModeOn** ()
  Hide the desk for object photo mode.

- void **photoModeOff** ()
  Hide the desk for object photo mode.

- **Animator** * **getAnimator** ()
  *Getter for **Animator**.*

- void **lockEditor** ()
  Lock the editor functions.

- void **freeEditor** ()
  Free the editor functions.

- bool **isEditorLoced** ()
  Return true if the editor locked.

- void **clearVehicles** ()
  Delete all active vehicles.

- void **setToFullScreen** ()
  Setter to full screen window mode.

- void **setToWindowed** ()
  Setter for windowed screen mode.

- void **setToBorderless** ()
  Setter for borderless screen mode.

- void **shaderLightUpdate** ()
  Update the light position in the shader.

- void **setTexture** (**Texture2D** &texture)
  Set the texture for the object rendering.

- void **shaderPreDrawingUpdate** (glm::mat4 worldMatrix, glm::vec4 color)
  Upload the shader data before rendering an object.

- void **drawMesh** (std::unique_ptr< **Mesh** > &mesh)
  Drawing a file preloaded mesh.

- void **drawVao** (**VertexArrayObject** &vao, int trianglesNumber)
  Drawing a dynamically generated object.

- void **rendering** ()

Swapping the OpenGL drawing and showed image buffers.

- void **setWindowTitle** (std::string title)
  Window title text setter.

- void **setWindowIcon** (SDL_Surface *windowIcon)
  Set the window icon.

- void **faceTestOn** ()
  ***Render** mode configurator - Face test on.*

- void **faceTestOff** ()
  ***Render** mode configurator - Face test off.*

- void **shaderCameraUpdate** ()
  ***Camera** position updating in the shader variable.*

- void **renderScrean** ()
  Composing the renderable scene from the renderableObjects and the renderableRoads storages.

- void **sceneInit** ()
  Initialize the scene.

- void **wireframeOn** ()
  ***Render** mode configurator - Wire frame object drawing on.*

- void **wireframeOff** ()
  ***Render** mode configurator - Wire frame object drawing off.*

- void **showHitSphere** (size_t renderID)
  ***Render** the hit spheres for each scene objects.*

- void **showVehicleHitSphere** (size_t vehicleRenderID)
  ***Render** the hit spheres for each scene vehicle objects.*

- void **showRoadHitSphere** (std::vector< glm::vec3 > &roadPoints, float size, glm::vec3 color)
  ***Render** the hit spheres for each scene road objects.*

- void **showAllRoadHitSpheres** ()
  ***Render** the hit spheres for each scene road objects.*

Detailed Description

The 3D Scene render implementation.

The 3D Scene render class Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Member Function Documentation

int Render::addObject (int *id*)

Add new renderable object from the preloaded object storage to the scene.

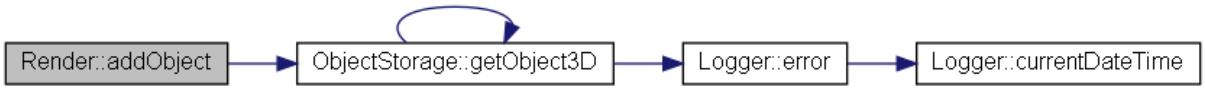
Parameters

id	New object storage ID.

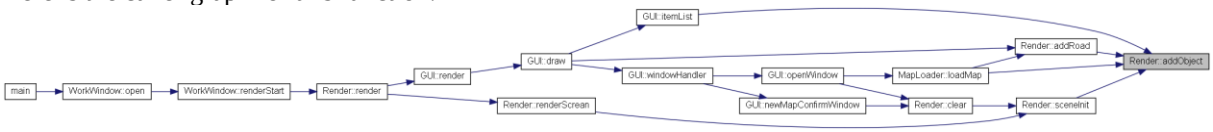
Returns

The new object render ID.

Here is the call graph for this function:



Here is the caller graph for this function:



int Render::addRoad ()

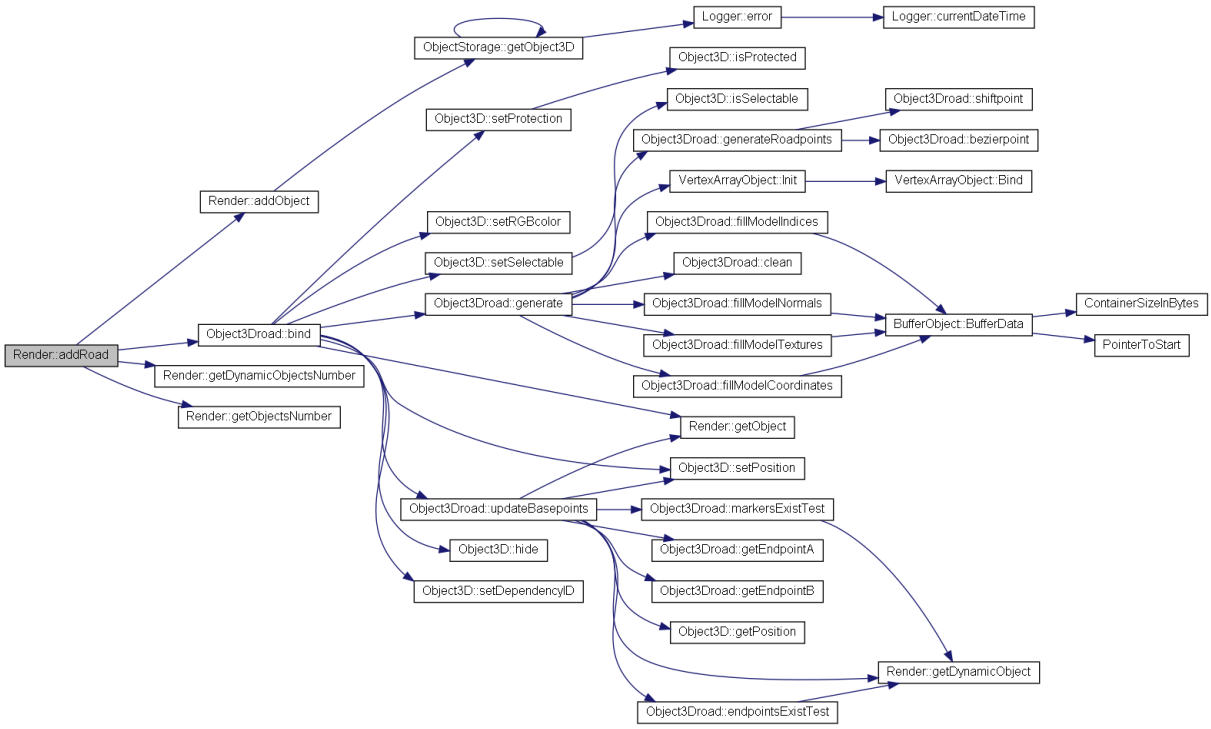
Add dynamically generated road to the scene.

/attention The road objects have 4 marker and 2 filler object for dependencies.

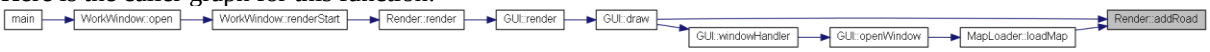
Returns

The new road dynamicRenderID.

Here is the call graph for this function:



Here is the caller graph for this function:



size_t Render::addVehicle (int *objectStorageID*, size_t *modelID*)

Add a new vehicle to the scene, from the objectStorage.

Parameters

objectStorageID	The addable vehicle object storage ID.
modelID	The addable vehicle object model ID.

Returns

The added vehicle renderID.

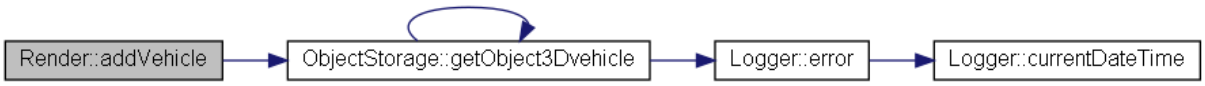
Parameters

objectStorageID	The addable vehicle object storage ID.

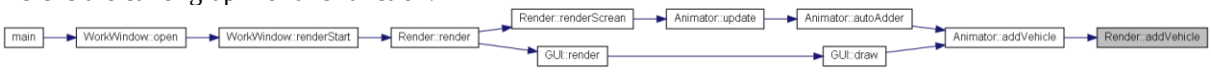
Returns

The added vehicle renderID.

Here is the call graph for this function:



Here is the caller graph for this function:



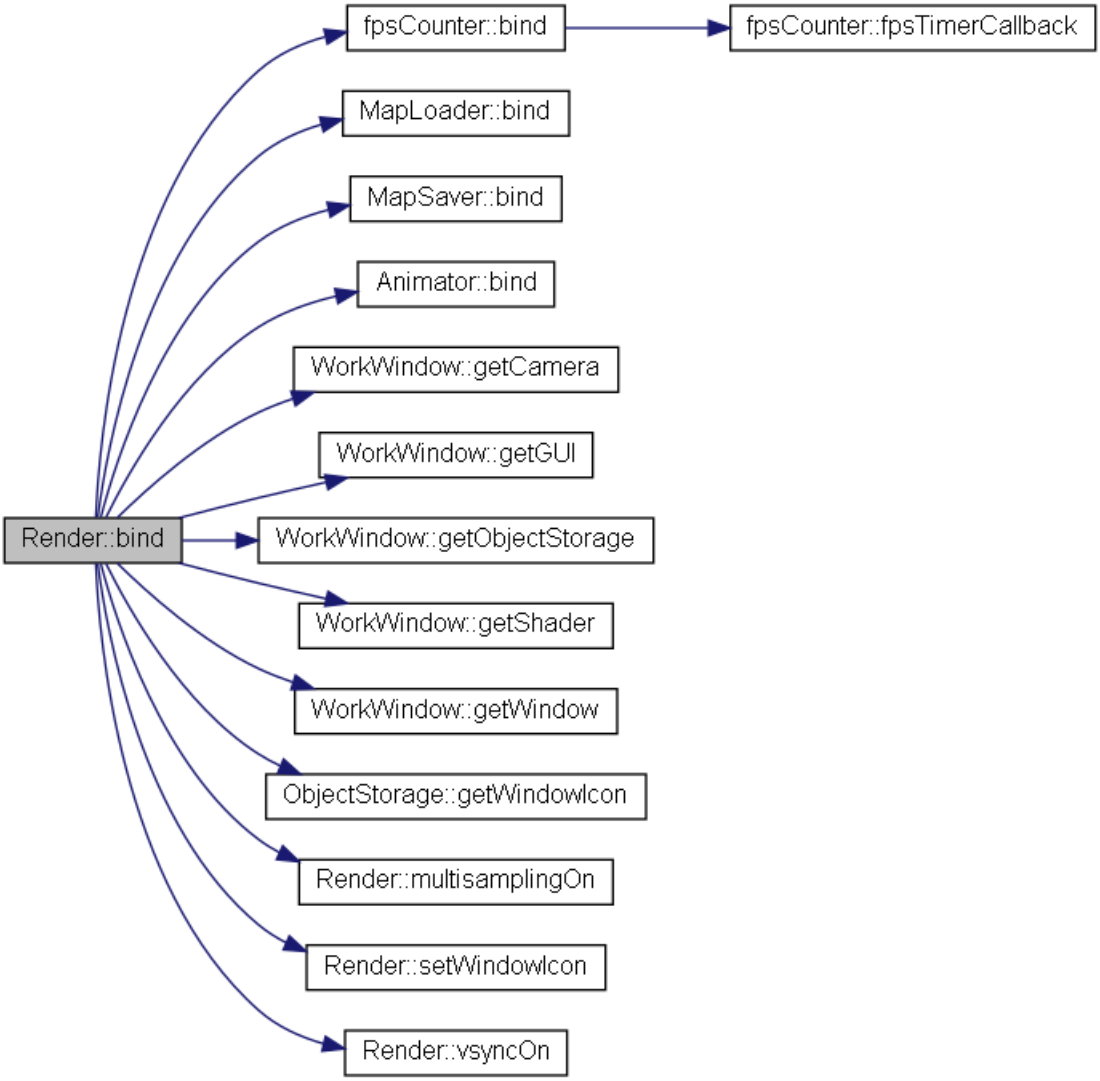
void Render::bind (WorkWindow * *currentWindow*)

Binding the render to the window.

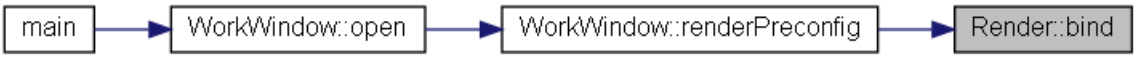
Parameters

window	The joinable window.

Here is the call graph for this function:



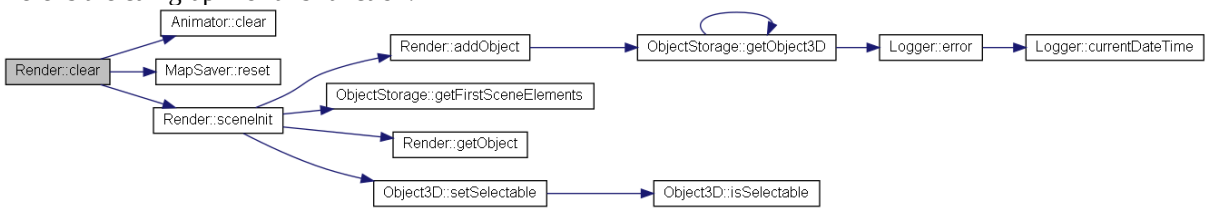
Here is the caller graph for this function:



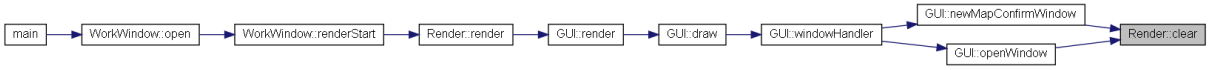
void Render::clear ()

Clean and renew the map editor.

Here is the call graph for this function:



Here is the caller graph for this function:



void Render::clearScrean ()

Clearing the scene.

void Render::clearVehicles ()

Delete all active vehicles.

Here is the caller graph for this function:

void Render::close ()

Exit program.

Here is the call graph for this function:

void Render::deleteObject (int *renderID*)

Delete a scene object by renderID.

/attention This cause pointer invalidation in the vector. Use the updateRenderIDs and rebindRoads for repairing pointers.

Parameters

renderID	renderID

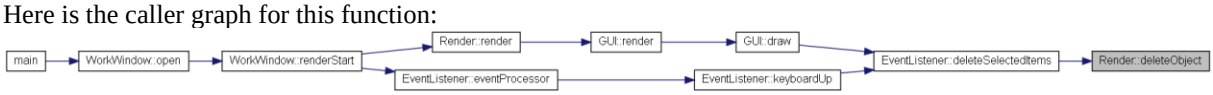
Attention

This cause pointer invalidation in the vector. Use the updateRenderIDs and rebindRoads for repairing pointers.

Parameters

renderID	renderableObjects for the

Here is the caller graph for this function:



void Render::deleteRoad (int *dynamicRenderID*)

Delete a scene road object by dynamicRenderID.

Delete a scene object by renderID.

/attention This cause pointer invalidation in the vector. Use the updateRenderIDs and rebindRoads for repairing pointers.

Parameters

dynamicRenderID	dynamicRenderID for the renderable roads.

/attention This cause pointer invalidation in the vector. Use the updateRenderIDs and rebindRoads for repairing pointers.

Parameters

renderID	renderID

Here is the caller graph for this function:

void Render::deleteVehicle (size_t *renderableVehicleID*)

Delete a vehicle by renderID.

Parameters

renderableVehicleID	The deletable vehicle renderID.

Here is the caller graph for this function:

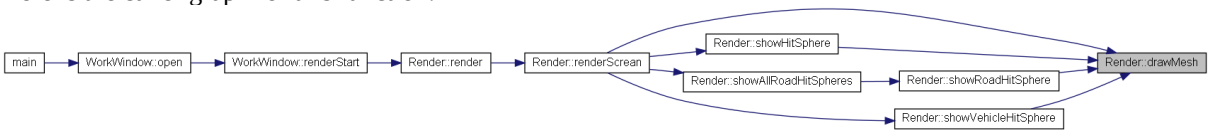
void Render::drawMesh (std::unique_ptr< Mesh > & *mesh*) `[private]`

Drawing a file preloaded mesh.

Parameters

mesh	The preloaded mesh for drawing.

Here is the caller graph for this function:



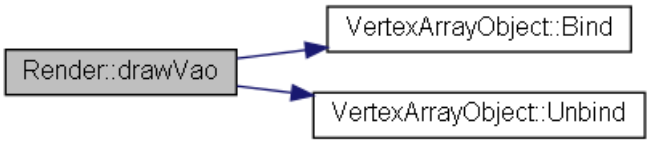
void Render::drawVao (VertexArrayObject & *vao*, int *trianglesNumber*) `[private]`

Drawing a dynamically generated object.

Parameters

VAO	The dynamically generated object for drawing.
trianglesNumber	The drawable object triangle number.

Here is the call graph for this function:



Here is the caller graph for this function:

void Render::faceTestOff () `[private]`

Render mode configurator - Face test off.

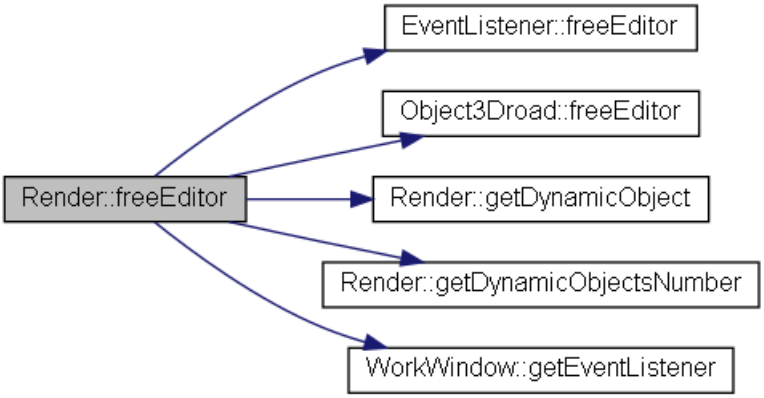
void Render::faceTestOn () `[private]`

Render mode configurator - Face test on.

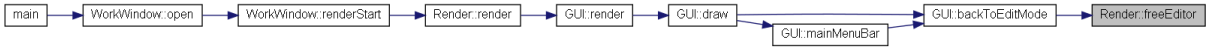
void Render::freeEditor ()

Free the editor functions.

Here is the call graph for this function:



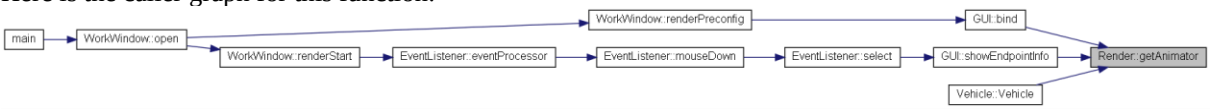
Here is the caller graph for this function:



Animator * Render::getAnimator ()

Getter for **Animator**.

Here is the caller graph for this function:



Object3Droad * Render::getDynamicObject (size_t *dynamicRenderID*)

Getter for the RenderableRoads.

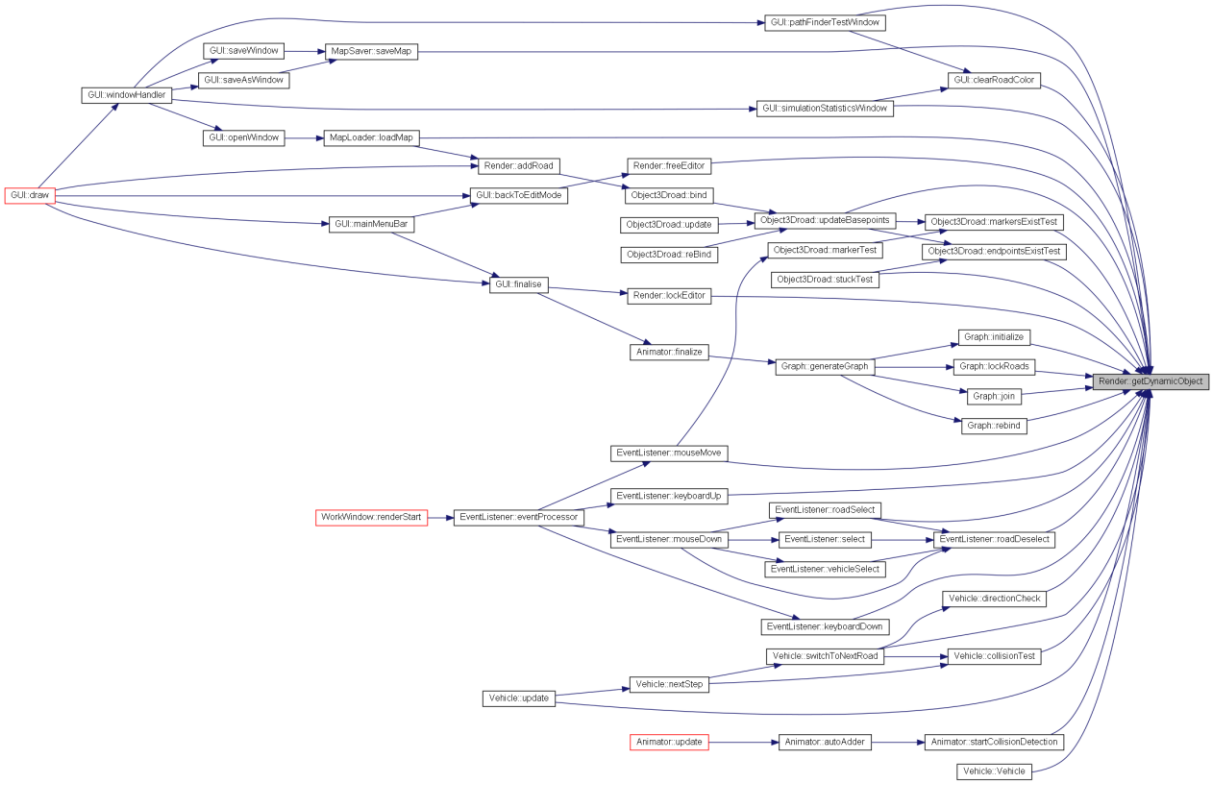
Parameters

renderID	The needed object dynamicRenderID.

Returns

The needed object.

Here is the caller graph for this function:



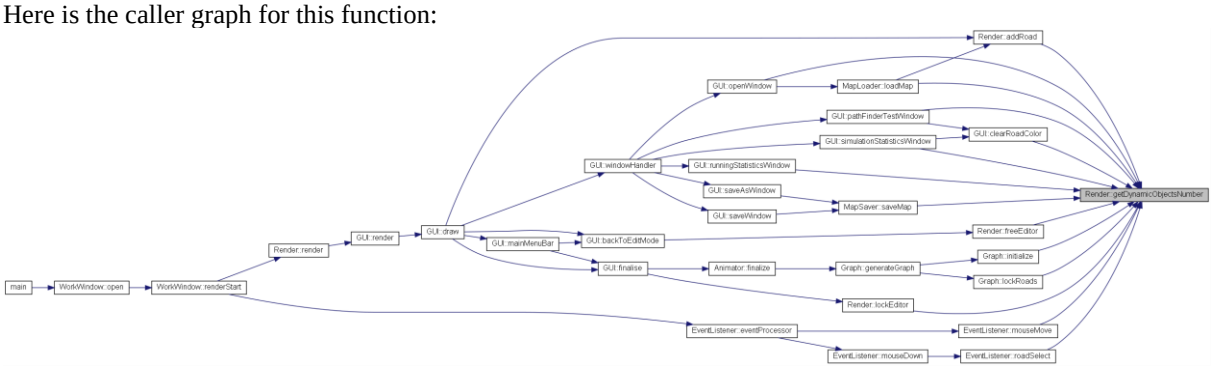
size_t Render::getDynamicObjectsNumber ()

Getter for the renderableRoads size.

Returns

The renderableRoads size.

Here is the caller graph for this function:



GUI * Render::getGui ()

Getter for the **GUI**.

Returns

The current **GUI**.

MapLoader * Render::getMapLoader ()

Getter for the map loader program.

Returns

pointer for the map loader instance.

Here is the caller graph for this function:

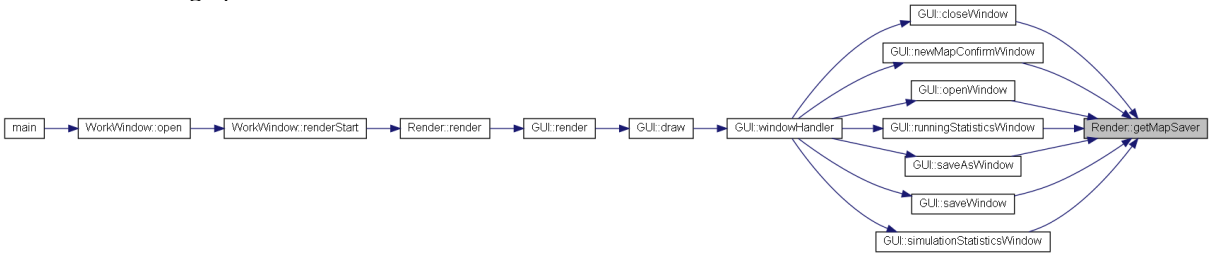
MapSaver * Render::getMapSaver ()

Getter for the map saver program.

Returns

Pointer for the map loader instance.

Here is the caller graph for this function:



Object3D * Render::getObject (size_t *id*)

Get scene preloaded object from the scene render storage.

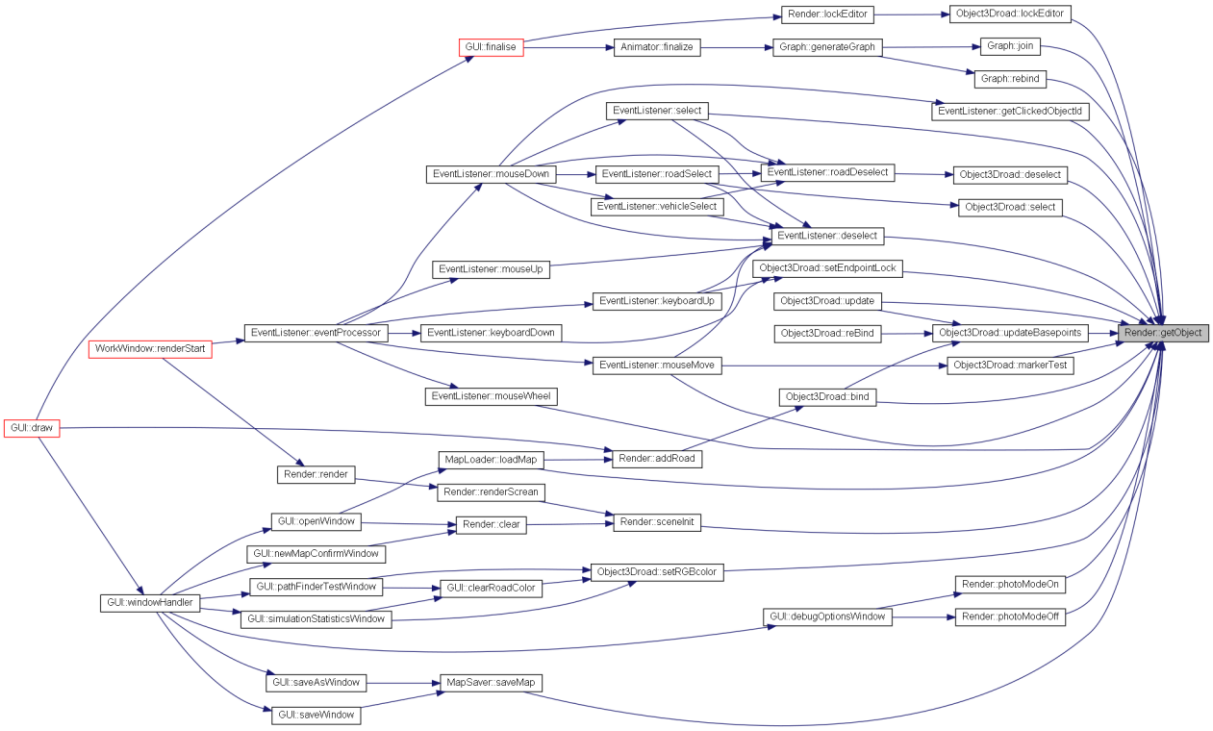
Parameters

id	renderID

Returns

The scene object.

Here is the caller graph for this function:



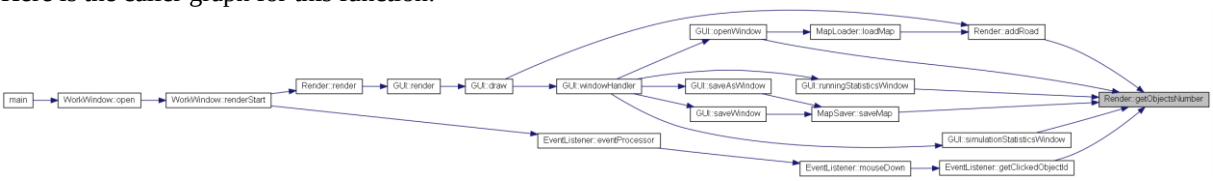
size_t Render::getObjectsNumber ()

Getter for the RenderableRoads size.

Returns

The renderableRoads size.

Here is the caller graph for this function:



Object3Dvehicle * Render::getVehicle (size_t *renderableVehicleID*)

Getter for the renderableVehicles;.

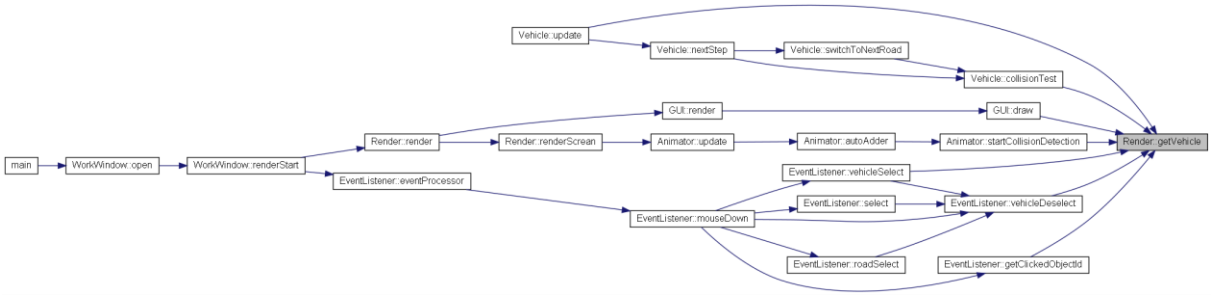
Parameters

renderableVehicleID	The needed vehicle's id.

Returns

The needed vehicle.

Here is the caller graph for this function:



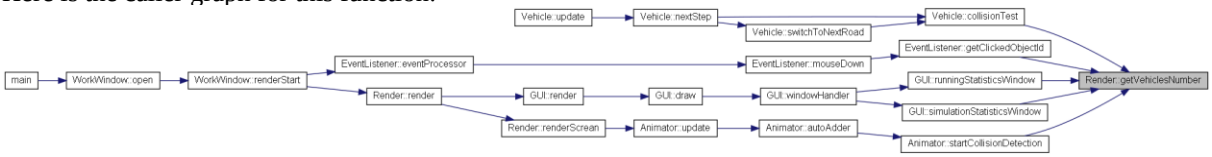
size_t Render::getVehiclesNumber ()

Getter for the renderableVehicles size.

Returns

The renderableVehicles size.

Here is the caller graph for this function:



WorkWindow * Render::getWorkingWindow ()

Getter for the working window pointer.

Returns

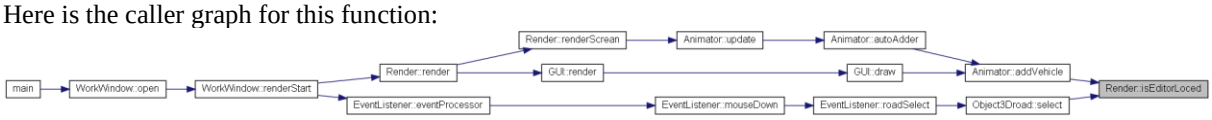
Pointer for the current working window.

Here is the caller graph for this function:

bool Render::isEditorLoced ()

Return true if the editor locked.

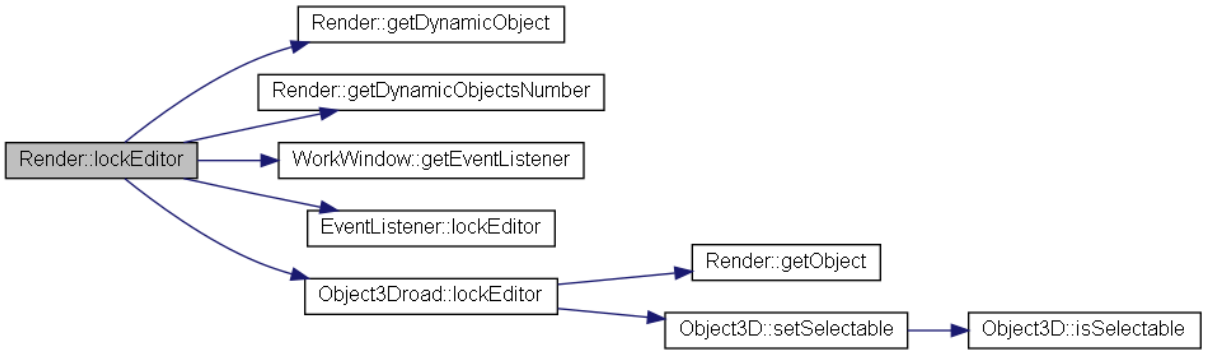
Here is the caller graph for this function:



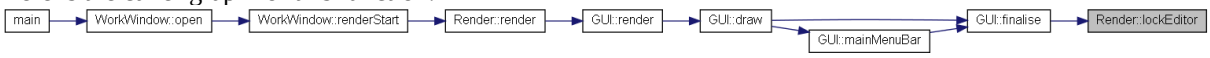
void Render::lockEditor ()

Lock the editor functions.

Here is the call graph for this function:



Here is the caller graph for this function:



void Render::multisamplingOff ()

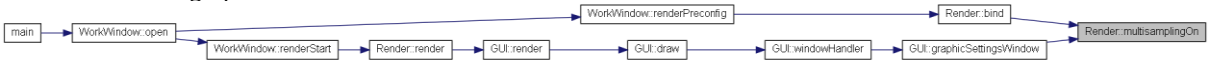
Render mode configurator - Multisampling off.

Here is the caller graph for this function:

void Render::multisamplingOn ()

Render mode configurator - Multisampling on.

Here is the caller graph for this function:

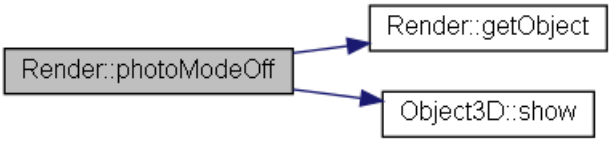


void Render::photoModeOff ()

Hide the desk for object photo mode.

Function off. (Debug function)

Here is the call graph for this function:



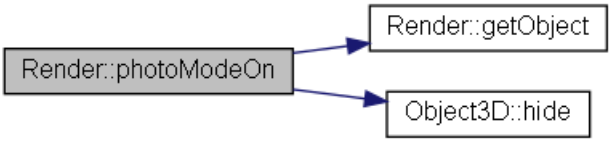
Here is the caller graph for this function:

void Render::photoModeOn ()

Hide the desk for object photo mode.

Function on.(Debug function)

Here is the call graph for this function:



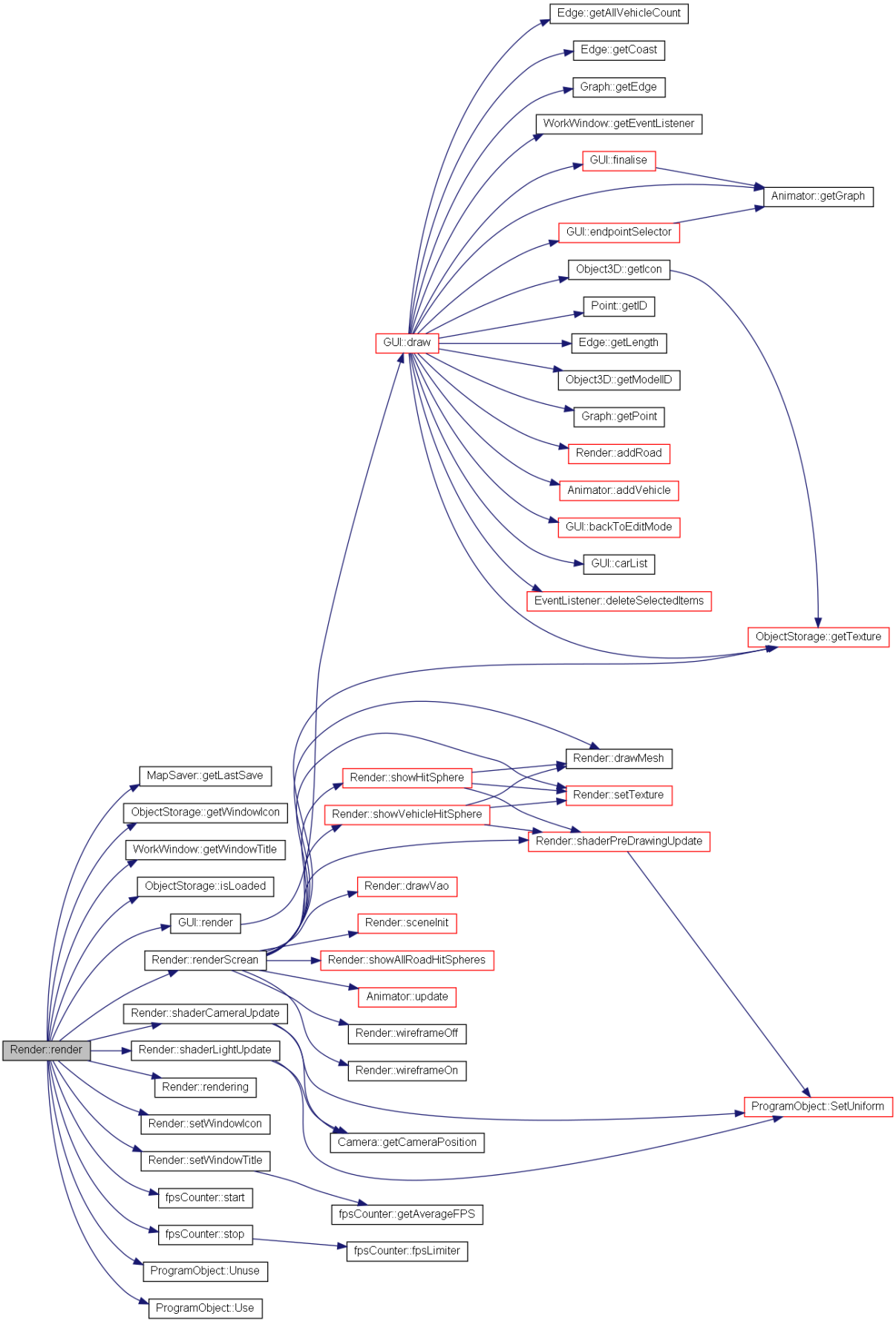
Here is the caller graph for this function:

void Render::render ()

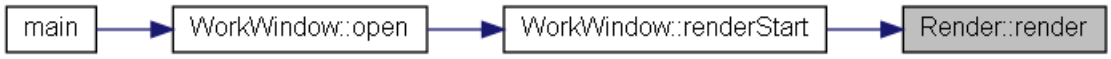
Render the next frame.

Use the shader, update the camera and the **GUI**. **Render** the objects and the current scene.

