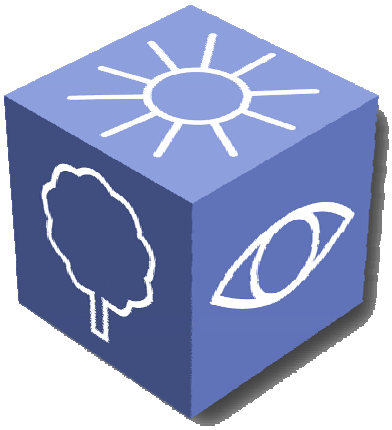


GAMETOOLS

ADVANCED TOOLS FOR DEVELOPING HIGHLY REALISTIC COMPUTER GAMES

DUMMY MODULE FOR ILLUMINATION ALGORITHMS

Document identifier:	**GameTools-5-D5.1-01-1-0-Dummy Module for Illumination Algorithms**
Date: (use "update field" Word function, right mouse button)	**28/04/2005**
Work package:	**WP05: Illumination**
Partner(s):	**BUTE, UdG, Unilim**
Leading Partner:	**BUTE**
Document status:	**DRAFT**
Deliverable identifier:	**D5.1**

Abstract: Dummy Module Report for Illumination Algorithms

Delivery Slip

	Name	Partner	Date	Signature
From	László Szirmay-Kalos	BUTE	25-04-2005	
Reviewed by	Moderator and reviewers	ALL	28-04-2005	
Approved by	Moderator and reviewers	ALL	28-04-2005	

Document Log

Issue	Date	Comment	Author
1-0	25-04-2005	First draft	László Szirmay-Kalos
1-1	28-04-2005	Final Version	László Szirmay-Kalos

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files / URL
Word	gametools-ist-2-004363-5-d5.1-01-1-1-dummy module for illumination algorithms.doc (use “update field” Word function)

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Overview

The illumination workpackage contains modules to compute the direct and indirect illumination of the surfaces taking into account point, directional, area and environment lighting. These algorithms are responsible for calculating the color of the visible points based on object geometry, material data, and lighting conditions. To aim at interactive rates, the proposed algorithms take simplifying assumptions. The modules solve rendering problems taking different compromises between speed and image quality and are good for simulating different lighting phenomena. The modules are similar in that they all consist of a two main phases, called as pre-computing runs and final rendering runs. Pre-computing runs render into textures, which can store not only color but also geometry, shadow, indirect illumination etc. information. Then, in the final rendering run, this information is used to obtain the final image. The modules have complementary character, thus they should be selected according to the requirements of the actual game, or may even be combined to compute different components of the light transport. In the following sections, we first introduce our convenience classes encapsulating Ogre Materials and render textures, used to realize separate passes. Then, the software architecture is described, how different computation modules are organized into a hierarchy and are composed into a rendering pipeline. Finally, the particular computational modules are described.

3.1 Convenience classes of the illumination module

The class **OgreEffectWrapper** encapsulates shader setup and uniform parameter setup. It uses convenience functions for materials with one pass per technique. It hides the complexity of the engine's uniform parameter, and texture unit setup and shader management. The class **ManagedOgreRenderTexturePass** encapsulates a rendering pass. Currently we can render either with the default graphics pipeline, or we can use vertex and pixel shaders to make dynamic calculations in the pipeline. The default pipeline can be simulated via shaders, so we are currently supporting passes that use shaders only. A typical rendering pass encapsulates rendering to a buffer, which can be either the back buffer, any render texture surface, including cube map render targets. Modern 3D hardware is also capable of rendering to multiple textures at once (the feature is called Multiple Render Targets, **MRT**s). If this feature is set up correctly, then multiple outputs of the current pixel shader are written to multiple textures. While the OGRE engine has a feature with the same name, it cannot handle multiple shader outputs at once, but instead renders to multiple viewports at the same time. The OGRE engine is already supporting back buffers and simple render to textures. The class uses a new method we created to support MRTs, and cube map face render targets. To do this we had to re-implement the OGRE methods **RenderSystem::setViewport()** and **SceneManager::manualRender()**. These were low-level APIs to the underlying rendering system. Currently we support these features for the Direct3D9 rendering system. The class has a vector of render targets and one of cube map faces. They are paired and set to render on. If the current render target is not a cube map, then the cube map face parameter is ignored. After using MRTs, the render system switches the feature off. Shaders are supported via OGRE

material scripts. Transform matrices are automatically set from material scripts. They are extracted from the first render target's first viewport's camera. For a pass to work we need to set up the texture units and the uniform parameters of the shaders correctly. This is done via an instance of the **OgreEffectWrapper** convenience class.

3.2 Workpackage software architecture

The illumination workpackage rendering modules are composed to create an interface for rendering objects very much similar to the standard rendering pipeline in OGRE. However, the illumination of an entity is computed in a more complex manner. Several texture maps have to be pre-computed, possibly also involving intermediate computation textures, and final shading can also include multiple steps. Therefore, a single Material is not sufficient to describe shading properties any more.

Additional shading information is stored in **EntityRenderingObjects**, derived from **UserDefinedObject**. So that other custom data can still be attached to Entities, **MultipleUserDefinedObject** is introduced, which is a hash map to include any number of user defined objects. **MultipleUserDefinedObject** is derived from **UserDefinedObject** itself. Actually, every Entity should own a **MultipleUserDefinedObject**, containing an **EntityRenderingObject**. It is the responsibility of the creator of the Entity to attach an appropriate **EntityRenderingObject**, describing the desired rendering qualities for the Entity.

An **EntityRenderingObject** encapsulates a **FinalRenderingRun**, and multiple **PreComputingRuns**. The **FinalRenderingRun** describes how the Entity is rendered to the screen, and what pre-computed resources are required, updated with what frequency. The **FinalRenderingRun** subclass and parametrization, and the **PreComputingRun**s' parametrization are all extracted from the **RenderingType** structure, passed to the constructor of the **EntityRenderingObject**.

A **RenderingRun** is an empty superclass conceptualizing the common structure of **PreComputingRun** and **FinalRenderingRun** derived classes. Typically, they all will be composed of passes derived from **ManagedOgreRenderTexturePath** (note the associated render target may be the back buffer).

PreComputingRun is the superclass for algorithms computing any kind of illumination data. Typically, these are going to be textures. A class derived from **PreComputingRun** stores some pre-computed information, and has an overloaded update method, which is supposed to re-compute the stored data. Typically, the constructor of a class derived from **PreComputingRun** accepts a few parameters describing the owner Entity and computation quality (number of samples, map resolution, etc.), but it does not reveal all the parameters of underlying passes. These parameters should be included in the **RenderingType** structure, and used by **EntityRenderingObject** to instantiate the **PreComputingRun**s.

Instances of classes derived from **PreComputingRun** are stored with objects to with the pre-computed data is associated. Most of them are stored with Entities (e.g. illumination maps), some with lights (e.g. depth maps), and some are global (e.g. impostors). A **PreComputingRun** may not keep resource-heavy intermediate results (textures) in a persistent manner.

A **FinalRenderingRun** is internally composed similarly to a **PreComputingRun**, save for the fact that the last pass (**ManagedOgreRenderTexturePath**) renders to a specified render target, rather than to an own render texture. Typically, for simple illumination, there is only one pass encapsulating a Material that applies the appropriate shading and textures. On the interface level, however, **FinalRenderingRun** has a rich set of methods querying illumination properties, used to create and regularly update pre-computed data used by the **FinalRenderingRun**-derived instance. A **FinalRenderingRun** is always stored with an Entity, and describes how it is rendered. In every frame, for every Entity, its **FinalRenderingRun** is queried to determine which pre-computed data are to be updated. Thereafter, for every Entity, the **FinalRenderingRun::renderSingleEntity** method is invoked, specifying a render target. The method should perform its internal passes, finally invoking the final pass to render to the render target (typically the back buffer of an environment cube face).

The **IlluminationManager** is a wrapper to manage extended resources (those external to to OGRE

pipeline) and invoke rendering operations. There are some global parameters to be set like light depth map resolution. Lists of different OGRE lights, complete with additional information and pre-computed data, are stored, and synchronized to actual OGRE lights in every frame.

IlluminationManager::update is the method to invoke the Illumination Work-package rendering pipeline. First, light data is synchronized, and, if necessary, maps are updated (method **updateStaticLightData**). Then, for every Entity, the pre-computing runs due in the frame are executed, updating its maps. Thereafter every Entity is rendered, one by one, using their own **FinalRenderingRun**-derived instances for shading, building up the scene on the back buffer.

3.2.1 Environment map generation modules (classes FrensnelEnvironmentMapRun, DiffuseEnvironmentMapRun, SpecularEnvironmentMapRun)

Environment map generation modules gather information about the scene from given reference points and store this information in cube maps. The classes represent a pre-processing rendering phase responsible for the refreshing of the environment **cubemap**. The scene is rendered to the cubemap faces with the same methods used for the main back buffer. The only difference is that the alpha channel is used to hold the pixels's distance from the center of the cube. This can be done with a simple shader based ambient-like pass.

- The **FresnelEnvironmentMapRun** generates a cubemap for ideal reflections and refractions, that stores illumination of a reference point for each incoming direction together with the distance of the visible point from the reference point. During final gathering, for ideal reflections, illumination of the reference point can be determined with a simple lookup (more precisely, the incoming radiance is weighted with the Fresnel term - hence the name of the module). For points other than the reference point, we apply a simple iteration scheme based on the distance values, which provides a good approximation of the visible point. To maintain good image quality, in case of moving objects the cubemap is regularly recalculated.
- In case of diffuse surfaces, determining the irradiance at a point requires an integral over the hemisphere defined by the surface normal. To compute this integral efficiently, we prepare a diffuse map in the **DiffuseEnvironmentMapRun** to store these integral values for a reference point. The diffuse map stores the pre-computed convolution between the cosine function and the incoming illumination together with the average distance from the reference point. Thus, the diffuse illumination of the reference point can be determined with a simple lookup towards the direction of the surface normal. For other points, we use the distance information to obtain precise results.
- In case of specular surfaces, **SpecularEnvironmentMapRun** generates separate cubemaps for each shininess level. Each cubemap is prepared in a way similar to the diffuse case, except that the lookup direction is not the normal but the reflection direction.

The <FEM/DEM/SEM>FinalRenderingRun class implements the final rendering phase of the environment mapping. It makes use of the cubemap rendered by <Fresnel/Diffise/Specular>EnvironmentMapRun. This pass can be considered as the environment illuminating the object as a composite light. The actual rendering and shader managment is done with a **FEMFinalGatheringPass** object. It performs shading computations and localized environment map look-ups in a complex shader.

3.2.2 The caustics generation module (class CausticMapRun)

The caustics generator module creates caustic effect for caustic generator objects. The algorithm runs on GPU. The algorithm renders the scene from the light source position, where the generation of a pixel value is equivalent to sending a ray through the pixel. The rays which hit the caustic generator object are

refracted on its surface. With the help of the previously generated cubemaps, from the refraction directions the algorithm generates hit positions on the surface of the light receiver. These hit points are stored in a texture. This texture is used the corresponding final rendering run when the light receiver object is rendered. The algorithm renders a Photon Impostor at every hit point into the reflected illumination texture of the light receiver object. If we have more light sources or more caustic generator objects, we run the algorithm for every light source, caustic receiver pairs.

3.2.3 Hard shadow module (classes PointLightDepthCubeRun, SpotLightDepthMapRun, DirectionalLightDepthMapRun)

Instances of classes **PointLightDepthCubeRun**, **SpotLightDepthMapRun**, **DirectionalLightDepthMapRun** are stored by the **IlluminationManager**, associated to OGRE lights of different types. They encapsulate a depth map texture rendered for the lights.

3.2.4 Soft shadow module (class SoftShadowMapRun)

The soft shadow module generates shadow map for the entities in the scene which can cast soft shadow, if at least one area light source is used. In case of area light sources the shadow edges are detected and filtered. Filtering is executed with regard to geometrical distances (between source and occluder, and between occluder and receiver) and the viewable size of the light source from the actual pixel. The output of the algorithm is a texture which contains the filtered soft shadow. If there are more area light sources in the scene, theirs effects are gathered into one texture. One output texture must be generated for every soft shadow caster. They will be used, when the shadow receiver object is rendered.

3.2.5 Cloud Rendering Module (class HPSLightIlluminationRun, HPSCompositeRun)

The cloud rendering module extends OGRE's particle rendering capabilities to achieve realistically shaded cloudy volumes. The algorithm also solves the problem of visual artifacts (clipping) during billboard particle rendering, with the use of depth impostors. To speed up rendering, the particles are treated hierarchically: they are grouped into blocks, and calculations are made per block instead of per particles. The realistic shading is made with a use of a light illumination texture, which stores the extinct light intensity within the volume in different depths. To display the billboards without artifacts, a depth impostor texture of a particle block is generated. During final rendering (class **HPSFinalRenderingRun**) we compute light illumination texture for a particle system made of particle blocks. With the use of the depth impostor texture of a particle block, visually correct display of volume data can be achieved, as we set the density of a single block according to how much a ray can travel in the block until it reaches an object (if the object is in front of the block density will be zero, if the object is behind the block full density will be used, if the object is located in the block the density will be scaled). This calculation needs the scene depth information.

3.2.6 Stochastic iteration global illumination module (class LightMapRun)

The stochastic iteration global illumination module implements a GPU based finite-element type global illumination method assuming only diffuse reflections of the multiple light bounces and diffuse-glossy light reflection towards the eye. The algorithm first renders all emissive triangles into an emission map then calculates a map from the patch indices. The next step is choosing a shooter. After finding a suitable shooter, the scene is rendered from the point of view of the shooter, generating a depth map for all five **hemicube** sides. Then we calculate the lighting contribution for all five sides. After that we average the current radiosity with the existing and start again from choosing a new shooter. We do that until we get a radiosity representation.

3.2.7 Photon mapping module (class PhotonMapFilteringRun)

The photon mapping module represents pre-computing runs that generate photon hits in a texture by ray-shooting, and execute photon map filtering corresponding to the surface and scene parameters. The filtering method needs the area and normal properties of the entity's surface, which are collected into two texture maps. These texture maps called area map and normal map. While the area map is generated parallel with the photon map generation by the CPU and is represented by the member **areaMap** texture pointer, the normal map is generated as one of the first steps of the filtering phase. The final result of the filtering task can be obtained by getting the render texture object of the member **photonMapFilteringPass**, which is a **PMFFilteringPass** type pointer. The **PMFFilteringPass** class has three important parameters: area compensation, normal threshold and filter kernel type. These parameters can be modified calling their setter methods.

3.2.8 Precomputed radiance map module (class PRMRun)

PRMRun computes the **Pre-computed Radiance Map** (or **PRM** for short) for an entity, encoding self-illumination properties. Ray-casting and the virtual light sources method is used to render the illumination caused by unit irradiance incident at reference points. In the final rendering run, the tile of the PRM can be combined according to the actual lighting conditions, to render the self-illumination of the object.

3.2.9 Image based lighting module (class ImageLightingSamplesRun)

ImageLightingSamplesRun is a special **PreComputingRun**. It is global in the sense that it does not belong to an entity or light. Neither does it provide a texture as a result. It computes directional light samples of a HDR image using importance sampling, Bowyer-Watson algorithm for generating a Delanuay mesh to find the Voronoi decomposition, and Lloyd's relaxation. A corresponding **FinalRenderingRun** would use passes **IBLDepthMapPass** and **IBLBlendAddIllumPass** to render the contribution of image lighting to the scene.

3.2.10 Image based rendering module (class ImageBasedRenderRun)

The image based rendering module generates semi-transparent impostors to reduce the geometric complexity. The objective is to achieve an accurate representation of the original polygons applying simplifications using images. For the generation of trees a set of preprocessing steps are applied to the original tree meshes to generate image based impostors. The **PlanesGenerator** class implements a clustering algorithm, that form clusters of leaves in a way that all leaves belong to a cluster lie approximately on the same impostor plane and replace the whole group by a single impostor. The **TextureGenerator** class generates a single texture for each impostor plane that includes all the grouped leaves. Finally, the **SubMeshesPlanesGenerator** class compacts all the impostor planes generating a compact mesh that will replace the original complex geometry of the leaves.

3.2.11 Obscurances rendering module(class ObscuranceRun)

Obscurance rendering modules generate **obscurance map**s that are used to modulate ambient lighting during final rendering. This class has two children implementing different obscurance generation algorithms, namely, **ObscurancesRayTracingRun** works with CPU ray tracing and **ObscurancesDepthpeelingRun** runs on the GPU.

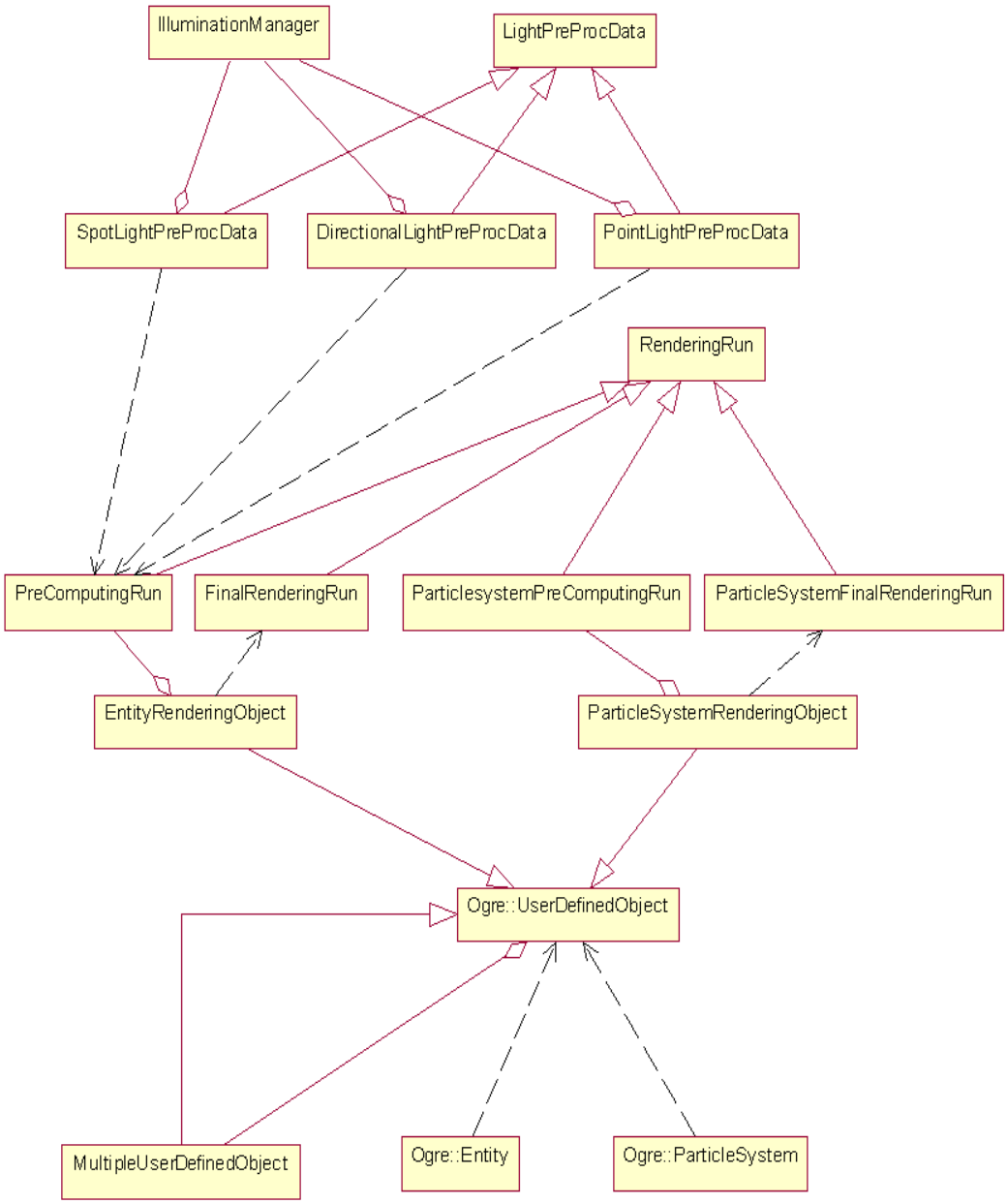


Figure 3.1: Class diagram: main classes

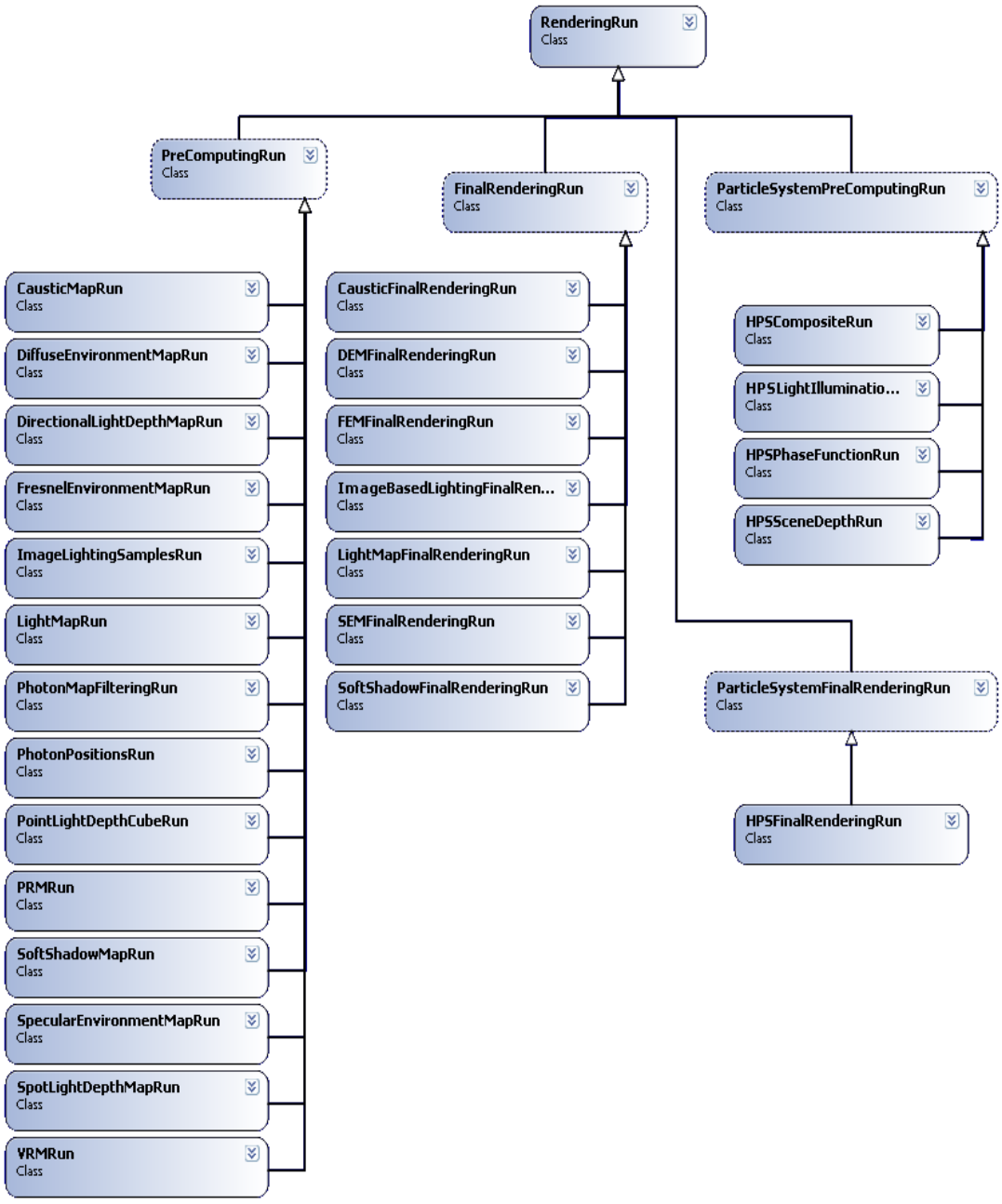


Figure 3.2: Class diagram: run classes

Chapter 4

Class Documentation

4.1 AdvancedParticleSystemManager Class Reference

Hierarcical, shaded particle system manager class.

Public Member Functions

- AdvancedParticleSystemManager (void)

  Constructor.

- ~AdvancedParticleSystemManager (void)

  Destructor.

- void setAllInvisible ()

  sets all entities and billboardsets invisible To restore previous visibility the AdvencedParticleSystemManager saves visibility information before hiding.

- void CreateHierSystem (String name, String blockscript, String pscript, String depthtexfront="", String depthtexback="")

  Creates a hierarchical particle system.

- void CreateShadableSystem (String name, String pscript, String depthtexfront="", String depthtexback="")

  Creates a shaded particle system.

- void UpdateSystems (unsigned long frameNumber)

  Updates the hierarchical and shaded systems created whith the manager.

Static Public Member Functions

- static void render (RenderTarget *rt, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)

- static void update (unsigned long frameNumber, RenderTarget *rt, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)

Static Public Attributes

- static std::vector< bool > entityVisibilityList

  Vector to store entity visibilities.