Here is the call graph for this function:



Here is the caller graph for this function:

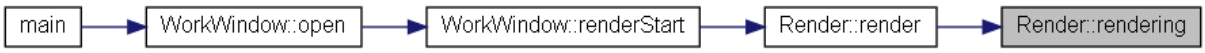


void Render::rendering () [private]

Swapping the OpenGL drawing and showed image buffers.

This is the true drawing.

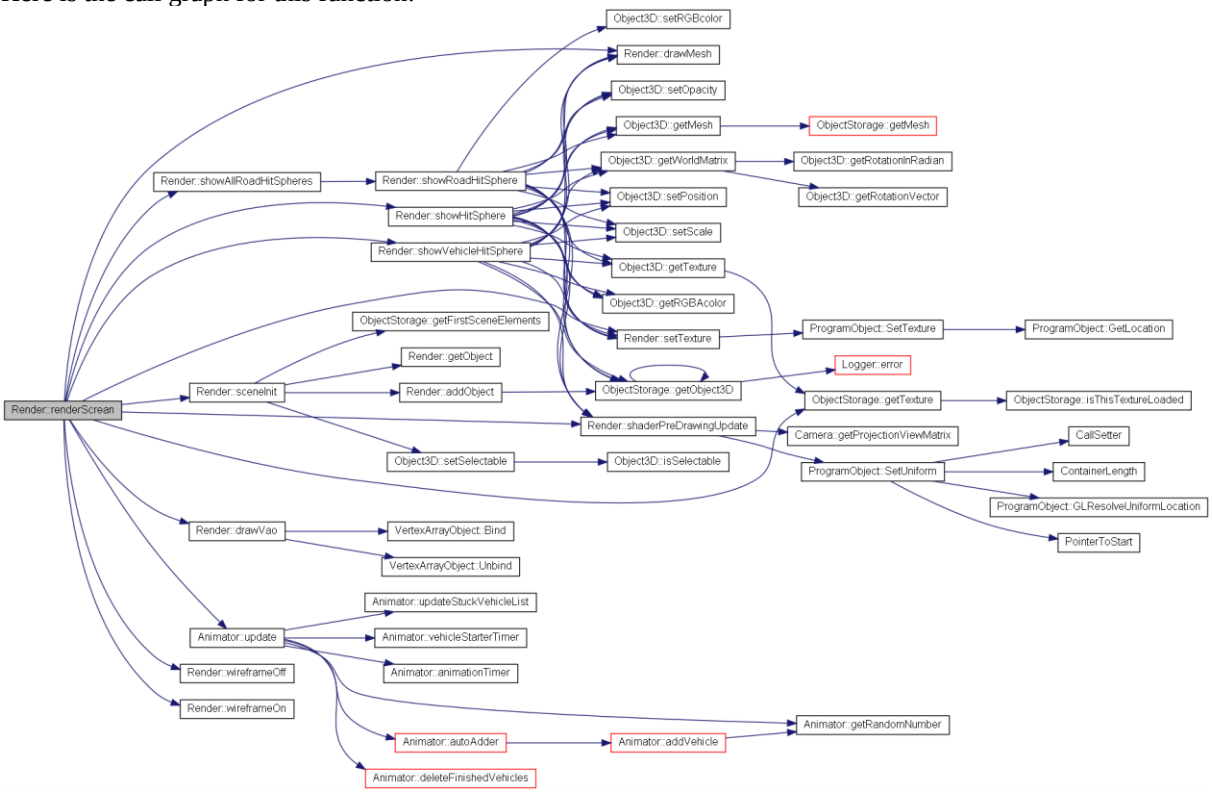
Here is the caller graph for this function:



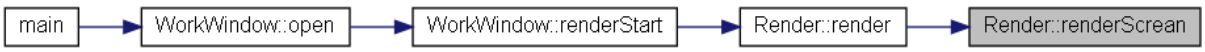
void Render::renderScrean () [private]

Composing the renderable scene from the renderableObjects and the renderableRoads storages.

Here is the call graph for this function:



Here is the caller graph for this function:

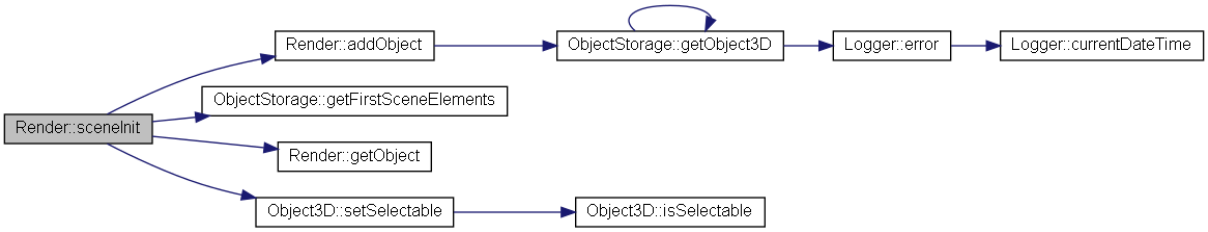


void Render::sceneInit () [private]

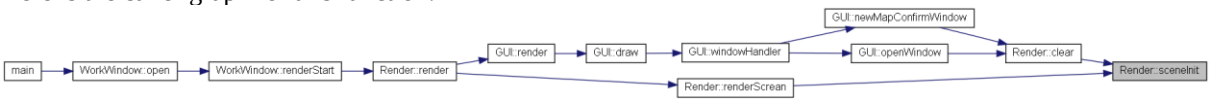
Initialize the scene.

Render the first objects.

Here is the call graph for this function:



Here is the caller graph for this function:



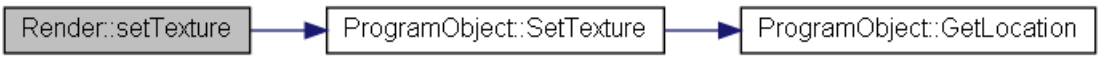
void Render::setTexture (Texture2D & *texture*) `[private]`

Set the texture for the object rendering.

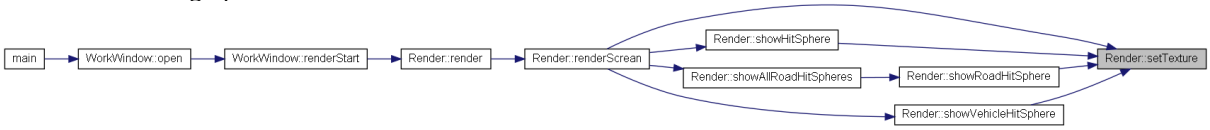
Parameters

texture	The usable texture.

Here is the call graph for this function:



Here is the caller graph for this function:



void Render::setToBorderless ()

Setter for borderless screen mode.

Here is the caller graph for this function:

void Render::setToFullScreen ()

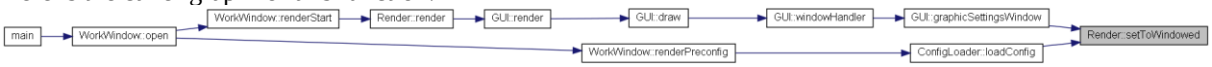
Setter to full screen window mode.

Here is the caller graph for this function:

void Render::setToWindowed ()

Setter for windowed screen mode.

Here is the caller graph for this function:



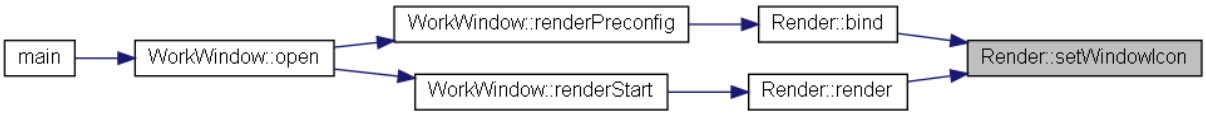
void Render::setWindowIcon (SDL_Surface * *windowIcon*) `[private]`

Set the window icon.

Parameters

windowIcon	The new window icon.

Here is the caller graph for this function:



void Render::setWindowTitle (std::string *title*) `[private]`

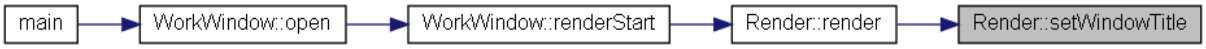
Window title text setter.

Parameters

title	The new window title.

Here is the call graph for this function:

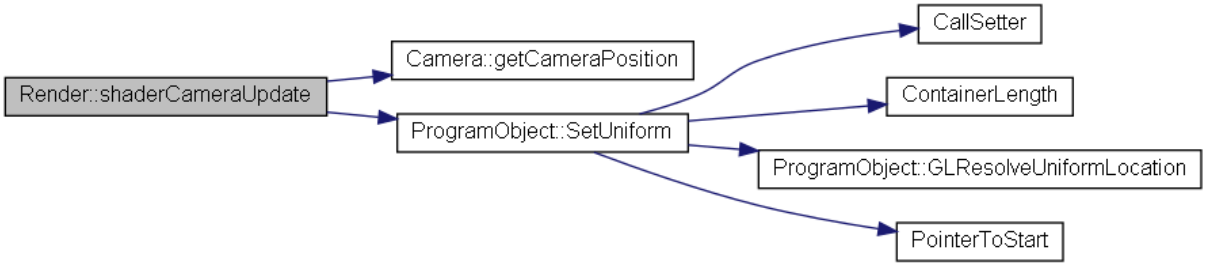
Here is the caller graph for this function:



void Render::shaderCameraUpdate () `[private]`

Camera position updating in the shader variable.

Here is the call graph for this function:

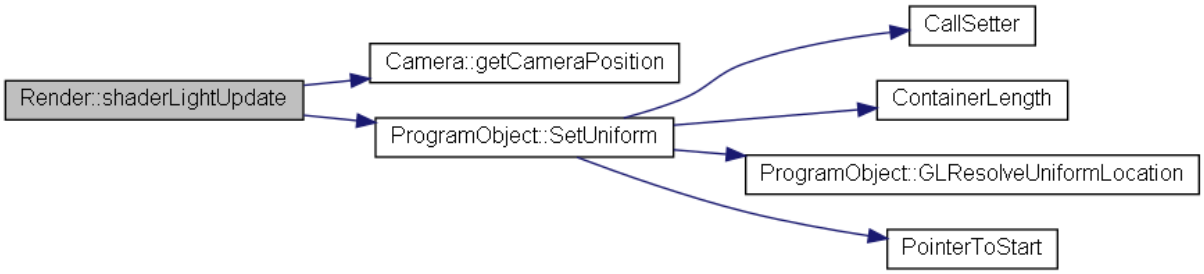


Here is the caller graph for this function:

void Render::shaderLightUpdate () `[private]`

Update the light position in the shader.

Here is the call graph for this function:



Here is the caller graph for this function:

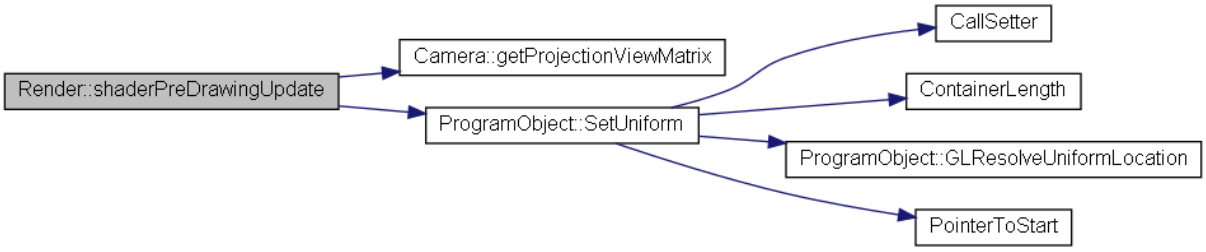
void Render::shaderPreDrawingUpdate (glm::mat4 *worldMatrix*, glm::vec4 *color*) `[private]`

Upload the shader data before rendering an object.

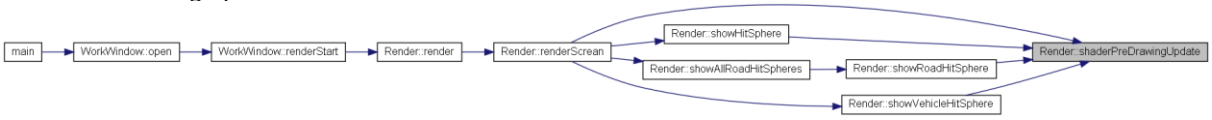
Parameters

worldMatrix	The world matrix for the shader.
color	The rendering RGBA color.

Here is the call graph for this function:



Here is the caller graph for this function:

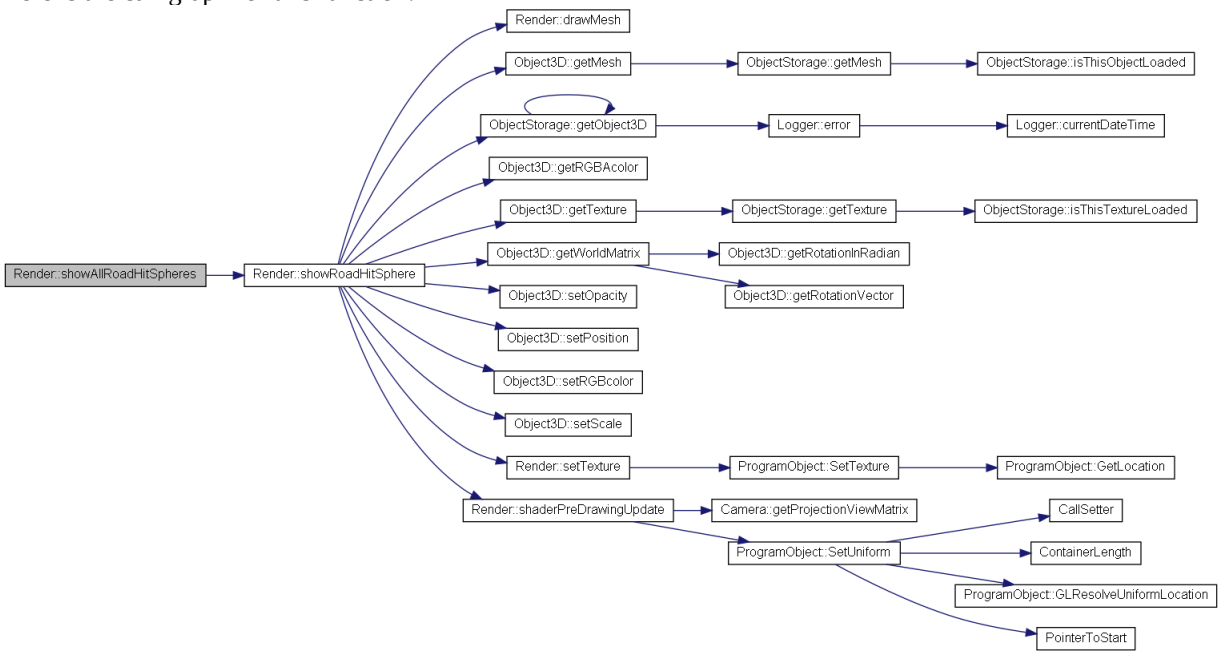


void Render::showAllRoadHitSpheres () `[private]`

Render the hit spheres for each scene road objects.

(Debug function)

Here is the call graph for this function:



Here is the caller graph for this function:

void Render::showHitSphere (size_t *renderID*) `[private]`

Render the hit spheres for each scene objects.

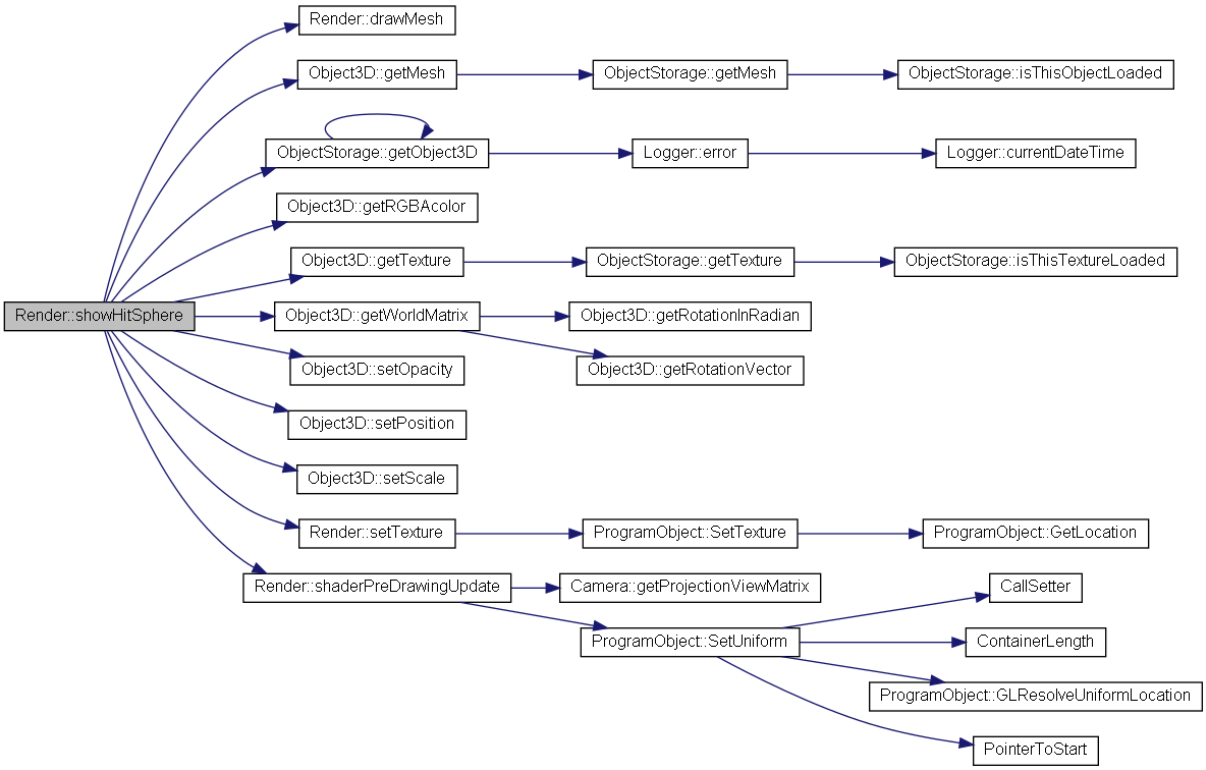
(Debug function)

Parameters

renderID	The actual object render ID.

(Debug function)

Here is the call graph for this function:



Here is the caller graph for this function:

void Render::showRoadHitSphere (std::vector< glm::vec3 > & *roadPoints*, float *size*, glm::vec3 *color*) `[private]`

Render the hit spheres for each scene road objects.

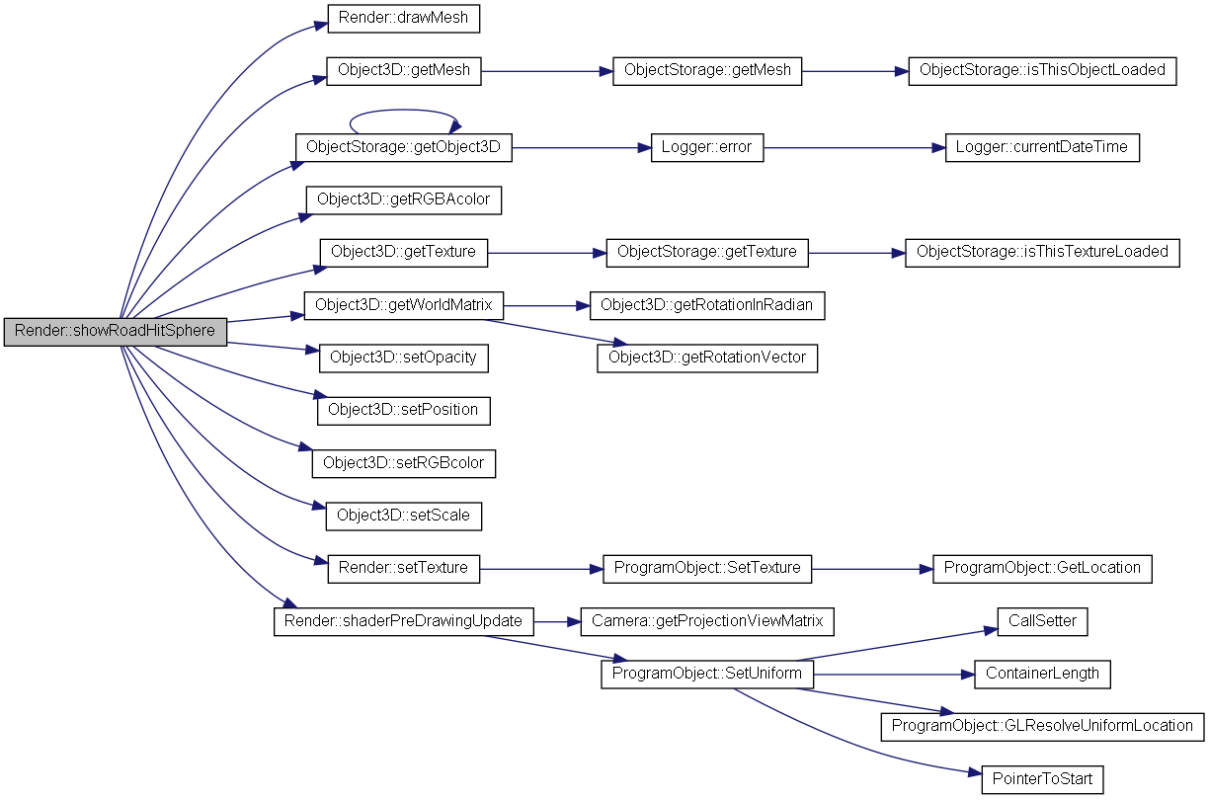
(Debug function)

Parameters

roadPoints	The drawable points
size	the marker size
vec3	the marker colors

(Debug function)

Here is the call graph for this function:



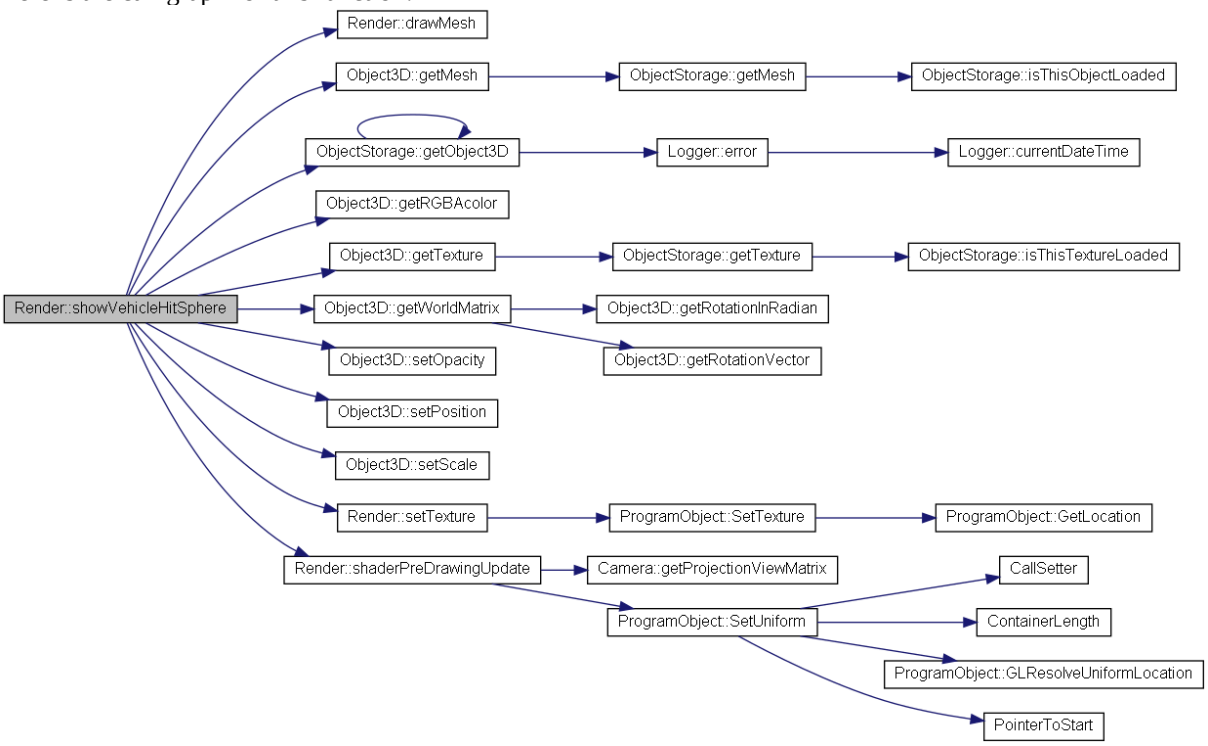
Here is the caller graph for this function:

void Render::showVehicleHitSphere (size_t *vehicleRenderID*) `[private]`

Render the hit spheres for each scene vehicle objects.

(Debug function)

Here is the call graph for this function:



Here is the caller graph for this function:

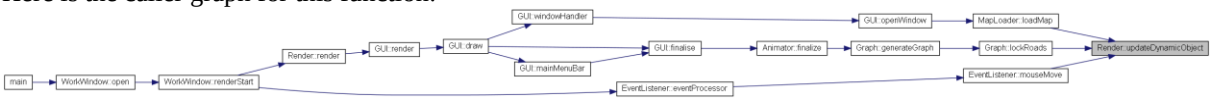
void Render::updateDynamicObject (size_t *dynamicRenderID*)

Update a dynamically generated object status by dynamicRenderID.

Parameters

The	updatable dynamicRenderID

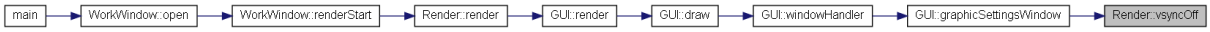
Here is the caller graph for this function:



void Render::vsyncOff ()

Render mode configurator - Vsync off.

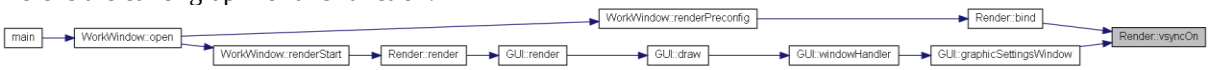
Here is the caller graph for this function:



void Render::vsyncOn ()

Render mode configurator - Vsync on.

Here is the caller graph for this function:



void Render::wireframeOff () `[private]`

Render mode configurator - Wire frame object drawing off.

Here is the caller graph for this function:

void Render::wireframeOn () `[private]`

Render mode configurator - Wire frame object drawing on.

Here is the caller graph for this function:

Field Documentation

glm::vec4 Render::ambientLight = glm::vec4(0.1f, 0.1f, 0.1f, 1.0f)

Ambient light.

glm::vec4 Render::ambientMaterial = glm::vec4(1.0f, 1.0f, 1.0f, 0.0f)

Ambient material properties.

Animator Render::animator `[private]`

The render's animator instance.

Camera* Render::camera `[private]`

The access pointer for the current camera object.

glm::vec4 Render::diffuseLight = glm::vec4(0.75f, 0.75f, 0.75f, 1.0f)

Diffuse light.

glm::vec4 Render::diffuseMaterial = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f)

Diffuse material.

bool Render::editorLock = false `[private]`

Flag for editor locking.

bool Render::firstScreen = true `[private]`

Flag variable for the first scene configurations.
/attention Default value is TRUE.

GUI* Render::gui `[private]`

The access pointer for the current **GUI** object.

bool Render::hitSphare = false

Active the hit sphere drawing.
(DebugFunction)

bool Render::lightCameraTracking = true

If it is true the light position is same with the **Camera** position.

glm::vec3 Render::lightPosition = glm::vec3(55, 55, 55)

The current light position, if the light position is fixed.

MapLoader Render::mapLoader `[private]`

The map loader class.

MapSaver Render::mapSaver `[private]`

The map saver class.

ObjectStorage* Render::objectStorage `[private]`

The access pointer for the current object storage (This store the preloaded meshes and textures, allways in one instance.).

bool Render::objectsWireframe = false

Active the wire frame function for the objects.

(DebugFunction)

std::vector<Object3D> Render::renderableObjects `[private]`

List of the renderable scene objects.

(Preloaded meshes)

std::vector<Object3Droad*> Render::renderableRoads `[private]`

List of the renderable scene roads.

(Dynamically generated shapes)

std::vector<Object3Dvehicle> Render::renderableVehicles `[private]`

The lost of renderable vehicle.

bool Render::roadHhitSphare = false

Active the road hit sphere drawing.

(DebugFunction)

bool Render::roadWireframe = false

Active the wire frame function for the roads.

(DebugFunction)

ProgramObject* Render::shader `[private]`

The access pointer for the current shader program.

bool Render::showRoadHitSphereMiddle = true

Show the road hit spheres.

(DebugFunction)

bool Render::showRoadSide01 = false

Show road side 1 markers.
(DebugFunction)

bool Render::showRoadSide02 = false

Show road side 2 markers.
(DebugFunction)

bool Render::showRoadTrack01 = false

Show road track 1 markers.
(DebugFunction)

bool Render::showRoadTrack02 = false

Show road track 2 markers.
(DebugFunction)

glm::vec4 Render::specularLight = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f)

Specular light.

glm::vec4 Render::specularMaterial = glm::vec4(1.0f, 1.0f, 1.0f, 0.0f)

Specular material.

float Render::specularPower = 32.0f

Specular light power.

bool Render::vehicleHitSphare = false

Active the vehicle hit sphere drawing.
(DebugFunction)

const char * Render::version = "v1.3" `[static]`

The current version number.

SDL_Window* Render::window `[private]`

The access pointer for the current window object.

WorkWindow* Render::workingWindow `[private]`

The access pointer for the current working window object.

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

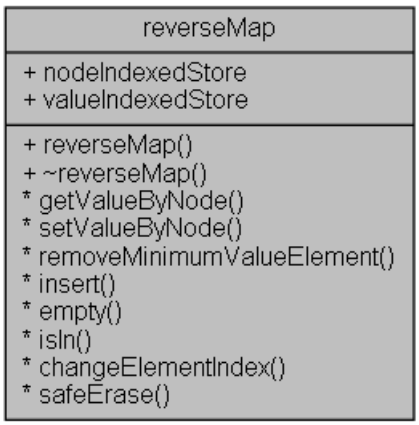
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**Render.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**Render.cpp**

reverseMap Class Reference

The reverse map implementation for the **Dijkstra** algorithm.

```
#include <reverseMap.h>
```

Collaboration diagram for reverseMap:



Public Member Functions

- **reverseMap** ()
  Constructor for the reverse map.

- **~reverseMap** ()
  Destructor for the reverse map.

Data Fields

- std::map< size_t, int > **nodeIndexedStore**
  THe node index storage.

- std::multimap< int, size_t > **valueIndexedStore**
  The value index storage.

Traffic Simulation

- int **getValueByNode** (size_t node)
  Getter for a value by node.

- void **setValueByNode** (size_t node, int newValue)
  Set a value by node.

- std::pair< size_t, int > **removeMinimumValueElement** ()
  Remove the minimum value element and return it.

- void **insert** (size_t node, int value)
  Insert an instance to the reverse map.

- bool **empty** ()
  The map is empty?

- bool **isIn** (size_t node)
  The node is in the map?

- void **changeElementIndex** (const size_t oldIndex, const size_t newIndex, const int element)
  Change the index of an element.

- void **safeErase** (size_t node, int value)
  Erase safely an element.

Detailed Description

The reverse map implementation for the **Dijkstra** algorithm.

Constructor & Destructor Documentation

reverseMap::reverseMap () `[inline]`

Constructor for the reverse map.

reverseMap::~reverseMap () `[inline]`

Destructor for the reverse map.

Member Function Documentation

void reverseMap::changeElementIndex (const size_t *oldIndex*, const size_t *newIndex*, const int *element*) `[private]`

Change the index of an element.

Parameters

oldIndex	The old index.
newIndex	The new index.
element	The element.

Here is the caller graph for this function:

bool reverseMap::empty ()

The map is empty?

Returns

The map is empty?

Here is the caller graph for this function:

int reverseMap::getValueByNode (size_t *node*)

Getter for a value by node.

Parameters

node	The node.

Returns

The value.

Here is the caller graph for this function:

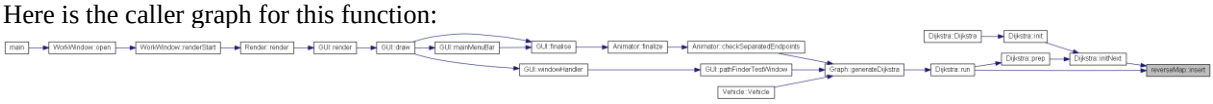
void reverseMap::insert (size_t *node*, int *value*)

Insert an instance to the reverse map.

Parameters

node	The node.
value	The value.

Here is the caller graph for this function:



bool reverseMap::isIn (size_t *node*)

The node is in the map?

Parameters

node	The node.

Returns

Is in the map?

Here is the caller graph for this function:

std::pair< size_t, int > reverseMap::removeMinimumValueElement ()

Remove the minimum value element and return it.

Returns

std::pair<node,value>

Here is the call graph for this function:

Here is the caller graph for this function:

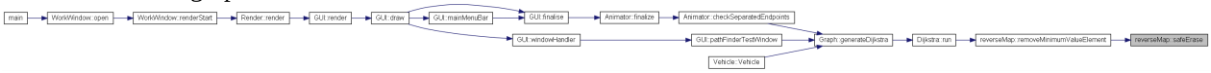
void reverseMap::safeErase (size_t *node*, int *value*) `[private]`

Erase safely an element.

Parameters

node	The node.
value	The value.

Here is the caller graph for this function:



void reverseMap::setValueByNode (size_t *node*, int *newValue*)

Set a value by node.

Parameters

node	The node.
newValue	The new value.

Here is the call graph for this function:

Here is the caller graph for this function:

Field Documentation

std::map<size_t, int> reverseMap::nodeIndexedStore

THe node index storage.

std::multimap<int, size_t> reverseMap::valueIndexedStore

The value index storage.

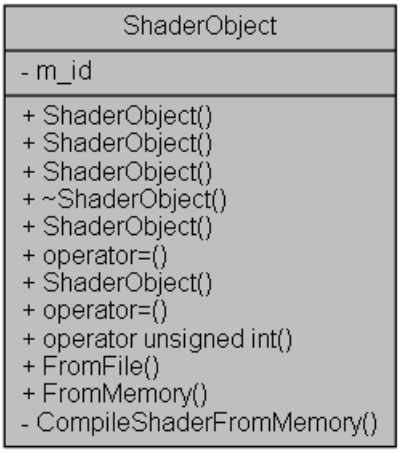
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**reverseMap.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**reverseMap.cpp**

ShaderObject Class Reference

`#include <ShaderObject.h>`

Collaboration diagram for ShaderObject:



Public Member Functions

- **ShaderObject** (GLenum pType)
- **ShaderObject** (GLenum pType, const std::string &)
- **ShaderObject** (const **TypeSourcePair** &pInfo)
- **~ShaderObject** ()
- **ShaderObject** (const **ShaderObject** &)=delete
- **ShaderObject** & **operator=** (const **ShaderObject** &)=delete
- **ShaderObject** (**ShaderObject** &&)
- **ShaderObject** & **operator=** (**ShaderObject** &&)
- **operator unsigned int** () const
- bool **FromFile** (GLenum _shaderType, const char *_filename)
- bool **FromMemory** (GLenum _shaderType, const std::string &_source)

Private Member Functions

- GLuint **CompileShaderFromMemory** (const GLuint _shaderObject, const std::string &_source)

Private Attributes

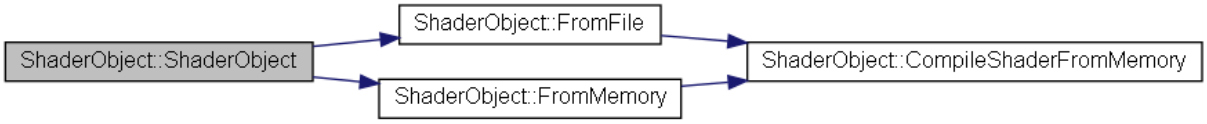
- GLuint **m_id**

Constructor & Destructor Documentation

ShaderObject::ShaderObject (GLenum *pType*)

ShaderObject::ShaderObject (GLenum *pType*, const std::string & *pFilenameOrSource*)

Here is the call graph for this function:



ShaderObject::ShaderObject (const TypeSourcePair & *pInfo*) `[inline]`

ShaderObject::~ShaderObject ()

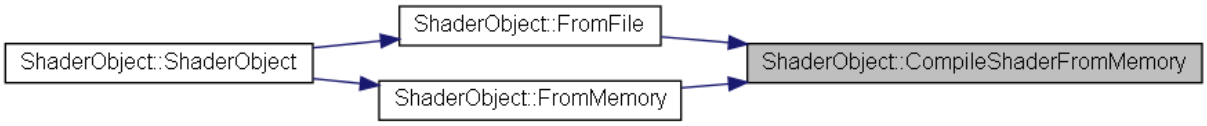
ShaderObject::ShaderObject (const ShaderObject &) `[delete]`

ShaderObject::ShaderObject (ShaderObject && *rhs*)

Member Function Documentation

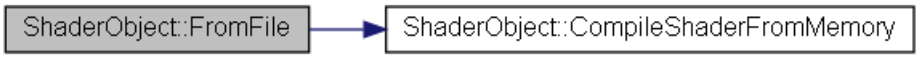
GLuint ShaderObject::CompileShaderFromMemory (const GLuint *_shaderObject*, const std::string & *_source*) `[private]`

Here is the caller graph for this function:



bool ShaderObject::FromFile (GLenum *_shaderType*, const char * *_filename*)

Here is the call graph for this function:



Here is the caller graph for this function:

bool ShaderObject::FromMemory (GLenum *_shaderType*, const std::string & *_source*)

Here is the call graph for this function:

Here is the caller graph for this function:

ShaderObject::operator unsigned int () const `[inline]`

ShaderObject & ShaderObject::operator= (const ShaderObject &) `[delete]`

ShaderObject & ShaderObject::operator= (ShaderObject && *rhs*)

Field Documentation

GLuint ShaderObject::m_id `[private]`

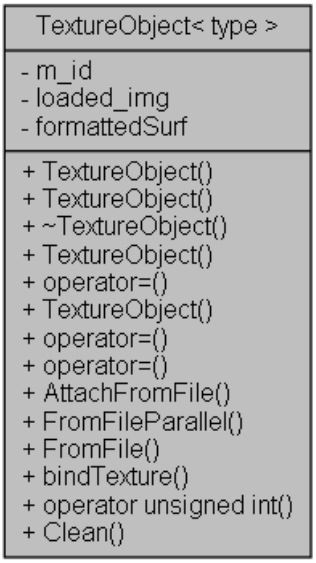
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ShaderObject.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**ShaderObject.cpp**

TextureObject< type > Class Template Reference

`#include <TextureObject.h>`

Collaboration diagram for TextureObject< type >:



Public Member Functions

- **TextureObject** ()
- **TextureObject** (const std::string &)
- **~TextureObject** ()
- **TextureObject** (const **TextureObject** &)=delete
- **TextureObject** & **operator=** (const **TextureObject** &)=delete
- **TextureObject** (**TextureObject** &&rhs)
- **TextureObject** & **operator=** (**TextureObject** &&rhs)
- **TextureObject** & **operator=** (const std::string &s)
- void **AttachFromFile** (const std::string &, bool generateMipMap=true, GLuint role=static_cast< GLuint >(type))
- void **FromFileParallel** (const std::string &)
- void **FromFile** (const std::string &)
- void **bindTexture** ()
- **operator unsigned int** () const
- void **Clean** ()

Private Attributes

- GLuint **m_id** {}
- SDL_Surface * **loaded_img**
- SDL_Surface * **formattedSurf**

Constructor & Destructor Documentation

template<TextureType type> TextureObject< type >::TextureObject `[inline]`

template<TextureType type> TextureObject< type >::TextureObject (const std::string & *s*) `[inline]`

template<TextureType type> TextureObject< type >::~TextureObject `[inline]`

template<TextureType type = TextureType::Texture2D> TextureObject< type >::TextureObject (const TextureObject< type > &) `[delete]`

template<TextureType type> TextureObject< type >::TextureObject (TextureObject< type > && *rhs*) `[inline]`

Member Function Documentation

template<TextureType type> void TextureObject< type >::AttachFromFile (const std::string & *filename*, bool *generateMipMap* = `true`, GLuint *role* = `static_cast<GLuint>(type)`) `[inline]`

Here is the call graph for this function:

template<TextureType type> void TextureObject< type >::bindTexture `[inline]`

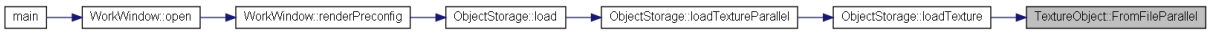
template<TextureType type> void TextureObject< type >::Clean `[inline]`

template<TextureType type> void TextureObject< type >::FromFile (const std::string & *s*) `[inline]`

template<TextureType type> void TextureObject< type >::FromFileParallel (const std::string & *filename*) `[inline]`

Here is the call graph for this function:

Here is the caller graph for this function:



template<TextureType type = TextureType::Texture2D> TextureObject< type >::operator unsigned int () const `[inline]`

template<TextureType type> TextureObject< type > & TextureObject< type >::operator= (const std::string & *s*) `[inline]`

template<TextureType type = TextureType::Texture2D> TextureObject & TextureObject< type >::operator= (const TextureObject< type > &) `[delete]`

template<TextureType type> TextureObject< type > & TextureObject< type >::operator= (TextureObject< type > && *rhs*) `[inline]`

Field Documentation

template<TextureType type = TextureType::Texture2D> SDL_Surface* TextureObject< type >::formattedSurf `[private]`

template<TextureType type = TextureType::Texture2D> SDL_Surface* TextureObject< type >::loaded_img `[private]`

template<TextureType type = TextureType::Texture2D> GLuint TextureObject< type >::m_id {} `[private]`

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

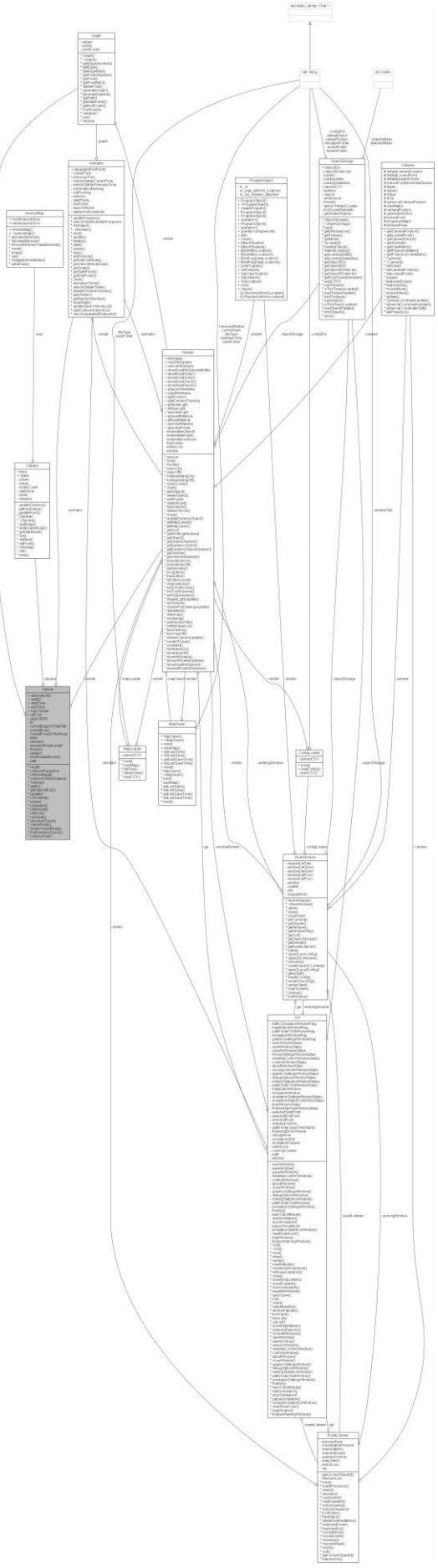
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**TextureObject.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**TextureObject.inl**

Vehicle Class Reference

Vehicle model representation.

```
#include <Vehicle.h>
```

Collaboration diagram for Vehicle:



Data Fields

- size_t **destinationID** = 0
  the destination point id.