- static std::vector< bool > billboardVisibilityList

  Vector to store billboardset visibilities.

4.1.1 Detailed Description

Hierarcical, shaded particle system manager class.

This class is a helper class to simplify the creation and use of hierarchical and shaded particle systems

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AdvancedParticleSystemManager::AdvancedParticleSystemManager (void)

Constructor.

4.1.2.2 AdvancedParticleSystemManager::~AdvancedParticleSystemManager (void)

Destructor.

4.1.3 Member Function Documentation

4.1.3.1 void AdvancedParticleSystemManager::CreateHierSystem (String *name*, String *blockscript*, String *pscript*, String *depthtexfront* = "", String *depthtexback* = "")

Creates a hierarchical particle system.

A hierarchical particle system is made of particle system blocks, thus it needs two particle scripts.

Parameters:

name the name of the system

blockscript the particle script which describes the characteristics of the blocks

pscript the particle script which describes the characteristics of the whole system

depthtexfront the name of the texture containing particle front depth information

depthtexback the name of the texture containing particle back depth information

4.1.3.2 void AdvancedParticleSystemManager::CreateShadableSystem (String *name*, String *pscript*, String *depthtexfront* = "", String *depthtexback* = "")

Creates a shaded particle system.

A shaded particle system is a particle system which is displayed with an algorithm that simulates lighting conditions. This algorithm can handle one main lightsource.

Parameters:

name the name of the system

pscript the particle script which describes the characteristics of the system

depthtexfront the name of the texture containing particle front depth information

depthtexback the name of the texture containing particle back depth information

4.1.3.3 static void AdvancedParticleSystemManager::render (RenderTarget * *rt*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [static]

4.1.3.4 void AdvancedParticleSystemManager::setAllInvisible ()

sets all entities and billboardsets invisible To restore previous visibility the AdvencedParticleSystemManager saves visibility information before hiding.

4.1.3.5 static void AdvancedParticleSystemManager::update (unsigned long *frameNumber*, RenderTarget * *rt*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [static]

4.1.3.6 void AdvancedParticleSystemManager::UpdateSystems (unsigned long *frameNumber*)

Updates the hierarchical and shaded systems created whith the manager.

Cycles throught all the shaded systems created whith the manager, and calls their update function.

Parameters:

frameNumber the actual framenumber

4.1.4 Member Data Documentation

4.1.4.1 std::vector<bool> AdvancedParticleSystemManager::billboardVisibilityList [static]

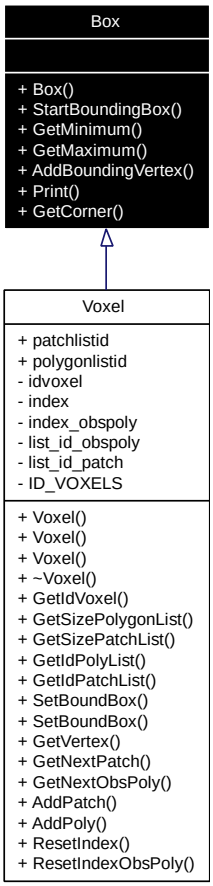
Vector to store billboardset visibilities.

4.1.4.2 std::vector<bool> AdvancedParticleSystemManager::entityVisibilityList [static]

Vector to store entity visibilities.

4.2 Box Class Reference

Inheritance diagram for Box:



Public Member Functions

- Box ()
- void StartBoundingBox ()
- Ogre::Vector3 GetMinimum ()
- Ogre::Vector3 GetMaximum ()
- void AddBoundingVertex (float x, float y, float z)
- void Print ()
- Vertex * GetCorner (int corner) const

4.2.1 Detailed Description

This class encaplusaltes the ogre::AxisAlignedBox.

Superclass: AxisAlignedBox Class: Box

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Box::Box () [inline]

4.2.3 Member Function Documentation

4.2.3.1 void Box::AddBoundingVertex (float *x*, float *y*, float *z*)

4.2.3.2 Vertex* Box::GetCorner (int *corner*) const

4.2.3.3 Ogre::Vector3 Box::GetMaximum () [inline]

4.2.3.4 Ogre::Vector3 Box::GetMinimum () [inline]

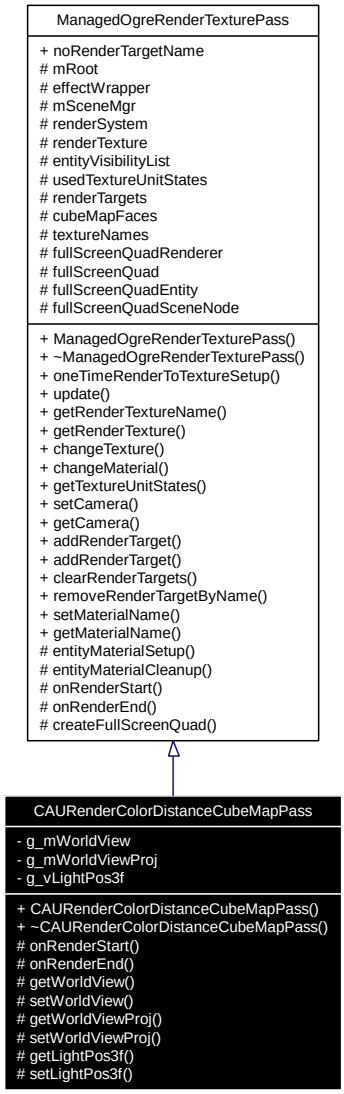
4.2.3.5 void Box::Print ()

4.2.3.6 void Box::StartBoundingBox ()

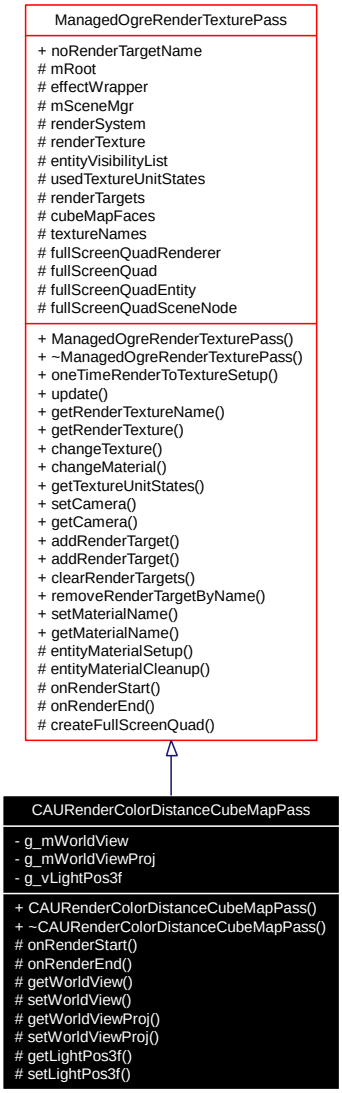
4.3 CAURenderColorDistanceCubeMapPass Class Reference

Generates a CubeMap to store color and distance information about the surrounding of the entity.

Inheritance diagram for CAURenderColorDistanceCubeMapPass:



Collaboration diagram for CAURenderColorDistanceCubeMapPass:



Public Member Functions

- CAURenderColorDistanceCubeMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~CAURenderColorDistanceCubeMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

- Matrix4 getWorldView ()
  Returns the value of g_mWorldView matrix.
- void setWorldView (Matrix4 matrix4)
  Sets the value of g_mWorldView matrix.
- Matrix4 getWorldViewProj ()
  Returns the value of g_mWorldViewProj matrix.
- void setWorldViewProj (Matrix4 matrix4)
  Sets the value of g_mWorldViewProj matrix.
- Vector3 getLightPos3f ()
  Returns the value of g_vLightPos3f.
- void setLightPos3f (Vector3 vector3)
  Sets the value of g_vLightPos3f.

4.3.1 Detailed Description

Generates a CubeMap to store color and distance information about the surrounding of the entity.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderColorDistanceCubeMapPass The instances of this class are to generate a cubemap texture. The resulting texture is a PF_FLOAT32_RGBA type texture.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 CAURenderColorDistanceCubeMapPass::CAURenderColorDistanceCubeMapPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.3.2.2 CAURenderColorDistanceCubeMapPass::~CAURenderColorDistanceCubeMapPass () [inline]

Destructor.

4.3.3 Member Function Documentation

4.3.3.1 Vector3 CAURenderColorDistanceCubeMapPass::getLightPos3f () [protected]

Returns the value of g_vLightPos3f.

4.3.3.2 Matrix4 CAURenderColorDistanceCubeMapPass::getWorldView () [protected]

Returns the value of g_mWorldView matrix.

4.3.3.3 Matrix4 CAURenderColorDistanceCubeMapPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.3.3.4 void CAURenderColorDistanceCubeMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.3.3.5 void CAURenderColorDistanceCubeMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.3.3.6 void CAURenderColorDistanceCubeMapPass::setLightPos3f (Vector3 *vector3*) [protected]

Sets the value of g_vLightPos3f.

Parameters:
vector3 Contains the new value of g_vLightPos3f.

4.3.3.7 void CAURenderColorDistanceCubeMapPass::setWorldView (Matrix4 *matrix4*) [protected]

Sets the value of g_mWorldView matrix.

Parameters:
matrix4 Contains the new value of g_mWorldView matrix.

4.3.3.8 void CAURenderColorDistanceCubeMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

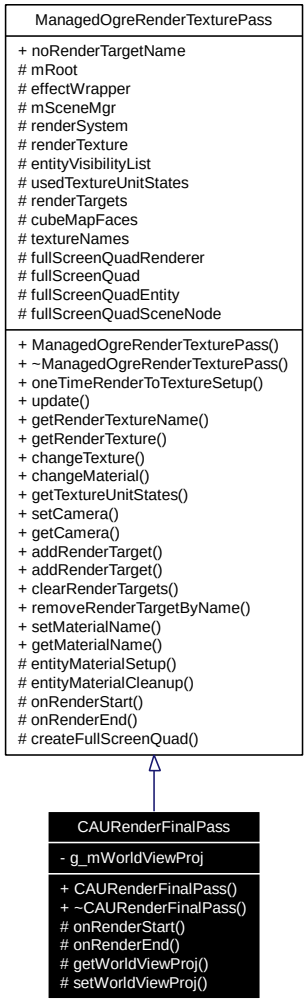
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

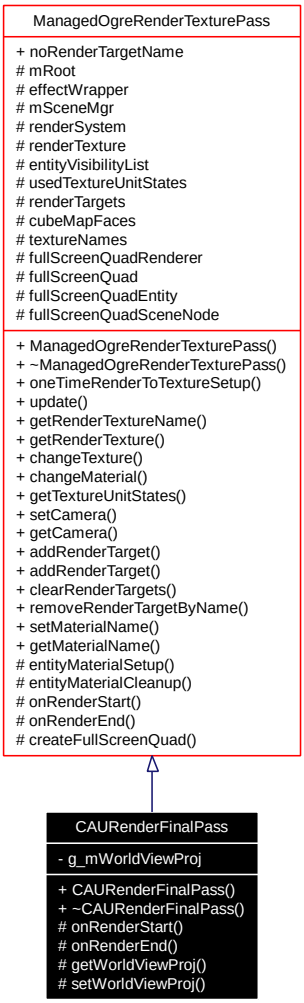
4.4 CAURenderFinalPass Class Reference

Renders the entity into the backbuffer.

Inheritance diagram for CAURenderFinalPass:



Collaboration diagram for CAURenderFinalPass:



Public Member Functions

- CAURenderFinalPass (Root *mRoot)

  Constructor.

- ~CAURenderFinalPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)
- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

4.4.1 Detailed Description

Renders the entity into the backbuffer.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderFinalPass

4.4.2 Constructor & Destructor Documentation

4.4.2.1 CAURenderFinalPass::CAURenderFinalPass (Root * *mRoot*)

Constructor.

Constructor.

Parameters:
mRoot Root, The root class of the Ogre system.

4.4.2.2 CAURenderFinalPass::~CAURenderFinalPass () [inline]

Destructor.

4.4.3 Member Function Documentation

4.4.3.1 Matrix4 CAURenderFinalPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.4.3.2 void CAURenderFinalPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.4.3.3 void CAURenderFinalPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.4.3.4 void CAURenderFinalPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

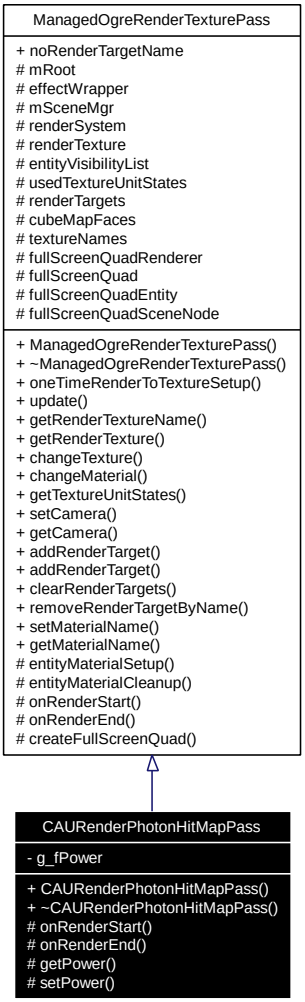
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

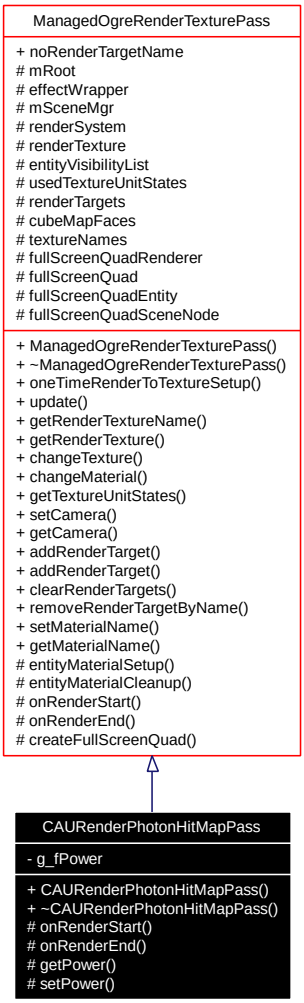
4.5 CAURenderPhotonHitMapPass Class Reference

Creates the Photon hit effects in the texture.

Inheritance diagram for CAURenderPhotonHitMapPass:



Collaboration diagram for CAURenderPhotonHitMapPass:



Public Member Functions

- CAURenderPhotonHitMapPass (Root *mRoot)

  Constructor.

- ~CAURenderPhotonHitMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)
- float getPower ()

  Returns the value of g_fPower.

- void setPower (float f)

  ! Sets the value of g_fPower.

4.5.1 Detailed Description

Creates the Photon hit effects in the texture.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderPhotonHitMapPass The instances of this class are to generate a texture. The resulting texture is a PF_FLOAT32_RGBA type texture.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 CAURenderPhotonHitMapPass::CAURenderPhotonHitMapPass (Root * *mRoot*)

Constructor.

Constructor.

Parameters:
mRoot Root, The root class of the Ogre system.

4.5.2.2 CAURenderPhotonHitMapPass::~CAURenderPhotonHitMapPass () `[inline]`

Destructor.

4.5.3 Member Function Documentation

4.5.3.1 float CAURenderPhotonHitMapPass::getPower () `[protected]`

Returns the value of g_fPower.

4.5.3.2 void CAURenderPhotonHitMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) `[protected, virtual]`

Runs after the render-texture object is updatedplace all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.5.3.3 void CAURenderPhotonHitMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) `[protected, virtual]`

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.5.3.4 void CAURenderPhotonHitMapPass::setPower (float *f*) [protected]

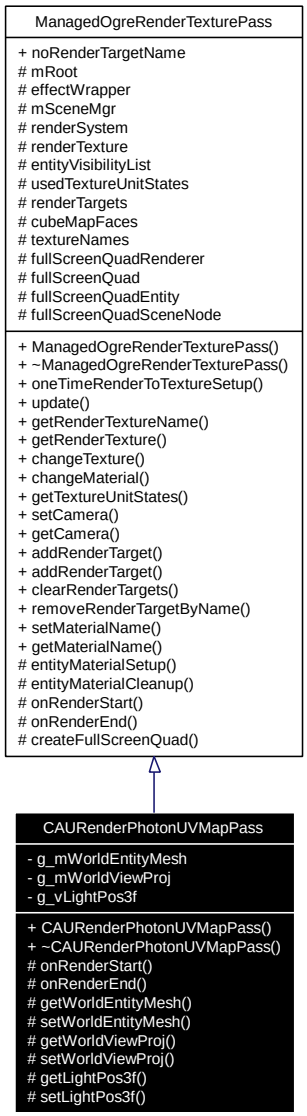
! Sets the value of g_fPower.

Parameters:
f Contains the new value of g_fPower.

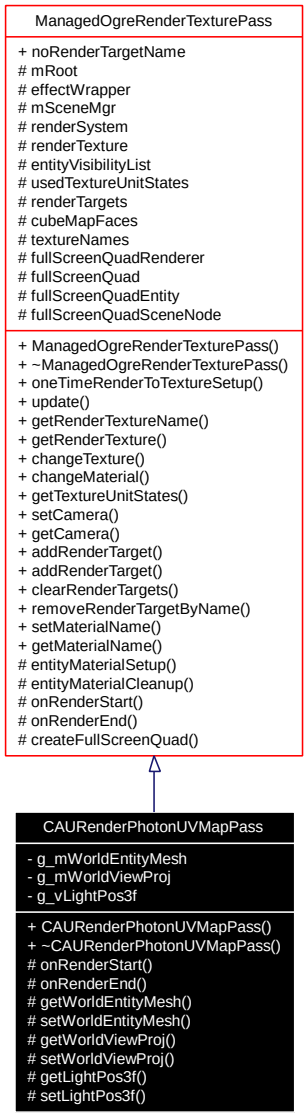
4.6 CAURenderPhotonUVMapPass Class Reference

Generates a texture to store UV information about the PhotonHits.

Inheritance diagram for CAURenderPhotonUVMapPass:



Collaboration diagram for CAURenderPhotonUVMapPass:



Public Member Functions

- CAURenderPhotonUVMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~CAURenderPhotonUVMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

- void onRenderEnd (NameValuePairList *namedParams=0)
- Matrix4 getWorldEntityMesh ()

  Returns the value of g_mWorldEntityMesh matrix.

- void setWorldEntityMesh (Matrix4 matrix4)

  Sets the value of g_mWorldEntityMesh matrix.

- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

- Vector3 getLightPos3f ()

  Returns the value of g_vLightPos3f.

- void setLightPos3f (Vector3 vector3)

  Sets the value of g_vLightPos3f.

4.6.1 Detailed Description

Generates a texture to store UV information about the PhotonHits.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderPhotonUVMapPass The instances of this class are to generate a texture. The resulting texture is a PF_FLOAT32_RGBA type texture.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 CAURenderPhotonUVMapPass::CAURenderPhotonUVMapPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.6.2.2 CAURenderPhotonUVMapPass::~CAURenderPhotonUVMapPass () [inline]

Destructor.

4.6.3 Member Function Documentation

4.6.3.1 Vector3 CAURenderPhotonUVMapPass::getLightPos3f () [protected]

Returns the value of g_vLightPos3f.

4.6.3.2 Matrix4 CAURenderPhotonUVMapPass::getWorldEntityMesh () [protected]

Returns the value of g_mWorldEntityMesh matrix.

4.6.3.3 Matrix4 CAURenderPhotonUVMapPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.6.3.4 void CAURenderPhotonUVMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.6.3.5 void CAURenderPhotonUVMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.6.3.6 void CAURenderPhotonUVMapPass::setLightPos3f (Vector3 *vector3*) [protected]

Sets the value of g_vLightPos3f.

Parameters:
vector3 Contains the new value of g_vLightPos3f.

4.6.3.7 void CAURenderPhotonUVMapPass::setWorldEntityMesh (Matrix4 *matrix4*) [protected]

Sets the value of g_mWorldEntityMesh matrix.

Parameters:
matrix4 Contains the new value of g_mWorldEntityMesh matrix.

4.6.3.8 void CAURenderPhotonUVMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

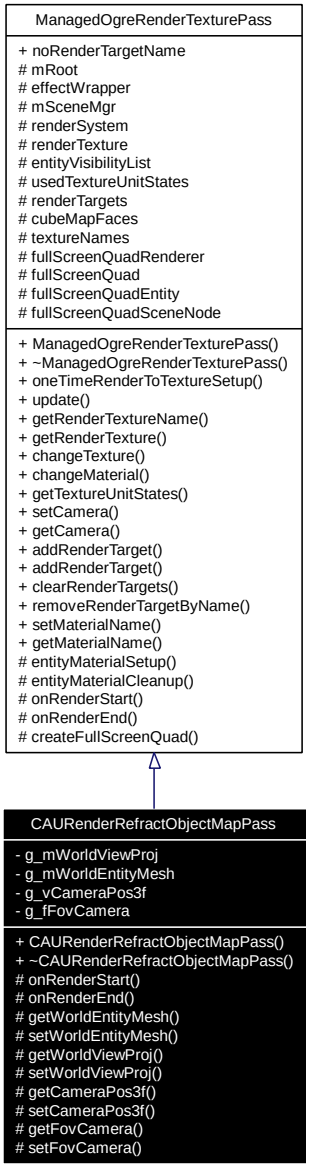
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

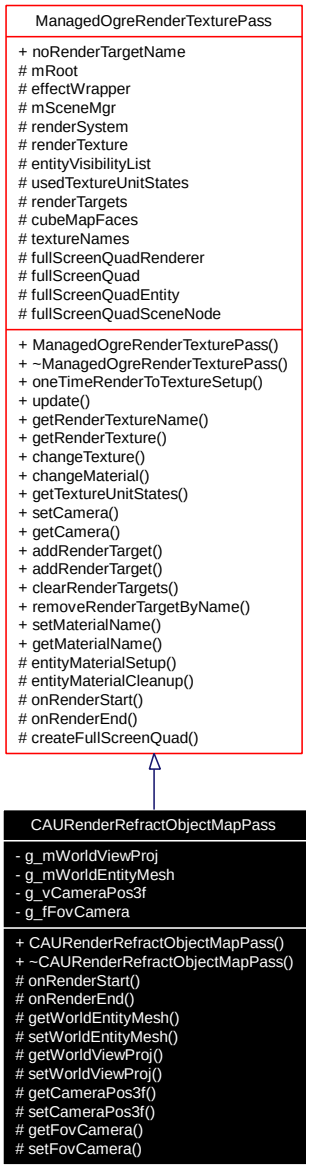
4.7 CAURenderRefractObjectMapPass Class Reference

Generates a texture to store color information about the refractor entity.

Inheritance diagram for CAURenderRefractObjectMapPass:



Collaboration diagram for CAURenderRefractObjectMapPass:



Public Member Functions

- CAURenderRefractObjectMapPass (Root *mRoot)

  Constructor.

- ~CAURenderRefractObjectMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

- Matrix4 getWorldEntityMesh ()

  Returns the value of g_mWorldEntityMesh matrix.

- void setWorldEntityMesh (Matrix4 matrix4)

  Sets the value of g_mWorldEntityMesh matrix.

- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

- Vector3 getCameraPos3f ()

  Returns the value of g_vCameraPos3f.

- void setCameraPos3f (Vector3 vector3)

  Sets the value of g_vCameraPos3f.

- float getFovCamera ()

  Returns the value of g_fFovCamera.

- void setFovCamera (float f)

  Sets the value of g_fFovCamera.

4.7.1 Detailed Description

Generates a texture to store color information about the refractor entity.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderRefractObjectMapPass The instances of this class are to generate a texture. The resulting texture is a PF_FLOAT32_RGBA type texture.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 CAURenderRefractObjectMapPass::CAURenderRefractObjectMapPass (Root * *mRoot*)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

4.7.2.2 CAURenderRefractObjectMapPass::~CAURenderRefractObjectMapPass () `[inline]`

Destructor.

4.7.3 Member Function Documentation

4.7.3.1 Vector3 CAURenderRefractObjectMapPass::getCameraPos3f () [protected]

Returns the value of g_vCameraPos3f.

4.7.3.2 float CAURenderRefractObjectMapPass::getFovCamera () [protected]

Returns the value of g_fFovCamera.

4.7.3.3 Matrix4 CAURenderRefractObjectMapPass::getWorldEntityMesh () [protected]

Returns the value of g_mWorldEntityMesh matrix.

4.7.3.4 Matrix4 CAURenderRefractObjectMapPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.7.3.5 void CAURenderRefractObjectMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.7.3.6 void CAURenderRefractObjectMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:

namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.7.3.7 void CAURenderRefractObjectMapPass::setCameraPos3f (Vector3 *vector3*) [protected]

Sets the value of g_vCameraPos3f.

Parameters:
vector3 Contains the new value of g_vCameraPos3f.

4.7.3.8 void CAURenderRefractObjectMapPass::setFovCamera (float *f*) [protected]

Sets the value of g_fFovCamera.

Parameters:
f float, contains the new value of g_fFovCamera.