- size_t **startID** = 0
  The start point ID.

- Uint32 **startTime** = 0
  The start time.

- Uint32 **endTime** = 0
  The end time.

- size_t **hopCounter** = 0
  The hop counter.

- int **allCost** = 0
  The cost calculator.

Private Attributes

- size_t **object3DiD** = 0
  The render ID.

- size_t **ID** = 0
  The vehicle moddel ID.

- size_t **currentEdgeOnThePath** = 0
  The current road number on the path.

- size_t **currentRoad** = 0
  The current road ID.

- size_t **currentPointOnTheRoad** = 0
  The current point on the current road.

- char **track** = '1'
  The track marker.

- char **direction** = 'a'
  the traveling direction marker.

- const size_t **standardRoadLenght** = 100
  The standard road length constant.

- bool **finished** = false
  The finished flag.

- bool **deleted** = false
  The deleted flag.

- bool **nextRoadIsBlocked** = false
  The blocked flag.

- **Graph** * **graph**
  The pointer to the current graph.

- **Render** * **render**
  The pointer to the current render.

- **Dijkstra** * **dijkstra**
  *The pointer to the vehicle's own **Dijkstra** object.*

- **Animator** * **animator**
  The pointer to the current animator object.

- std::vector< size_t > **path**
  The current calculated path.

Traffic Simulation

- static bool **repath** = true
  Repath function static flag.

- static bool **collisionPrevention** = true
  Collision detection static flag.

- static int **vehicleWeight** = 100
  The vehicle weight in the graph.

- static int **collisionCheckDistance** = 15
  The collision detection static distance.

- **Vehicle** (**Graph** ***graph**, **Render** ***render**, size_t **startID**, size_t **destinationID**, size_t **object3DiD**, size_t **ID**)
  *Constructor for the **Vehicle**.*

- size_t **getID** ()
  *Getter for the **Vehicle** ID.*

- size_t **getObject3DiD** ()
  Getter for the render ID.

- void **update** ()
  *Update the **Vehicle** status.*

- bool **isFinished** ()
  Getter for the finish flag.

- void **erase** ()
  Set the vehicle deleted.

- bool **isDeleted** ()
  Getter for the deleted flag.

- bool **isBlocked** ()
  Getter for the blocked flag.

- void **unblock** ()
  Unblock the vehicle.

- void **nextStep** ()
  Step to the next point.

- char **directionCheck** ()
  Check the vehicle direction and track.

- void **checkFinish** ()
  Check the path finishing.

- void **switchToNextRoad** ()
  Check the next road.

- void **firstDirectionCheck** ()
  Calculate the first start direction and track.

- bool **collisionTest** (int checkDistance, size_t localCurrentRoad, size_t localCurrentPointOnTheRoad, char **direction**, char **track**)
  Check the road collision.

Detailed Description

Vehicle model representation.

Vehicle model representation. Contact: `sandorbalazs9402@gmail.com`.

Author

Sándor Balázs - AZA6NL

Date

2021.11.08.

Constructor & Destructor Documentation

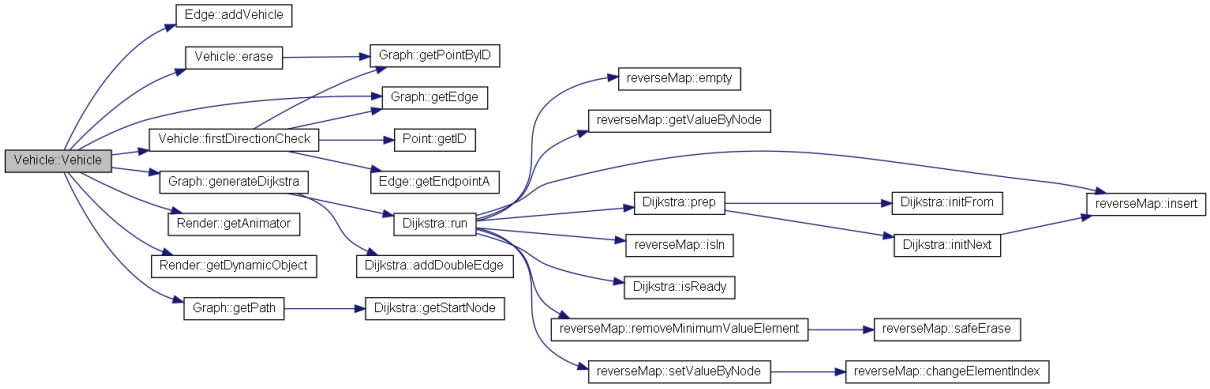
Vehicle::Vehicle (Graph * *graph*, Render * *render*, size_t *startID*, size_t *destinationID*, size_t *object3DiD*, size_t *ID*)

Constructor for the **Vehicle**.

Parameters

graph	The current graph.
render	The pointer for the render.
startID	The startpoint ID.
destinationID	The destination endpoint.
object3DiD	The 3D render ID.
ID	The current ID.

Here is the call graph for this function:



Member Function Documentation

void Vehicle::checkFinish () `[private]`

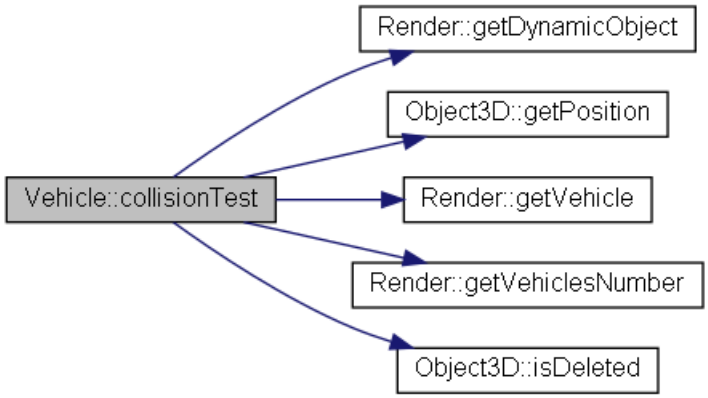
Check the path finishing.

Here is the caller graph for this function:

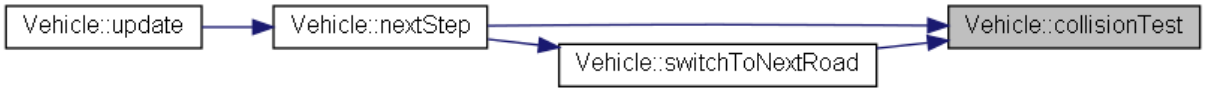
bool Vehicle::collisionTest (int *checkDistance*, size_t *localCurrentRoad*, size_t *localCurrentPointOnTheRoad*, char *direction*, char *track*) `[private]`

Check the road collision.

Here is the call graph for this function:



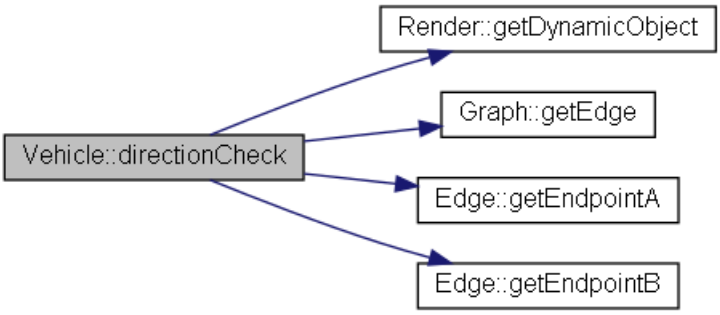
Here is the caller graph for this function:



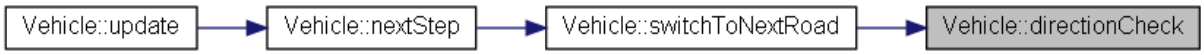
char Vehicle::directionCheck () [private]

Check the vehicle direction and track.

Here is the call graph for this function:



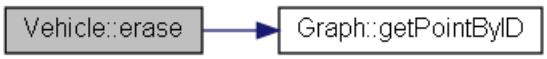
Here is the caller graph for this function:



void Vehicle::erase ()

Set the vehicle deleted.

Here is the call graph for this function:

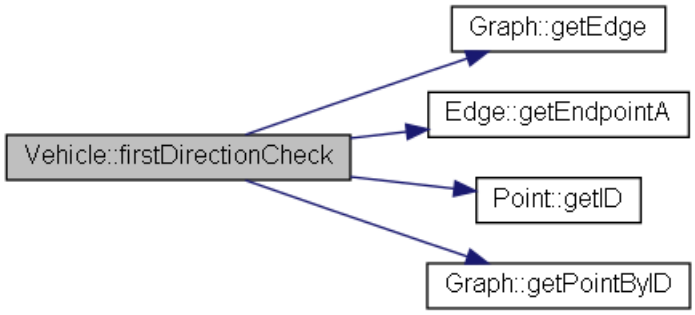


Here is the caller graph for this function:

void Vehicle::firstDirectionCheck () [private]

Calculate the first start direction and track.

Here is the call graph for this function:



Here is the caller graph for this function:

size_t Vehicle::getID ()

Getter for the **Vehicle** ID.

Returns

vehicle ID.

size_t Vehicle::getObject3DiD ()

Getter for the render ID.

Returns

The render ID.

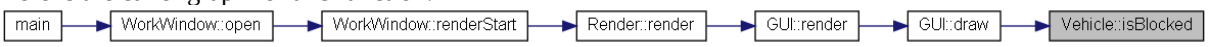
bool Vehicle::isBlocked ()

Getter for the blocked flag.

Returns

The blocked flag.

Here is the caller graph for this function:



bool Vehicle::isDeleted ()

Getter for the deleted flag.

Returns

The deleted flag.

bool Vehicle::isFinished ()

Getter for the finish flag.

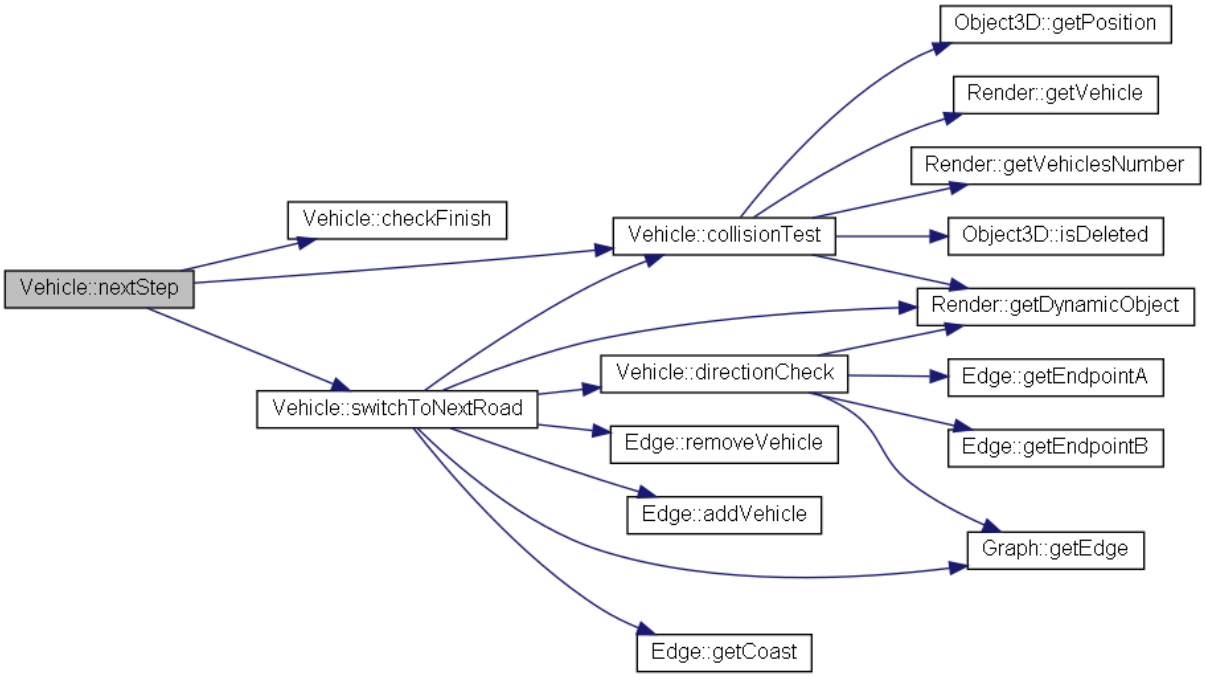
Returns

Here is the caller graph for this function:

void Vehicle::nextStep () [private]

Step to the next point.

Here is the call graph for this function:

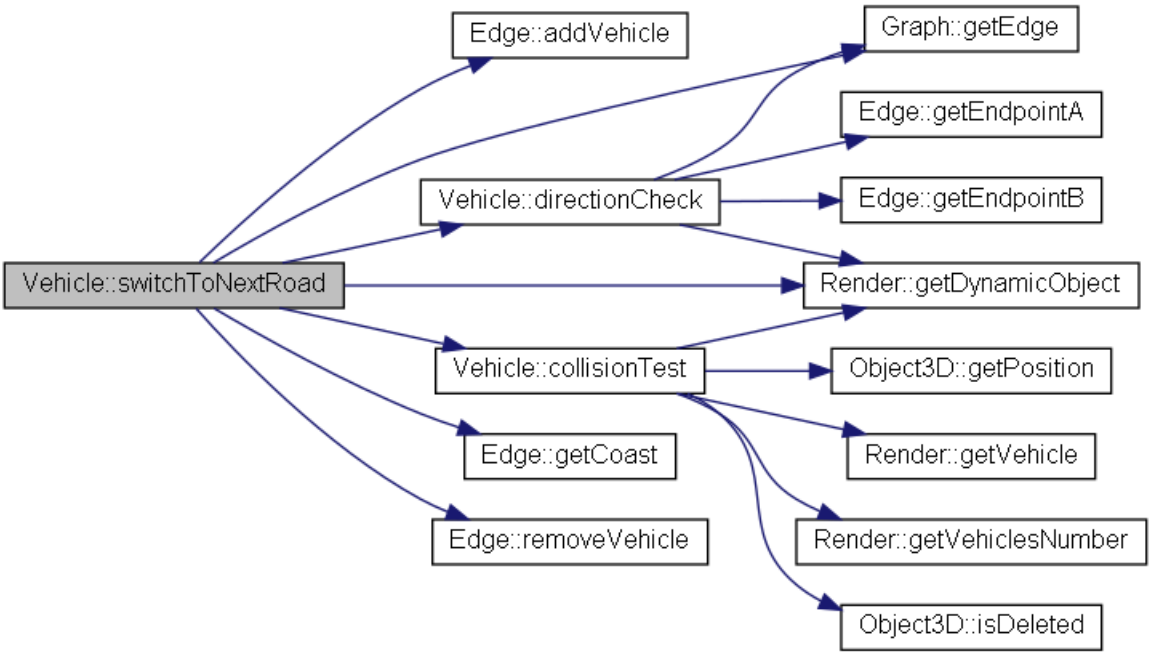


Here is the caller graph for this function:

void Vehicle::switchToNextRoad () [private]

Check the next road.

Here is the call graph for this function:



Here is the caller graph for this function:

void Vehicle::unblock ()

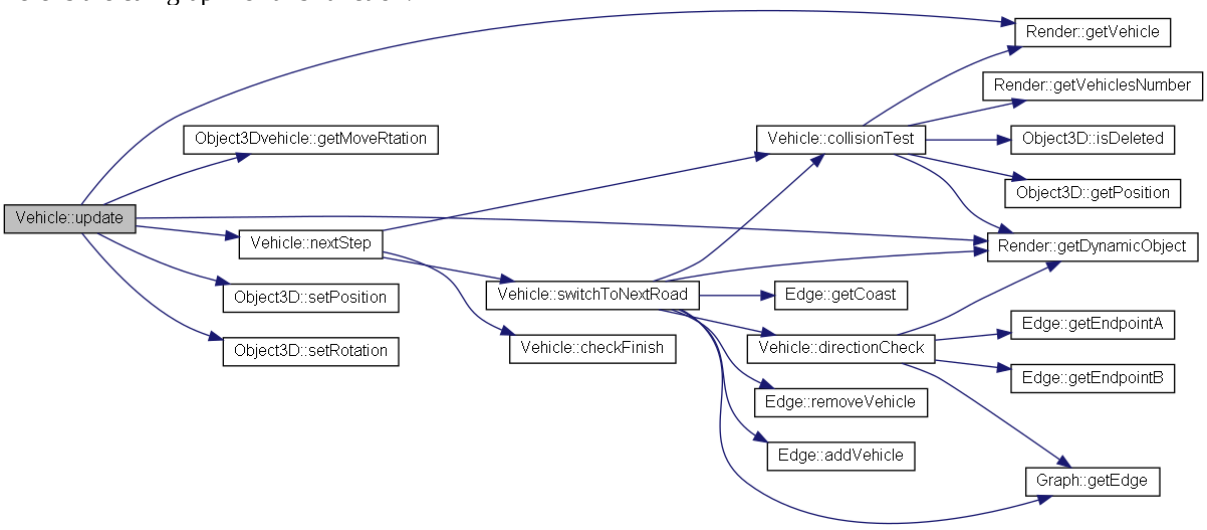
Unblock the vehicle.

Here is the caller graph for this function:

void Vehicle::update ()

Update the **Vehicle** status.

Here is the call graph for this function:



Field Documentation

int Vehicle::allCost = 0

The cost calculator.

Animator* Vehicle::animator [private]

The pointer to the current animator object.

int Vehicle::collisionCheckDistance = 15 [static]

The collision detection static distance.

The collision prevention distance.

bool Vehicle::collisionPrevention = true [static]

Collision detection static flag.

The collision prevention flag.

size_t Vehicle::currentEdgeOnThePath = 0 [private]

The current road number on the path.

size_t Vehicle::currentPointOnTheRoad = 0 [private]

The current point on the current road.

size_t Vehicle::currentRoad = 0 [private]

The current road ID.

bool Vehicle::deleted = false [private]

The deleted flag.

size_t Vehicle::destinationID = 0

the destination point id.

Dijkstra* Vehicle::dijkstra [private]

The pointer to the vehicle's own **Dijkstra** object.

char Vehicle::direction = 'a' `[private]`

the traveling direction marker.

Uint32 Vehicle::endTime = 0

The end time.

bool Vehicle::finished = false `[private]`

The finished flag.

Graph* Vehicle::graph `[private]`

The pointer to the current graph.

size_t Vehicle::hopCounter = 0

The hop counter.

size_t Vehicle::ID = 0 `[private]`

The vehicle moddel ID.

bool Vehicle::nextRoadIsBlocked = false `[private]`

The blocked flag.

size_t Vehicle::object3DiD = 0 `[private]`

The render ID.

std::vector<size_t> Vehicle::path `[private]`

The current calculated path.

Render* Vehicle::render `[private]`

The pointer to the current render.

bool Vehicle::repath = true `[static]`

Repath function static flag.

The rapath function flag.

const size_t Vehicle::standardRoadLenght = 100 `[private]`

The standard road length constant.

size_t Vehicle::startID = 0

The start point ID.

Uint32 Vehicle::startTime = 0

The start time.

char Vehicle::track = '1' `[private]`

The track marker.

int Vehicle::vehicleWeight = 100 `[static]`

The vehicle weight in the graph.

The default vehicle weight.

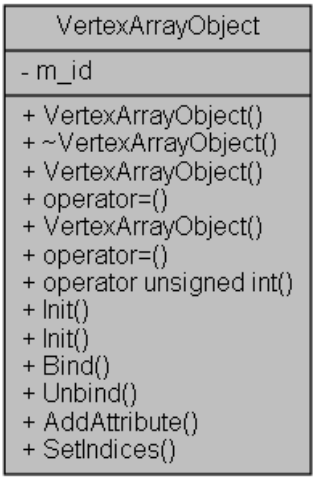
The documentation for this class was generated from the following files:

- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Vehicle.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/**Vehicle.cpp**

VertexArrayObject Class Reference

`#include <VertexArrayObject.h>`

Collaboration diagram for VertexArrayObject:



Public Member Functions

- **VertexArrayObject** ()
- **~VertexArrayObject** ()
- **VertexArrayObject** (const **VertexArrayObject** &)=delete
- **VertexArrayObject** & **operator=** (const **VertexArrayObject** &)=delete
- **VertexArrayObject** (**VertexArrayObject** &&)
- **VertexArrayObject** & **operator=** (**VertexArrayObject** &&)
- **operator unsigned int** () const
- void **Init** (std::initializer_list< std::pair< **AttributeData**, const **ArrayBuffer** & > > pDataBuffers)
- void **Init** (std::initializer_list< std::pair< **AttributeData**, const **ArrayBuffer** & > > pDataBuffers, const **IndexBuffer** &pIndexBuffer)
- **VertexArrayObject** & **Bind** ()
- void **Unbind** ()
- template<**BufferType** type, **BufferUsage** usage> **VertexArrayObject** & **AddAttribute** (**AttributeData** &, **BufferObject**< type, usage > &)
- **VertexArrayObject** & **SetIndices** (const **IndexBuffer** &)

Private Attributes

- GLuint **m_id** = 0

Constructor & Destructor Documentation

VertexArrayObject::VertexArrayObject ()

VertexArrayObject::~VertexArrayObject ()

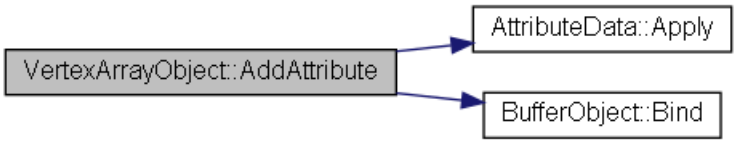
VertexArrayObject::VertexArrayObject (const VertexArrayObject &) `[delete]`

VertexArrayObject::VertexArrayObject (VertexArrayObject && *rhs*)

Member Function Documentation

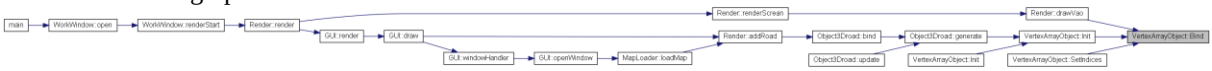
template<BufferType type, BufferUsage usage> VertexArrayObject & VertexArrayObject::AddAttribute (AttributeData & *pAttrib*, BufferObject< type, usage > & *pBuffer*) `[inline]`

Here is the call graph for this function:



VertexArrayObject & VertexArrayObject::Bind ()

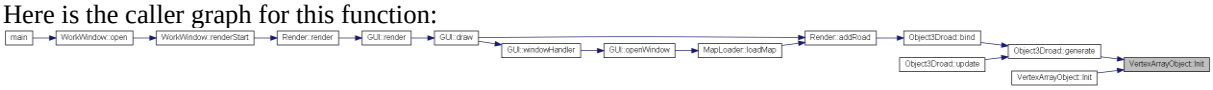
Here is the caller graph for this function:



void VertexArrayObject::Init (std::initializer_list< std::pair< AttributeData, const ArrayBuffer & > > *pDataBuffers*)

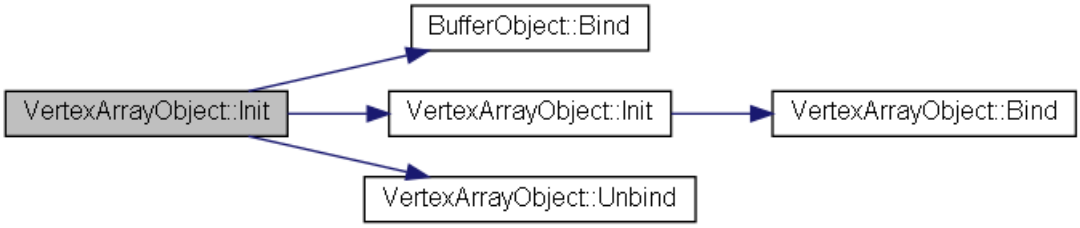
Here is the call graph for this function:

Here is the caller graph for this function:



void VertexArrayObject::Init (std::initializer_list< std::pair< AttributeData, const ArrayBuffer & > > *pDataBuffers*, const IndexBuffer & *pIndexBuffer*)

Here is the call graph for this function:



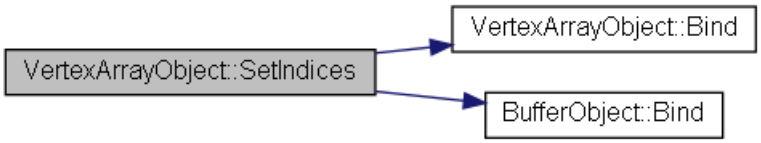
VertexArrayObject::operator unsigned int () const `[inline]`

VertexArrayObject & VertexArrayObject::operator= (const VertexArrayObject &) `[delete]`

VertexArrayObject & VertexArrayObject::operator= (VertexArrayObject && *rhs*)

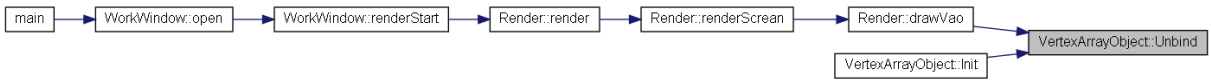
VertexArrayObject & VertexArrayObject::SetIndices (const IndexBuffer & *pIndexBuffer*)

Here is the call graph for this function:



void VertexArrayObject::Unbind ()

Here is the caller graph for this function:



Field Documentation

GLuint VertexArrayObject::m_id = 0 `[private]`

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

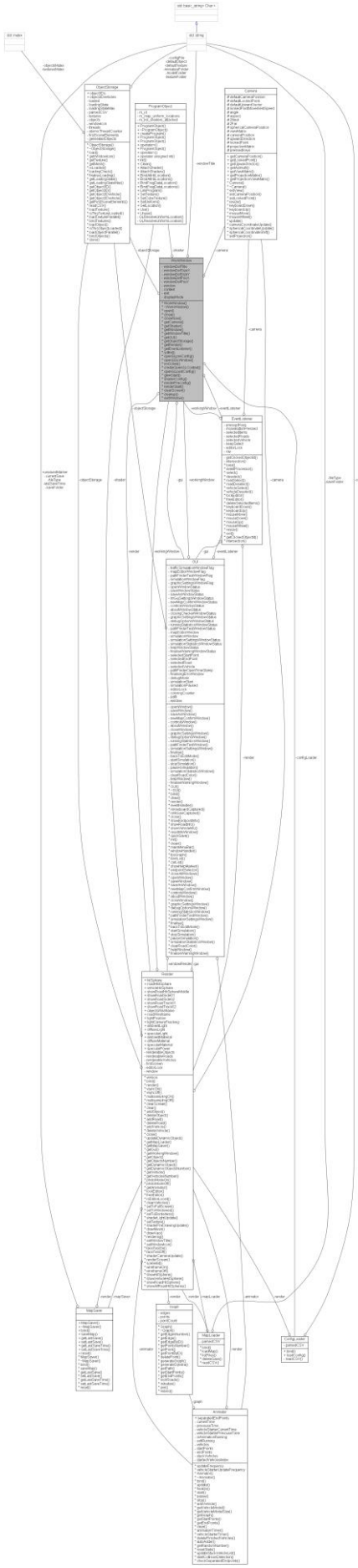
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**VertexArrayObject.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**VertexArrayObject.cpp**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/**VertexArrayObject.inl**

WorkWindow Class Reference

The OpenGL 3D window implementation.

```
#include <WorkWindow.h>
```

Collaboration diagram for WorkWindow:



Private Attributes

- const char * **windowDefTitle** = "Traffic Simulator"
  The main window default tittle text.

- const int **windowDefSizeX** = 640
  The default windows with.

- const int **windowDefSizeY** = 480
  The default window height.

- const int **windowDefPosX** = 100
  The default window horizontal position.

- const int **windowDefPosY** = 100
  The default window vertical position.

- std::string **windowTitle** = "Traffic simulator"
  The main window base tittle text.

- SDL_Window * **window**
  The SDL window pointer.

- SDL_GLContext **context**
  The openGL context object.

- **ProgramObject shader**
  The shader object.

- **Camera camera**
  The camera controller object.

- **EventListener eventListener**
  The event listener object.

- **Render render**
  The render object.

- **ConfigLoader configLoader**
  Configuration file loader.

- **GUI gui**
  *The **GUI** controller object.*

- **ObjectStorage objectStorage**
  The 3D Object storage object.

- bool **exit** = false
  The render loop running parameter.

- int **displayMode** = 0
  The current window mode.

Traffic Simulation

- **WorkWindow** (void)
  OpenGL 3D Window constructor.

- **~WorkWindow** (void)
  OpenGL 3D Window destructor.

- int **open** ()
  Open and show the window.

- void **close** ()
  Start the closing procedure.

- void **closeNow** ()
  Stop the render and start the closing procedure.

- **Camera** * **getCamera** ()
  Getter for the camera controller.

- **ProgramObject** * **getShader** ()
  Getter for the 3D Window compiled shader program.

- SDL_Window * **getWindow** ()
  Getter for the SDL Windows object.

- std::string **getWindowTitle** ()
  Getter for the window title.

- **GUI** * **getGUI** ()
  *Getter for the **GUI** representer.*

- **ObjectStorage** * **getObjectStorage** ()
  Getter for the object storage.

- **Render** * **getRender** ()
  Getter for the render object.

- **EventListener** * **getEventListener** ()
  Getter for the event listener.

- int **sdlInit** ()
  SDL system initialization.

- int **openGLpreConfig** ()
  OpenGL pre-configuration.

- int **openSDLWindow** ()
  SDL window opening.

- int **imGUIinit** ()
  inGUI GUS system start.

- int **createOpenGLContext** ()
  OpenGL Context creation.

- int **openGLpostConfig** ()
  OpenGL Variables setting.

- int **glewStart** ()
  GLEW System start.

- int **shaderConfig** ()
  OpenGL Shader configuration compiling.

- int **renderPreconfig** ()
  ***Render** variable setting.*

- int **renderStart** ()
  ***Render** Start!*

- void **clearScreen** ()
  Clear the screen.

- void **cleanup** ()
  Cleanup before closing.

- static void **exitWindow** ()
  Windows close callback function.

Detailed Description

The OpenGL 3D window implementation.

Constructor & Destructor Documentation

WorkWindow::WorkWindow (void)

OpenGL 3D Window constructor.

WorkWindow::~WorkWindow (void)

OpenGL 3D Window destructor.

Here is the call graph for this function:

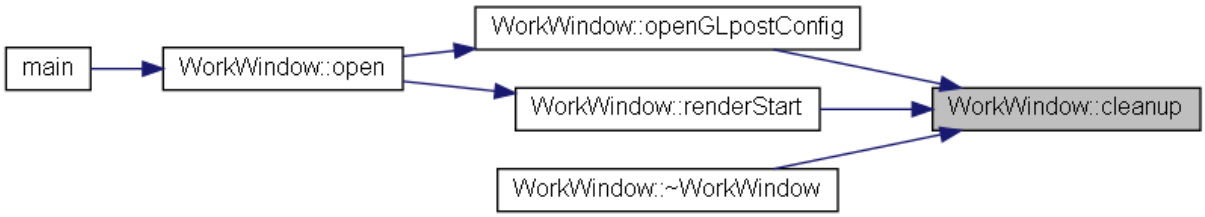
Member Function Documentation

void WorkWindow::cleanup () [private]

Cleanup before closing.

Here is the call graph for this function:

Here is the caller graph for this function:



void WorkWindow::clearScreen () [private]

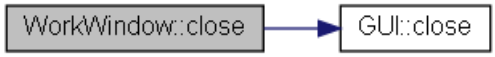
Clear the screen.

Here is the caller graph for this function:

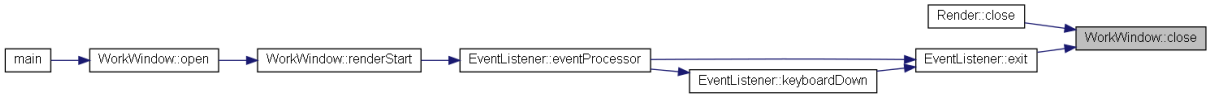
void WorkWindow::close ()

Start the closing procedure.

Here is the call graph for this function:



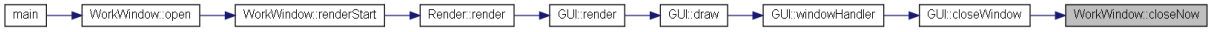
Here is the caller graph for this function:



void WorkWindow::closeNow ()

Stop the render and start the closing procedure.

Here is the caller graph for this function:



int WorkWindow::createOpenGLContext () `[private]`

OpenGL Context creation.

Returns

Status ID. 0 is OK status.

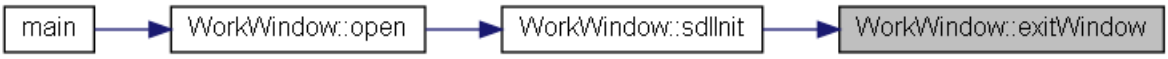
Here is the call graph for this function:

Here is the caller graph for this function:

void WorkWindow::exitWindow () `[static], [private]`

Windows close callback function.

Here is the caller graph for this function:



Camera * WorkWindow::getCamera ()

Getter for the camera controller.

Returns

pointer for the camera controller.

Here is the caller graph for this function:

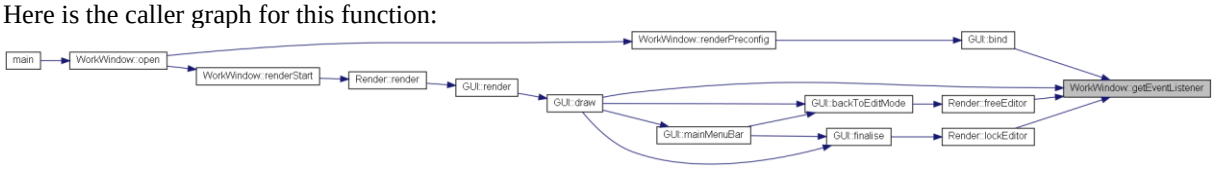
EventListener * WorkWindow::getEventListener ()

Getter for the event listener.

Returns

Pointer for the current event listener.

Here is the caller graph for this function:



GUI * WorkWindow::getGUI ()

Getter for the **GUI** representer.

Returns

Pointer to the **GUI** controller class.

Here is the caller graph for this function:

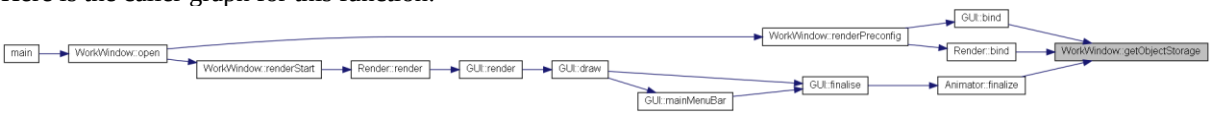
ObjectStorage * WorkWindow::getObjectStorage ()

Getter for the object storage.

Returns

Pointer to the object storage.

Here is the caller graph for this function:



Render * WorkWindow::getRender ()

Getter for the render object.

Returns

Pointer to the current render.

Here is the caller graph for this function:

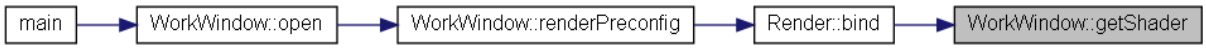
ProgramObject * WorkWindow::getShader ()

Getter for the 3D Window compiled shader program.

Returns

The OpõenGL 3D Shader.