4.7.3.9 void CAURenderRefractObjectMapPass::setWorldEntityMesh (Matrix4 *matrix4*) [protected]

Sets the value of g_mWorldEntityMesh matrix.

Parameters:
matrix4 Contains the new value of g_mWorldEntityMesh matrix.

4.7.3.10 void CAURenderRefractObjectMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

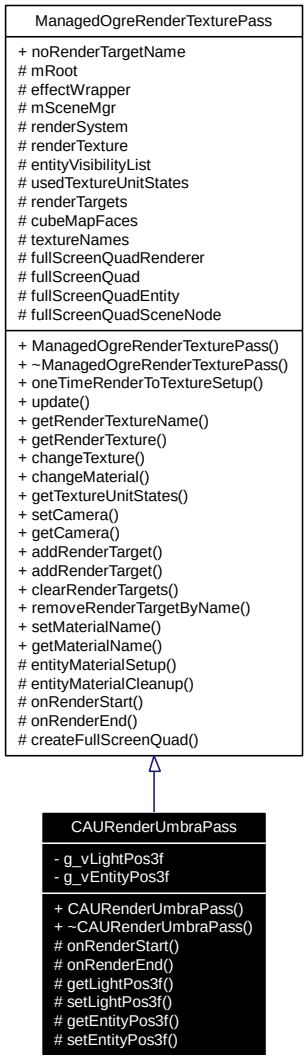
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

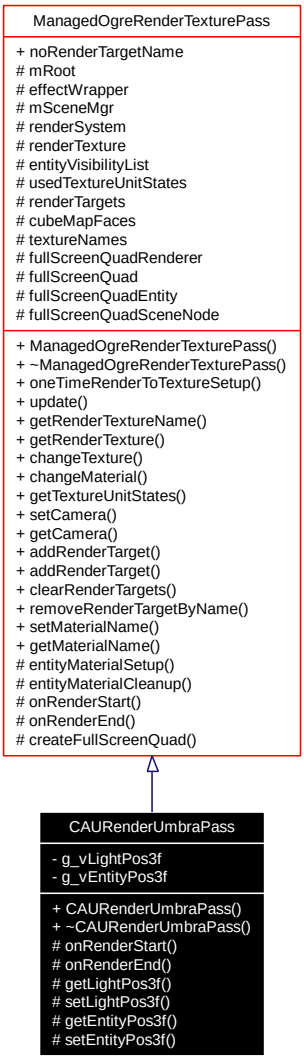
4.8 CAURenderUmbraPass Class Reference

Modify a texture to create umbra for the entity.

Inheritance diagram for CAURenderUmbraPass:



Collaboration diagram for CAURenderUmbraPass:



Public Member Functions

- CAURenderUmbraPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~CAURenderUmbraPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)
- Vector3 getLightPos3f ()

Returns the value of g_vLightPos3f.

- void setLightPos3f (Vector3 vector3)

  Sets the value of g_vLightPos3f.

- Vector3 getEntityPos3f ()

  Returns the value of g_vEntityPos3f.

- void setEntityPos3f (Vector3 vector3)

  Sets the value of g_vEntityPos3f.

4.8.1 Detailed Description

Modify a texture to create umbra for the entity.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderUmbraPass

4.8.2 Constructor & Destructor Documentation

4.8.2.1 CAURenderUmbraPass::CAURenderUmbraPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.8.2.2 CAURenderUmbraPass::~CAURenderUmbraPass () `[inline]`

Destructor.

4.8.3 Member Function Documentation

4.8.3.1 Vector3 CAURenderUmbraPass::getEntityPos3f () [protected]

Returns the value of g_vEntityPos3f.

4.8.3.2 Vector3 CAURenderUmbraPass::getLightPos3f () [protected]

Returns the value of g_vLightPos3f.

4.8.3.3 void CAURenderUmbraPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.8.3.4 void CAURenderUmbraPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.8.3.5 void CAURenderUmbraPass::setEntityPos3f (Vector3 *vector3*) [protected]

Sets the value of g_vEntityPos3f.

Parameters:
vector3 Contains the new value of g_vEntityPos3f.

4.8.3.6 void CAURenderUmbraPass::setLightPos3f (Vector3 *vector3*) [protected]

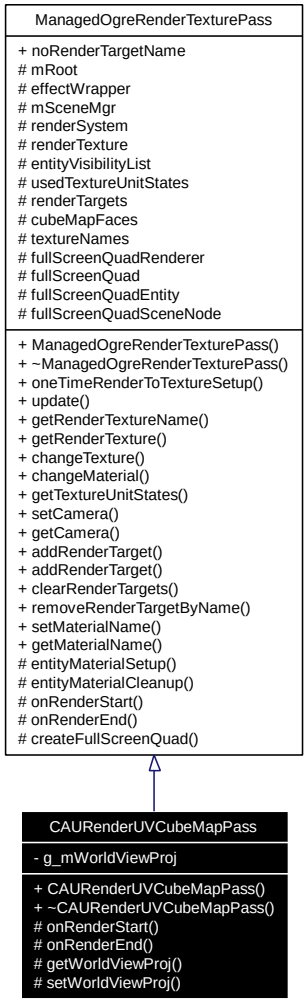
Sets the value of g_vLightPos3f.

Parameters:
vector3 Contains the new value of g_vLightPos3f.

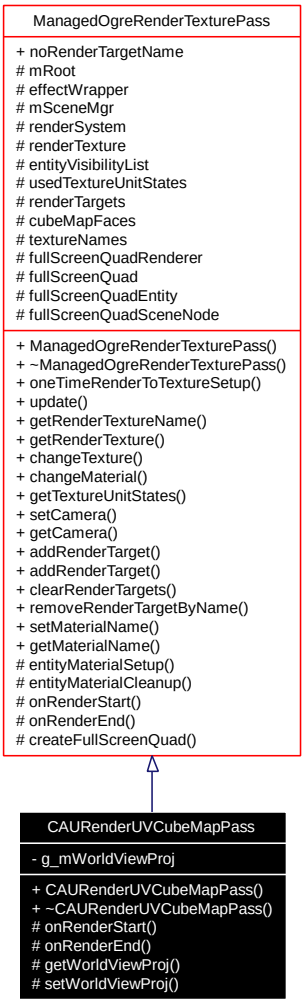
4.9 CAURenderUVCubeMapPass Class Reference

Generates a CubeMap to store UV information about the surrounding of the entity.

Inheritance diagram for CAURenderUVCubeMapPass:



Collaboration diagram for CAURenderUVCubeMapPass:



Public Member Functions

- CAURenderUVCubeMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~CAURenderUVCubeMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)
- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

4.9.1 Detailed Description

Generates a CubeMap to store UV information about the surrounding of the entity.

SuperClass: ManagedOgreRenderTexturePass Class: CAURenderUVCubeMapPass The instances of this class are to generate a cubemap texture. The resulting texture is a PF_FLOAT32_RGBA type texture.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 CAURenderUVCubeMapPass::CAURenderUVCubeMapPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.9.2.2 CAURenderUVCubeMapPass::~CAURenderUVCubeMapPass () `[inline]`

Destructor.

4.9.3 Member Function Documentation

4.9.3.1 Matrix4 CAURenderUVCubeMapPass::getWorldViewProj () `[protected]`

Returns the value of g_mWorldViewProj matrix.

4.9.3.2 void CAURenderUVCubeMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.9.3.3 void CAURenderUVCubeMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.9.3.4 void CAURenderUVCubeMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

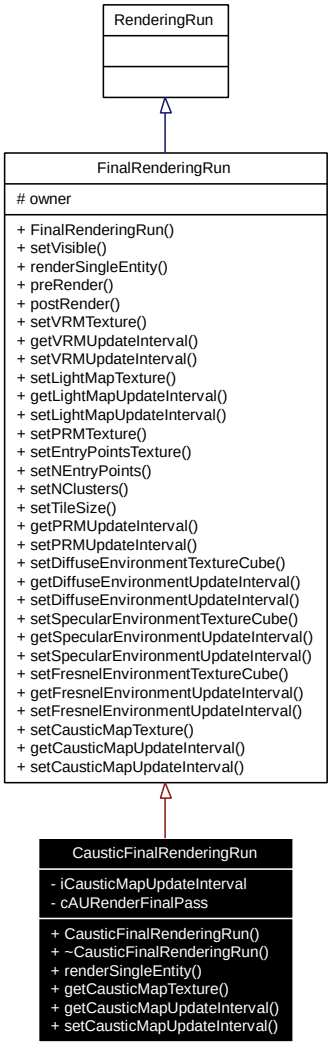
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

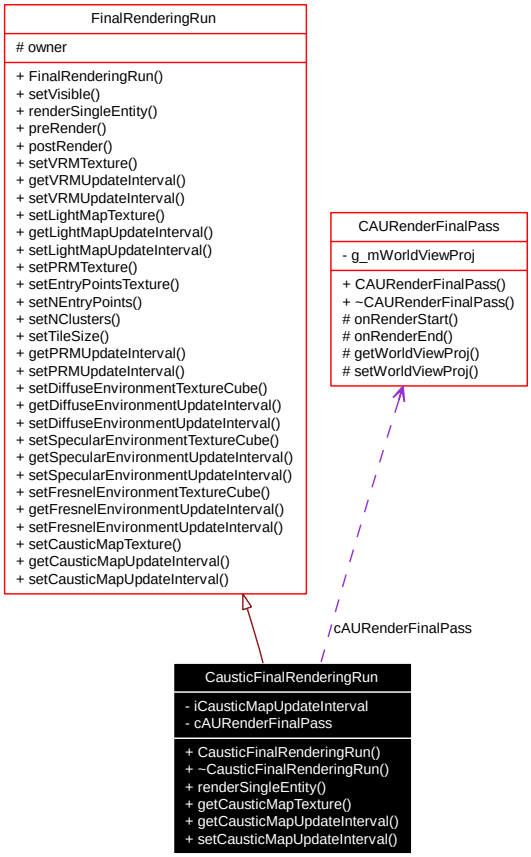
4.10 CausticFinalRenderingRun Class Reference

Draws the entity into the backbuffer.

Inheritance diagram for CausticFinalRenderingRun:



Collaboration diagram for CausticFinalRenderingRun:



Public Member Functions

- CausticFinalRenderingRun (Entity *entity)
- ~CausticFinalRenderingRun ()

  Destructor.

- void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_-POSITIVE_X)

  Renders a sinle entity into the screen.

- const String & getCausticMapTexture ()

  Returns with the causticMapTexture.

- unsigned int getCausticMapUpdateInterval ()

  Gets the number of frame of CausticMapUpdateInterval.

- void setCausticMapUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Sets the number of frame of CausticMapUpdateInterval.

4.10.1 Detailed Description

Draws the entity into the backbuffer.

SuperClass: FinalRenderingRun Class: CausticFinalRenderingRun

4.10.2 Constructor & Destructor Documentation

4.10.2.1 CausticFinalRenderingRun::CausticFinalRenderingRun (Entity * *entity*)

Constructor.

Parameters:
entity The owner entity.

4.10.2.2 CausticFinalRenderingRun::~CausticFinalRenderingRun ()

Destructor.

4.10.3 Member Function Documentation

4.10.3.1 const String& CausticFinalRenderingRun::getCausticMapTexture ()

Returns with the causticMapTexture.

4.10.3.2 unsigned int CausticFinalRenderingRun::getCausticMapUpdateInterval () [virtual]

Gets the number of frame of CausticMapUpdateInterval.

Returns:
The number of frame of CausticMapUpdateInterval.

Reimplemented from FinalRenderingRun.

4.10.3.3 void CausticFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Renders a sinle entity into the screen.

Parameters:
backBuffer RenderTarget, The screen.
cf CubeMapFaces, A CubeMap face.

Implements FinalRenderingRun.

4.10.3.4 void CausticFinalRenderingRun::setCausticMapUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [virtual]

Sets the number of frame of CausticMapUpdateInterval.

Parameters:

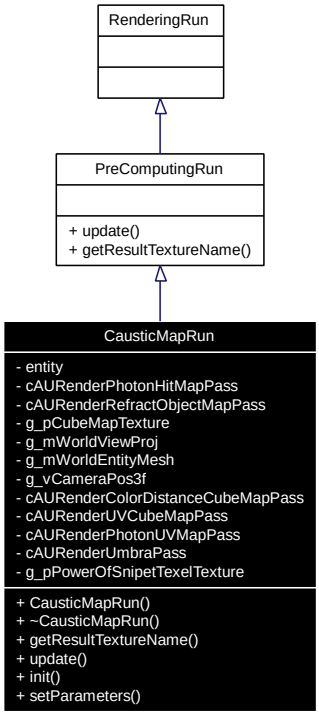
updateIntervalNumOfFrames int, The number of frame of CausticMapUpdateInterval.

Reimplemented from FinalRenderingRun.

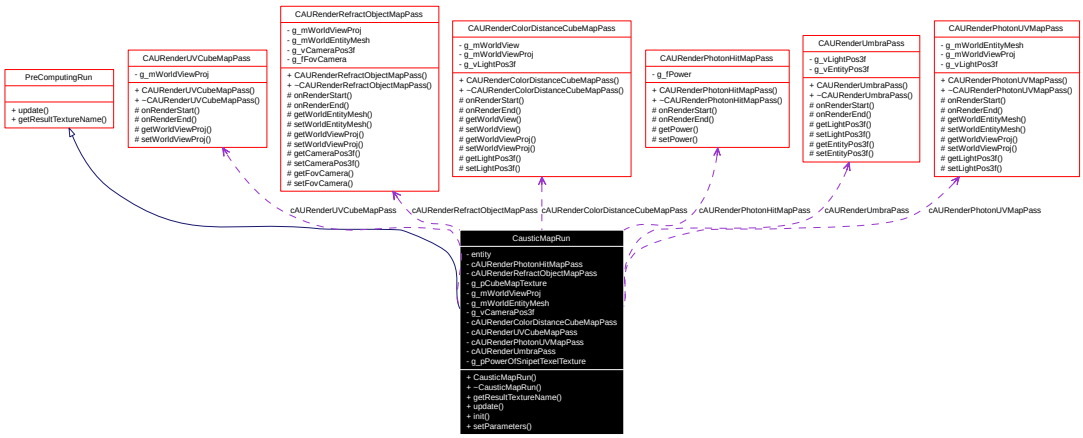
4.11 CausticMapRun Class Reference

Generates a texture with caustic effect.

Inheritance diagram for CausticMapRun:



Collaboration diagram for CausticMapRun:



Public Member Functions

- CausticMapRun (Entity *entity, unsigned int width, unsigned int height)

  Constructor.

- ~CausticMapRun ()

  Destructor.

- const String & getResultTextureName ()

  Returns with the created texture.

- void update ()

  Recalculates the passes.

- void init ()

  Runs passes which should run just once.

- void setParameters ()

  Changes the value of the parameters.

4.11.1 Detailed Description

Generates a texture with caustic effect.

SuperClass: PreComputingRun Class: CausticMapRun The instances of this class are to generate a texture for the surface which receives caustics. The resulting texture is a PF_FLOAT32_RGBA type texture.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 CausticMapRun::CausticMapRun (Entity * *entity*, unsigned int *width*, unsigned int *height*)

Constructor.

Constructor.

Parameters:

entity The owner entity of an entity-bound precomputing run.

width The width of the texture.

height The height of the texture.

4.11.2.2 CausticMapRun::~CausticMapRun ()

Destructor.

4.11.3 Member Function Documentation

4.11.3.1 const String& CausticMapRun::getResultTextureName () [virtual]

Returns with the created texture.

Reimplemented from PreComputingRun.

4.11.3.2 void CausticMapRun::init ()

Runs passes which should run just once.

4.11.3.3 void CausticMapRun::setParameters ()

Changes the value of the parameters.

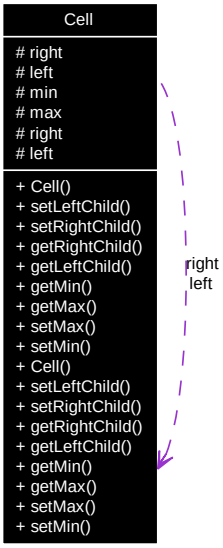
4.11.3.4 void CausticMapRun::update () [virtual]

Recalculates the passes.

Implements PreComputingRun.

4.12 Cell Class Reference

Collaboration diagram for Cell:



Public Member Functions

- Cell ()
- void setLeftChild (Cell ∗lChild)
- void setRightChild (Cell ∗rChild)
- Cell ∗ getRightChild ()
- Cell ∗ getLeftChild ()
- Vector4 getMin ()
- Vector4 getMax ()
- void setMax (Vector4 vmax)
- void setMin (Vector4 vmin)
- Cell ()
- void setLeftChild (Cell ∗lChild)
- void setRightChild (Cell ∗rChild)
- Cell ∗ getRightChild ()
- Cell ∗ getLeftChild ()
- Vector4 getMin ()
- Vector4 getMax ()
- void setMax (Vector4 vmax)
- void setMin (Vector4 vmin)

Protected Attributes

- Cell * right
- Cell * left
- Vector4 min
- Vector4 max
- Cell * right
- Cell * left

4.12.1 Constructor & Destructor Documentation

4.12.1.1 Cell::Cell () [inline]

4.12.1.2 Cell::Cell () [inline]

4.12.2 Member Function Documentation

4.12.2.1 Cell* Cell::getLeftChild () [inline]

4.12.2.2 Cell* Cell::getLeftChild () [inline]

4.12.2.3 Vector4 Cell::getMax () [inline]

4.12.2.4 Vector4 Cell::getMax () [inline]

4.12.2.5 Vector4 Cell::getMin () [inline]

4.12.2.6 Vector4 Cell::getMin () [inline]

4.12.2.7 Cell* Cell::getRightChild () [inline]

4.12.2.8 Cell* Cell::getRightChild () [inline]

4.12.2.9 void Cell::setLeftChild (Cell * *lChild*) [inline]

4.12.2.10 void Cell::setLeftChild (Cell * *lChild*) [inline]

4.12.2.11 void Cell::setMax (Vector4 *vmax*) [inline]

4.12.2.12 void Cell::setMax (Vector4 *vmax*) [inline]

4.12.2.13 void Cell::setMin (Vector4 *vmin*) [inline]

4.12.2.14 void Cell::setMin (Vector4 *vmin*) [inline]

4.12.2.15 void Cell::setRightChild (Cell * *rChild*) [inline]

4.12.2.16 void Cell::setRightChild (Cell * *rChild*) [inline]

4.12.3 Member Data Documentation

4.12.3.1 Cell* Cell::left [protected]

4.12.3.2 Cell* Cell::left [protected]

4.12.3.3 Vector4 Cell::max [protected]

4.12.3.4 Vector4 Cell::min [protected]

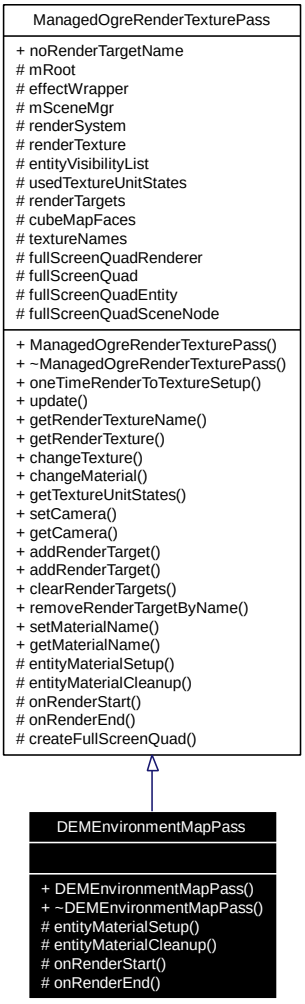
4.12.3.5 Cell* Cell::right [protected]

4.12.3.6 Cell* Cell::right [protected]

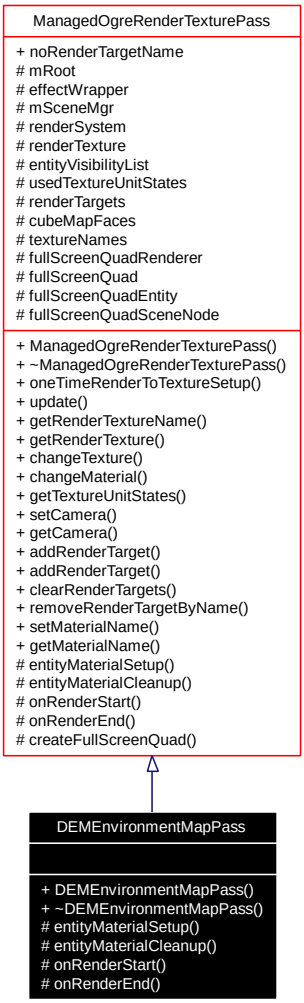
4.13 DEMEnvironmentMapPass Class Reference

Performs the actual pre-processing steps for the Environment Mapping effect.

Inheritance diagram for DEMEnvironmentMapPass:



Collaboration diagram for DEMEnvironmentMapPass:



Public Member Functions

- DEMEnvironmentMapPass (Root *mRoot, unsigned int width, unsigned int height)
- ~DEMEnvironmentMapPass (void)

Protected Member Functions

- virtual void entityMaterialSetup ()
- virtual void entityMaterialCleanup ()
- virtual void onRenderStart (NameValuePairList *namedParams=0)
- virtual void onRenderEnd (NameValuePairList *namedParams=0)

4.13.1 Detailed Description

Performs the actual pre-processing steps for the Environment Mapping effect.

SuperClass ManagedOgreRenderTexturePass

Class DEMEnvironmentMapPass

4.13.2 Constructor & Destructor Documentation

4.13.2.1 DEMEnvironmentMapPass::DEMEnvironmentMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor

Parameters:

mRoot Pointer to the Ogre Root object

width The width of the environment cube-map

height The height of the environment cube-map

Remarks:

The width and height parameters must be equal and the power of 2.

4.13.2.2 DEMEnvironmentMapPass::~DEMEnvironmentMapPass (void)

Destructor

4.13.3 Member Function Documentation

4.13.3.1 virtual void DEMEnvironmentMapPass::entityMaterialCleanup () [protected, virtual]

Cleans up the material of the rendered entity. Can be overriden, if different functionality is desired.

Reimplemented from ManagedOgreRenderTexturePass.

4.13.3.2 virtual void DEMEnvironmentMapPass::entityMaterialSetup () [protected, virtual]

Sets up the material of the rendered entity. Can be overriden, if different functionality is desired.

Reimplemented from ManagedOgreRenderTexturePass.

4.13.3.3 virtual void DEMEnvironmentMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.13.3.4 virtual void DEMEnvironmentMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:

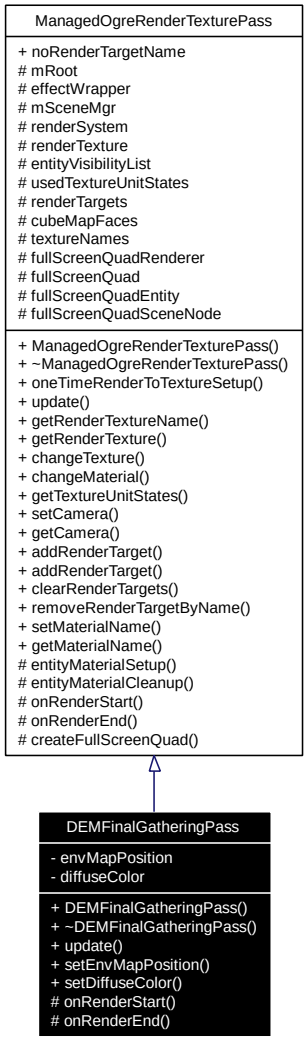
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

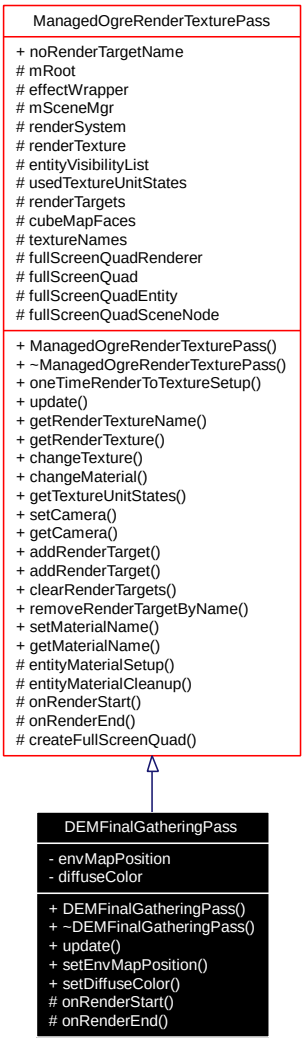
4.14 DEMFinalGatheringPass Class Reference

Performs the actual rendering of the Environment Mapping Effect.

Inheritance diagram for DEMFinalGatheringPass:



Collaboration diagram for DEMFinalGatheringPass:



Public Member Functions

- DEMFinalGatheringPass (Root *mRoot)
- ~DEMFinalGatheringPass (void)
- void update (void)
- void setEnvMapPosition (Vector3 envMapPosition)
- void setDiffuseColor (Vector3 diffuseColor)

Protected Member Functions

- virtual void onRenderStart (NameValuePairList *namedParams=0)
- virtual void onRenderEnd (NameValuePairList *namedParams=0)

4.14.1 Detailed Description

Performs the actual rendering of the Environment Mapping Effect.