Here is the caller graph for this function:



SDL_Window * WorkWindow::getWindow ()

Getter for the SDL Windows object.

Returns

The SDL OpenGL window object.

Here is the caller graph for this function:

std::string WorkWindow::getWindowTitle ()

Getter for the window title.

Returns

The window title.

Here is the caller graph for this function:

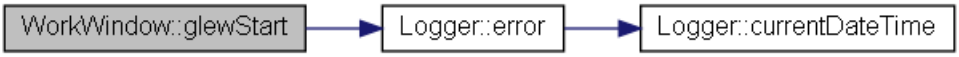
int WorkWindow::glewStart () `[private]`

GLEW System start.

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

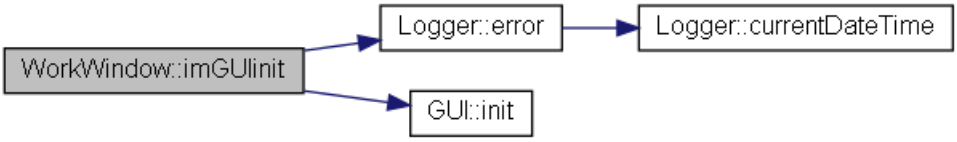
int WorkWindow::imGUIinit () `[private]`

inGUI GUS system start.

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

int WorkWindow::open ()

Open and show the window.

This run all the preparation function and start the rendering.

Returns

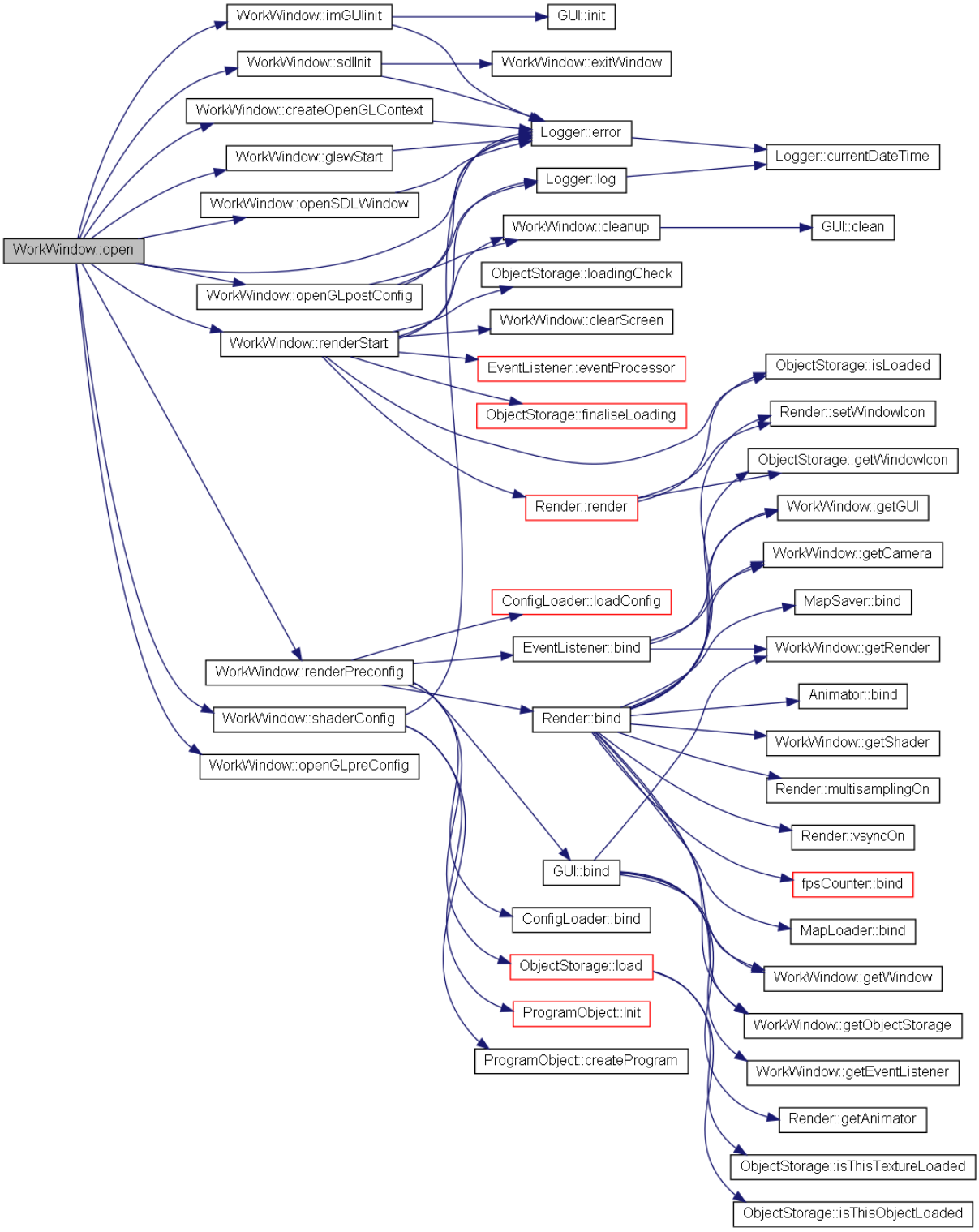
The starting process status number. 0 is OK Other ID-s mean errors.

This run all the preparation function and start the rendering. Possible error codes: 0 - Everything is fine. 1 - SDL Initialization error. 2 - OpenGL 3D Window creation error. 3 - imGUI **GUI** system Initialization error. 4 - OpenGL Context creation error. 5 - GLEW system initialization error. 6 - OpenGL starting or version error. 7 - Shader compiling or starting error.

Returns

The starting process status number. 0 is OK Other ID-s mean errors.

Here is the call graph for this function:



Here is the caller graph for this function:

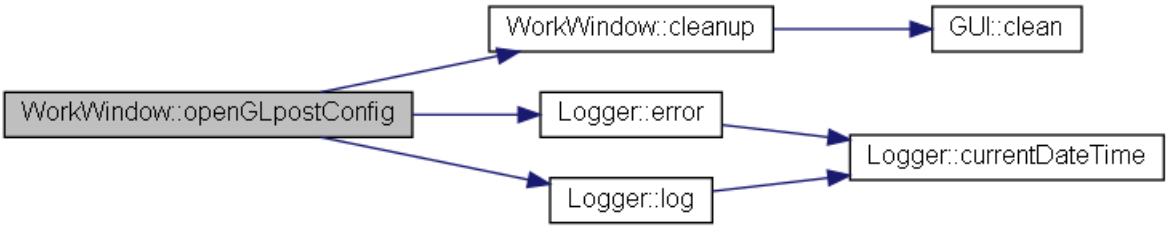
int WorkWindow::openGLpostConfig () [private]

OpenGL Variables setting.

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

int WorkWindow::openGLpreConfig () [private]

OpenGL pre-configuration.

Returns

Status ID. 0 is OK status.

Here is the caller graph for this function:

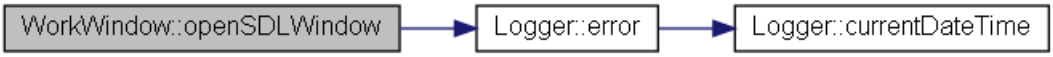
int WorkWindow::openSDLWindow () [private]

SDL window opening.

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

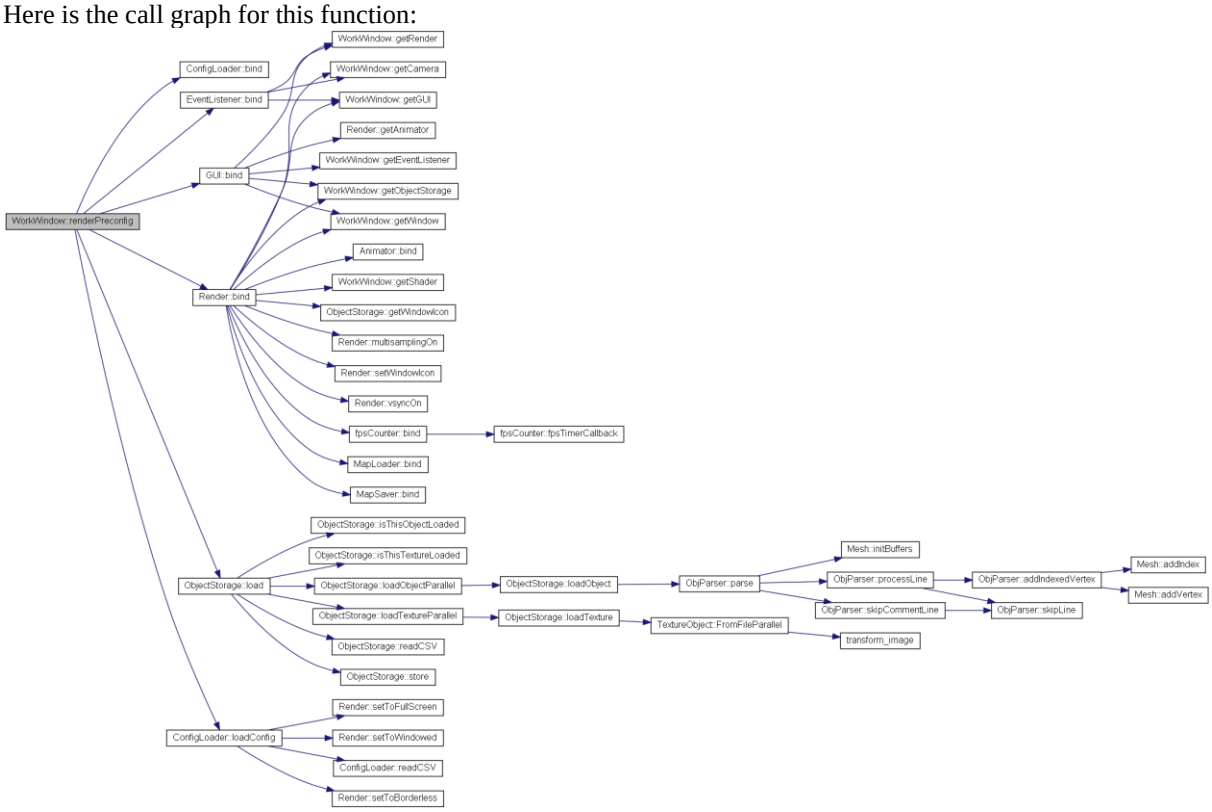
int WorkWindow::renderPreconfig () [private]

Render variable setting.

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

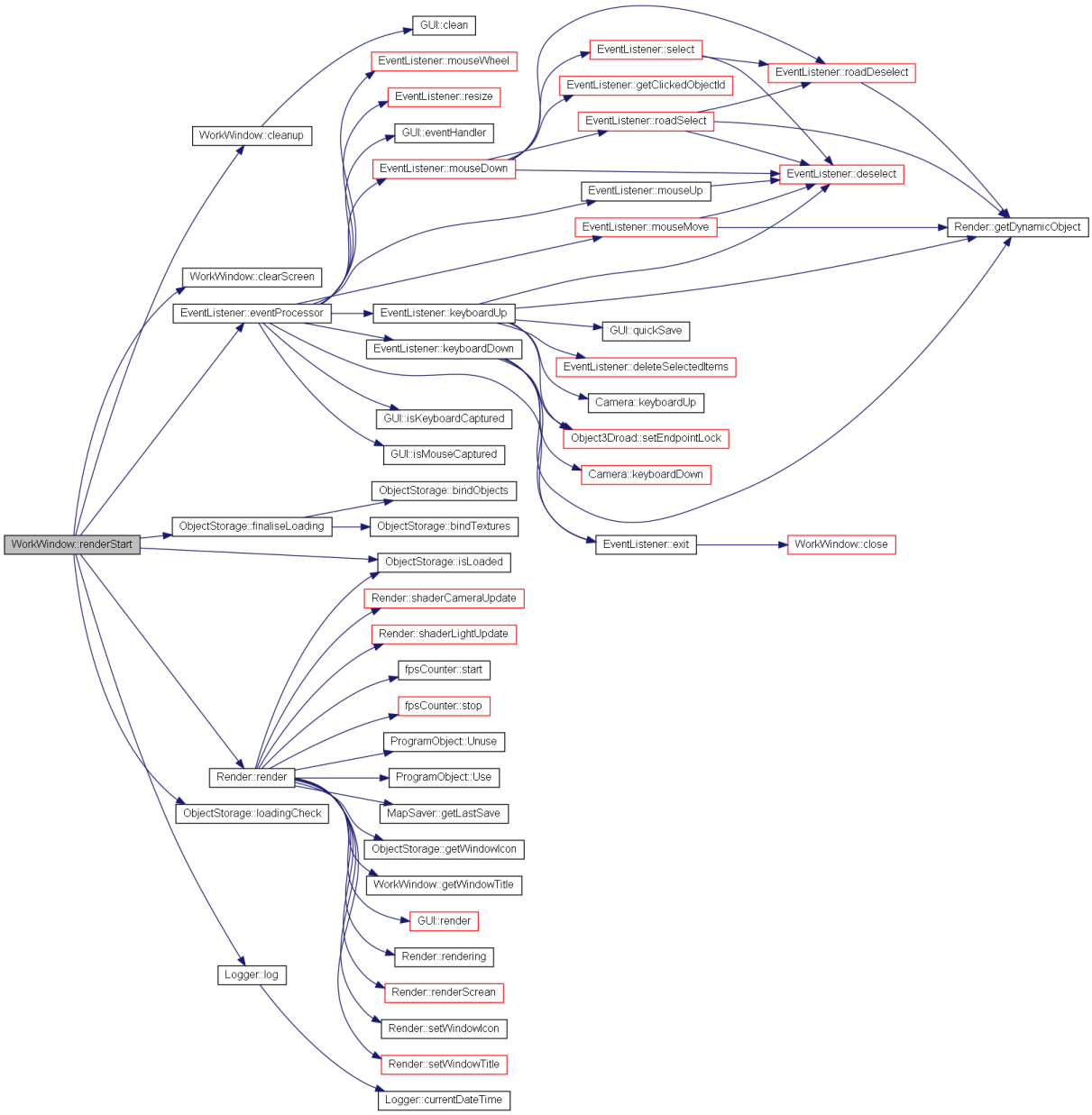
int WorkWindow::renderStart () `[private]`

Render Start!

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

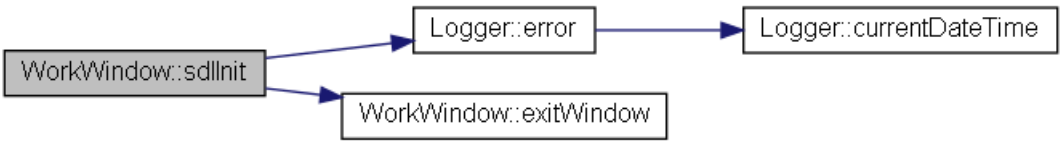
int WorkWindow::sdlInit () [private]

SDL system initialization.

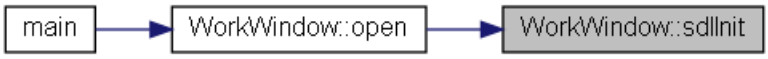
Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:



int WorkWindow::shaderConfig () [private]

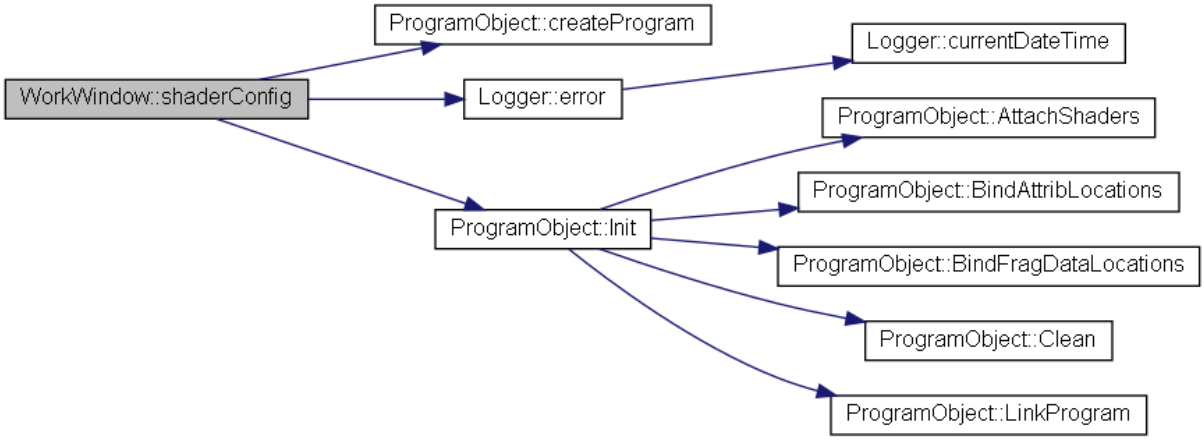
OpenGL Shader configuration compiling.

OpenGL Shader configuration and compiling.

Returns

Status ID. 0 is OK status.

Here is the call graph for this function:



Here is the caller graph for this function:

Field Documentation

Camera WorkWindow::camera [private]

The camera controller object.

ConfigLoader WorkWindow::configLoader [private]

Configuration file loader.

SDL_GLContext WorkWindow::context [private]

The openGL context object.

int WorkWindow::displayMode = 0 [private]

The current window mode.

0 - Windowed 1 - Borderless 2 - Full screen

EventListener WorkWindow::eventListener [private]

The event listener object.

bool WorkWindow::exit = false `[private]`

The render loop running parameter.

GUI WorkWindow::gui `[private]`

The **GUI** controller object.

ObjectStorage WorkWindow::objectStorage `[private]`

The 3D Object storage object.

Render WorkWindow::render `[private]`

The render object.

ProgramObject WorkWindow::shader `[private]`

The shader object.

SDL_Window* WorkWindow::window `[private]`

The SDL window pointer.

const int WorkWindow::windowDefPosX = 100 `[private]`

The default window horizontal position.

const int WorkWindow::windowDefPosY = 100 `[private]`

The default window vertical position.

const int WorkWindow::windowDefSizeX = 640 `[private]`

The default windows with.

const int WorkWindow::windowDefSizeY = 480 `[private]`

The default window height.

const char* WorkWindow::windowDefTitle = "Traffic Simulator" `[private]`

The main window default tittle text.

std::string WorkWindow::windowTitle = "Traffic simulator" `[private]`

The main window base tittle text.

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

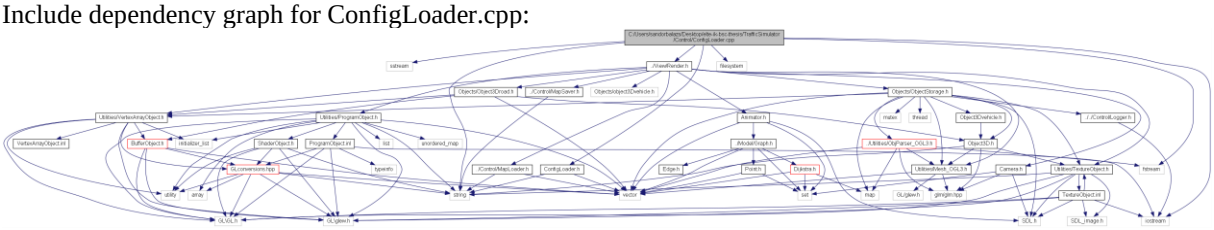
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**WorkWindow.h**
- C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/**WorkWindow.cpp**

File Documentation

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigLoader.cpp File Reference

```
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <filesystem>
#include "ConfigLoader.h"
#include "../View/Render.h"
```

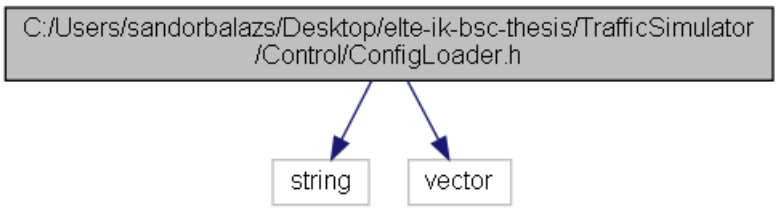
Include dependency graph for ConfigLoader.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigLoader.h File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for ConfigLoader.h:



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

Data Structures

- class **ConfigLoader**
  The configuration loader class.

ConfigLoader.h

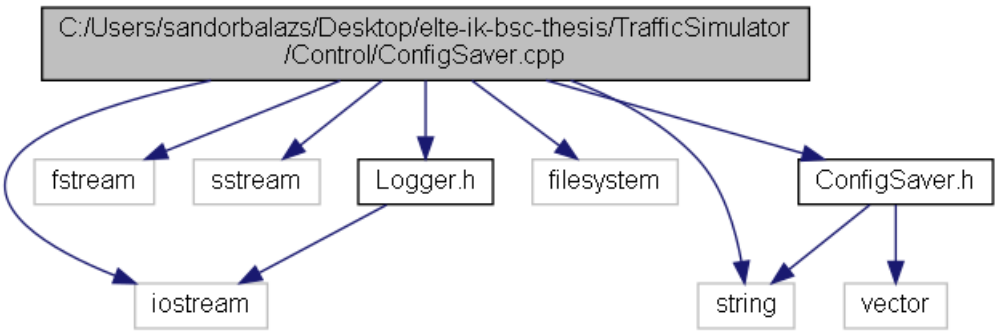
Go to the documentation of this file.1

```
//Understanding is a three edged sword.

#pragma once

//Avoiding the circular include.
class Render;

#include <string>
#include <vector>

class ConfigLoader {
public:
        void bind(Render* render);

        void loadConfig();

private:

        std::string configFile = "config.csv";

        Render* render = NULL;

        std::vector<std::vector<std::string>> parsedCSV;

        void readCSV(std::string fileName);
};
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigSaver.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <filesystem>
#include "ConfigSaver.h"
#include "Logger.h"
```

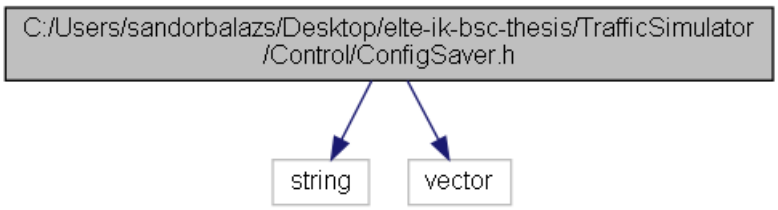
Include dependency graph for ConfigSaver.cpp:



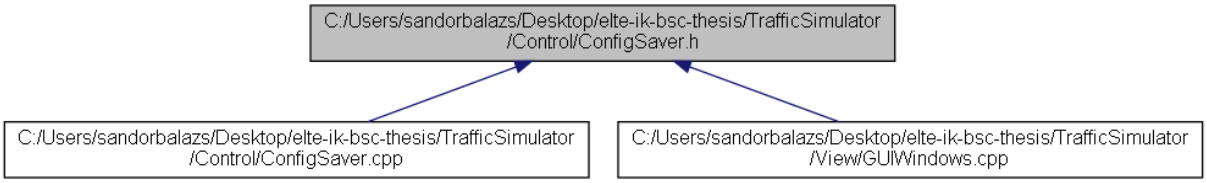
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/ConfigSaver.h File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for ConfigSaver.h:



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



Data Structures

- class **ConfigSaver**
  Configuration saver class.

ConfigSaver.h

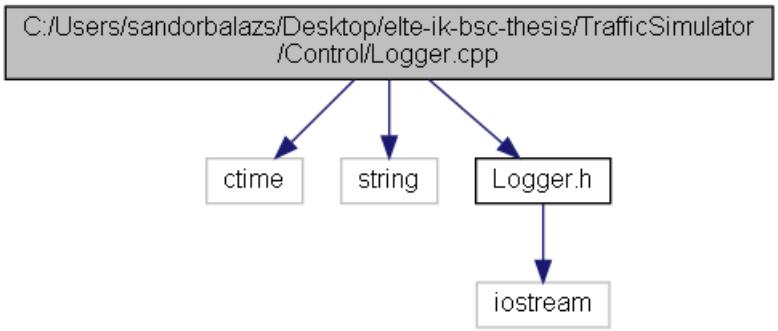
Go to the documentation of this file.1

```
//Alarm 1202, Alarm 1201

#pragma once


#include <string>
#include <vector>

class ConfigSaver {
public:

        void setProperty(std::string propertyName, std::string value);

private:

        std::string configFile = "config.csv";

        std::vector<std::vector<std::string>> parsedCSV;

        void writeCSV(std::string fileName);

        void readCSV(std::string fileName);
};
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/Logger.cpp File Reference

```
#include <ctime>
#include <string>
#include "Logger.h"
```

Include dependency graph for Logger.cpp:



Macros

- Traffic Simulation#define **_CRT_SECURE_NO_WARNINGS**

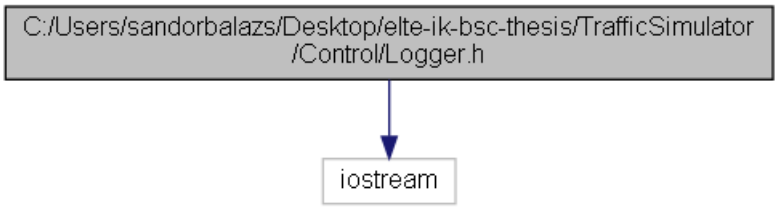
Macro Definition Documentation

#define _CRT_SECURE_NO_WARNINGS

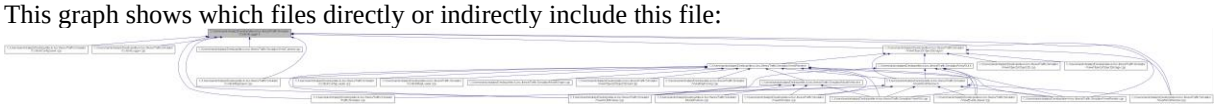
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/Logger.h File Reference

```
#include <iostream>
```

Include dependency graph for Logger.h:



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



Data Structures

- class **Logger**
  Logging system implementation.

Logger.h

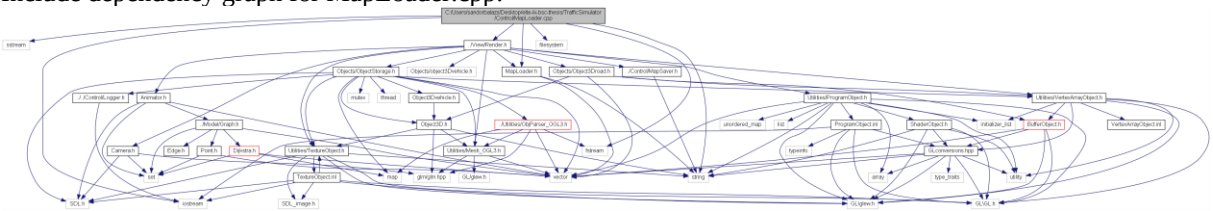
Go to the documentation of this file.1

```
11 #pragma once
12
13 #include <iostream>
14
18 class Logger {
19 public:
20
24         static void startFileLogging();
25
29         static void stopFileLogging();
30
35         static std::string currentDateTime();
36
41         static void log(std::string message);
42
47         static void error(std::string message);
48
54         static void error(std::string message, int errorCode);
55
56 private:
57
61         Logger() {}
62 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapLoader.cpp File Reference

```
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <filesystem>
#include "MapLoader.h"
#include "../View/Render.h"
```

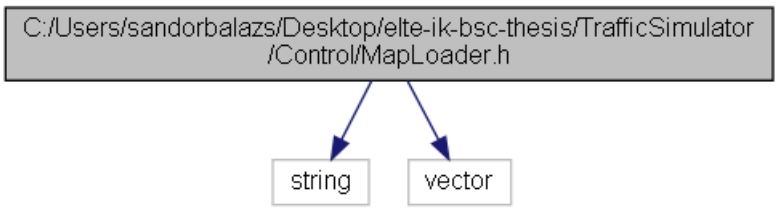
Include dependency graph for MapLoader.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapLoader.h File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for MapLoader.h:



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

Data Structures

- class **MapLoader**
  Map loader class.

MapLoader.h

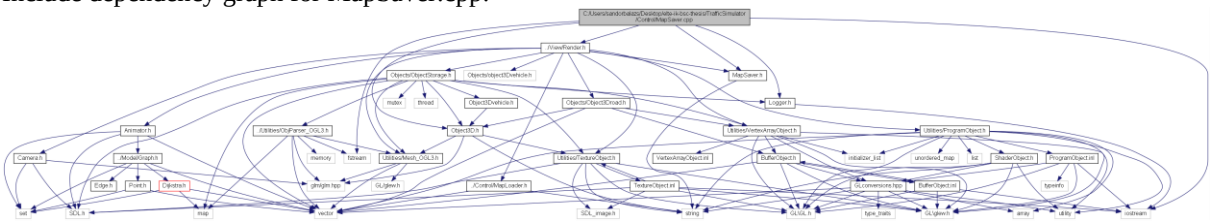
Go to the documentation of this file.1

```
11 #pragma once
12
16 class Render;
17
18 #include <string>
19 #include <vector>
20
24 class MapLoader {
25 public:
26
31         void bind(Render* render);
32
37         void loadMap(std::string fileName);
38
43         std::vector<std::string> listFiles();
44
50         int deleteSave(std::string fileName);
51
52 private:
53
57         const std::string fileType = ".csv";
58
62         std::string saveFolder = "Saves/";
63
67         Render* render = NULL;
68
72         std::vector<std::vector<std::string>> parsedCSV;
73
78         void readCSV(std::string fileName);
79 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapSaver.cpp File Reference

```
#include <iostream>
#include <fstream>
#include "MapSaver.h"
#include "Logger.h"
#include "../View/Render.h"
#include "../View/Objects/Object3D.h"
```

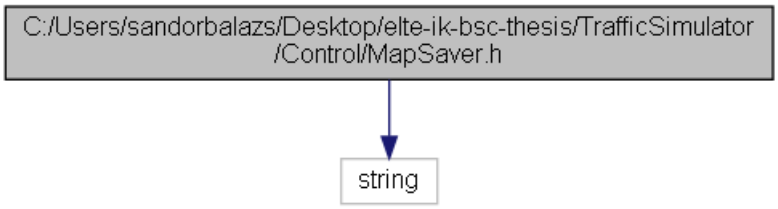
Include dependency graph for MapSaver.cpp:



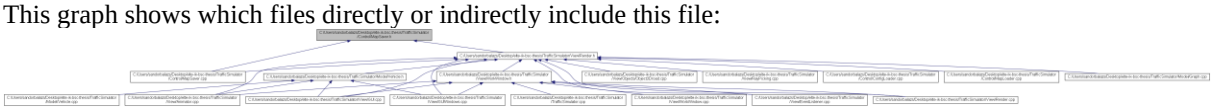
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Control/MapSaver.h File Reference

`#include <string>`

Include dependency graph for MapSaver.h:



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



Data Structures

- class **MapSaver**
  Map saver mechanism implementation.

MapSaver.h

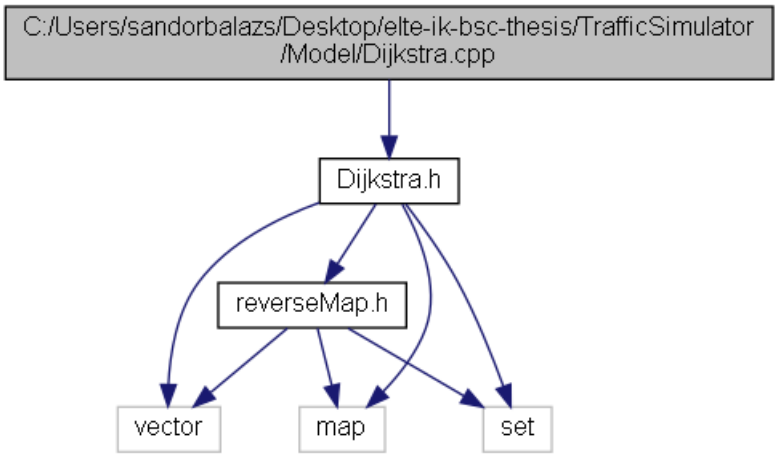
Go to the documentation of this file.1

```
11 #pragma once
12
16 class Render;
17
18 #include <string>
19
23 class MapSaver {
24 public:
25
30         MapSaver(void);
31
36         ~MapSaver(void);
37
42         void bind(Render* render);
43
48         void saveMap(std::string fileName);
49
54         std::string getLastSave();
55
60         void setLastSave(std::string loadedMap);
61
66         std::string getLastSaveTime();
67
72         void setLastSaveTime();
73
77         void reset();
78
82         const std::string unsavedMarker = "Unsaved";
83
84 private:
85
89         const std::string fileType = ".csv";
90
94         std::string currentSave = unsavedMarker;
95
99         const std::string saveFolder = "Saves\\";
100
104         std::string lastSaveTime = unsavedMarker;
105
109         Render* render;
110 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Dijkstra.cpp File Reference

```
#include "Dijkstra.h"
```

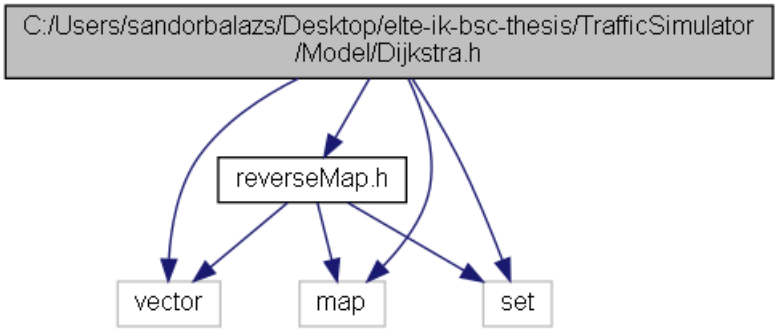
Include dependency graph for Dijkstra.cpp:



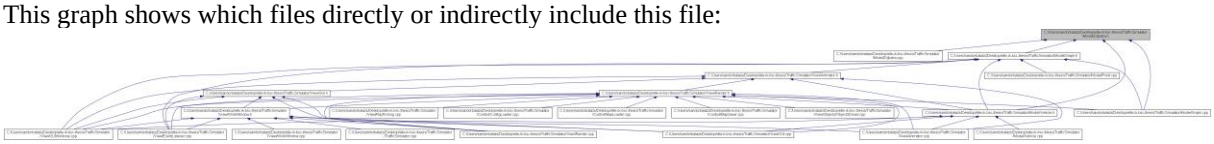
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Dijkstra.h File Reference

```
#include <vector>
#include <map>
#include <set>
#include "reverseMap.h"
```

Include dependency graph for Dijkstra.h:



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



Data Structures

- class **Dijkstra**
  Dijkstra *algorithm implementation.*

Dijkstra.h

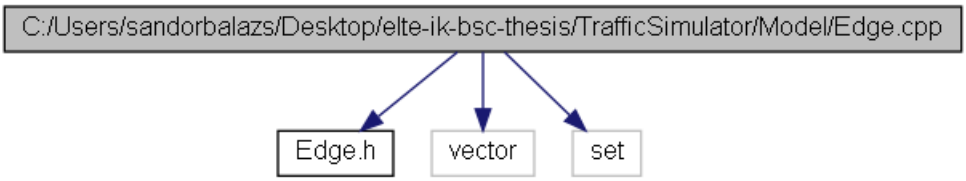
Go to the documentation of this file.1

```
11 #pragma once
12
13 #include <vector>
14 #include <map>
15 #include <set>
16 #include "reverseMap.h"
17
21 class Dijkstra {
22 public:
28     Dijkstra(size_t nodesCount, size_t startNode);
29
33     ~Dijkstra();
34
41     void addEdge(size_t fromNode, size_t toNode, int weight);
42
49     void addDoubleEdge(size_t node1, size_t node2, int weight);
50
55     size_t getStartNode();
56
60     void run();
61
65     std::vector<size_t> from;
66
70     std::vector<std::vector<int>> matrix;
71
72 private:
73
77     const int infinite = INT_MAX - 1;
78
82     const int empty = -1;
83
87     size_t nodesCount;
88
92     size_t startNode;
93
97     std::set<size_t> ready;
98
102     std::vector<int> distance;
103
107     reverseMap next;
108
113     void updateDistance(size_t node);
114
119     void initNext(size_t startNode);
120
125     void initFrom(size_t startNode);
126
132     size_t getNextValue(size_t node);
133
138     void updateFrom(size_t node);
139
145     bool isReady(size_t node);
146
150     void init();
151
155     void prep();
156 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Edge.cpp File Reference

```
#include "Edge.h"
#include <vector>
#include <set>
```

Include dependency graph for Edge.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Edge.h File Reference

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

Data Structures

- class **Edge**
  *The abstract **Edge** implementation.*

Edge.h

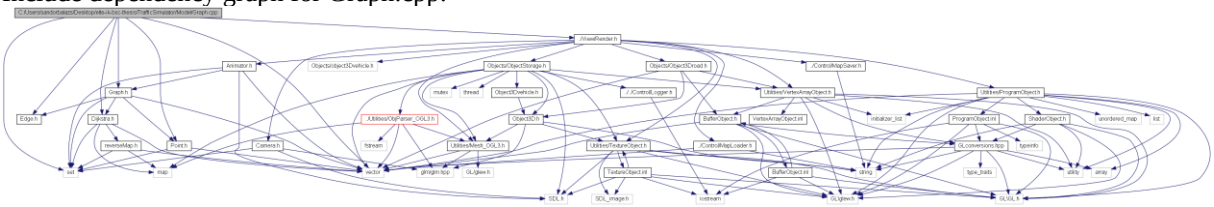
Go to the documentation of this file.1

```
10 #pragma once
11
15 class Edge {
16 public:
25         Edge(size_t id, size_t road3DiD, size_t endpointA, size_t endpointB, int
coast);
26
30         ~Edge();
31
37         void changePoint(size_t originalPoint, size_t newPoint);
38
43         size_t getID();
44
49         size_t getRoad3DiD();
50
55         size_t getEndpointA();
56
61         size_t getEndpointB();
62
67         int getCoast();
68
73         int getLength();
74
79         int getVehicleCoast();
80
85         size_t getVehicleCount();
86
91         size_t getAllVehicleCount();
92
98         void addVehicle(int c, bool repath);
99
105         void removeVehicle(int c, bool repath);
106
113         bool match(size_t endpointA, size_t endpointB);
114
118         void resetStats();
119
120 private:
121
125         size_t id = -1;
126
130         size_t road3DiD = -1;
131
135         size_t endpointA = -1;
136
140         size_t endpointB = -1;
141
145         int coast = 0;
146
150         int vehicleCoast = 0;
151
155         size_t vehicleCount = 0;
156
160         size_t allVehicleCount = 0;
161 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Graph.cpp File Reference

```
#include "Graph.h"
#include "Edge.h"
#include "Point.h"
#include "Dijkstra.h"
#include "../View/Render.h"
#include <vector>
#include <set>
```

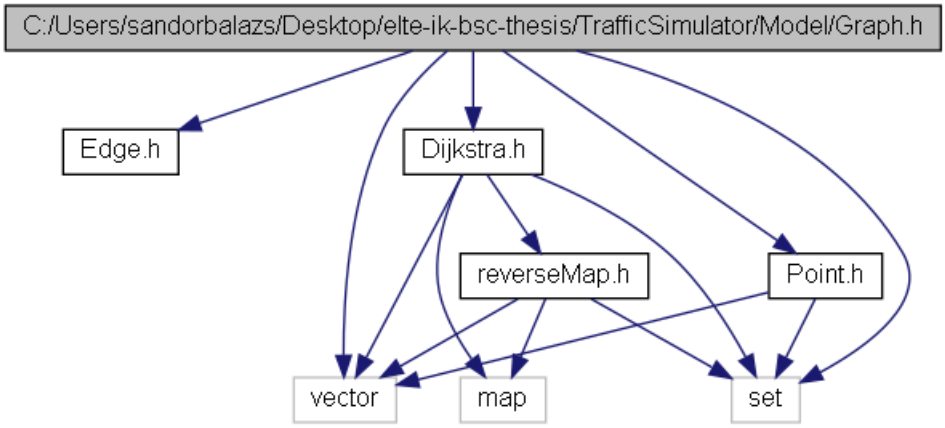
Include dependency graph for Graph.cpp:



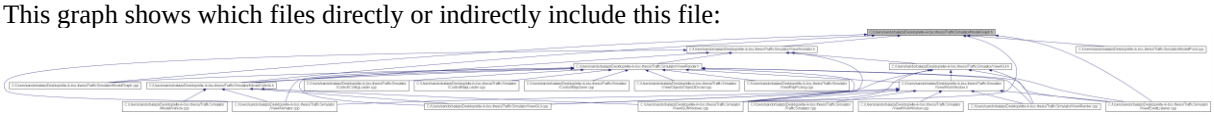
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Graph.h File Reference

```
#include "Edge.h"
#include "Point.h"
#include "Dijkstra.h"
#include <vector>
#include <set>
```

Include dependency graph for Graph.h:



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



Data Structures

- class **Graph**
  *The abstract **Edge** implementation.*

Graph.h

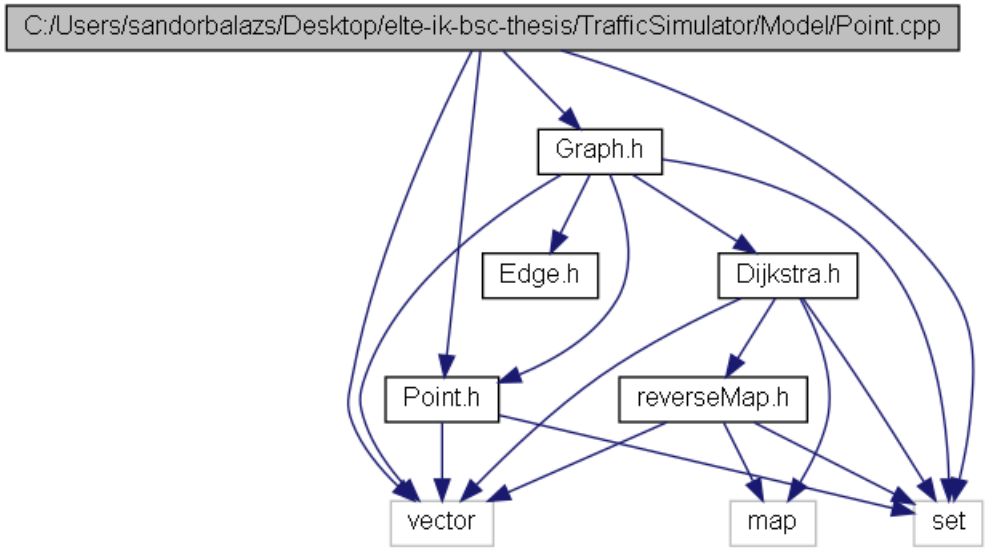
Go to the documentation of this file.1

```
10 #pragma once
11
15 class Render;
16
17 #include "Edge.h"
18 #include "Point.h"
19 #include "Dijkstra.h"
20 #include <vector>
21 #include <set>
22
26 class Graph {
27 public:
28
33         Graph(Render* render);
34
38         ~Graph();
39
44         size_t getEdgesNumber();
45
51         Edge* getEdge(size_t edge);
52
58         Edge* getEdgeByID(size_t edge);
59
64         size_t getPointsNumber();
65
71         Point* getPoint(size_t point);
72
78         Point* getPointByID(size_t point);
79
84         void deletePoint(size_t point);
85
89         void generateGraph();
90
96         Dijkstra* generateDijkstra(size_t startPoint);
97
104         std::vector<size_t> getPath(Dijkstra* dijkstra, size_t target);
105
110         std::vector<size_t> getStartPoints();
111
116         std::vector<size_t> getEndPoints();
117
118         private:
119
123         Render* render;
124
128         std::vector<Edge*> edges;
129
133         std::vector<Point*> points;
134
138         size_t pointCount = 0;
139
143         void lockRoads();
144
148         void initialize();
149
153         void join();
154
158         void rebind();
159 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Point.cpp File Reference

```
#include "Point.h"
#include "Graph.h"
#include <vector>
#include <set>
```

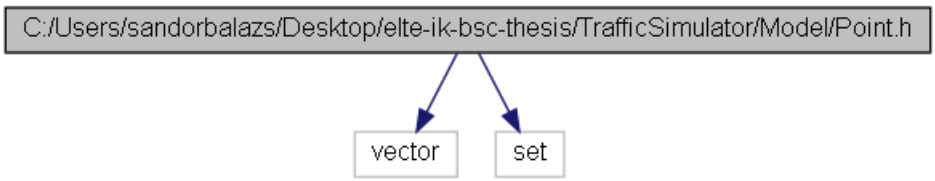
Include dependency graph for Point.cpp:



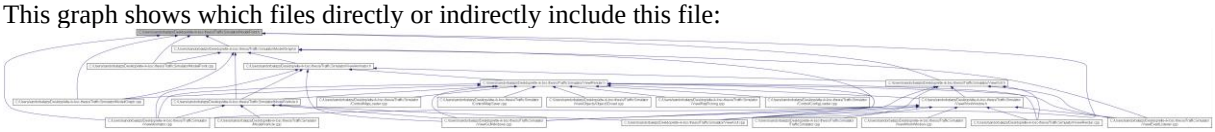
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Point.h File Reference

```
#include <vector>
#include <set>
```

Include dependency graph for Point.h:



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



Data Structures

- class **Point**
  The abstract point implementation.

Point.h

Go to the documentation of this file.1

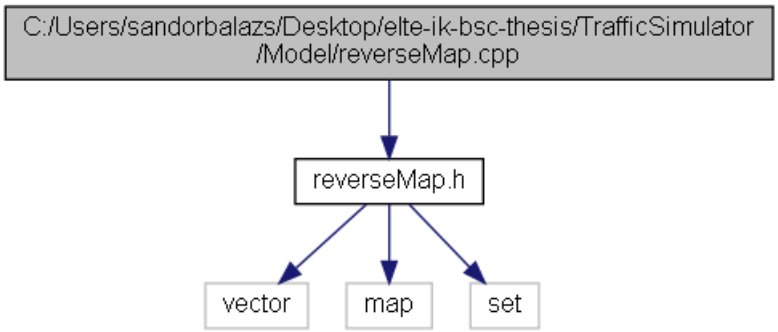
```
10 #pragma once
11
12 class Graph;
13
14 #include <vector>
15 #include <set>
16
20 class Point {
21 public:
22
30         Point(size_t id, size_t edge, Graph* myGraph, char side);
31
32         /*
33         * @brief Point destructor. (Important for delete)
34         */
35         ~Point(void);
36
41         void join(Point* point);
42
47         void reID(size_t id);
48
52         size_t getID();
53
57         size_t getEdge();
58
62         char getSide();
63
67         std::set<size_t> getEdges();
68
72         void erase();
73
78         bool isErased();
79
83         void setAsStartPoint();
84
88         void setAsEndpoint();
89
94         bool isStartPoint();
95
100         bool isEndPoint();
101
106         size_t getRenderID();
107
111         void setRenderID(size_t id);
112
116         void resetStats();
117
121         std::vector<size_t> startConfiguration;
122
126         std::vector<bool> endpointsList;
127
131         const int baseStartableVehicleCount = 50;
132
136         int startableVehicles = baseStartableVehicleCount;
137
141         int receivedVehicles = 0;
142
147         std::vector<size_t> activeVehicles();
148
153         std::vector<size t> activeEndpoints();
154
155 private:
156
160         size_t id = -1;
161
165         size t edge = -1;
166
170         size_t renderID = -1;
171
175         char side = 'Q';
176
180         bool erased = false;
```

```
181
185         bool startPoint = false;
186
190         bool endPoint = false;
191
195         Graph* myGraph;
196
200         std::set<size_t> connectedEdges;
201 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/reverseMap.cpp File Reference

```
#include "reverseMap.h"
```

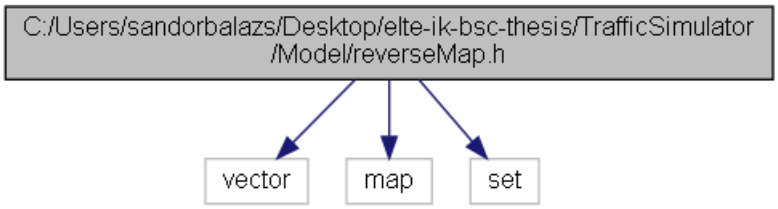
Include dependency graph for reverseMap.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/reverseMap.h File Reference

```
#include <vector>
#include <map>
#include <set>
```

Include dependency graph for reverseMap.h:



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

Data Structures

- class **reverseMap**
  *The reverse map implementation for the **Dijkstra** algorithm.*

reverseMap.h

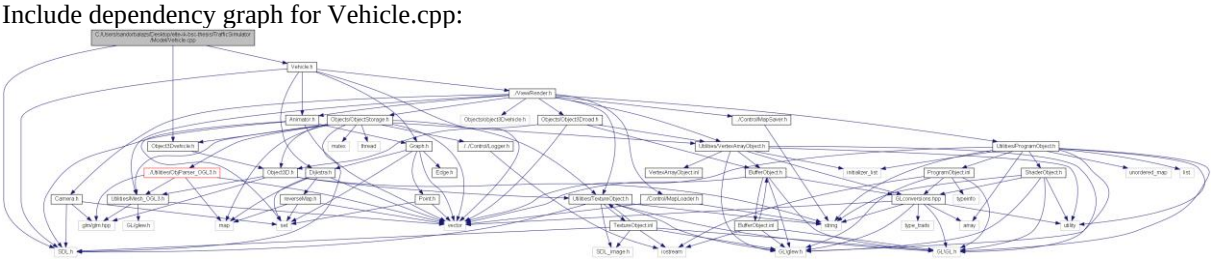
Go to the documentation of this file.1

```
11 #pragma once
12
13 #include <vector>
14 #include <map>
15 #include <set>
16
20 class reverseMap {
21 public:
22
26     reverseMap() {}
27
31     ~reverseMap() {}
32
38     int getValueByNode(size_t node);
39
45     void setValueByNode(size_t node, int newValue);
46
51     std::pair<size_t, int> removeMinimumValueElement();
52
58     void insert(size_t node, int value);
59
64     bool empty();
65
71     bool isIn(size_t node);
72
76     std::map<size_t, int> nodeIndexedStore;
77
81     std::multimap<int, size_t> valueIndexedStore;
82
83 private:
84
91     void changeElementIndex(const size_t oldIndex, const size_t newIndex, const int
element);
92
98     void safeErase(size_t node, int value);
99 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Vehicle.cpp File Reference

```
#include <SDL.h>
#include "Vehicle.h"
#include "../View/Objects/Object3Dvehicle.h"
```

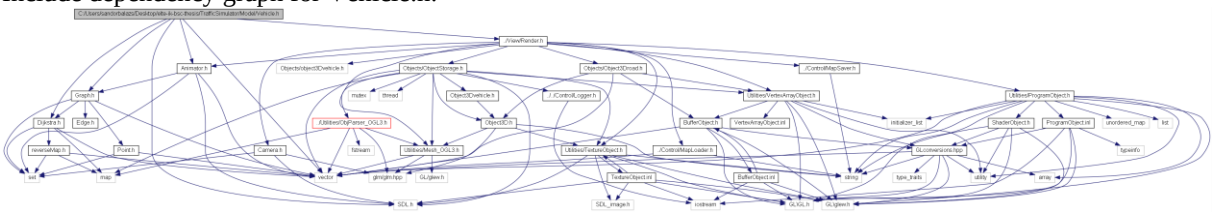
Include dependency graph for Vehicle.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/Model/Vehicle.h File Reference

```
#include "Graph.h"
#include "Dijkstra.h"
#include "../View/Render.h"
#include "../View/Animator.h"
#include <vector>
#include <SDL.h>
```

Include dependency graph for Vehicle.h:



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

Data Structures

- class **Vehicle**
  Vehicle *model representation.*

Vehicle.h

Go to the documentation of this file.1

```
11 //NEVER FADE AWAY!
12
13 #pragma once
14
15 #include "Graph.h"
16 #include "Dijkstra.h"
17 #include "../View/Render.h"
18 #include "../View/Animator.h"
19 #include <vector>
20 #include <SDL.h>
21
25 class Vehicle {
26 public:
27
37         Vehicle(Graph* graph, Render* render, size_t startID, size_t destinationID,
size_t object3DiD, size_t ID);
38
43         size_t getID();
44
49         size_t getObject3DiD();
50
54         void update();
55
60         bool isFinished();
61
65         void erase();
66
71         bool isDeleted();
72
77         bool isBlocked();
78
82         void unblock();
83
87         static bool repath;
88
92         static bool collisionPrevention;
93
97         static int vehicleWeight;
98
102         static int collisionCheckDistance;
103
107         size_t destinationID = 0;
108
112         size_t startID = 0;
113
117         Uint32 startTime = 0;
118
122         Uint32 endTime = 0;
123
127         size_t hopCounter = 0;
128
132         int allCost = 0;
133
134 private:
135
139         size_t object3DiD = 0;
140
144         size_t ID = 0;
145
149         size_t currentEdgeOnThePath = 0;
150
154         size_t currentRoad = 0;
155
159         size_t currentPointOnTheRoad = 0;
160
164         char track = '1';
165
169         char direction = 'a';
170
174         const size_t standardRoadLenght = 100;
175
179         bool finished = false;
```

```
180
184         bool deleted = false;
185
189         bool nextRoadIsBlocked = false;
190
194         Graph* graph;
195
199         Render* render;
200
204         Dijkstra* dijkstra;
205
209         Animator* animator;
210
214         std::vector<size_t> path;
215
219         void nextStep();
220
224         char directionCheck();
225
229         void checkFinish();
230
234         void switchToNextRoad();
235
239         void firstDirectionCheck();
240
244         bool collisionTest(int checkDistance, size_t localCurrentRoad, size_t
localCurrentPointOnTheRoad, char direction, char track);
245 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/README.md File Reference

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/resource.h File Reference

Macros

- #define **IDI_ICON1** 101

Macro Definition Documentation

#define IDI_ICON1 101

resource.h

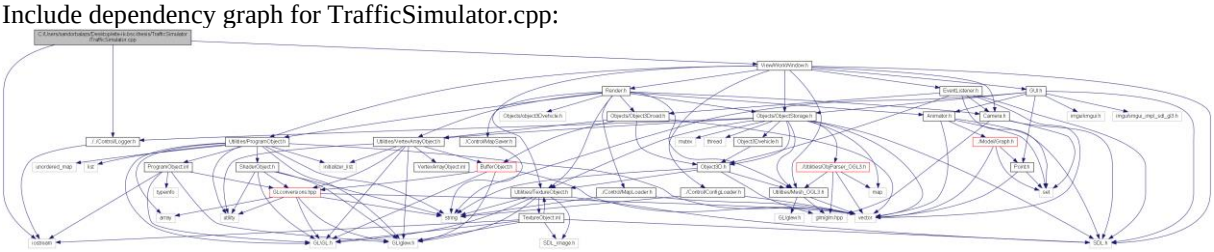
Go to the documentation of this file.

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by TrafficSimulator.rc
//
#define IDI_ICON1                       101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/TrafficSimulator.cpp File Reference

```
#include <iostream>
#include "View/WorkWindow.h"
#include "Control/Logger.h"
```

Include dependency graph for TrafficSimulator.cpp:



Functions

- Traffic Simulationint **main** (int argc, char *args[])
  Program main function.

Function Documentation

int main (int *argc*, char * *args*[])

Program main function.

Possible error codes: 0 - Everything is fine. 1 - SDL Initialization error. 2 - OpenGL 3D Window creation error. 3 - imGUI **GUI** system initialization error. 4 - OpenGL Context creation error. 5 - GLEW system initialization error. 6 - OpenGL starting or version error. 7 - Shader compiling or starting error.

Returns

The starting process status number. 0 is OK Other ID-s mean errors.

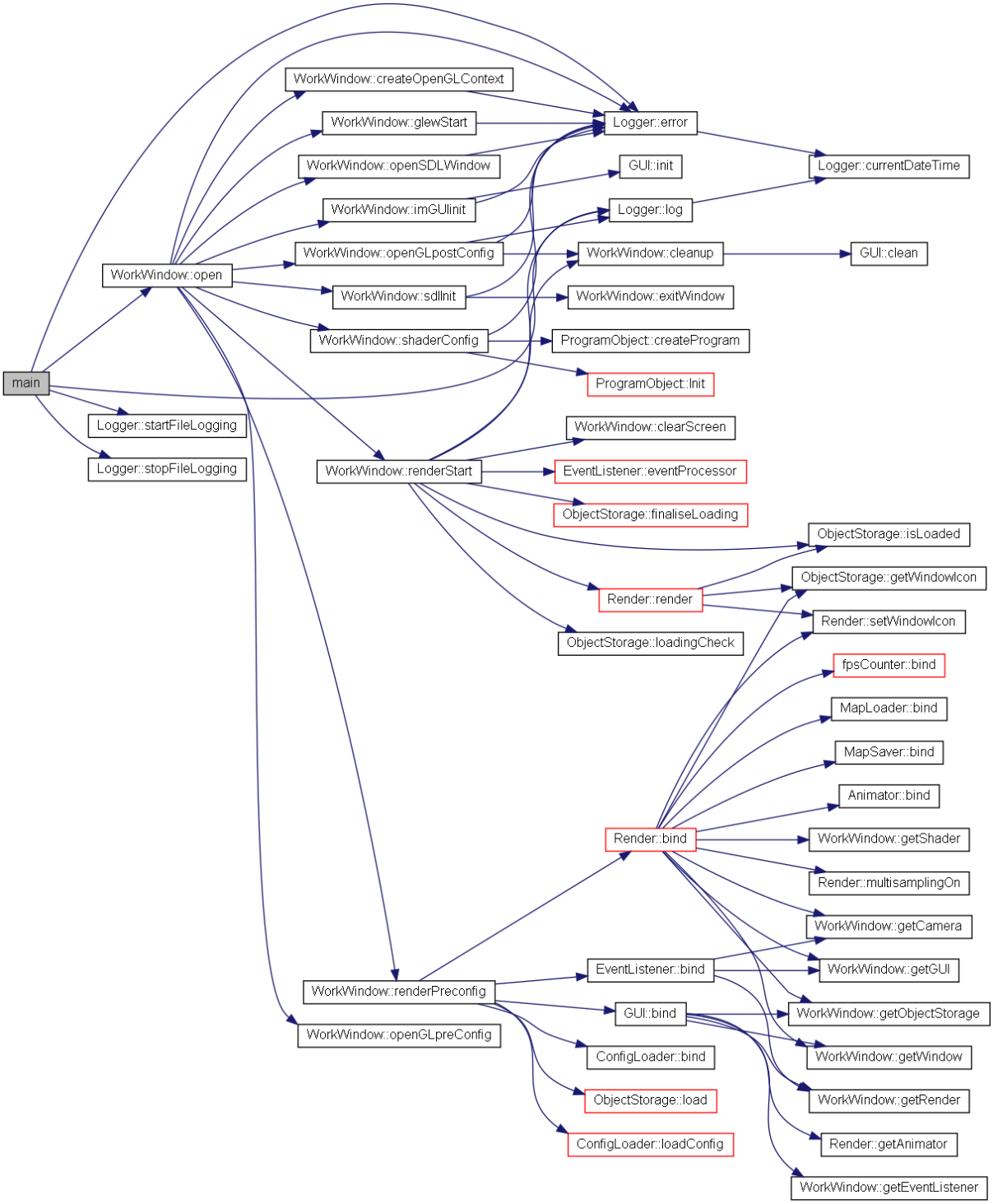
Parameters

argc	Console Argument count.
args	Console arguments array.

Returns

program exit status.

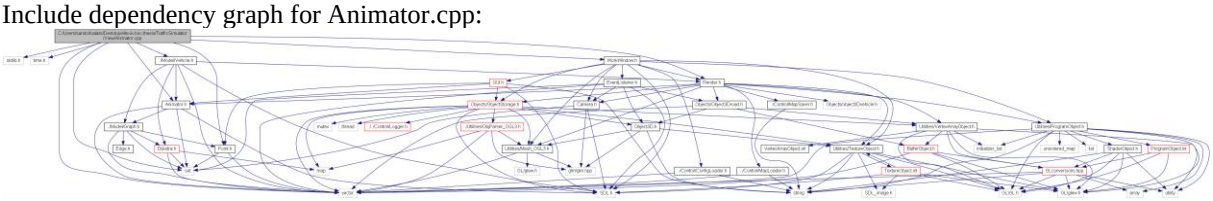
Here is the call graph for this function:


main
Logger::startFileLogging
Logger::stopFileLogging
WorkWindow::open
WorkWindow::createOpenGLContext
WorkWindow::glewStart
WorkWindow::openSDLWindow
WorkWindow::imGUIinit
WorkWindow::openGLpostConfig
WorkWindow::sdlInit
WorkWindow::shaderConfig
WorkWindow::renderStart
WorkWindow::renderPreconfig
WorkWindow::openGLpreConfig
Logger::error
GUI::init
Logger::log
WorkWindow::cleanup
WorkWindow::exitWindow
ProgramObject::createProgram
ProgramObject::Init
Logger::currentDateTime
GUI::clean
WorkWindow::clearScreen
EventListener::eventProcessor
ObjectStorage::finaliseLoading
ObjectStorage::isLoaded
ObjectStorage::getWindowIcon
Render::render
Render::setWindowIcon
ObjectStorage::loadingCheck
fpsCounter::bind
MapLoader::bind
MapSaver::bind
Animator::bind
Render::bind
WorkWindow::getShader
Render::multisamplingOn
WorkWindow::getCamera
EventListener::bind
WorkWindow::getGUI
GUI::bind
WorkWindow::getObjectStorage
ConfigLoader::bind
WorkWindow::getWindow
ObjectStorage::load
WorkWindow::getRender
ConfigLoader::loadConfig
Render::getAnimator
WorkWindow::getEventListener

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Animator.cpp File Reference

```
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <set>
#include "Animator.h"
#include "Render.h"
#include "WorkWindow.h"
#include "../Model/Vehicle.h"
#include "../Model/Point.h"
```

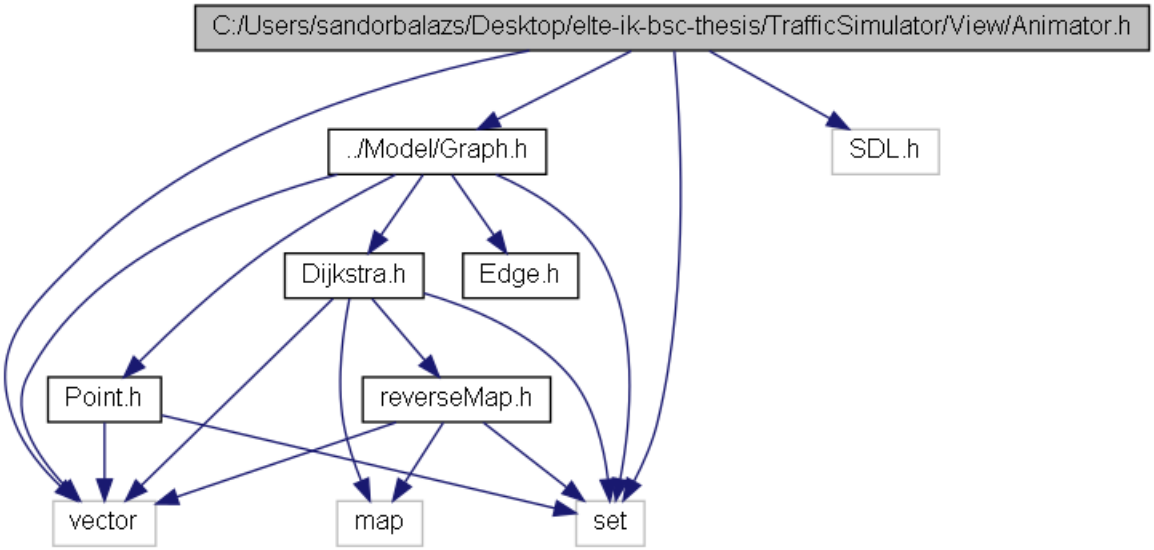
Include dependency graph for Animator.cpp:



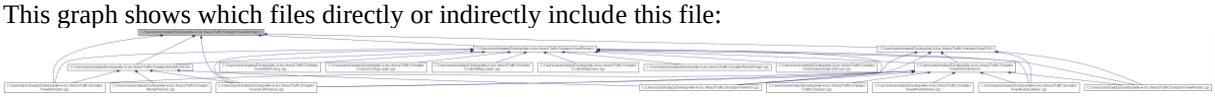
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Animator.h File Reference

```
#include <vector>
#include <set>
#include <SDL.h>
#include "../Model/Graph.h"
```

Include dependency graph for Animator.h:



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



Data Structures

- class **Animator**
  The animation controller class.

Animator.h

Go to the documentation of this file.1

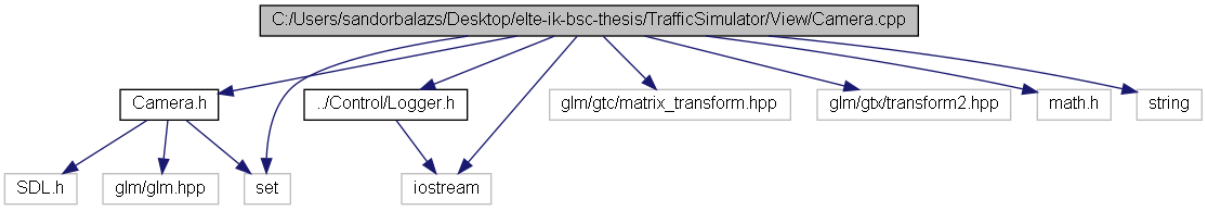
```
11 //Spirit In The Sky
12
13 #pragma once
14
15 //Empty classes for avoiding the circular includes.
16 class Render;
17 class Vehicle;
18
19 #include<vector>
20 #include<set>
21 #include<SDL.h>
22 #include "../Model/Graph.h"
23
27 class Animator {
28 public:
29
34         Animator(void);
35
40         ~Animator(void);
41
46         void bind(Render* render);
47
51         void update();
52
56         void finalize();
57
61         void start();
62
66         void pause();
67
71         void stop();
72
78         void addVehicle(size_t startPoint, bool singleVehicle);
79
85         Vehicle* getVehicleModel(size_t modelID);
86
91         size_t getVehicleModelSize();
92
97         Graph* getGraph();
98
102         std::vector<size_t> getStartPoints();
103
107         std::vector<size_t> getEndPoints();
108
112         std::map<size_t,std::set<size_t>> separatedEndPoints;
113
117         void clear();
118
122         static int updateFrequency;
123
127         static int vehicleStarterUpdateFrequency;
128
129 private:
130
134         Render* render = NULL;
135
139         Graph* graph = NULL;
140
144         Uint32 currentTime = 0;
145
149         Uint32 prevouseTime = 0;
150
154         Uint32 vehicleStarterCurrentTime = 0;
155
159         Uint32 vehicleStarterPrevouseTime = 0;
160
164         bool isAnimationRunning = false;
165
169         bool softRunning = false;
170
174         std::vector<Vehicle> vehicles;
175
```

```
179         std::vector<size_t> startPoints;
180
184         std::vector<size_t> endPoints;
185
189         std::set<size_t> stuckVehicles;
190
194         size_t startedVehiclesIndex = 0;
195
200         bool animationTimer();
201
206         bool vehicleStarterTimer();
207
211         void deleteFinishedVehicles();
212
216         void autoAdder();
217
224         int getRandomNumber(int min, int max);
225
229         void resetStats();
230
234         void updateStuckVehicleList();
235
240         bool startCollisionDetection();
241
245         void checkSeparatedEndpoints();
246 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Camera.cpp File Reference

```
#include "Camera.h"
#include "../Control/Logger.h"
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform2.hpp>
#include <set>
#include <math.h>
#include <iostream>
#include <string>
```

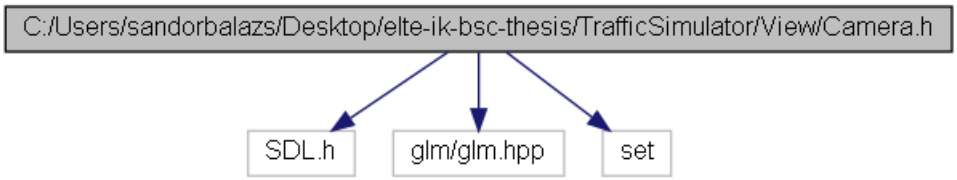
Include dependency graph for Camera.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Camera.h File Reference

```
#include <SDL.h>
#include <glm/glm.hpp>
#include <set>
```

Include dependency graph for Camera.h:



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

Data Structures

- class **Camera**
  The own OpenGL camera class.

Camera.h

Go to the documentation of this file.1

```
11 #pragma once
12 #include <SDL.h>
13 #include <glm/glm.hpp>
14 #include <set>
15
20 class Camera {
21 public:
26         Camera(void);
27
32         ~Camera(void);
33
40         void setView(glm::vec3 cameraPosition, glm::vec3 lookedPoint);
41
46         void setCameraPosition(glm::vec3 cameraPosition);
47
52         void setLookedPoint(glm::vec3 lookedPoint);
53
58         glm::vec3 getCameraPosition() {
59                 return cameraPosition;
60         }
61
66         glm::vec3 getLookedPoint() {
67                 return lookedPoint;
68         }
69
74         glm::vec3 getUpwardVector() {
75                 return upwardDirection;
76         }
77
90         float getAzimuth() {
91                 return sphericalCameraPosition.z;
92         }
93
98         glm::mat4 getViewMatrix() {
99                 return viewMatrix;
100         }
101
106         glm::mat4 getProjectionMatrix() {
107                 return projectionMatrix;
108         }
109
114         glm::mat4 getProjectionViewMatrix() {
115                 return projectionMatrix * viewMatrix;
116         }
117
123         void resize(int width, int height);
124
131         void keyboardDown(SDL_KeyboardEvent& key);
132
138         void keyboardUp(SDL_KeyboardEvent& key);
139
144         void mouseMove(SDL_MouseMotionEvent& mouse);
145
150         void mouseWheel(SDL_MouseWheelEvent& wheel);
151
152 protected:
156         const glm::vec3 defaultCameraPosition = glm::vec3(45.0f, 45.0f, 45.0f);
157
161         const glm::vec3 defaultLookedPoint = glm::vec3(0.0f);
162
166         const glm::vec3 defaultUpwardVector = glm::vec3(0.0f, 10.0f, 0.0f);
167
171         float lookedPointMovementSpeed = 1.0f;
172
176         const float angle = 45.0f;
177
181         float aspect = 640 / 480.0f;
182
186         const float zNear = 0.8f;
187
191         const float zFar = 350.0f;
192
```

```
196         void update();
197
208         void cameraCoordinateUpdate();
209
219         void sphericalCoordinateUpdate();
220
227         void sphericalCoordinateShift(float radius, float azimuth, float
inclination);
228
236         void setProjection(float angle, float aspect, float zNear, float zFar);
237
244         glm::vec3       sphericalCameraPosition;
245
249         glm::mat4       viewMatrix;
250
254         glm::vec3       cameraPosition;
255
259         glm::vec3       upwardDirection;
260
264         glm::vec3       lookedPoint;
265
269         glm::mat4       projectionMatrix;
270
274         std::set<int> pressedKeys;
275 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/EventListener.cpp File Reference

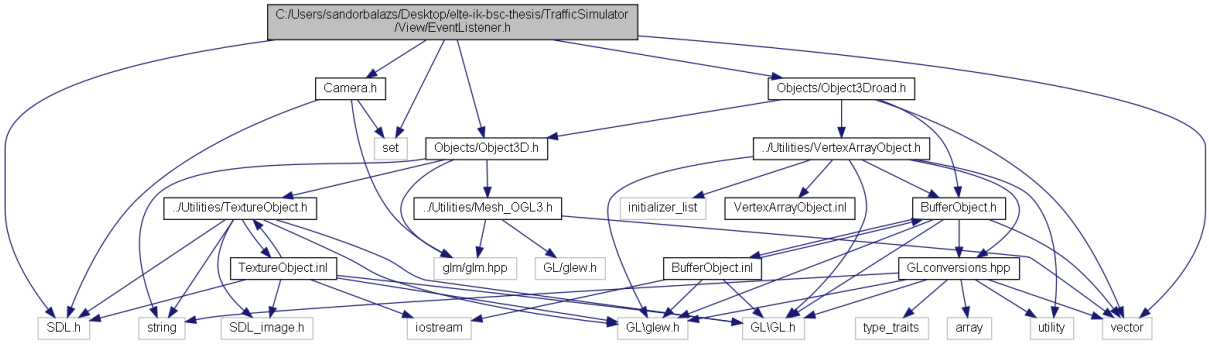
```
#include "EventListener.h"
#include "WorkWindow.h"
#include "Render.h"
#include "../Model/Graph.h"
#include "GUI.h"
#include <glm/gtx/transform2.hpp>
#include <iostream>
```

Include dependency graph for EventListener.cpp:

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/EventListener.h File Reference

```
#include <SDL.h>
#include <vector>
#include <set>
#include "Camera.h"
#include "Objects/Object3D.h"
#include "Objects/Object3Droad.h"
```

Include dependency graph for EventListener.h:



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

Data Structures

- class **EventListener**
  Event listener implementation. Contact: `sandorbalazs9402@gmail.com`.

EventListener.h

Go to the documentation of this file.1

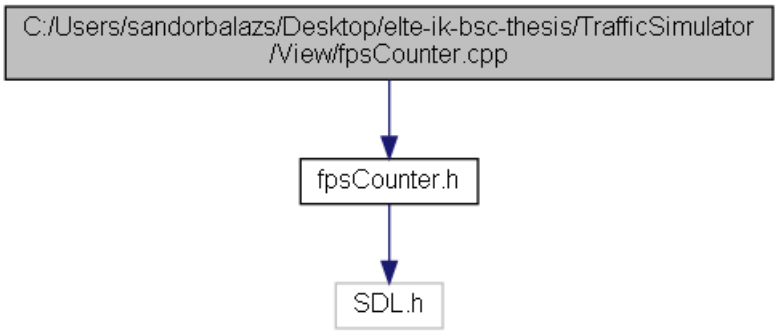
```
#pragma once

//Empty classes for avoiding circular includes.
class WorkWindow;
class GUI;
class Render;

#include<SDL.h>
#include<vector>
#include<set>
#include"Camera.h"
#include"Objects/Object3D.h"
#include"Objects/Object3Droad.h"

class EventListener {
public:
        void bind(WorkWindow* currentWindow);

        void eventProcessor();

        void select(size_t objectID);

        void deselect();

        void roadSelect();

        void roadDeselect();

        void vehicleSelect(size_t vehicleID);

        void vehicleDeselect();

        void lockEditor();

        void freeEditor();

        void deleteSelectedItems();

private:

        WorkWindow* workingWindow;

        Camera* camera;

         GUI* gui;

         Render* render;

         std::set<int> pressedKeys;

         std::set<int> moseButtonPressed;

         std::vector<size_t> selectedItems;

         std::vector<size_t> selectedRoads;

         size_t selectedVehicle = -1;

         bool keepSelect = false;

         bool editorLock = false;

         glm::vec3 ray;

         void keyboardDown(SDL_KeyboardEvent&);

         void keyboardUp(SDL_KeyboardEvent&);

         void mouseMove(SDL_MouseMotionEvent&);

         void mouseDown(SDL_MouseButtonEvent&);

```

```
175         void mouseUp(SDL_MouseButtonEvent&);
176
181         void mouseWheel(SDL_MouseWheelEvent&);
182
187         void resize(SDL_WindowEvent& window);
188
192         void exit();
193
209         int getClickedObjectId(SDL_MouseButtonEvent& mouse, bool vehicleCheck =
false);
210
219         float intersection(glm::vec3 ray, glm::vec3 cameraPosition, glm::vec3
hitSphereCenter, float hitSphereRadius);
220 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/fpsCounter.cpp File Reference

```
#include "fpsCounter.h"
```

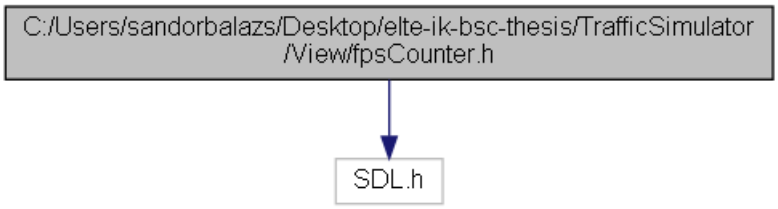
Include dependency graph for fpsCounter.cpp:



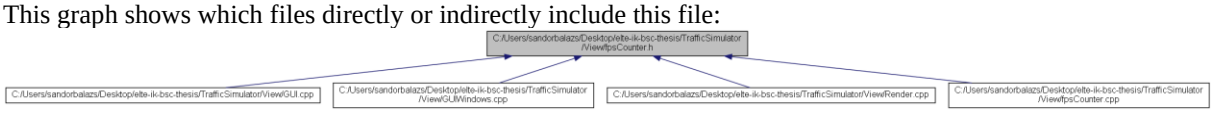
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/fpsCounter.h File Reference

`#include <SDL.h>`

Include dependency graph for fpsCounter.h:



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



Data Structures

- class **fpsCounter**
  FPS counter, logger and limiter static class.

fpsCounter.h

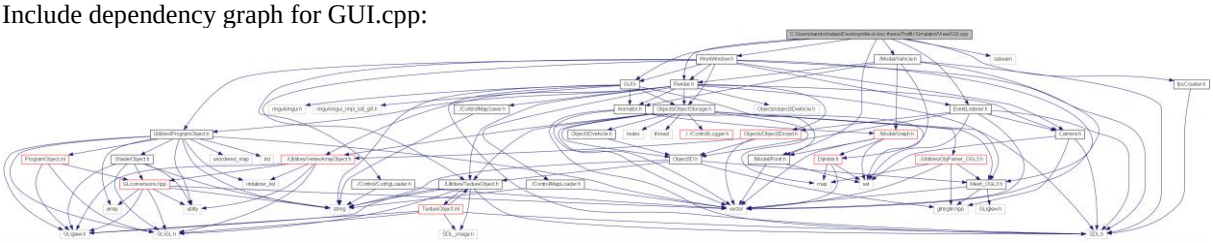
Go to the documentation of this file.1

```
11 #pragma once
12
13 #include <SDL.h>
14
18 class fpsCounter {
19 public:
20
24         static void bind();
25
29         static void start();
30
34         static void stop();
35
39         static void fpsLimiterOn();
40
44         static void fpsLimiterOff();
45
50         static void setFpsLimit(float limit);
51
56         static float getCurrentFPS();
57
62         static float getAverageFPS();
63
67         static float fpsLog[100];
68
69 private:
70
74         fpsCounter();
75
79         ~fpsCounter();
80
84         static const int averageFPSupdateTime = 500;
85
89         static int fpsLogCounter;
90
94         static float currentFPS;
95
99         static float averageFPS;
100
104         static float fpsLimit;
105
109         static bool isLimiterOn;
110
114         static Uint32 limiterCurrentTime;
115
119         static Uint32 currentTime;
120
124         static Uint32 previousTime;
125
129         static SDL_atomic_t atomicFrameCounter;
130
135         static void fpsLimiter(float limit);
136
143         static Uint32 fpsTimerCallback(Uint32 interval, void* data);
144 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/GUI.cpp File Reference

```
#include "GUI.h"
#include "WorkWindow.h"
#include "Render.h"
#include "EventListener.h"
#include "fpsCounter.h"
#include "../Model/Point.h"
#include "../Model/Vehicle.h"
#include <sstream>
```