SuperClass ManagedOgreRenderTexturePass

Class DEMFinalGatheringPass

4.14.2 Constructor & Destructor Documentation

4.14.2.1 DEMFinalGatheringPass::DEMFinalGatheringPass (Root * *mRoot*)

Constructor

Parameters:
mRoot The Ogre Root object

4.14.2.2 DEMFinalGatheringPass::~DEMFinalGatheringPass (void)

Destructor

4.14.3 Member Function Documentation

4.14.3.1 virtual void DEMFinalGatheringPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.14.3.2 virtual void DEMFinalGatheringPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.14.3.3 void DEMFinalGatheringPass::setDiffuseColor (Vector3 *diffuseColor*)

Sets the material's diffuse color for the DiffuseEnvMap shader.

Parameters:
diffuseColor The diffuse color.

4.14.3.4 void DEMFinalGatheringPass::setEnvMapPosition (Vector3 *envMapPosition*)

Sets the world-space position of the environment cube-map for the EnvMap shader. This is used, because the cubemap is not regenerated in every frame.

Parameters:
envMapPosition The position vector

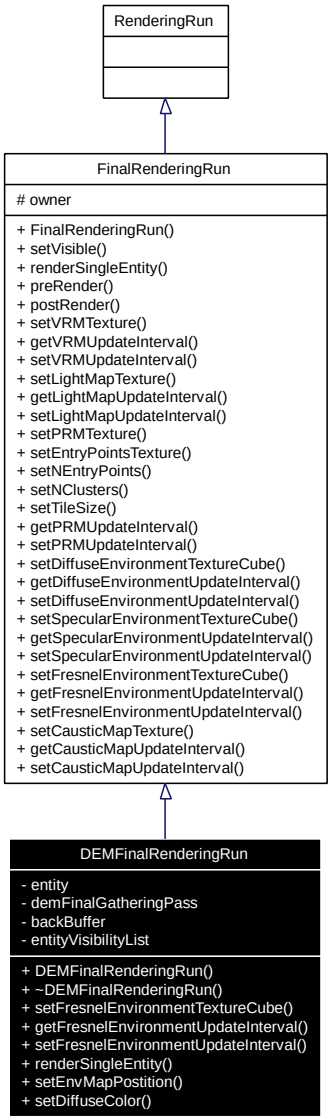
4.14.3.5 void DEMFinalGatheringPass::update (void)

Performs the rendering

4.15 DEMFinalRenderingRun Class Reference

Controlls the rendering of the Environment Mapping effect.

Inheritance diagram for DEMFinalRenderingRun:



Collaboration diagram for DEMFinalRenderingRun:

Public Member Functions

- DEMFinalRenderingRun (Entity *ent)
- ~DEMFinalRenderingRun (void)
- virtual void setFresnelEnvironmentTextureCube (const String &fresnelEnvironmentTextureCube-Name)

  Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getFresnelEnvironmentUpdateInterval ()
- virtual void setFresnelEnvironmentUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the DEM udate interval desired for the owner entity. If DEM is not used, the method should have no effect.

- virtual void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_-FACE_POSITIVE_X)

  Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

- void setEnvMapPostition (Vector3 envMapPosition)

- void setDiffuseColor (float diffuseColor)

4.15.1 Detailed Description

Controlls the rendering of the Environment Mapping effect.

SuperClass FinalRenderingRun

Class DEMFinalRenderingRun

4.15.2 Constructor & Destructor Documentation

4.15.2.1 DEMFinalRenderingRun::DEMFinalRenderingRun (Entity * *ent*)

Constructor

Parameters:
ent Owner entity.

4.15.2.2 DEMFinalRenderingRun::~DEMFinalRenderingRun (void)

Destructor

4.15.3 Member Function Documentation

4.15.3.1 virtual unsigned int DEMFinalRenderingRun::getFresnelEnvironmentUpdateInterval () [inline, virtual]

Returns:
0 if Fresnel Enviroment Map is not used, the desired length of the update interval otherwise.

Reimplemented from FinalRenderingRun.

4.15.3.2 virtual void DEMFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

This method is supposed to reproduce the behaviour of rendering an object using the standard OGRE pipeline. Thus, it is forbidden to commit any of the following:

- clear the color, depth or stencil of the backbuffer
- alter the depth testing, stencil testing, alpha blending render state without restoring it
- render with altered depth testing, stencil testing, alpha blending to the backbuffer
- alter entity or billboard visibilities without restoring them

Parameters:

backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Implements FinalRenderingRun.

4.15.3.3 void DEMFinalRenderingRun::setDiffuseColor (float *diffuseColor*)

Sets the material's diffuse color for the DiffuseEnvMap shader.

Parameters:

diffuseColor The diffuse color.

4.15.3.4 void DEMFinalRenderingRun::setEnvMapPostition (Vector3 *envMapPosition*)

Sets the world-space position of the environment cube-map for the EnvMap shader. This is used, because the cubemap is not regenerated in every frame.

Parameters:

envMapPosition The position vector

4.15.3.5 virtual void DEMFinalRenderingRun::setFresnelEnvironmentTextureCube (const String & *fresnelEnvironmentTextureCubeName*) [virtual]

Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

fresnelEnvironmentTextureCubeName The precomputed Fresnel Environment Map texture's name, as returned by FresnelEnvironmentRenderingRun::getResultTextureName().

Reimplemented from FinalRenderingRun.

4.15.3.6 virtual void DEMFinalRenderingRun::setFresnelEnvironmentUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [virtual]

Set the DEM udate interval desired for the owner entity. If DEM is not used, the method should have no effect.

Parameters:

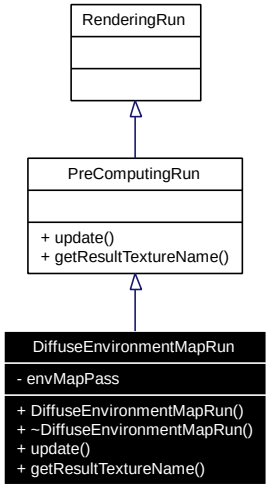
updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the DEM.

Reimplemented from FinalRenderingRun.

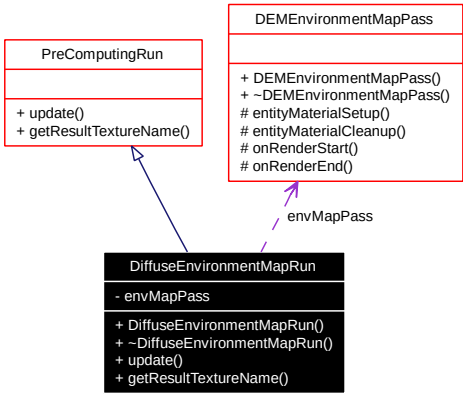
4.16 DiffuseEnvironmentMapRun Class Reference

Controlls the actual pre-processing steps for the Environment Mapping effect.

Inheritance diagram for DiffuseEnvironmentMapRun:



Collaboration diagram for DiffuseEnvironmentMapRun:



Public Member Functions

- DiffuseEnvironmentMapRun (Entity *entity, unsigned int resolution)
- ~DiffuseEnvironmentMapRun (void)
- virtual void update (void)
- virtual const String & getResultTextureName ()

  This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

4.16.1 Detailed Description

Controlls the actual pre-processing steps for the Environment Mapping effect.

SuperClass PreComputingRun Class DiffuseEnvironmentMapRun

4.16.2 Constructor & Destructor Documentation

4.16.2.1 DiffuseEnvironmentMapRun::DiffuseEnvironmentMapRun (Entity * *entity*, unsigned int *resolution*)

Constructor

Parameters:

entity The owner entity.

resolution The resolution of the texture.

Remarks:

The resolution parameter must be equal and the power of 2.

4.16.2.2 DiffuseEnvironmentMapRun::~DiffuseEnvironmentMapRun (void)

Destructor

4.16.3 Member Function Documentation

4.16.3.1 virtual const String& DiffuseEnvironmentMapRun::getResultTextureName () [virtual]

This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Returns:

the main result texture's name

Reimplemented from PreComputingRun.

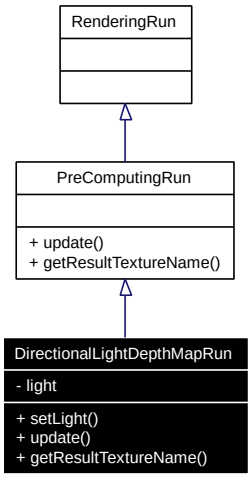
4.16.3.2 virtual void DiffuseEnvironmentMapRun::update (void) [virtual]

Performs the update of the environment cube-map. LOD can be implemented by varying the frequency of calls.

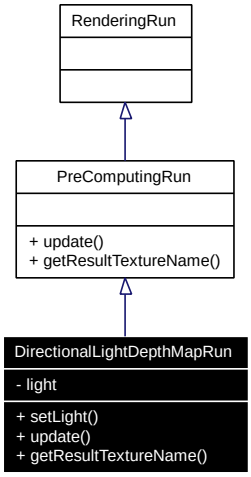
Implements PreComputingRun.

4.17 DirectionalLightDepthMapRun Class Reference

Inheritance diagram for DirectionalLightDepthMapRun:



Collaboration diagram for DirectionalLightDepthMapRun:



Public Member Functions

- void setLight (Light *light)
- virtual void update ()
- virtual const String & getResultTextureName ()

4.17.1 Detailed Description

Computes a depth map for a directional light

4.17.2 Member Function Documentation

4.17.2.1 virtual const String& DirectionalLightDepthMapRun::getResultTextureName () [inline, virtual]

Reimplemented from PreComputingRun.

4.17.2.2 void DirectionalLightDepthMapRun::setLight (Light * *light*) [inline]

Parameters:

light The owner light of an light-bound precomputing run.

4.17.2.3 virtual void DirectionalLightDepthMapRun::update (void) [inline, virtual]

Implements PreComputingRun.

4.18 EffectWrapper Class Reference

Wraps vertex and fragment shader setup. Convinience class based upon effect framework interfaces. (See D3D9 Effect framework or CGFX.).

Public Member Functions

- EffectWrapper ()

  Default constructor.

- EffectWrapper (Material *material)

  Constructor.

- ~EffectWrapper ()

  Destructor.

- bool SetFloat (const String &name, float value)
- bool SetMatrix (const String &name, Matrix4 &value)
- bool SetInt (const String &name, int value)
- bool SetMatrixTranspose (const String &name, Matrix4 &value)
- bool SetFloatArray (const String &name, const float *value, size_t count)
- bool SetIntArray (const String &name, const int *value, size_t count)
- bool SetDoubleArray (const String &name, double *value, size_t count)
- bool SetMatrixArray (const String &name, Matrix4 *value, size_t numEntries)
- bool SetMatrixTransposeArray (const String &name, Matrix4 *value, size_t numEntries)
- bool SetTexture (int textureUnit, const String &textureName)
- bool BeginPass ()

  Begins a new pass.

- void EndPass ()

  Signals the ending of the pass.

- bool SetVector4 (const String &name, const Vector4 &vec)
- bool SetVector3 (const String &name, const Vector3 &vec)
- bool SetReal (const String &name, Real value)
- bool SetColourValue (const String &name, const ColourValue &colour)
- void SetVertexProgramParameters ()

  Switches to vertex program parameter setup.

- void SetFragmentProgramParameters ()

  Switches to fragment program parameter setup.

- void SetShadowCasterProgramParameters ()

  Switches to shadow caster program parameter setup.

- void SetShadowReceiverProgramParameters ()

Switches to shadow receiver program parameter setup.

- bool SetTechniqueToUse (int numberOfTechnique)
  Setup for the material techniques.
- unsigned short GetNumTechniques (void) const
  Accessor for the number of techniques in the material.
- void setMaterial (Material *material)
  Does a material setup by pointer.
- void createMaterial (const String &materialName)
  Does a material setup by name.
- MaterialPtr & getMaterial ()
  Retrieves the material pointer object of the material.

4.18.1 Detailed Description

Wraps vertex and fragment shader setup. Convinience class based upon effect framework interfaces. (See D3D9 Effect framework or CGFX.).

4.18.2 Constructor & Destructor Documentation

4.18.2.1 EffectWrapper::EffectWrapper ()

Default constructor.

4.18.2.2 EffectWrapper::EffectWrapper (Material * *material*)

Constructor.

Parameters:
material Material pointer to the Ogre material which the wrapper will use.

4.18.2.3 EffectWrapper::~EffectWrapper ()

Destructor.