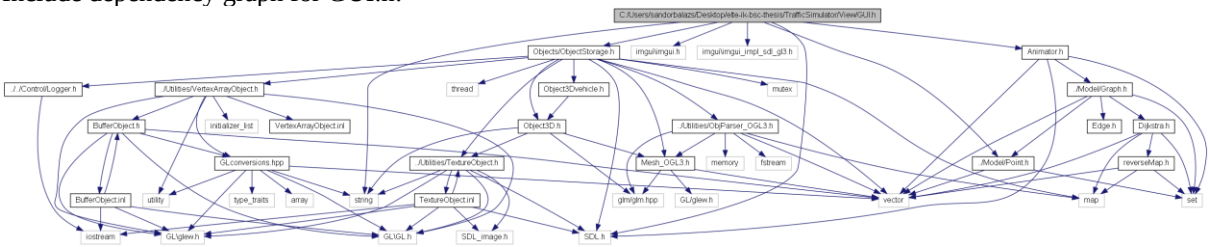
Include dependency graph for GUI.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/GUI.h File Reference

```
#include <SDL.h>
#include <string>
#include "imgui\imgui.h"
#include "imgui\imgui_impl_sdl_gl3.h"
#include "Objects/ObjectStorage.h"
#include "../Model/Point.h"
#include "Animator.h"
```

Include dependency graph for GUI.h:



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

Data Structures

- class **GUI**
  ***GUI** Descriptor class. Contact: `sandorbalazs9402@gmail.com`.*

GUI.h

Go to the documentation of this file.1

```
11 // Flanker-C
12
13 #pragma once
14 class WorkWindow;
15 class Render;
16 class EventListener;
17
18 #include <SDL.h>
19 #include <string>
20 #include "imgui\imgui.h"
21 #include "imgui\imgui_impl_sdl_gl3.h"
22 #include "Objects/ObjectStorage.h"
23 #include "../Model/Point.h"
24 #include "Animator.h"
25
26 class GUI {
27 public:
28
32         GUI(void);
33
37         ~GUI(void);
38
44         static bool init(SDL_Window* window);
45
49         static void clean();
50
55         void bind(WorkWindow* currentWindow);
56
60         void draw();
61
65         void render();
66
71         void eventHandler(SDL_Event* event);
72
77         bool isKeyboardCaptured();
78
83         bool isMouseCaptured();
84
88         void close();
89
94         void showEndpointInfo(size_t pointModelID);
95
100         void showRoadInfo(size t edgeModelID);
101
106         void showVehicleInfo(size_t vehicleModelID);
107
111         void resetInfoWindow();
112
116         void quickSave();
117
118 private:
119
120         //Window configurators
121         ImGuiWindowFlags trafficSimulationWindowFlag = 0;
122         ImGuiWindowFlags mapEditorWindowFlag = 0;
123         ImGuiWindowFlags pathFinderTestWindowFlag = 0;
124         ImGuiWindowFlags simulationWindowFlag = 0;
125         ImGuiWindowFlags graphicSettingsWindowFlag = 0;
126
127         //Window open flags.
128         bool openWindowStatus = false;
129         bool saveWindowStatus = false;
130         bool saveAsWindowStatus = false;
131         bool ImGuiSettingsWindowStatus = false;
132         bool newMapConfirmWindowStatus = false;
133         bool controlsWindowStatus = false;
134         bool aboutWindowStatus = false;
135         bool closingCheckerWindowStatus = false;
136         bool graphicSettingsWindowStatus = false;
137         bool debugOptionsWindowStatus = false;
138         bool runningStatisticsWindowStatus = false;
139         bool pathFinderTestWindowStatus = false;
```

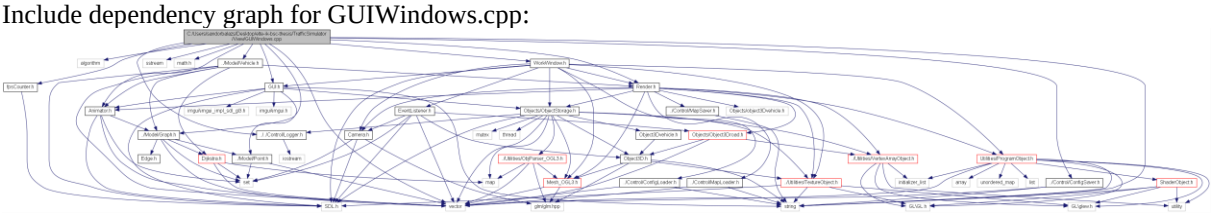
```
140         bool mapEditorWindow = true;
141         bool simulationWindow = false;
142         bool simulationSettingsWindowStatus = false;
143         bool simulationStatisticsWindowStatus = false;
144         bool helpWindowStatus = false;
145         bool finaliseWarningWindowStatus = false;
146
147         //Selection handlers
148         size_t selectedStartPoint = -1;
149         size_t selectedEndPoint = -1;
150         size_t selectedRoad = -1;
151         size_t selectedVehicle = -1;
152         Uint32 pathFinderOpenTimeStamp;
153
154         //Mode flags
155         bool finalisingErrorWindow = false;
156         const bool debugMode = false;
157
161         bool simulationStart = false;
162
166         bool simulationPaused = false;
167
171         bool editorLock = false;
172
176         size_t coloringCounter = 0;
177
181         std::vector<size_t> path;
182
186         WorkWindow* workingWindow;
187
191         SDL_Window* window;
192
196         Render* windowRender;
197
201         ObjectStorage* objectStorage;
202
206         EventListener* eventListener;
207
211         Animator* animator;
212
216         void mainMenuBar();
217
221         void windowHandler();
222
226         void fpsGraph();
227
231         void openWindow();
232
236         void saveWindow();
237
241         void saveAsWindow();
242
246         void newMapConfirmWindow();
247
251         void itemList();
252
256         void carList(Point* point);
257
261         void controlsWindow();
262
266         void aboutWindow();
267
271         void closeWindow();
272
276         void graphicSettingsWindow();
277
281         void debugOptionsWindow();
282
286         void runningStatisticsWindow();
287
291         void pathFinderTestWindow();
292
297         void showHelpMarker(const char* desc);
298
303         void endpointSelector(Point* point);
304
308         void simulationSettingsWindow();
```

```
309
313         void finalise();
314
318         void backToEditMode();
319
323         void startSimulation();
324
328         void stopSimulation();
329
333         void pauseSimulation();
334
338         void simulationStatisticsWindow();
339
343         void clearRoadColor();
344
348         void closeAllWindows(std::string exeption);
349
353         void helpWindow();
354
358         void finaliseWarningWindow();
359 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/GUIWindows.cpp File Reference

```
#include <algorithm>
#include <vector>
#include <string>
#include <sstream>
#include <math.h>
#include "GUI.h"
#include "fpsCounter.h"
#include "Render.h"
#include "WorkWindow.h"
#include "../Control/Logger.h"
#include "../Control/ConfigSaver.h"
#include "../Model/Graph.h"
#include "../Model/Vehicle.h"
#include "Animator.h"
```

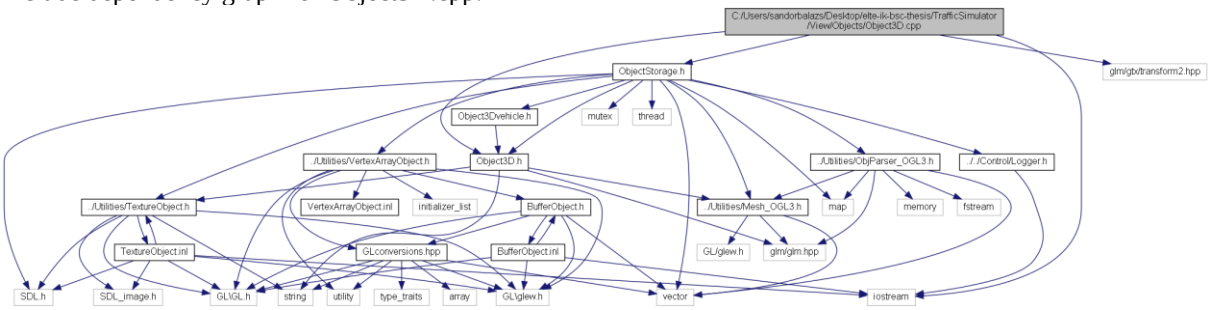
Include dependency graph for GUIWindows.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3D.cpp File Reference

```
#include "Object3D.h"
#include "ObjectStorage.h"
#include <glm/gtx/transform2.hpp>
#include <iostream>
```

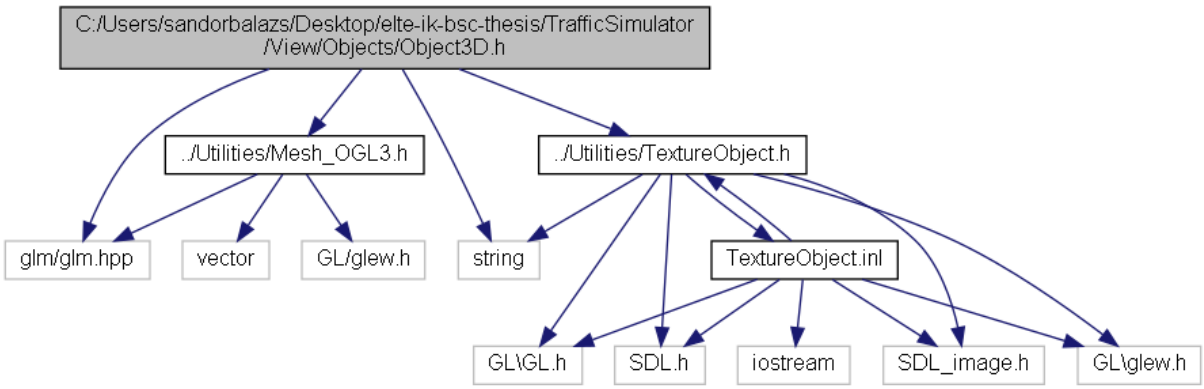
Include dependency graph for Object3D.cpp:



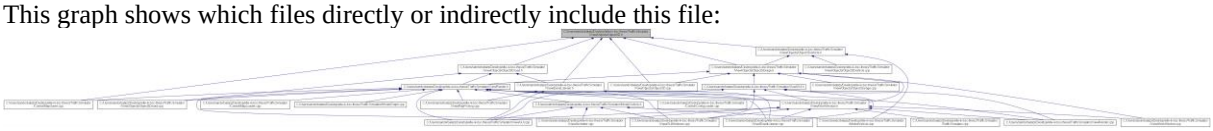
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3D.h File Reference

```
#include <glm/glm.hpp>
#include <string>
#include "../Utilities/TextureObject.h"
#include "../Utilities/Mesh_OGL3.h"
```

Include dependency graph for Object3D.h:



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



Data Structures

- class **Object3D**
  The 3D renderable object representation.

Object3D.h

Go to the documentation of this file.1

```
#pragma once

#include <glm/glm.hpp>
#include <string>
#include "../Utilities/TextureObject.h"
#include "../Utilities/Mesh_OGL3.h"

class ObjectStorage;

class Object3D {
        public:

                Object3D(ObjectStorage* objectStorage);

                Object3D(int id, std::string name, std::string type, std::string meshID, std::string textureID, std::string iconID,
                        glm::vec3 initPosition, glm::vec3 initScale, glm::vec4 initRotation,
                        glm::vec4 color, glm::vec4      hitSphere, glm::vec4 moveSphere, ObjectStorage* objectStorage);

                ~Object3D(void);

                void                                    setDependencyID(int dependencyID);

                void                                    setMeshID(std::string meshID);

                void                                    setTextureID(std::string textureID);

                void                                    setIconID(std::string iconID);

                void                                    setPosition(glm::vec3 position);

                void                                    setRotation(glm::vec4 rotation);

                void                                    setScale(glm::vec3 scale);

                void                                    move(glm::vec3 move);

                void                                    rotateLeft(float rotateLeft);

                void                                    rotateRight(float rotateRight);

                void                                    setRGBAcolor(glm::vec4 RGBAcolor);

                void                                    setRGBcolor(glm::vec3 RGBcolor);

                void                                    setOpacity(float opacity);

                void                                    setRenderID(int newRenderID);

                void                                    setProtection(bool isProtected);

                void                                    setSelectable(bool isSelectable);

                int                                     getId();

                int                                     getRenderID();

```

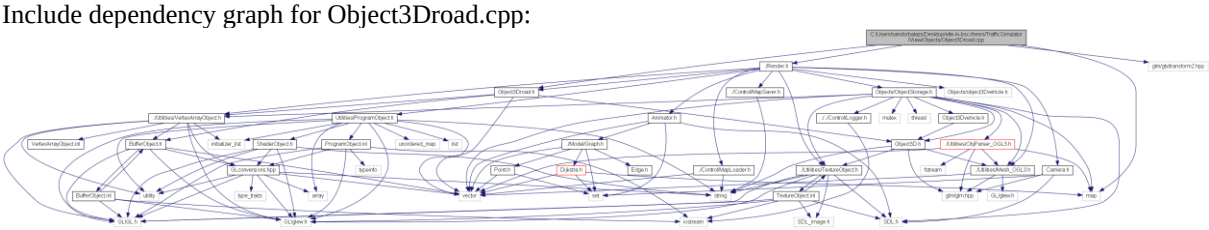
```
173                 int                                         getDependencyID();
174
179                 std::string                             getName();
180
185                 std::string                             getType();
186
191                 std::unique_ptr<Mesh>&  getMesh();
192
197                 Texture2D&                              getTexture();
198
203                 std::string                             getTextureID();
204
209                 Texture2D&                              getIcon();
210
215                 std::string                             getIconID();
216
221                 glm::vec3                               getPosition();
222
227                 glm::vec4                               getRotation();
228
233                 float                                   getRotationInDegree();
234
239                 float                                   getRotationInRadian();
240
245                 glm::vec3                               getRotationVector();
246
251                 glm::vec3                               getScale();
252
257                 glm::vec4                               getRGBAcolor();
258
263                 glm::vec3                               getRGBcolor();
264
269                 float                                   getOpacity();
270
275                 glm::mat4                               getWorldMatrix();
276
281                 glm::vec4                               getHitSphere();
282
287                 glm::vec4                               getMoveSphere();
288
293                 bool                                    isProtected();
294
299                 bool                                    isSelectable();
300
305                 Object3D                                copy();
306
312                 Object3D                                copy(int renderID);
313
318                 bool                                    isHidden();
319
323                 void                                    hide();
324
328                 void                                    show();
329
333                 void                                    erase();
334
339                 bool                                    isDeleted();
340
345                 size_t                                  getModelID();
346
351                 void                                    setModelID(size t id);
352
356                 void                                    select();
357
361                 void                                    deSelect();
362
367                 bool                                    isSelected();
368
369         protected:
370
374                 ObjectStorage*                  objectStorage;
375
379                 int                                     id = -1;
380
384                 int                                     renderID = -1;
385
389                 int                                         dependencyID = -1;
```

```
390
394                 std::string                             name = "";
395
399                 std::string                             type = "";
400
404                 std::string                             meshID = "";
405
409                 std::string                             textureID = "white.png";
410
414                 std::string                             iconID =
"sphere_mark_mini.png";
415
419                 glm::vec4                               color = glm::vec4(1.0f,
1.0f, 1.0f, 1.0f);
420
424                 glm::vec3                               defaultPosition =
glm::vec3(0.0f, 0.0f, 0.0f);
425
429                 glm::vec3                               position = glm::vec3(0.0f,
0.0f, 0.0f);
430
434                 glm::vec3                               scale = glm::vec3(1.0f,
1.0f, 1.0f);
435
439                 glm::vec4                               rotation = glm::vec4(0.0f,
0.0f, 1.0f, 0.0f);
440
444                 glm::vec4                               worldMatrix =
glm::vec4(1.0f);
445
449                 glm::vec4                               hitSphere = glm::vec4(0.0f,
1.0f, 0.0f, 1.0f);
450
454                 glm::vec4                               moveSphere =
glm::vec4(0.0f, 0.0f, 0.0f, 1.0f);
455
459                 bool                                    hidden = false;
460
464                 bool                                    protection = false;
465
469                 bool                                    selectable = true;
470
474                 bool                                    deleted = false;
475
479                 size_t                                  modelID = -1;
480
484                 bool                                    selected = true;
485         };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Droad.cpp File Reference

```
#include "Object3Droad.h"
#include "../Render.h"
#include <glm/gtx/transform2.hpp>
#include <map>
```

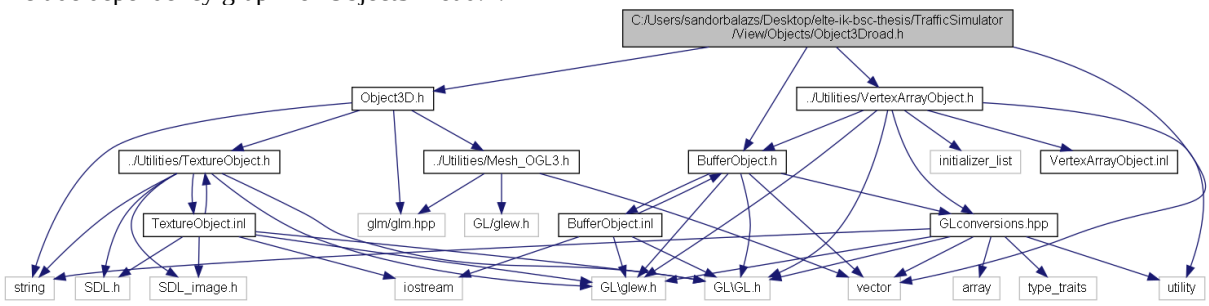
Include dependency graph for Object3Droad.cpp:



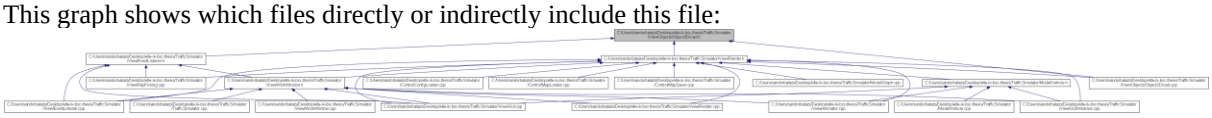
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Droad.h File Reference

```
#include "Object3D.h"
#include "../Utilities/VertexArrayObject.h"
#include "../Utilities/BufferObject.h"
#include <vector>
```

Include dependency graph for Object3Droad.h:



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



Data Structures

- class **Object3Droad**
  The parametric generated road implementation.

Object3Droad.h

Go to the documentation of this file.1

```
11 //T'ana
12
13 #pragma once
14
15 //Empty lass for avoiding the circular includes.
16 class Render;
17
18 #include "Object3D.h"
19 #include "../Utilities/VertexArrayObject.h"
20 #include "../Utilities/BufferObject.h"
21 #include <vector>
22
26 class Object3Droad {
27 public:
28
32         Object3Droad();
33
38         Object3Droad(const Object3Droad&);
39
44         ~Object3Droad(void);
45
57         void bind(Render* render, size_t newRenderID, size_t trackBall_0, size_t
trackBall_1, size_t trackBall_2, size_t trackBall_3, size_t roadEndCircle_1, size_t
roadEndCircle_2);
58
68         void reBind(size_t trackBall_0, size_t trackBall_1, size_t trackBall_2, size_t
trackBall_3, size_t roadEndCircle_1, size_t roadEndCircle_2);
69
73         void generate();
74
78         void update();
79
84         VertexArrayObject& getVAO();
85
90         size_t* getTrackBalls();
91
96         size_t* getRoadEndCircles();
97
102         glm::vec3* getBasepoints();
103
108         size_t getRenderID();
109
114         void setRGBAcolor(glm::vec4 RGBAcolor);
115
120         void setRGBcolor(glm::vec3 RGBcolor);
121
126         void setOpacity(float opacity);
127
132         glm::mat4 getWorldMatrix();
133
138         glm::vec4 getRGBAcolor();
139
144         glm::vec3 getRGBcolor();
145
150         float getOpacity();
151
156         bool isSelected();
157
161         void select();
162
166         void deselect();
167
174         bool isClicked(glm::vec3 cameraPosition, glm::vec3 ray);
175
180         glm::vec3 getEndpointA();
181
186         glm::vec3 getEndpointB();
187
192         void stuckTest(size_t road);
193
199         char markerTest(size_t marker);
200
```

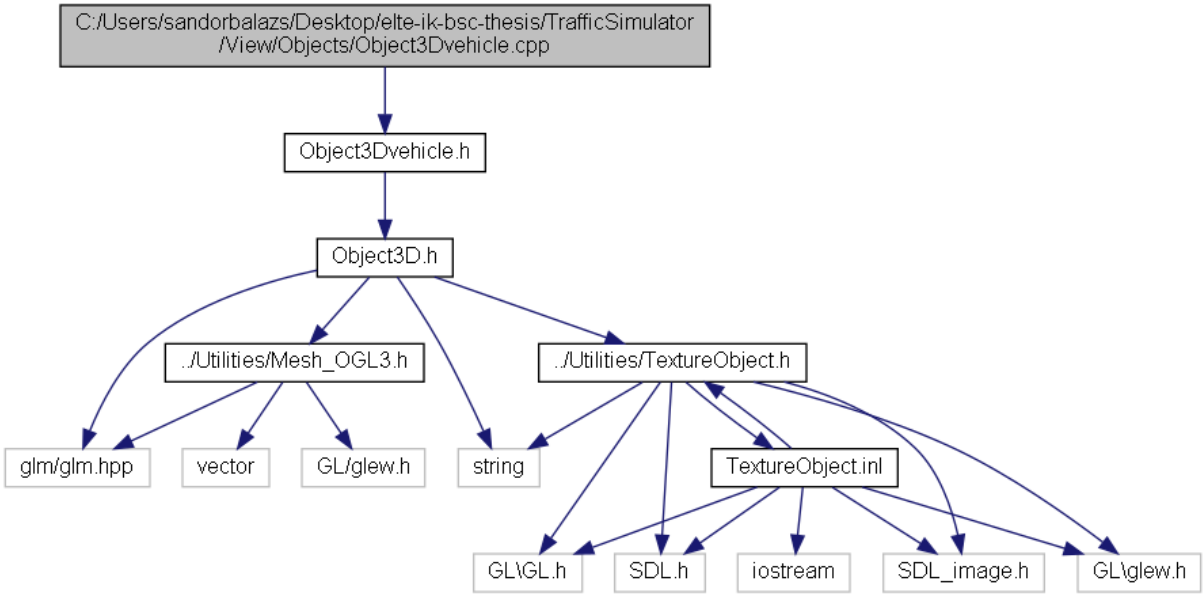
```
205         void setEndpointLock(bool lock);
206
210         void lockEditor();
211
215         void freeEditor();
216
221         float getRoadLength();
222
226         std::vector<glm::vec3> points;
227
231         std::vector<glm::vec3> shiftedpoints_1;
232
236         std::vector<glm::vec3> shiftedpoints_2;
237
241         std::vector<glm::vec3> trackOne;
242
246         std::vector<glm::vec3> trackTwo;
247
251         size_t stickA = -1;
252
256         char stickMarkA = 'Q';
257
261         size_t stickB = -1;
262
266         char stickMarkB = 'Q';
267
271         size_t markerA = -1;
272
276         size_t markerB = -1;
277
281         size_t modelID = -1;
282
283 private:
284
288         void generateRoadpoints();
289
295         glm::vec3 bezierpoint(float u);
296
305         glm::vec3 shiftpoint(glm::vec3 point1, glm::vec3 point2, float currentShift,
float yPosition);
306
310         void updateBasepoints();
311
315         void clean();
316
320         void endpointsExistTest();
321
325         void markersExistTest();
326
330         void fillModelCoordinates();
331
335         void fillModelNormals();
336
340         void fillModelTextures();
341
345         void fillModelIndices();
346
350         size_t renderID = -1;
351
355         bool selected = true;
356
360         bool endpointLock = true;
361
365         bool editorLock = false;
366
370         float shift = 2.0f;
371
375         glm::vec4 color;
376
380         size_t trackBalls[4];
381
385         size_t roadEndCircles[2];
386
390         glm::vec3 basepoints[4];
391
395         float roadYposition;
396
```

```
400         size_t  modelpointCount;
401
405         VertexArrayObject model;
406
410         ArrayBuffer modelCoordinates;
411
415         IndexBuffer modelIndices;
416
420         ArrayBuffer modelNormals;
421
425         ArrayBuffer modelTextures;
426
430         Render* render;
431 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Dvehicle.cpp File Reference

```
#include "Object3Dvehicle.h"
```

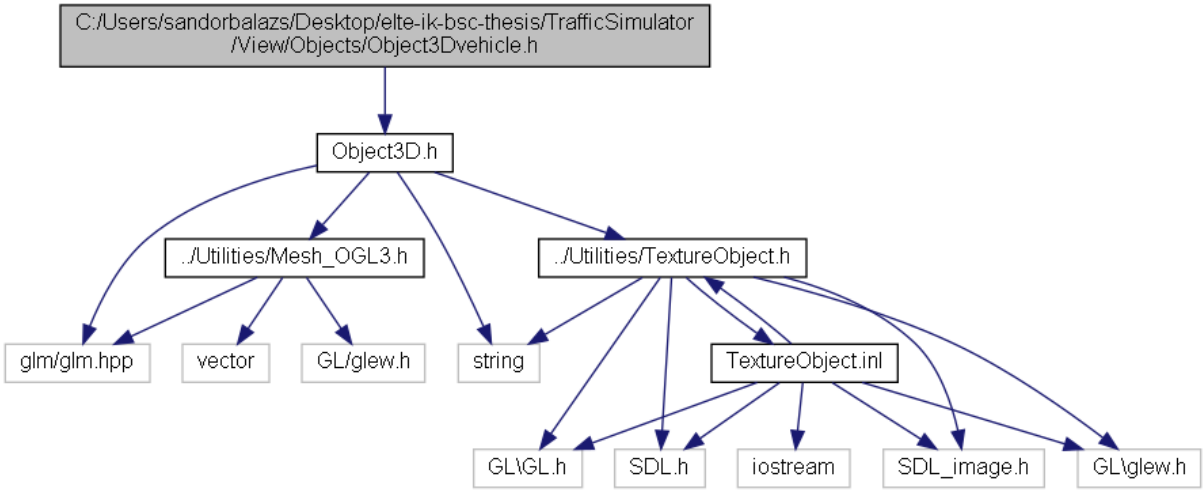
Include dependency graph for Object3Dvehicle.cpp:



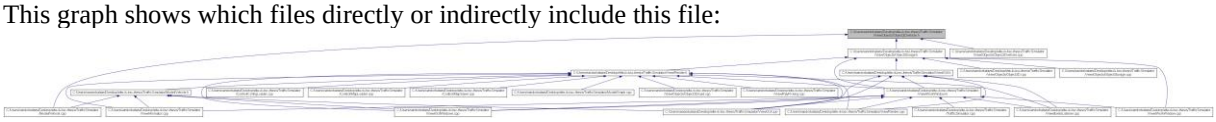
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/Object3Dvehicle.h File Reference

```
#include "Object3D.h"
```

Include dependency graph for Object3Dvehicle.h:



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



Data Structures

- class **Object3Dvehicle**
  The renderable vehicle representation.

Object3Dvehicle.h

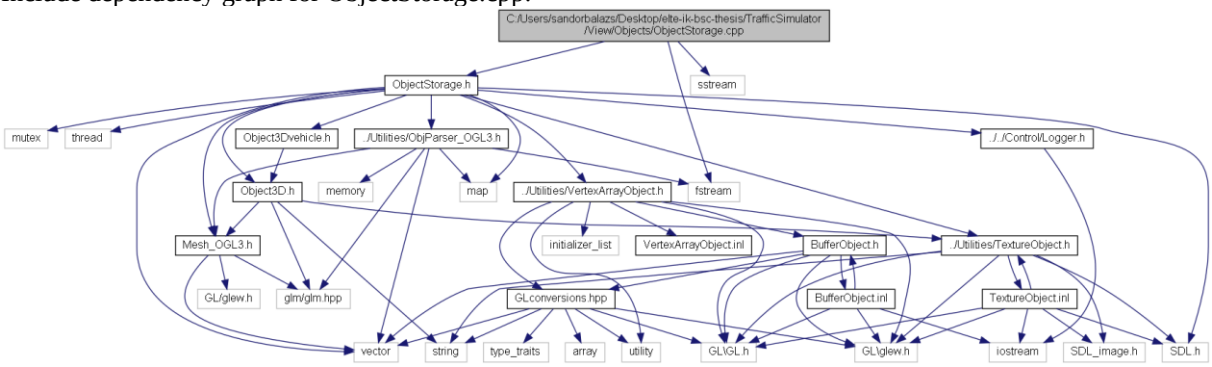
Go to the documentation of this file.1

```
11 #pragma once
12
13 #include "Object3D.h"
14
18 class Object3Dvehicle: public Object3D {
19 public:
20
25         Object3Dvehicle(ObjectStorage* objectStorage);
26
47         Object3Dvehicle(int id, std::string name, std::string type, std::string
meshID, std::string textureID, std::string lightTexture, std::string breakTexture,
48         std::string rightTexture, std::string leftTexture, std::string iconID,
49         glm::vec3 initPosition, glm::vec3 initScale, glm::vec4 initRotation,
50         glm::vec4 color, glm::vec4      hitSphere, glm::vec4 moveSphere,
ObjectStorage* objectStorage);
51
56         ~Object3Dvehicle(void);
57
62         void setLightTexture(int);
63
68         void setBreakTexture(int);
69
74         void setRightTexture(int);
75
80         void setLeftTexture (int);
81
86         Texture2D getLightTexture();
87
92         Texture2D getBreakTexture();
93
98         Texture2D getRightTexture();
99
104         Texture2D getLeftTexture ();
105
110         Object3Dvehicle copy();
111
117         Object3Dvehicle copy(int renderID);
118
125         static float getMoveRtation(glm::vec3 pointA, glm::vec3 pointB);
126
127 protected:
131         std::string lightTexture;
132
136         std::string breakTexture;
137
141         std::string rightTexture;
142
146         std::string leftTexture;
147 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/ObjectStorage.cpp File Reference

```
#include "ObjectStorage.h"
#include <sstream>
#include <fstream>
```

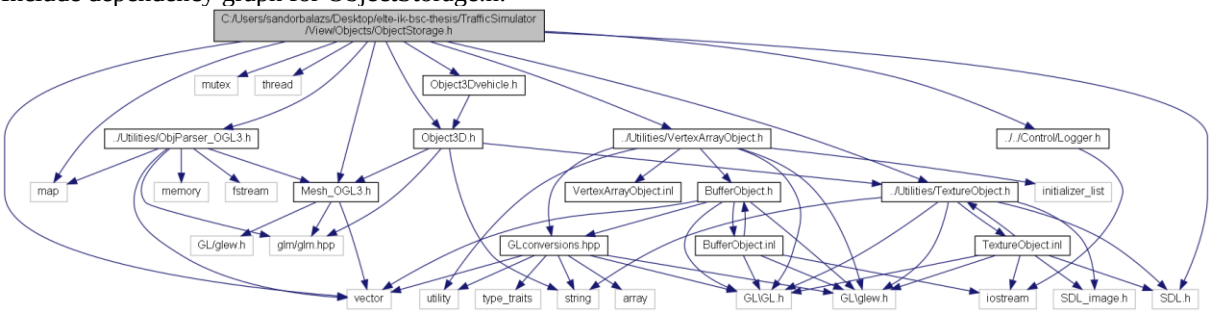
Include dependency graph for ObjectStorage.cpp:



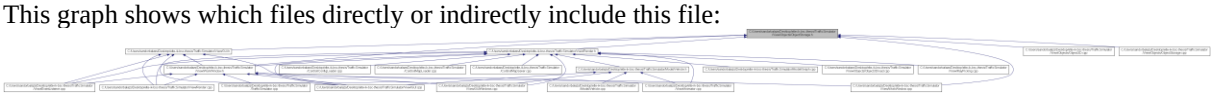
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Objects/ObjectStorage.h File Reference

```
#include <vector>
#include <map>
#include <mutex>
#include <thread>
#include <SDL.h>
#include "../../Control/Logger.h"
#include "../Utilities/ObjParser_OGL3.h"
#include "../Utilities/TextureObject.h"
#include "../Utilities/Mesh_OGL3.h"
#include "../Utilities/VertexArrayObject.h"
#include "Object3D.h"
#include "Object3Dvehicle.h"
```

Include dependency graph for ObjectStorage.h:



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



Data Structures

- class **ObjectStorage**
  Load and store the 3D Models and Textures and create the render objects.

ObjectStorage.h

Go to the documentation of this file.1

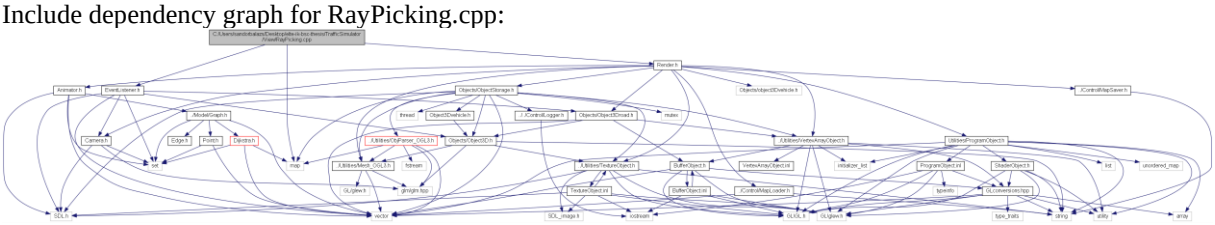
```
#pragma once

#include <vector>
#include <map>
#include <mutex>
#include <thread>
#include <SDL.h>
#include "../../Control/Logger.h"
#include "../Utilities/ObjParser_OGL3.h"
#include "../Utilities/TextureObject.h"
#include "../Utilities/Mesh_OGL3.h"
#include "../Utilities/VertexArrayObject.h"
#include "Object3D.h"
#include "Object3Dvehicle.h"

class ObjectStorage {
public:

        ObjectStorage(void);

        ~ObjectStorage(void);

        void load();

        SDL_Surface* getWindowIcon();

        Texture2D& getTexture(std::string textureName);

        std::unique_ptr<Mesh>& getMesh(std::string meshName);

        bool isLoaded();

        bool loadingCheck();

        void finaliseLoading();

        int getLoadingState();

        int getLoadingStateMax();

         Object3D getObject3D(int object3Did);

         Object3D getObject3D(int object3Did, int renderID);

         Object3Dvehicle getObject3Dvehicle(int object3Did);

         Object3Dvehicle getObject3Dvehicle(int object3Did, int renderID);

         std::vector<int>& getFirstSceneElements();

         std::map<int, Object3D> object3Ds;

         std::map<int, Object3Dvehicle> object3Dvehicles;

private:

         const std::string defaultTexture = "default.png";

         const std::string defaultObject = "sphere_mark.obj";

         const std::string configFile = "3Dobjects/3d_object_library.csv";

         const std::string textureFolder = "3Dobjects/textures/";

         const std::string miniatureFolder = "miniatures/";

         const std::string modelFolder = "3Dobjects/models/";

         bool loaded = false;

         int loadingState = 0;

```

```
189         int loadingStateMax = 0;
190
194         std::vector<std::vector<std::string>> parsedCSV;
195
199         std::mutex texturesMutex;
200
204         std::mutex objectsMutex;
205
209         std::map<std::string, Texture2D> textures;
210
214         std::map<std::string, std::unique_ptr<Mesh>> objects;
215
219         SDL_Surface* windowIcon;
220
224         std::vector<std::thread> threads;
225
229         SDL_atomic_t atomicThreadCounter;
230
234         std::vector<int> firstSceneElements;
235
239         std::map<std::string, VertexArrayObject> generatedObjects;
240
244         void readCSV();
245
250         void loadTexture(std::string fileName);
251
257         bool isThisTextureLoaded(std::string textureName);
258
264         std::thread loadTextureParallel(std::string fileName);
265
269         void bindTextures();
270
275         void loadObject(std::string fileName);
276
282         bool isThisObjectLoaded(std::string objectName);
283
288         std::thread loadObjectParallel(std::string fileName);
289
293         void bindObjects();
294
299         void store(int csvID);
300 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/RayPicking.cpp File Reference

```
#include "EventListener.h"
#include "Render.h"
#include <map>
```

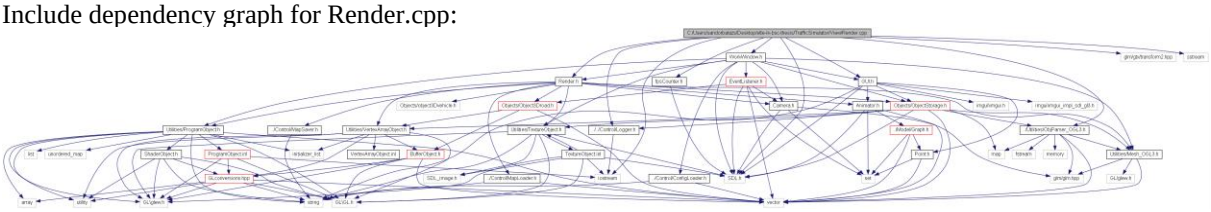
Include dependency graph for RayPicking.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Render.cpp File Reference

```
#include "Render.h"
#include "WorkWindow.h"
#include "GUI.h"
#include "fpsCounter.h"
#include "Utilities/ObjParser_OGL3.h"
#include <glm/gtx/transform2.hpp>
#include <SDL.h>
#include <iostream>
#include <sstream>
#include "../Control/Logger.h"
```

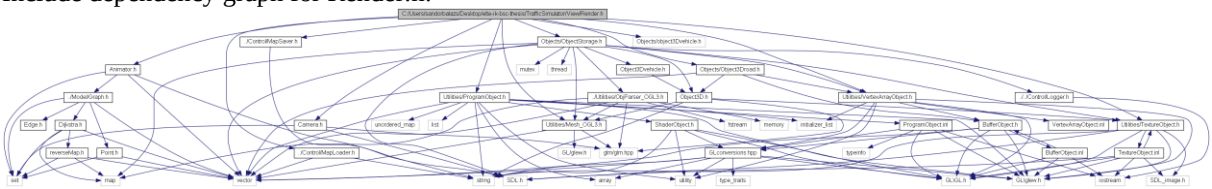
Include dependency graph for Render.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Render.h File Reference

```
#include "Camera.h"
#include "Animator.h"
#include "Utilities/ProgramObject.h"
#include "Utilities/TextureObject.h"
#include "Utilities/Mesh_OGL3.h"
#include "Utilities/VertexArrayObject.h"
#include "Objects/ObjectStorage.h"
#include "Objects/Object3Droad.h"
#include "Objects/object3Dvehicle.h"
#include "../Control/MapLoader.h"
#include "../Control/MapSaver.h"
```

Include dependency graph for Render.h:



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

Data Structures

- class **Render**
  The 3D Scene render implementation.

Render.h

Go to the documentation of this file.1

```
11 #pragma once
15 class WorkWindow;
16 class GUI;
17
18 #include "Camera.h"
19 #include "Animator.h"
20 #include "Utilities/ProgramObject.h"
21 #include "Utilities/TextureObject.h"
22 #include "Utilities/Mesh_OGL3.h"
23 #include "Utilities/VertexArrayObject.h"
24 #include "Objects/ObjectStorage.h"
25 #include "Objects/Object3Droad.h"
26 #include "Objects/object3Dvehicle.h"
27 #include "../Control/MapLoader.h"
28 #include "../Control/MapSaver.h"
29
33 class Render {
34 public:
39         void bind(WorkWindow* window);
40
44         void render();
45
49         void vsyncOn();
50
54         void vsyncOff();
55
59         void multisamplingOn();
60
64         void multisamplingOff();
65
69         void clearScrean();
70
74         void clear();
75
81         int addObject(int objectStorageID);
82
88         void deleteObject(int renderID);
89
95         int addRoad();
96
102         void deleteRoad(int dynamicRenderID);
103
110         size_t addVehicle(int objectStorageID, size_t modelID);
111
116         void deleteVehicle(size_t renderableVehicleID);
117
121         void close();
122
127         void updateDynamicObject(size_t dynamicRenderID);
128
133         MapLoader* getMapLoader();
134
139         MapSaver* getMapSaver();
140
145         GUI* getGui();
146
151         WorkWindow* getWorkingWindow();
152
158         Object3D* getObject(size_t renderID);
159
164         size_t getObjectsNumber();
165
171         Object3Droad* getDynamicObject(size_t dynamicRenderID);
172
177         size_t getDynamicObjectsNumber();
178
184         Object3Dvehicle* getVehicle(size_t renderableVehicleID);
185
190         size_t getVehiclesNumber();
191
195         void photoModeOn();
196
```

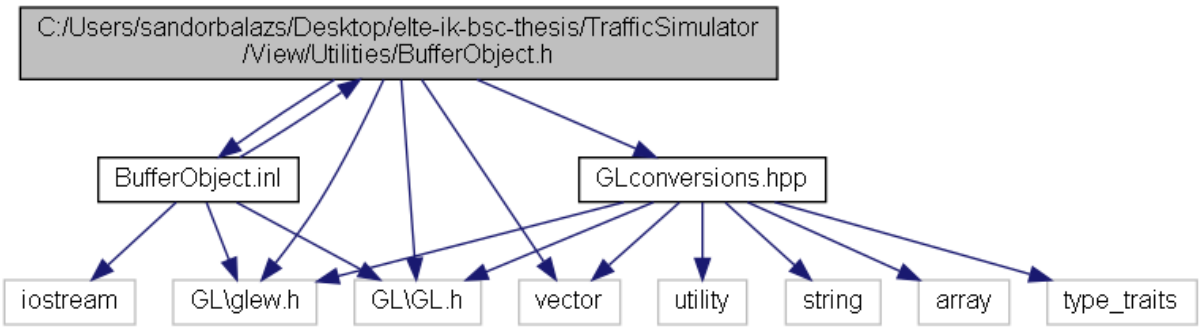
```
200         void photoModeOff();
201
205         bool hitSphare = false;
206
210         bool roadHhitSphare = false;
211
215         bool vehicleHitSphare = false;
216
220         bool showRoadHitSphereMiddle = true;
221
225         bool showRoadSide01 = false;
226
230         bool showRoadSide02 = false;
231
235         bool showRoadTrack01 = false;
236
240         bool showRoadTrack02 = false;
241
245         bool objectsWireframe = false;
246
250         bool roadWireframe = false;
251
255         Animator* getAnimator();
256
260         void lockEditor();
261
265         void freeEditor();
266
270         bool isEditorLoced();
271
275         void clearVehicles();
276
280         void setToFullScreen();
281
285         void setToWindowed();
286
290         void setToBorderless();
291
295         glm::vec3 lightPosition = glm::vec3(55, 55, 55);
296
300         bool lightCameraTracking = true;
301
305         glm::vec4 ambientLight = glm::vec4(0.1f, 0.1f, 0.1f, 1.0f);
306
310         glm::vec4 diffuseLight = glm::vec4(0.75f, 0.75f, 0.75f, 1.0f);
311
315         glm::vec4 specularLight = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f);
316
320         glm::vec4 ambientMaterial = glm::vec4(1.0f, 1.0f, 1.0f, 0.0f);
321
325         glm::vec4 diffuseMaterial = glm::vec4(1.0f, 1.0f, 1.0f, 1.0f);
326
330         glm::vec4 specularMaterial = glm::vec4(1.0f, 1.0f, 1.0f, 0.0f);
331
335         float specularPower = 32.0f;
336
340         static const char* version;
341
342 private:
343
347         void shaderLightUpdate();
348
352         std::vector<Object3D> renderableObjects;
353
357         std::vector<Object3Droad*> renderableRoads;
358
362         std::vector<Object3Dvehicle> renderableVehicles;
363
368         bool firstScreen = true;
369
373         bool editorLock = false;
374
378         WorkWindow* workingWindow;
379
383         SDL_Window* window;
384
388         Camera* camera;
```

```
389
393         ProgramObject* shader;
394
398         GUI* gui;
399
403         ObjectStorage* objectStorage;
404
408         MapLoader mapLoader;
409
413         MapSaver mapSaver;
414
418         Animator animator;
419
424         void setTexture(Texture2D& texture);
425
431         void shaderPreDrawingUpdate(glm::mat4 worldMatrix, glm::vec4 color);
432
437         void drawMesh(std::unique_ptr<Mesh>& mesh);
438
444         void drawVao(VertexArrayObject& vao, int trianglesNumber);
445
449         void rendering();
450
455         void setWindowTitle(std::string title);
456
461         void setWindowIcon(SDL_Surface* windowIcon);
462
466         void faceTestOn();
467
471         void faceTestOff();
472
476         void shaderCameraUpdate();
477
481         void renderScrean();
482
486         void sceneInit();
487
491         void wireframeOn();
492
496         void wireframeOff();
497
502         void showHitSphere(size_t renderID);
503
507         void showVehicleHitSphere(size_t vehicleRenderID);
508
515         void showRoadHitSphere(std::vector<glm::vec3>& roadPoints, float size,
glm::vec3 color);
516
520         void showAllRoadHitSpheres();
521 };
```

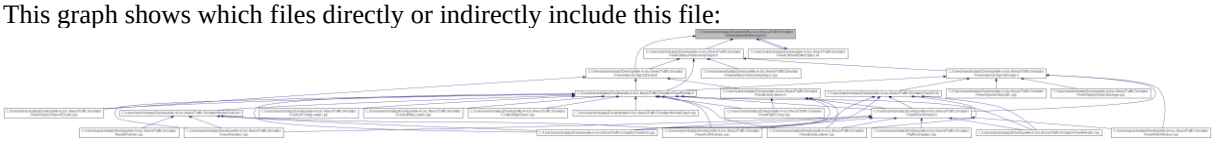
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/BufferObject.h File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include <vector>
#include "GLconversions.hpp"
#include "BufferObject.inl"
```

Include dependency graph for BufferObject.h:



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



Data Structures

- class **BufferObject< target, usage >**

Typedefs

- using **ArrayBuffer** = **BufferObject**<>
- using **IndexBuffer** = **BufferObject**< **BufferType::ElementArray** >

Enumerations

- enum class **BufferType** : GLenum { **Array** = GL_ARRAY_BUFFER, **AtomicCounter** = GL_ATOMIC_COUNTER_BUFFER, **CopyRead** = GL_COPY_READ_BUFFER, **CopyWrite** = GL_COPY_WRITE_BUFFER, **DispatchIndirect** = GL_DISPATCH_INDIRECT_BUFFER, **DrawIndirect** = GL_DRAW_INDIRECT_BUFFER, **ElementArray** = GL_ELEMENT_ARRAY_BUFFER, **PixelPack** = GL_PIXEL_PACK_BUFFER, **PixelUnpack** = GL_PIXEL_UNPACK_BUFFER, **Query** = GL_QUERY_BUFFER, **ShaderStorage** = GL_SHADER_STORAGE_BUFFER, **Texture** = GL_TEXTURE_BUFFER, **TransformFeedback** = GL_TRANSFORM_FEEDBACK_BUFFER, **Uniform** = GL_UNIFORM_BUFFER }
- enum class **BufferUsage** : GLenum { **StreamDraw** = GL_STREAM_DRAW, **StreamRead** = GL_STREAM_READ, **StreamCopy** = GL_STREAM_COPY, **StaticDraw** = GL_STATIC_DRAW, **StaticRead** = GL_STATIC_READ, **StaticCopy** = GL_STATIC_COPY, **DynamicDraw** = GL_DYNAMIC_DRAW, **DynamicRead** = GL_DYNAMIC_READ, **DynamicCopy** = GL_DYNAMIC_COPY }

Typedef Documentation

using ArrayBuffer = BufferObject<>

using IndexBuffer = BufferObject<BufferType::ElementArray>

Enumeration Type Documentation

enum class BufferType : GLenum `[strong]`

Enumerator:

Array	
AtomicCounter	
CopyRead	
CopyWrite	
DispatchIndirect	
DrawIndirect	
ElementArray	
PixelPack	
PixelUnpack	
Query	
ShaderStorage	
Texture	
TransformFeedback	
Uniform	

enum class BufferUsage : GLenum `[strong]`

Enumerator:

StreamDraw	
StreamRead	
StreamCopy	
StaticDraw	
StaticRead	
StaticCopy	
DynamicDraw	
DynamicRead	
DynamicCopy	

BufferObject.h

Go to the documentation of this file.

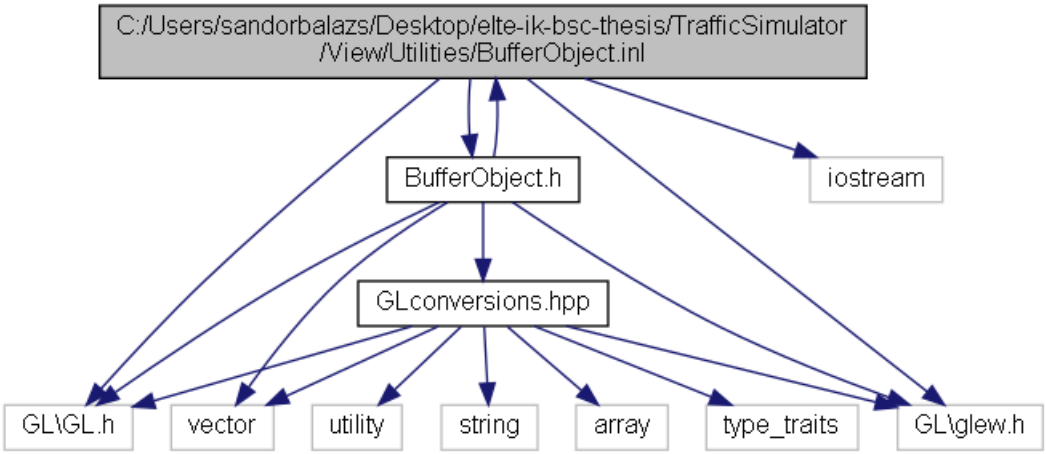
```
1 #pragma once
2
3 #include <GL\glew.h>
4 #include <GL\GL.h>
5
6 #include <vector>
7
8 #include "GLconversions.hpp"
9
10 /*
11         BufferType is an enum class that stands for OpenGL bind targets (from
https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glBufferData.xhtml - OpenGL
4.6)
12 */
13 enum class BufferType : GLenum
14 {
15         Array                           = GL_ARRAY_BUFFER,
16         AtomicCounter           = GL_ATOMIC_COUNTER_BUFFER,
17         CopyRead                        = GL_COPY_READ_BUFFER,
18         CopyWrite                       = GL_COPY_WRITE_BUFFER,
19         DispatchIndirect        = GL_DISPATCH_INDIRECT_BUFFER,
20         DrawIndirect            = GL_DRAW_INDIRECT_BUFFER,
21         ElementArray            = GL_ELEMENT_ARRAY_BUFFER,
22         PixelPack                       = GL_PIXEL_PACK_BUFFER,
23         PixelUnpack                     = GL_PIXEL_UNPACK_BUFFER,
24         Query                           = GL_QUERY_BUFFER,
25         ShaderStorage           = GL_SHADER_STORAGE_BUFFER,
26         Texture                         = GL_TEXTURE_BUFFER,
27         TransformFeedback       = GL_TRANSFORM_FEEDBACK_BUFFER,
28         Uniform                         = GL_UNIFORM_BUFFER
29 };
30
31 enum class BufferUsage : GLenum
32 {
33         StreamDraw              = GL_STREAM_DRAW,
34         StreamRead              = GL_STREAM_READ,
35         StreamCopy              = GL_STREAM_COPY,
36         StaticDraw              = GL_STATIC_DRAW,
37         StaticRead              = GL_STATIC_READ,
38         StaticCopy              = GL_STATIC_COPY,
39         DynamicDraw             = GL_DYNAMIC_DRAW,
40         DynamicRead             = GL_DYNAMIC_READ,
41         DynamicCopy             = GL_DYNAMIC_COPY
42 };
43
44 template <BufferType target = BufferType::Array , BufferUsage usage =
BufferUsage::StreamDraw>
45 class BufferObject final
46 {
47 public:
48         BufferObject();
49         ~BufferObject();
50
51         BufferObject(const BufferObject&) = delete;
52         BufferObject& operator=(const BufferObject&) = delete;
53
54         BufferObject(BufferObject&& rhs);
55         BufferObject& operator=(BufferObject&& rhs);
56
57         template <typename T>
58         explicit BufferObject(const std::vector<T>&);
59
60         template <typename T, size_t N>
61         explicit BufferObject(const std::array<T, N>&);
62
63         operator unsigned int() const { return m_id; }
64
65         void Clean();
66
67         template <typename T>
68         IsContiguousContainer<T> BufferData(const T& pArr);
69
70         void BufferData(GLsizeiptr pSize, const GLvoid* pSource = nullptr);
```

```
71
72          void BufferSubData(GLintptr pOffset, GLsizeiptr pSize, const GLvoid* pSource
= nullptr);
73
74          inline void Bind() const;
75
76          template <typename T>
77          BufferObject& operator=(const T& pArr);
78
79          // returns a read-only copy of the buffer contents as an std::vector
80          template <typename T>
81          operator std::vector<T>() const;
82
83          // returns a read-only copy of the buffer contents as an array
84          template <typename T, size_t N>
85          operator std::array<T, N>() const;
86
87 private:
88          GLuint m_id{};
89          GLsizeiptr m_sizeInBytes{};
90          static GLuint g_lastBound;
91 };
92
93 #include "BufferObject.inl"
94
95 using ArrayBuffer = BufferObject<>;
96 using IndexBuffer = BufferObject<BufferType::ElementArray>;
97
```

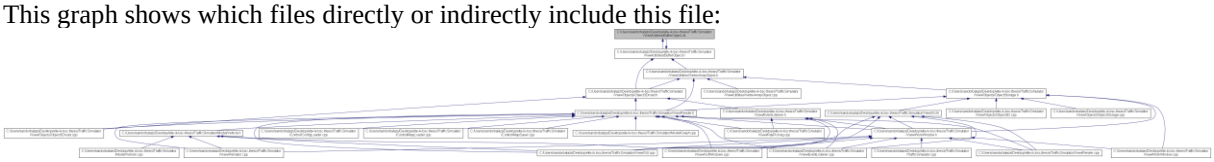
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/BufferObject.inl File Reference

```
#include "BufferObject.h"
#include <GL\glew.h>
#include <GL\GL.h>
#include <iostream>
```

Include dependency graph for BufferObject.inl:



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

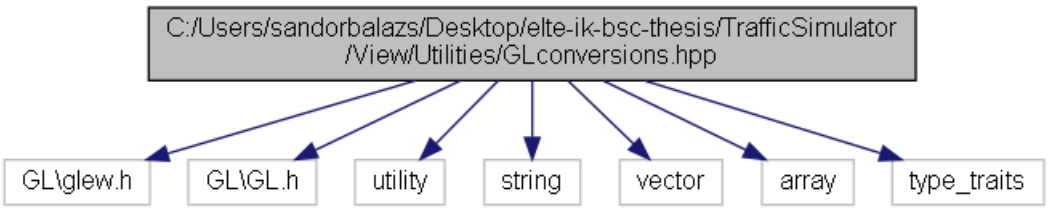


C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/GLconversions.cpp File Reference

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/GLconversions.hpp File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include <utility>
#include <string>
#include <vector>
#include <array>
#include <type_traits>
```

Include dependency graph for GLconversions.hpp:



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

Data Structures

- struct **GLExtractPrimitiveType< T, IsValidBaseType< T > >**
- struct **ElementInfo< T >**
- struct **ElementInfo< std::vector< T > >**
- struct **ElementInfo< std::array< T, N > >**
- struct **ElementInfo< T[N]>**
- struct **HasContiguousStorage< T >**
- struct **HasContiguousStorage< std::vector< T > >**
- struct **HasContiguousStorage< std::array< T, N > >**
- struct **HasContiguousStorage< T[N]>**

Macros

- #define **CALL_SETTER_VECTOR_SPECIALIZATION**(TYPE, DIM_X, FUNCTION) template <> inline void **CallSetter**<TYPE, DIM_X, 1>(GLint pLoc, GLsizei pCount, const TYPE* pArg) { FUNCTION(pLoc, pCount, pArg); }
- #define **CALL_SETTER_MATRIX_SPECIALIZATION**(TYPE, DIM_X, DIM_Y, FUNCTION) template <> inline void **CallSetter**<TYPE, DIM_X, DIM_Y>(GLint pLoc, GLsizei pCount, const TYPE* pArg) { FUNCTION(pLoc, pCount, GL_FALSE, pArg); }

Typedefs

- using **TypeSourcePair** = std::pair< GLuint, std::string >
- template<typename T > using **IsValidBaseType** = std::enable_if_t< std::is_same_v< T, GLint >||std::is_same_v< T, GLuint >||std::is_same_v< T, GLfloat >||std::is_same_v< T, GLdouble > >
- template<typename T , typename R = void> using **IsContiguousContainer** = std::enable_if_t< **HasContiguousStorage**< T >::value, R >

Functions

- **TypeSourcePair operator""_vs** (const char *str, size_t len)

- **TypeSourcePair operator""_tcs** (const char *str, size_t len)
- **TypeSourcePair operator""_tes** (const char *str, size_t len)
- **TypeSourcePair operator""_gs** (const char *str, size_t len)
- **TypeSourcePair operator""_fs** (const char *str, size_t len)
- **TypeSourcePair operator""_comp** (const char *str, size_t len)
- template<typename T > constexpr GLenum **NativeTypeToOpenGLType** ()
- template<> constexpr GLenum **NativeTypeToOpenGLType< float >** ()
- template<> constexpr GLenum **NativeTypeToOpenGLType< double >** ()
- template<typename T > constexpr std::pair< size_t, size_t > **ComponentCount** ()
- template<typename T , int N, int M> void **CallSetter** (GLint, GLsizei, const T *)
- template<typename T > constexpr GLsizei **ContainerLength** (const T &)
- template<typename T , size_t N> constexpr GLsizei **ContainerLength** (const std::array< T, N > &)
- template<typename T , size_t N> constexpr GLsizei **ContainerLength** (const T(&)[N])
- template<typename T > constexpr GLsizei **ContainerLength** (const std::vector< T > &pArr)
- template<typename T > constexpr GLsizei **ContainerSizeInBytes** (const T &)
- template<typename T , size_t N> constexpr GLsizei **ContainerSizeInBytes** (const std::array< T, N > &)
- template<typename T , size_t N> constexpr GLsizei **ContainerSizeInBytes** (const T(&)[N])
- template<typename T > constexpr GLsizei **ContainerSizeInBytes** (const std::vector< T > &pArr)
- template<typename T > constexpr const T * **PointerToStart** (const T &pArg)
- template<typename T , size_t N> constexpr const T * **PointerToStart** (const std::array< T, N > &pArg)
- template<typename T , size_t N> constexpr const T * **PointerToStart** (const T(&pArg)[N])
- template<typename T > constexpr const T * **PointerToStart** (const std::vector< T > &pArg)

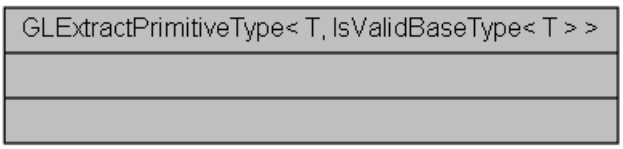
Data Structure Documentation

struct GLExtractPrimitiveType< T, IsValidBaseType< T > >

template<typename T>

struct GLExtractPrimitiveType< T, IsValidBaseType< T > >

Collaboration diagram for GLExtractPrimitiveType< T, IsValidBaseType< T > >:



Data Fields:

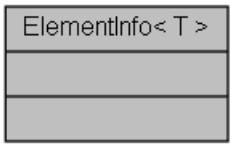
typedef T	primitive_type	

struct ElementInfo

template<typename T>

struct ElementInfo< T >

Collaboration diagram for ElementInfo< T >:



Data Fields:

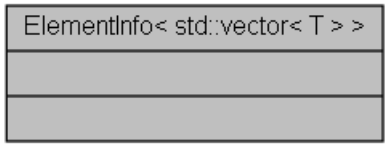
typedef T	value_type	

struct ElementInfo< std::vector< T > >

template<typename T>

struct ElementInfo< std::vector< T > >

Collaboration diagram for ElementInfo< std::vector< T > >:



Data Fields:

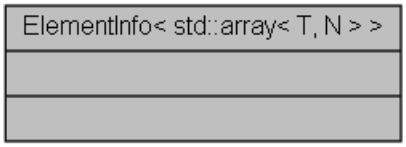
typedef T	value_type	

struct ElementInfo< std::array< T, N > >

template<typename T, size_t N>

struct ElementInfo< std::array< T, N > >

Collaboration diagram for ElementInfo< std::array< T, N > >:



Data Fields:

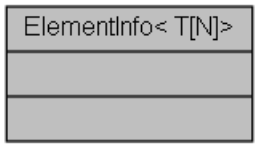
typedef T	value_type	

struct ElementInfo< T[N]>

template<typename T, size_t N>

struct ElementInfo< T[N]>

Collaboration diagram for ElementInfo< T[N]>:



Data Fields:

typedef T	value_type	

Macro Definition Documentation

#define CALL_SETTER_MATRIX_SPECIALIZATION(TYPE, DIM_X, DIM_Y, FUNCTION) template <> inline void CallSetter<TYPE, DIM_X, DIM_Y>(GLint pLoc, GLsizei pCount, const TYPE* pArg) { FUNCTION(pLoc, pCount, GL_FALSE, pArg); }

#define CALL_SETTER_VECTOR_SPECIALIZATION(TYPE, DIM_X, FUNCTION) template <> inline void CallSetter<TYPE, DIM_X, 1>(GLint pLoc, GLsizei pCount, const TYPE* pArg) { FUNCTION(pLoc, pCount, pArg); }

Typedef Documentation

template<typename T , typename R = void> using IsContiguousContainer = std::enable_if_t<HasContiguousStorage< T>::value, R>

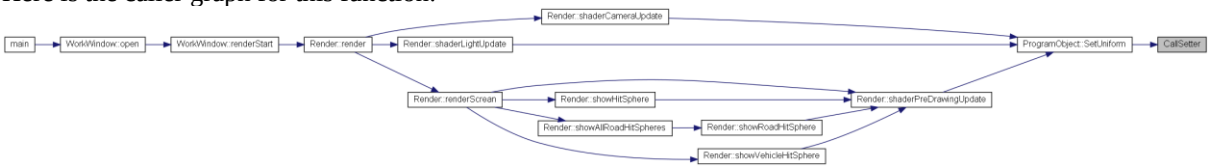
template<typename T > using IsValidBaseType = std::enable_if_t< std::is_same_v<T, GLint> || std::is_same_v<T, GLuint> || std::is_same_v<T, GLfloat> || std::is_same_v<T, GLdouble> >

using TypeSourcePair = std::pair<GLuint, std::string>

Function Documentation

template<typename T , int N, int M> void CallSetter (GLint , GLsizei , const T *)`[inline]`

Here is the caller graph for this function:



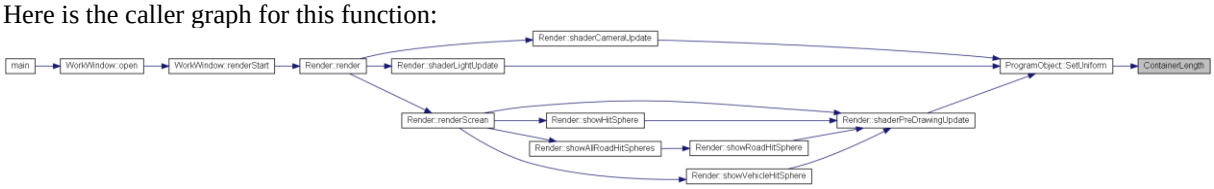
template<typename T > constexpr std::pair< size_t, size_t > ComponentCount () `[constexpr]`

template<typename T , size_t N> constexpr GLsizei ContainerLength (const std::array< T, N > &) `[constexpr]`

template<typename T > constexpr GLsizei ContainerLength (const std::vector< T > & *pArr*) `[constexpr]`

template<typename T > constexpr GLsizei ContainerLength (const T &) `[constexpr]`

Here is the caller graph for this function:



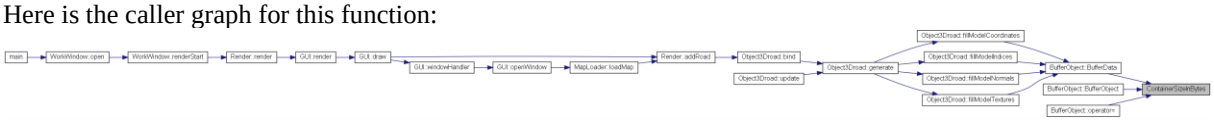
template<typename T , size_t N> constexpr GLsizei ContainerLength (const *T*(&)[N]) `[constexpr]`

template<typename T , size_t N> constexpr GLsizei ContainerSizeInBytes (const std::array< T, N > &) `[constexpr]`

template<typename T > constexpr GLsizei ContainerSizeInBytes (const std::vector< T > & *pArr*) `[constexpr]`

template<typename T > constexpr GLsizei ContainerSizeInBytes (const T &) `[constexpr]`

Here is the caller graph for this function:



template<typename T , size_t N> constexpr GLsizei ContainerSizeInBytes (const *T*(&)[N]) `[constexpr]`

template<typename T > constexpr GLenum NativeTypeToOpenGLType () `[constexpr]`

template<> constexpr GLenum NativeTypeToOpenGLType< double > () `[constexpr]`

template<> constexpr GLenum NativeTypeToOpenGLType< float > () `[constexpr]`

TypeSourcePair operator""_comp (const char * *str*, size_t *len*) `[inline]`

TypeSourcePair operator""_fs (const char * *str*, size_t *len*) `[inline]`

TypeSourcePair operator""_gs (const char * *str*, size_t *len*) `[inline]`

TypeSourcePair operator""_tcs (const char * *str*, size_t *len*) `[inline]`

TypeSourcePair operator""_tes (const char * *str*, size_t *len*) `[inline]`

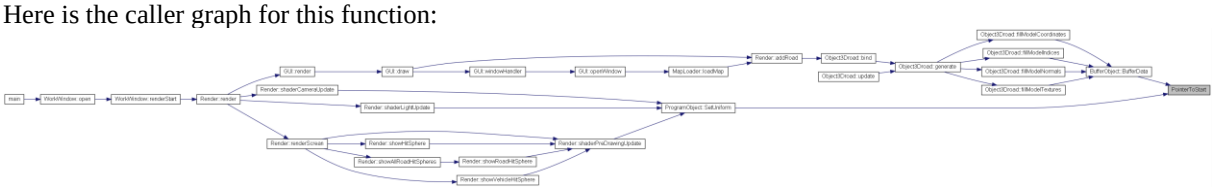
TypeSourcePair operator""_vs (const char * *str*, size_t *len*) `[inline]`

template<typename T , size_t N> constexpr const T * PointerToStart (const std::array< T, N > & *pArg*) `[constexpr]`

template<typename T > constexpr const T * PointerToStart (const std::vector< T > & *pArg*) `[constexpr]`

template<typename T > constexpr const T * PointerToStart (const T & *pArg*) `[constexpr]`

Here is the caller graph for this function:



template<typename T , size_t N> constexpr const T * PointerToStart (const T(&) *pArg*[N]) `[constexpr]`

GLconversions.hpp

Go to the documentation of this file.

```
1 #pragma once
2
3 #include <GL\glew.h>
4 #include <GL\GL.h>
5
6 #include <utility>
7 #include <string>
8 #include <vector>
9 #include <array>
10
11 #include <type_traits>
12
13 /*
14
15         Convenience conversion functions to simplify the creation of ShaderObject
objects using the _vs, _tcs, _tes, _gs, _fs, and _comp suffices.
16
17 */
18 using TypeSourcePair = std::pair<GLuint, std::string>;
19
20 inline TypeSourcePair operator"" _vs  (const char * str, size_t len) { return
std::make_pair(GL_VERTEX_SHADER,                  std::string{ str, len }); }
21 inline TypeSourcePair operator"" _tcs (const char * str, size_t len) { return
std::make_pair(GL_TESS_CONTROL_SHADER,    std::string{ str, len }); }
22 inline TypeSourcePair operator"" _tes (const char * str, size_t len) { return
std::make_pair(GL_TESS_EVALUATION_SHADER, std::string{ str, len }); }
23 inline TypeSourcePair operator"" _gs  (const char * str, size_t len) { return
std::make_pair(GL_GEOMETRY_SHADER,                std::string{ str, len }); }
24 inline TypeSourcePair operator"" _fs  (const char * str, size_t len) { return
std::make_pair(GL_FRAGMENT_SHADER,                std::string{ str, len }); }
25 inline TypeSourcePair operator"" _comp(const char * str, size_t len) { return
std::make_pair(GL_COMPUTE_SHADER,                 std::string{ str, len }); }
26
27 /*
28
29         Enabler templates to restruct SetUniform utility function instantiations to
supported types only.
30
31 */
32 template <typename T>
33 using IsValidBaseType = std::enable_if_t<
34         std::is_same_v<T, GLint> ||
35         std::is_same_v<T, GLuint> ||
36         std::is_same_v<T, GLfloat> ||
37         std::is_same_v<T, GLdouble>
38 >;
39
40 #ifdef GLM_VERSION
41         template <typename T, glm::precision P = glm::defaultp>
42         using IsValidGLMVectorType = std::enable_if_t<
43                 std::is_base_of_v<glm::tvec1<typename T::value_type, P>, T> ||
44                 std::is_base_of_v<glm::tvec2<typename T::value_type, P>, T> ||
45                 std::is_base_of_v<glm::tvec3<typename T::value_type, P>, T> ||
46                 std::is_base_of_v<glm::tvec4<typename T::value_type, P>, T>
47         >;
48
49         template <typename T, glm::precision P = glm::defaultp>
50         using IsValidGLMMatrixType = std::enable_if_t<
51                 std::is_base_of_v<glm::tmat2x2<typename T::value_type, P>, T> ||
52                 std::is_base_of_v<glm::tmat2x3<typename T::value_type, P>, T> ||
53                 std::is_base_of_v<glm::tmat2x4<typename T::value_type, P>, T> ||
54                 std::is_base_of_v<glm::tmat3x2<typename T::value_type, P>, T> ||
55                 std::is_base_of_v<glm::tmat3x4<typename T::value_type, P>, T> ||
56                 std::is_base_of_v<glm::tmat3x4<typename T::value_type, P>, T> ||
57                 std::is_base_of_v<glm::tmat4x2<typename T::value_type, P>, T> ||
58                 std::is_base_of_v<glm::tmat4x3<typename T::value_type, P>, T> ||
59                 std::is_base_of_v<glm::tmat4x4<typename T::value_type, P>, T>
60         >;
61 #endif
62
63 /*
64
```

```
65          Convert types to OpenGL type GLenum values listed in
https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glVertexAttribPointer.xhtml
66
67 */
68
69 template <typename T> constexpr GLenum NativeTypeToOpenGLType() { static_assert(true,
"Non-supported vertex attribute type"); }
70 template <> constexpr GLenum NativeTypeToOpenGLType<float>() { return GL_FLOAT; }
71 template <> constexpr GLenum NativeTypeToOpenGLType<double>() { return GL_DOUBLE; }
72
73 // we have to provide these extraction functions because tvec<*,*>::length(), cols(),
rows() are not defined as a constexpr...
74 template<typename T>    constexpr std::pair<size_t, size_t> ComponentCount() { return
std::make_pair(1, 1); }
75 #ifdef GLM_VERSION
76          #define COMPONENT_COUNT_SPECIALIZATION(TYPE, DIM_X, DIM_Y) template<>
constexpr std::pair<size_t, size_t> ComponentCount<TYPE>() { return std::make_pair(DIM_X,
DIM_Y); }
77          COMPONENT_COUNT_SPECIALIZATION(glm::vec2, 2, 1)
78          COMPONENT_COUNT_SPECIALIZATION(glm::vec3, 3, 1)
79          COMPONENT_COUNT_SPECIALIZATION(glm::vec4, 4, 1)
80          COMPONENT_COUNT_SPECIALIZATION(glm::mat2, 2, 2)
81          COMPONENT_COUNT_SPECIALIZATION(glm::mat3, 3, 3)
82          COMPONENT_COUNT_SPECIALIZATION(glm::mat4, 4, 4)
83          COMPONENT_COUNT_SPECIALIZATION(glm::mat4x3, 4, 3)
84          COMPONENT_COUNT_SPECIALIZATION(glm::mat4x2, 4, 2)
85          COMPONENT_COUNT_SPECIALIZATION(glm::mat3x2, 3, 2)
86          COMPONENT_COUNT_SPECIALIZATION(glm::mat3x4, 3, 4)
87          COMPONENT_COUNT_SPECIALIZATION(glm::mat2x3, 2, 3)
88          COMPONENT_COUNT_SPECIALIZATION(glm::mat2x4, 2, 4)
89 #endif
90 /*
91
92          Extract primitive type of argument, i.e. what is the base type (e.g. float) of
whatever construct (e.g. glm::vec3, glm::mat4, etc.)
93          we received.
94
95 */
96
97 template <typename T, typename C = void> struct GLExtractPrimitiveType;
98 template <typename T> struct GLExtractPrimitiveType<T, IsValidBaseType<T>>
{ using primitive_type = T;};
99 #ifdef GLM_VERSION
100          template <typename T> struct GLExtractPrimitiveType<T,
IsValidGLMVectorType<T>> { using primitive_type = typename T::value_type; };
101          template <typename T> struct GLExtractPrimitiveType<T,
IsValidGLMMatrixType<T>> { using primitive_type = typename T::value_type; };
102 #endif
103
104 /*
105
106          Selection of appropriate OpenGL glUniform* function, depending on argument
primitive type.
107
108 */
109 template <typename T, int N, int M> inline void CallSetter(GLint, GLsizei, const T*)
{ }
110
111 #define CALL_SETTER_VECTOR_SPECIALIZATION(TYPE, DIM_X, FUNCTION)
template <> inline void CallSetter<TYPE, DIM_X, 1>(GLint pLoc, GLsizei pCount, const TYPE*
pArg) { FUNCTION(pLoc, pCount, pArg); }
112 #define CALL_SETTER_MATRIX_SPECIALIZATION(TYPE, DIM_X, DIM_Y, FUNCTION) template <>
inline void CallSetter<TYPE, DIM_X, DIM_Y>(GLint pLoc, GLsizei pCount, const TYPE* pArg)
{ FUNCTION(pLoc, pCount, GL_FALSE, pArg); }
113
114 CALL_SETTER_VECTOR_SPECIALIZATION(GLint,                1, glUniform1iv )
115 CALL_SETTER_VECTOR_SPECIALIZATION(GLint,                2, glUniform2iv )
116 CALL_SETTER_VECTOR_SPECIALIZATION(GLint,                3, glUniform3iv )
117 CALL_SETTER_VECTOR_SPECIALIZATION(GLint,                4, glUniform4iv )
118
119 CALL_SETTER_VECTOR_SPECIALIZATION(GLuint,               1, glUniform1uiv)
120 CALL_SETTER_VECTOR_SPECIALIZATION(GLuint,               2, glUniform2uiv)
121 CALL_SETTER_VECTOR_SPECIALIZATION(GLuint,               3, glUniform3uiv)
122 CALL_SETTER_VECTOR_SPECIALIZATION(GLuint,               4, glUniform4uiv)
123
124 CALL_SETTER_VECTOR_SPECIALIZATION(GLfloat,              1, glUniform1fv )
125 CALL_SETTER_VECTOR_SPECIALIZATION(GLfloat,              2, glUniform2fv )
```

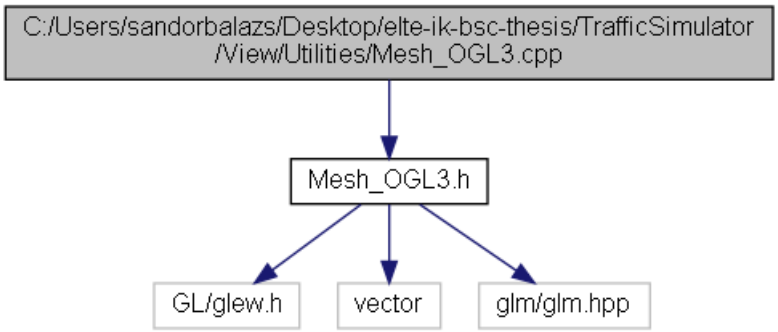
```
CALL_SETTER_VECTOR_SPECIALIZATION(GLfloat,              3, glUniform3fv )
CALL_SETTER_VECTOR_SPECIALIZATION(GLfloat,              4, glUniform4fv )

CALL_SETTER_VECTOR_SPECIALIZATION(GLdouble,             1, glUniform1dv )
CALL_SETTER_VECTOR_SPECIALIZATION(GLdouble,             2, glUniform2dv )
CALL_SETTER_VECTOR_SPECIALIZATION(GLdouble,             3, glUniform3dv )
CALL_SETTER_VECTOR_SPECIALIZATION(GLdouble,             4, glUniform4dv )

CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 4, 4, glUniformMatrix4fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 3, 3, glUniformMatrix3fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 2, 2, glUniformMatrix2fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 4, 3, glUniformMatrix4x3fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 4, 2, glUniformMatrix4x2fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 3, 4, glUniformMatrix3x4fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 2, 4, glUniformMatrix2x4fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 3, 2, glUniformMatrix3x2fv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLfloat, 2, 3, glUniformMatrix2x3fv)

CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     4, 4, glUniformMatrix4dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     3, 3, glUniformMatrix3dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     2, 2, glUniformMatrix2dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     4, 3, glUniformMatrix4x3dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     4, 2, glUniformMatrix4x2dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     3, 4, glUniformMatrix3x4dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     2, 4, glUniformMatrix2x4dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     3, 2, glUniformMatrix3x2dv)
CALL_SETTER_MATRIX_SPECIALIZATION(GLdouble,     2, 3, glUniformMatrix2x3dv)

/*

        Extract the type of the argument elements. In case of a scalar, this means the type of the argument. In case
        of containers this, boils down to the type of the array elements.

*/
template <typename T>                   struct ElementInfo { using value_type = T; };
template <typename T>                   struct ElementInfo<std::vector<T>> { using value_type = T; };
template <typename T, size_t N> struct ElementInfo<std::array<T, N>> { using value_type = T; };
template <typename T, size_t N> struct ElementInfo<T[N]> { using value_type = T; };

/*

        Compute the length of the container in _elements_ in types that our SetUniform supports.

*/
template <typename T>                   constexpr GLsizei ContainerLength(const T&) { return 1; }
template <typename T, size_t N> constexpr GLsizei ContainerLength(const std::array<T, N>&) { return N; }
template <typename T, size_t N> constexpr GLsizei ContainerLength(const T(&)[N]) { return N; }
template <typename T>                   constexpr GLsizei ContainerLength(const std::vector<T>& pArr) { return (GLsizei)pArr.size(); }

/*

Compute the length of the container in _bytes_.

*/
template <typename T>                   constexpr GLsizei ContainerSizeInBytes(const T&) { return sizeof(T); }
template <typename T, size_t N> constexpr GLsizei ContainerSizeInBytes(const std::array<T, N>&) { return N*sizeof(T); }
template <typename T, size_t N> constexpr GLsizei ContainerSizeInBytes(const T(&)[N]) { return N*sizeof(T); }
template <typename T>                   constexpr GLsizei ContainerSizeInBytes(const std::vector<T>& pArr) { return (GLsizei)pArr.size()*sizeof(T); }

/*

        Get a pointer to the beginning of the container. In case of non-contagious containers this should involve
        the creation of a contagious copy of the contents of the passed in containers.

```

```
190 */
191 template <typename T>                   constexpr const T* PointerToStart(const T&
pArg) { return &pArg; }
192 template <typename T, size_t N> constexpr const T* PointerToStart(const std::array<T,
N>& pArg) { return &pArg[0]; }
193 template <typename T, size_t N> constexpr const T* PointerToStart(const T(&pArg)[N])
{ return &pArg[0]; }
194 template <typename T>                   constexpr const T* PointerToStart(const
std::vector<T>& pArg) { return pArg.empty() ? nullptr: &pArg[0]; }
195
196 /*
197
198         Helper constexpr to determine if the supplied parameter is a supported
contagious container or not.
199
200 */
201
202 template<typename T> struct HasContiguousStorage : public std::false_type { };
203
204 template<typename T> struct HasContiguousStorage<std::vector<T>> : public
std::true_type {};
205 template<typename T, size_t N> struct HasContiguousStorage<std::array<T, N>> : public
std::true_type {};
206 template<typename T, size_t N> struct HasContiguousStorage< T[N]> : public
std::true_type {};
207
208 template <typename T, typename R = void>
209 using IsContiguousContainer = std::enable_if_t<HasContiguousStorage< T>::value, R>;
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/Mesh_OGL3.cpp File Reference

```
#include "Mesh_OGL3.h"
```

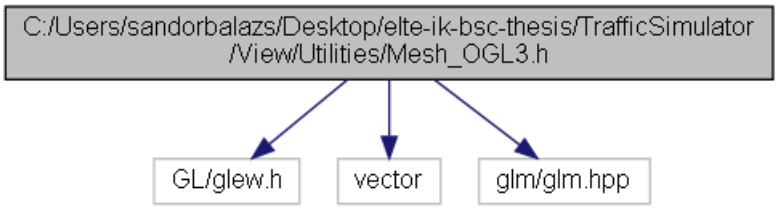
Include dependency graph for Mesh_OGL3.cpp:



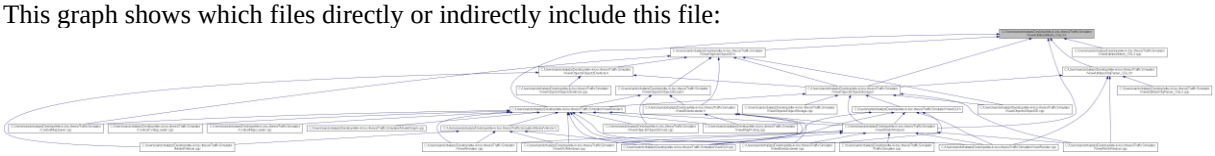
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/Mesh_OGL3.h File Reference

```
#include <GL/glew.h>
#include <vector>
#include <glm/glm.hpp>
```

Include dependency graph for Mesh_OGL3.h:



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



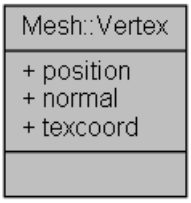
Data Structures

- class **Mesh**
- struct **Mesh::Vertex**

Data Structure Documentation

struct Mesh::Vertex

Collaboration diagram for Mesh::Vertex:



Data Fields:

vec3	normal	
vec3	position	
vec2	texcoord	

Mesh_OGL3.h

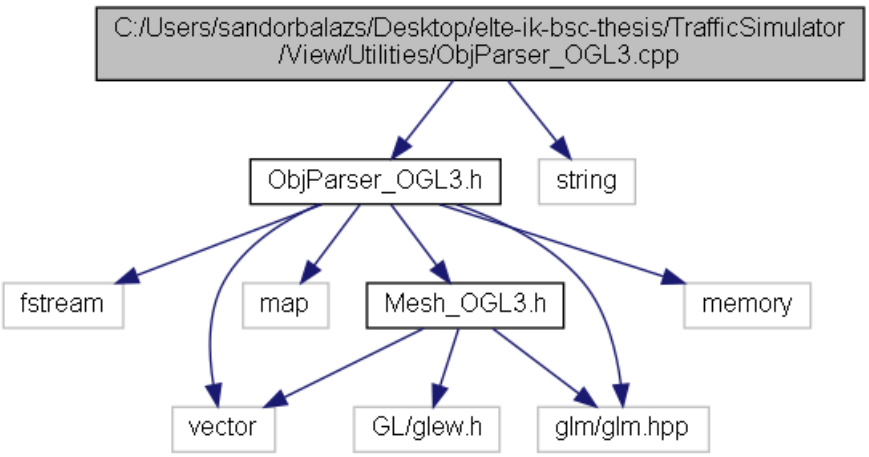
Go to the documentation of this file.

```
1 #pragma once
2
3 #include <GL/glew.h>
4
5 #include <vector>
6 #include <glm/glm.hpp>
7
8 class Mesh
9 {
10 public:
11         struct Vertex
12         {
13                 glm::vec3 position;
14                 glm::vec3 normal;
15                 glm::vec2 texcoord;
16         };
17
18         Mesh(void);
19         ~Mesh(void);
20
21         void initBuffers();
22         void draw();
23
24         void addVertex(const Vertex& vertex) {
25                 vertices.push_back(vertex);
26         }
27         void addIndex(unsigned int index) {
28                 indices.push_back(index);
29         }
30 private:
31         GLuint vertexArrayObject;
32         GLuint vertexBuffer;
33         GLuint indexBuffer;
34
35         std::vector<Vertex> vertices;
36         std::vector<unsigned int> indices;
37
38         bool inited = false;
39 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ObjParser_OGL3.cpp File Reference

```
#include "ObjParser_OGL3.h"
#include <string>
```

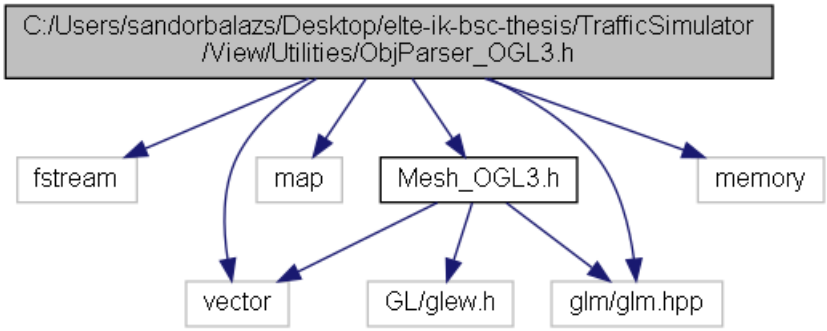
Include dependency graph for ObjParser_OGL3.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ObjParser_OGL3.h File Reference

```
#include <fstream>
#include <vector>
#include <map>
#include <glm/glm.hpp>
#include "Mesh_OGL3.h"
#include <memory>
```

Include dependency graph for ObjParser_OGL3.h:



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

Data Structures

- class **ObjParser**
- struct **ObjParser::IndexedVert**

ObjParser_OGL3.h

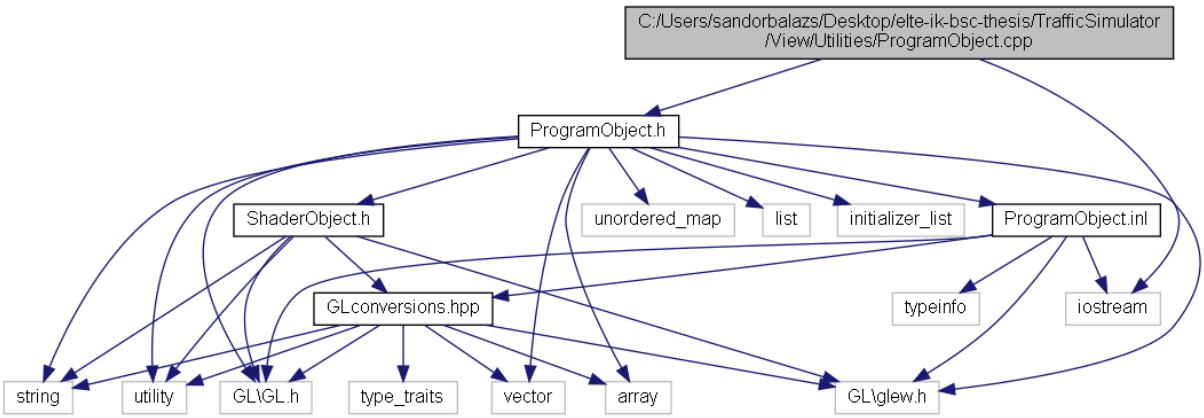
Go to the documentation of this file.

```
1 #pragma once
2
3 // Based on sroccaserra's (https://sourceforge.net/users/sroccaserra/) Obj loader
4 // http://sourceforge.net/projects/objloader/
5
6 #include <fstream>
7 #include <vector>
8 #include <map>
9 #include <glm/glm.hpp>
10
11 #include "Mesh_OGL3.h"
12
13 #include <memory>
14
15 class ObjParser
16 {
17 public:
18         static std::unique_ptr<Mesh> parse(const char* fileName, bool parallel =
false);
19
20         enum Exception { EXC_FILENOTFOUND };
21 private:
22         struct IndexedVert {
23                 int v, vt, vn;
24                 IndexedVert(int _v, int _vt, int _vn) : v(_v), vt(_vt), vn(_vn) {};
25                 bool operator<(const IndexedVert& rhs) const {
26                         return v < rhs.v || (v == rhs.v && (vt < rhs.vt || (vt == rhs.vt
&& vn < rhs.vn)));
27                 }
28         };
29
30         ObjParser(void) : mesh(0), nIndexedVerts(0) {}
31
32         bool processLine();
33         bool skipCommentLine();
34         void skipLine();
35         void addIndexedVertex(const IndexedVert& vertex);
36
37         Mesh* mesh;
38         std::ifstream ifs;
39
40         std::vector<glm::vec3> positions;
41         std::vector<glm::vec3> normals;
42         std::vector<glm::vec2> texcoords;
43
44         unsigned int nIndexedVerts;
45         std::map<IndexedVert, unsigned int> vertexIndices;
46 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ProgramObject.cpp File Reference

```
#include "ProgramObject.h"
#include <iostream>
```

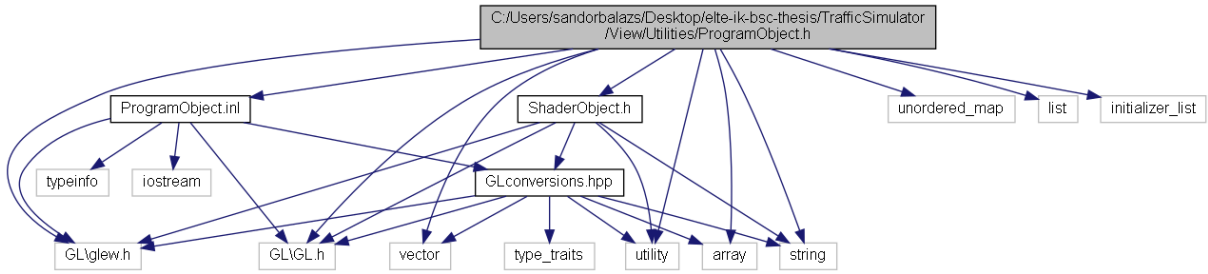
Include dependency graph for ProgramObject.cpp:



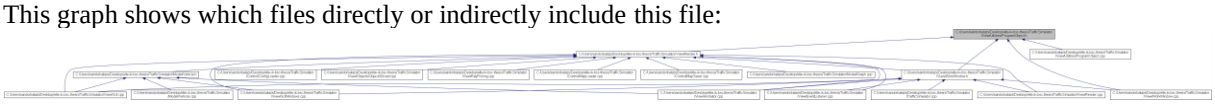
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ProgramObject.h File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include "ShaderObject.h"
#include <unordered_map>
#include <list>
#include <string>
#include <vector>
#include <array>
#include <initializer_list>
#include <utility>
#include "ProgramObject.inl"
```

Include dependency graph for ProgramObject.h:



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



Data Structures

- class **ProgramObject**

ProgramObject.h

Go to the documentation of this file.

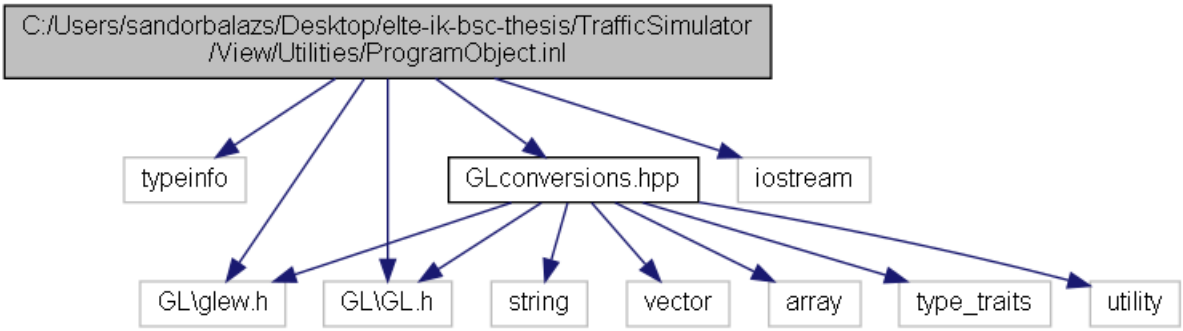
```
1 #pragma once
2
3 #include <GL\glew.h>
4 #include <GL\GL.h>
5
6 #include "ShaderObject.h"
7
8 #include <unordered_map>
9 #include <list>
10 #include <string>
11 #include <vector>
12 #include <array>
13
14 #include <initializer_list>
15 #include <utility>
16
17 class ProgramObject final
18 {
19 public:
20         using Binding = std::pair<int, const char*>;
21
22         ProgramObject();
23         ~ProgramObject();
24
25         void createProgram();
26
27         ProgramObject(std::initializer_list<ShaderObject>, std::initializer_list<
Binding > = {}, std::initializer_list< Binding > = {});
28
29         ProgramObject(const ProgramObject&)                              = delete;
30         ProgramObject& operator=(const ProgramObject&)  = delete;
31
32         ProgramObject(ProgramObject&& rhs);
33         ProgramObject& operator=(ProgramObject&& rhs);
34
35         operator unsigned int() const { return m_id; }
36
37         void Init(std::initializer list<ShaderObject>, std::initializer list<
Binding > = {}, std::initializer_list< Binding > = {});
38         void Clean();
39
40         ProgramObject& AttachShader(const ShaderObject&);
41         ProgramObject& AttachShaders(std::initializer_list<ShaderObject>);
42
43         void BindAttribLocation(int, const char*) const;
44         void BindAttribLocations( std::initializer_list< Binding > ) const;
45
46         void BindFragDataLocation(int, const char*) const;
47         void BindFragDataLocations(std::initializer_list< Binding >) const;
48
49         bool LinkProgram();
50
51         void SetTexture(const char* _uniform, int _sampler, GLuint _textureID);
52         void SetCubeTexture(const char* _uniform, int _sampler, GLuint _textureID);
53
54         template<typename U, typename T>
55         void SetUniform(U _uniform, const T& pArr);
56
57         GLint   GetLocation(const char* _uniform);
58
59         void Use() const;
60         void Unuse() const;
61 private:
62         GLuint m_id;
63
64         std::unordered_map< std::string, GLint >        m_map_uniform_locations;
65         std::vector< GLuint >
m_list_shaders_attached;
66
67         GLint GLResolveUniformLocation(GLint _uniform);
68         GLint GLResolveUniformLocation(const char* _uniform);
69 };
70
```

```
#include "ProgramObject.inl"
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ProgramObject.inl File Reference

```
#include <typeinfo>
#include <GL\glew.h>
#include <GL\GL.h>
#include <iostream>
#include "GLconversions.hpp"
```

Include dependency graph for ProgramObject.inl:

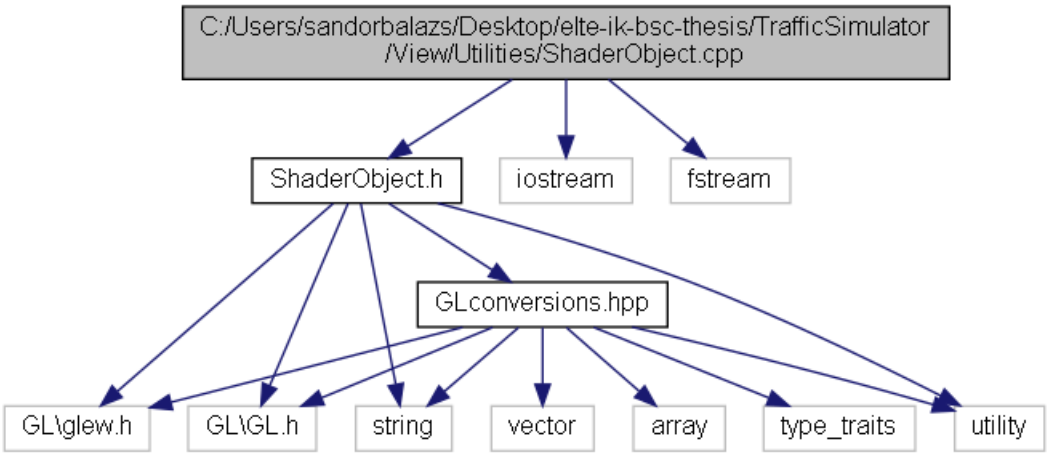


This graph shows which files directly or indirectly include this file:

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ShaderObject.cpp File Reference

```
#include "ShaderObject.h"
#include <iostream>
#include <fstream>
```

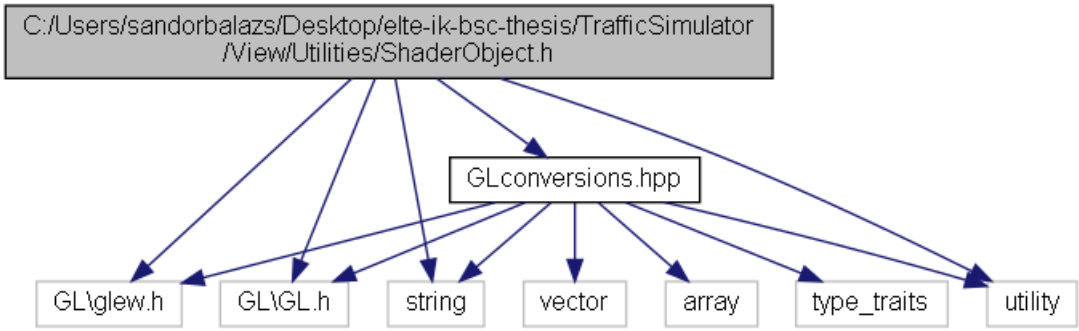
Include dependency graph for ShaderObject.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/ShaderObject.h File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include <string>
#include <utility>
#include "GLconversions.hpp"
```

Include dependency graph for ShaderObject.h:



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

Data Structures

- class **ShaderObject**

ShaderObject.h

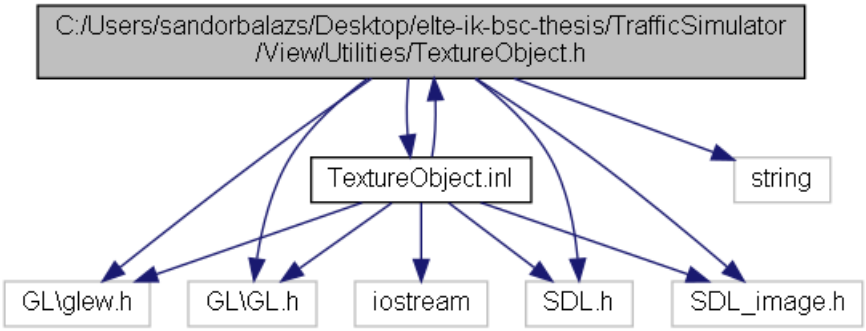
Go to the documentation of this file.

```
1 #pragma once
2
3 #include <GL\glew.h>
4 #include <GL\GL.h>
5
6 #include <string>
7 #include <utility>
8
9 #include "GLconversions.hpp"
10
11 class ShaderObject final
12 {
13 public:
14         ShaderObject(GLenum pType);
15         ShaderObject(GLenum pType, const std::string&);
16         ShaderObject(const TypeSourcePair& pInfo) : ShaderObject(pInfo.first,
pInfo.second) {}
17
18         ~ShaderObject();
19
20         ShaderObject(const ShaderObject&)                                = delete;
21         ShaderObject& operator=(const ShaderObject&)    = delete;
22
23         ShaderObject(ShaderObject&&);
24         ShaderObject& operator=(ShaderObject&&);
25
26         operator unsigned int() const { return m_id; }
27
28         bool FromFile(GLenum _shaderType, const char* _filename);
29         bool FromMemory(GLenum _shaderType, const std::string& _source);
30 private:
31         GLuint  CompileShaderFromMemory(const GLuint _shaderObject, const
std::string& _source);
32
33         GLuint  m_id;
34 };
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/TextureObject.h File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include <SDL.h>
#include <SDL_image.h>
#include <string>
#include "TextureObject.inl"
```

Include dependency graph for TextureObject.h:



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

Data Structures

- class **TextureObject< type >**

Typedefs

- using **Texture2D** = **TextureObject**<>

Enumerations

- enum class **TextureType** { **Texture1D** = GL_TEXTURE_1D, **Texture2D** = GL_TEXTURE_2D, **Texture3D** = GL_TEXTURE_3D, **Texture1DArray** = GL_TEXTURE_1D_ARRAY, **Texture2DArray** = GL_TEXTURE_2D_ARRAY, **TextureRectangle** = GL_TEXTURE_RECTANGLE, **TextureCubeMap** = GL_TEXTURE_CUBE_MAP, **TextureCubeMapArray** = GL_TEXTURE_CUBE_MAP_ARRAY, **Texture2DMultisample** = GL_TEXTURE_2D_MULTISAMPLE, **Texture2DMultisampleArray** = GL_TEXTURE_2D_MULTISAMPLE_ARRAY }

Typedef Documentation

using Texture2D = TextureObject<>

Enumeration Type Documentation

enum class TextureType [strong]

Enumerator:

Texture1D	
Texture2D	
Texture3D	
Texture1DArray	
Texture2DArray	
TextureRectangle	
TextureCubeMap	
TextureCubeMap Array	
Texture2DMultisa mple	
Texture2DMultisa mpleArray	

TextureObject.h

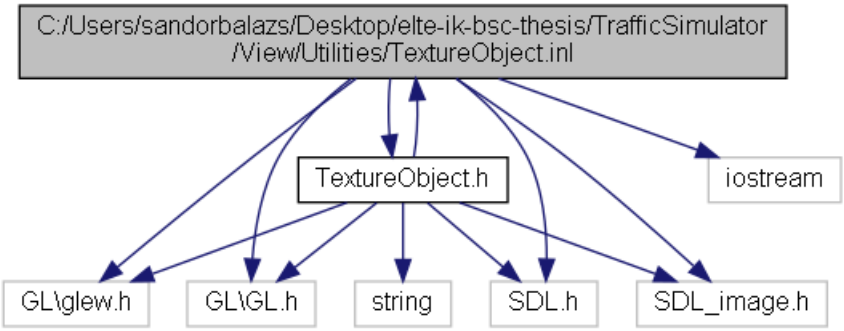
Go to the documentation of this file.

```
1 #pragma once
2
3 #include <GL\glew.h>
4 #include <GL\GL.h>
5 #include <SDL.h>
6 #include <SDL_image.h>
7 #include <string>
8
9 enum class TextureType
10 {
11         Texture1D                                    = GL_TEXTURE_1D,
12         Texture2D                                    = GL_TEXTURE_2D,
13         Texture3D                                    = GL_TEXTURE_3D,
14         Texture1DArray                         = GL_TEXTURE_1D_ARRAY,
15         Texture2DArray                         = GL_TEXTURE_2D_ARRAY,
16         TextureRectangle                       = GL_TEXTURE_RECTANGLE,
17         TextureCubeMap                         = GL_TEXTURE_CUBE_MAP,
18         TextureCubeMapArray                    = GL_TEXTURE_CUBE_MAP_ARRAY,
19         Texture2DMultisample           = GL_TEXTURE_2D_MULTISAMPLE,
20         Texture2DMultisampleArray      = GL_TEXTURE_2D_MULTISAMPLE_ARRAY
21 };
22
23 template <TextureType type = TextureType::Texture2D>
24 class TextureObject final
25 {
26 public:
27         TextureObject();
28         TextureObject(const std::string&);
29         ~TextureObject();
30
31         TextureObject(const TextureObject&) = delete;
32         TextureObject& operator=(const TextureObject&) = delete;
33
34         TextureObject(TextureObject&& rhs);
35         TextureObject& operator=(TextureObject&& rhs);
36
37         TextureObject& operator=(const std::string& s);
38
39         void AttachFromFile(const std::string&, bool generateMipMap = true, GLuint role
= static_cast<GLuint>(type));
40         void FromFileParallel(const std::string&);
41         void FromFile(const std::string&);
42         void bindTexture();
43
44         operator unsigned int() const { return m_id; }
45
46         void Clean();
47
48 private:
49         GLuint m_id{};
50         SDL_Surface* loaded_img;
51         SDL_Surface* formattedSurf;
52 };
53
54 #include "TextureObject.inl"
55
56 using Texture2D = TextureObject<>;
```

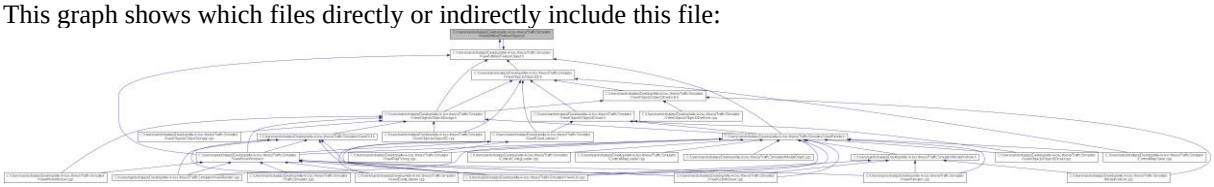
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/TextureObject.inl File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include "TextureObject.h"
#include <iostream>
#include <SDL.h>
#include <SDL_image.h>
```

Include dependency graph for TextureObject.inl:



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



Functions

- int **transform_image** (int pitch, int height, void *image_pixels)

Function Documentation

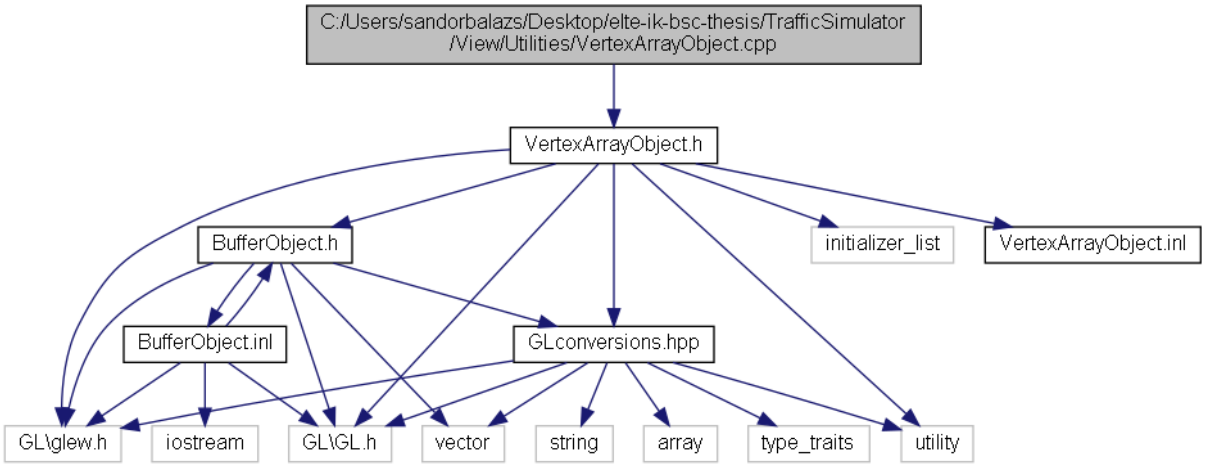
int transform_image (int *pitch*, int *height*, void * *image_pixels*) `[inline]`

Here is the caller graph for this function:

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/VertexArrayObject.cpp File Reference

```
#include "VertexArrayObject.h"
```

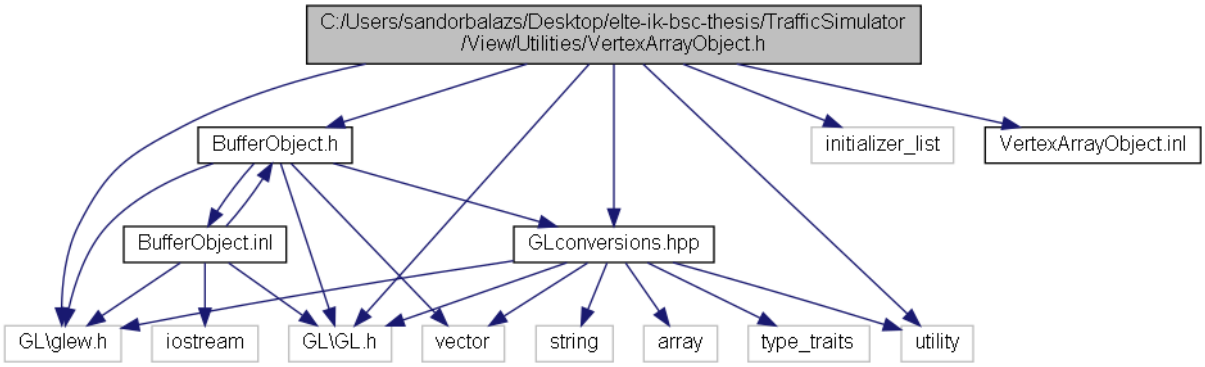
Include dependency graph for VertexArrayObject.cpp:



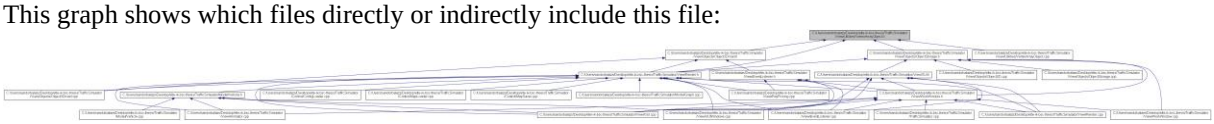
C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/VertexArrayObject.h File Reference

```
#include <GL\glew.h>
#include <GL\GL.h>
#include "GLconversions.hpp"
#include "BufferObject.h"
#include <initializer_list>
#include <utility>
#include "VertexArrayObject.inl"
```

Include dependency graph for VertexArrayObject.h:



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



Data Structures

- struct **AttributeData**
- class **VertexArrayObject**

Variables

- template<GLuint index, typename T , size_t pOffsetInBytes = 0, GLsizei pStrideInBytes = sizeof(T)> **AttributeData CreateAttribute** = **AttributeData**(index, sizeof(T) / sizeof(typename **GLExtractPrimitiveType**<T>::primitive_type), **NativeTypeToOpenGLType**<typename **GLExtractPrimitiveType**<T>::primitive_type>(), false, pStrideInBytes, (void*)pOffsetInBytes)

Variable Documentation

template<GLuint index, typename T , size_t pOffsetInBytes = 0, GLsizei pStrideInBytes = sizeof(T)> AttributeData CreateAttribute = AttributeData(index, sizeof(T) / sizeof(typename GLExtractPrimitiveType<T>::primitive_type), NativeTypeToOpenGLType<typename GLExtractPrimitiveType<T>::primitive_type>(), false, pStrideInBytes, (void*)pOffsetInBytes)

VertexArrayObject.h

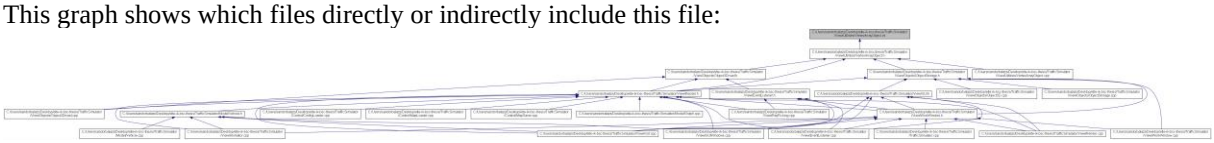
Go to the documentation of this file.

```
1 #pragma once
2
3 #include <GL\glew.h>
4 #include <GL\GL.h>
5
6 #include "GLconversions.hpp"
7 #include "BufferObject.h"
8
9 #include <initializer_list>
10 #include <utility>
11
12 struct AttributeData
13 {
14         AttributeData(GLuint pIndex, GLint pSize, GLenum pType, GLboolean pNormalized,
GLsizei pStride, void* pPtr) : index(pIndex), size(pSize), type(pType),
normalized(pNormalized), stride(pStride), ptr(pPtr)
15         { }
16
17         void Apply()
18         {
19                 glEnableVertexAttribArray(index);
20                 glVertexAttribPointer(index, size, type, normalized, stride, ptr);
21         }
22
23         GLuint          index{};
24         GLint           size{};
25         GLenum          type{};
26         GLboolean       normalized{};
27         GLsizei         stride{};
28         void*           ptr{};
29 };
30
31 // using template variable for ease of notation on call side
32 template <GLuint index, typename T, size_t pOffsetInBytes = 0, GLsizei pStrideInBytes
= sizeof(T)>
33 AttributeData CreateAttribute = AttributeData(index, sizeof(T) / sizeof(typename
GLExtractPrimitiveType<T>::primitive_type), NativeTypeToOpenGLType<typename
GLExtractPrimitiveType<T>::primitive type>(), false, pStrideInBytes,
(void*)pOffsetInBytes);
34 // otherwise, the following template function is more readable and serves as the function
equivalent of the above template variable:
35 /*
36 template <GLuint index, typename T>
37 AttributeData CreateAttribute(size t pOffsetInBytes = 0, GLsizei pStrideInBytes =
sizeof(T))
38 {
39         using PrimitiveType = typename GLExtractPrimitiveType<T>::primitive_type;
40         return AttributeData(index, sizeof(T) / sizeof(PrimitiveType),
NativeTypeToOpenGLType<PrimitiveType>(), false, pStrideInBytes, (void*)pOffsetInBytes);
41 }
42 */
43
44
45 class VertexArrayObject final
46 {
47 public:
48         VertexArrayObject();
49         ~VertexArrayObject();
50
51         VertexArrayObject(const VertexArrayObject&) = delete;
52         VertexArrayObject& operator=(const VertexArrayObject&) = delete;
53
54         VertexArrayObject(VertexArrayObject&&);
55         VertexArrayObject& operator=(VertexArrayObject&&);
56
57         operator unsigned int() const { return m_id;};
58
59         void Init(std::initializer_list< std::pair<AttributeData, const ArrayBuffer&>
> pDataBuffers);
60
61         void Init(std::initializer_list< std::pair<AttributeData, const ArrayBuffer&>
> pDataBuffers, const IndexBuffer& pIndexBuffer);
62
```

```
63          VertexArrayObject& Bind();
64          void Unbind();
65
66          template <BufferType type, BufferUsage usage>
67          VertexArrayObject& AddAttribute(AttributeData&, BufferObject<type, usage>&);
68
69          VertexArrayObject& SetIndices(const IndexBuffer&);
70
71 private:
72          GLuint m_id = 0;
73 };
74
75 #include "VertexArrayObject.inl"
```

C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/Utilities/VertexArrayObject.inl File Reference

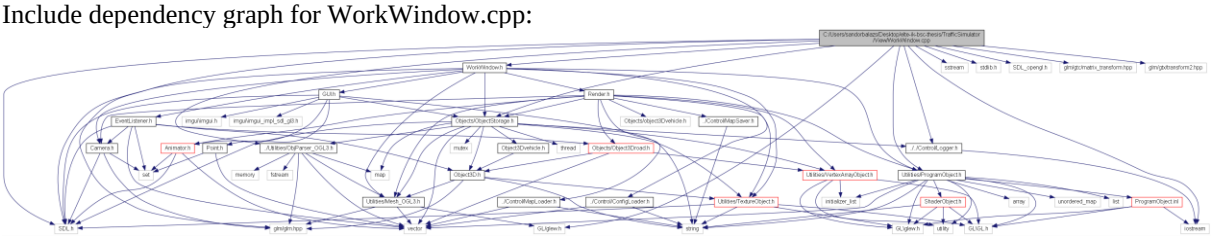
This graph shows which files directly or indirectly include this file:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/WorkWindow.cpp File Reference

```
#include "WorkWindow.h"
#include <iostream>
#include <sstream>
#include <stdlib.h>
#include <GL/glew.h>
#include <SDL.h>
#include <SDL_opengl.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform2.hpp>
#include "../Control/Logger.h"
#include "Objects/ObjectStorage.h"
#include "Utilities/ProgramObject.h"
#include "Utilities/ObjParser_OGL3.h"
```

Include dependency graph for WorkWindow.cpp:



C:/Users/sandorbalazs/Desktop/elte-ik-bsc-thesis/TrafficSimulator/View/WorkWindow.h File Reference

```
#include <SDL.h>
#include "Camera.h"
#include "Render.h"
#include "EventListener.h"
#include "GUI.h"
#include "Utilities/ProgramObject.h"
#include "Utilities/TextureObject.h"
#include "Utilities/Mesh_OGL3.h"
#include "Objects/ObjectStorage.h"
#include "../Control/ConfigLoader.h"
```

Include dependency graph for WorkWindow.h:

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

Data Structures

- class **WorkWindow**
  The OpenGL 3D window implementation.

WorkWindow.h

Go to the documentation of this file.1

```
11 #pragma once
12
13 #include <SDL.h>
14 #include "Camera.h"
15 #include "Render.h"
16 #include "EventListener.h"
17 #include "GUI.h"
18 #include "Utilities/ProgramObject.h"
19 #include "Utilities/TextureObject.h"
20 #include "Utilities/Mesh_OGL3.h"
21 #include "Objects/ObjectStorage.h"
22 #include "../Control/ConfigLoader.h"
23
27 class WorkWindow {
28 public:
29
33         WorkWindow(void);
34
38         ~WorkWindow(void);
39
44         int open();
45
49         void close();
50
54         void closeNow();
55
60         Camera* getCamera();
61
66         ProgramObject* getShader();
67
72         SDL_Window* getWindow();
73
78         std::string getWindowTitle();
79
84         GUI* getGUI();
85
90         ObjectStorage* getObjectStorage();
91
96         Render* getRender();
97
102         EventListener* getEventListener();
103
104 private:
105
109         const char* windowDefTitle = "Traffic Simulator";
110
114         const int windowDefSizeX = 640;
115
119         const int windowDefSizeY = 480;
120
124         const int windowDefPosX = 100;
125
129         const int windowDefPosY = 100;
130
134         std::string windowTitle = "Traffic simulator";
135
139         SDL_Window*             window;
140
144         SDL_GLContext   context;
145
149         ProgramObject   shader;
150
154         Camera                  camera;
155
159         EventListener   eventListener;
160
164         Render                  render;
165
169         ConfigLoader configLoader;
170
174         GUI                             gui;
175
```

```
179         ObjectStorage   objectStorage;
180
184         bool exit = false;
185
190         int sdlInit();
191
196         int openGLpreConfig();
197
202         int openSDLWindow();
203
208         int imGUIinit();
209
214         int createOpenGLContext();
215
220         int openGLpostConfig();
221
226         int glewStart();
227
232         int shaderConfig();
233
238         int renderPreconfig();
239
244         int renderStart();
245
249         void clearScreen();
250
254         void cleanup();
255
259         static void exitWindow();
260
267         int displayMode = 0;
268 };
```

Index

INDEX