4.18.3 Member Function Documentation

4.18.3.1 bool EffectWrapper::BeginPass () [inline]

Begins a new pass.

Returns:
Operation success or fail.

4.18.3.2 void EffectWrapper::createMaterial (const String & *materialName*) [inline]

Does a material setup by name.

Parameters:
materialName Material to create.

4.18.3.3 void EffectWrapper::EndPass () [inline]

Signals the ending of the pass.

4.18.3.4 MaterialPtr& EffectWrapper::getMaterial ()

Retrieves the material pointer object of the material.

Returns:
The material pointer object.

4.18.3.5 unsigned short EffectWrapper::GetNumTechniques (void) const [inline]

Accessor for the number of techniques in the material.

Returns:
Number of techniques in the effect currently wrapped.

4.18.3.6 bool EffectWrapper::SetColourValue (const String & *name*, const ColourValue & *colour*)

Parameters:
name Shader uniform parameter name.
colour Value of shader uniform parameter.

Returns:
Operation success or fail.

4.18.3.7 bool EffectWrapper::SetDoubleArray (const String & *name*, double * *value*, size_t *count*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

count Count of data.

Returns:

Operation success or fail.

4.18.3.8 bool EffectWrapper::SetFloat (const String & *name*, float *value*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

Returns:

Operation success or fail.

4.18.3.9 bool EffectWrapper::SetFloatArray (const String & *name*, const float * *value*, size_t *count*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

count Count of data.

Returns:

Operation success or fail.

4.18.3.10 void EffectWrapper::SetFragmentProgramParameters () [inline]

Switches to fragment program parameter setup.

4.18.3.11 bool EffectWrapper::SetInt (const String & *name*, int *value*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

Returns:

Operation success or fail.

4.18.3.12 bool EffectWrapper::SetIntArray (const String & *name*, const int * *value*, size_t *count*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

count Count of data.

Returns:

Operation success or fail.

4.18.3.13 void EffectWrapper::setMaterial (Material * *material*) [inline]

Does a material setup by pointer.

Parameters:

material Material to wrap.

4.18.3.14 bool EffectWrapper::SetMatrix (const String & *name*, Matrix4 & *value*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

Returns:

Operation success or fail.

4.18.3.15 bool EffectWrapper::SetMatrixArray (const String & *name*, Matrix4 * *value*, size_t *numEntries*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

numEntries Count of data.

Returns:

Operation success or fail.

4.18.3.16 bool EffectWrapper::SetMatrixTranspose (const String & *name*, Matrix4 & *value*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

Returns:

Operation success or fail.

4.18.3.17 bool EffectWrapper::SetMatrixTransposeArray (const String & *name*, Matrix4 * *value*, size_t *numEntries*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

numEntries Count of data.

Returns:

Operation success or fail.

4.18.3.18 bool EffectWrapper::SetReal (const String & *name*, Real *value*)

Parameters:

name Shader uniform parameter name.

value Value of shader uniform parameter.

Returns:

Operation success or fail.

4.18.3.19 void EffectWrapper::SetShadowCasterProgramParameters () [inline]

Switches to shadow caster program parameter setup.

4.18.3.20 void EffectWrapper::SetShadowReceiverProgramParameters () [inline]

Switches to shadow receiver program parameter setup.

4.18.3.21 bool EffectWrapper::SetTechniqueToUse (int *numberOfTechnique*)

Setup for the material techniques.

Parameters:
numberOfTechnique Which technique to use.

Returns:
Operation success or fail.

4.18.3.22 bool EffectWrapper::SetTexture (int *textureUnit*, const String & *textureName*)

Parameters:
textureUnit Which texture unit to populate with the texture.

textureName The name of the texture to use.

Returns:
Operation success or fail.

4.18.3.23 bool EffectWrapper::SetVector3 (const String & *name*, const Vector3 & *vec*)

Parameters:
name Shader uniform parameter name.

vec Value of shader uniform parameter.

Returns:
Operation success or fail.

4.18.3.24 bool EffectWrapper::SetVector4 (const String & *name*, const Vector4 & *vec*)

Parameters:
name Shader uniform parameter name.

vec Value of shader uniform parameter.

Returns:
Operation success or fail.

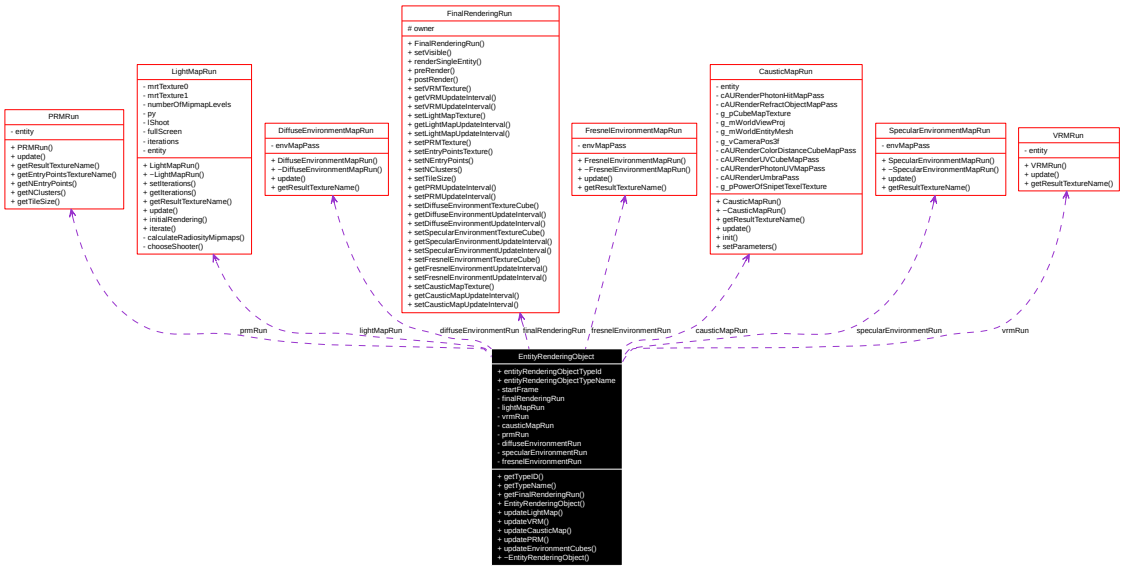
4.18.3.25 void EffectWrapper::SetVertexProgramParameters () [inline]

Switches to vertex program parameter setup.

4.19 EntityRenderingObject Class Reference

This class and the FinalRenderingRun class encapsulate the complete illumination model implemented in the illumination workpackage. A EntityRenderingObject instance is stored with all Entities.

Collaboration diagram for EntityRenderingObject:



Public Member Functions

- long getTypeID (void) const

  Inherted from Ogre::UserDefinedObject.

- const String & getTypeName (void) const

  Inherted from Ogre::UserDefinedObject.

- FinalRenderingRun ∗ getFinalRenderingRun ()
- EntityRenderingObject (Entity ∗owner, unsigned long startFrame, const RenderingType &rendering-Type)

  Constructor.

- void updateLightMap (long frameCount)

  Perform preprocessing necessary in this frame. Update the Light Map.

- void updateVRM (long frameCount)

  Perform preprocessing necessary in this frame. Update the Visibility Ratio Map.

- void updateCausticMap (long frameCount)

  Perform preprocessing necessary in this frame. Update the Visibility Ratio Map.

- void updatePRM (long frameCount)

  Perform preprocessing necessary in this frame. Update the Precomputed Radiance Map.

- void updateEnvironmentCubes (long frameCount)

  Perform preprocessing necessary in this frame. Update the environment cubes. They are not always simultaneously updated, but can be set a different update interval.

- ~EntityRenderingObject (void)

  Destructor.

Static Public Attributes

- static const long entityRenderingObjectTypeId
- static const String entityRenderingObjectTypeName

4.19.1 Detailed Description

This class and the FinalRenderingRun class encapsulate the complete illumination model implemented in the illumination workpackage. A EntityRenderingObject instance is stored with all Entities.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 EntityRenderingObject::EntityRenderingObject (Entity * *owner*, unsigned long *startFrame*, const RenderingType & *renderingType*)

Constructor.

Parameters:

owner The entity the EntityRenderingObject is linked to.

startFrame The current frame number. Update intervals starts from this frame.

renderingType The descriptor of the final rendering algorithm the Entity should use.

4.19.2.2 EntityRenderingObject::~EntityRenderingObject (void)

Destructor.

4.19.3 Member Function Documentation

4.19.3.1 FinalRenderingRun* EntityRenderingObject::getFinalRenderingRun () [inline]

Returns:
The encapsulated FinalRenderingRun instance.

4.19.3.2 long EntityRenderingObject::getTypeID (void) const [inline]

Inherted from Ogre::UserDefinedObject.

Returns:
The UserDefinedObject subclass ID, for type reflection.

4.19.3.3 const String& EntityRenderingObject::getTypeName (void) const [inline]

Inherted from Ogre::UserDefinedObject.

Returns:
The UserDefinedObject subclass name, for type reflection.

4.19.3.4 void EntityRenderingObject::updateCausticMap (long *frameCount*)

Perform preprocessing necessary in this frame. Update the Visibility Ratio Map.

Parameters:
frameCount The current frame number. This is used to determine whether the preprocessed data should be updated.

4.19.3.5 void EntityRenderingObject::updateEnvironmentCubes (long *frameCount*)

Perform preprocessing necessary in this frame. Update the environment cubes. They are not always simultaneously updated, but can be set a different update interval.

Parameters:
frameCount The current frame number. This is used to determine whether the preprocessed data should be updated.

4.19.3.6 void EntityRenderingObject::updateLightMap (long *frameCount*)

Perform preprocessing necessary in this frame. Update the Light Map.

Parameters:

frameCount The current frame number. This is used to determine whether the preprocessed data should be updated.

4.19.3.7 void EntityRenderingObject::updatePRM (long *frameCount*)

Perform preprocessing necessary in this frame. Update the Precomputed Radiance Map.

Parameters:

frameCount The current frame number. This is used to determine whether the preprocessed data should be updated.

4.19.3.8 void EntityRenderingObject::updateVRM (long *frameCount*)

Perform preprocessing necessary in this frame. Update the Visibility Ratio Map.

Parameters:

frameCount The current frame number. This is used to determine whether the preprocessed data should be updated.

4.19.4 Member Data Documentation

4.19.4.1 const long EntityRenderingObject::entityRenderingObjectTypeId [static]

Ogre::UserDefinedObject subclass ID, for type reflection.

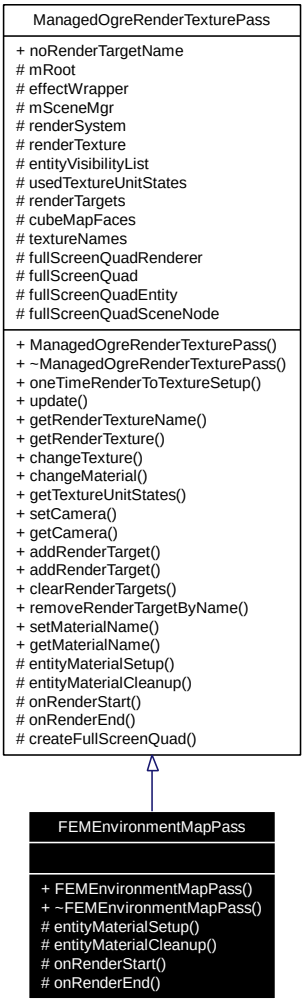
4.19.4.2 const String EntityRenderingObject::entityRenderingObjectTypeName [static]

Ogre::UserDefinedObject subclass name, for type reflection.

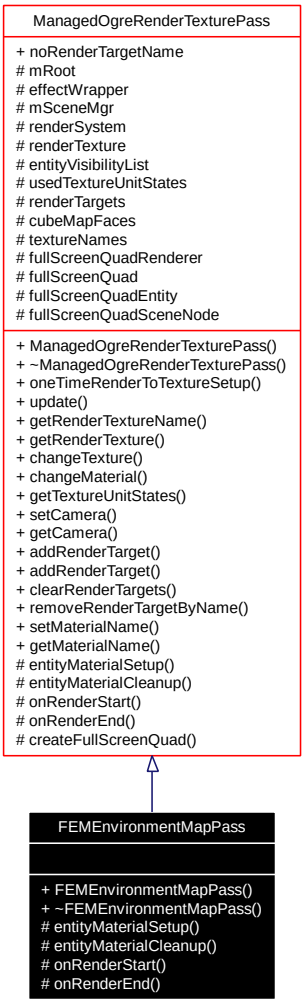
4.20 FEMEnvironmentMapPass Class Reference

Performs the actual pre-processing steps for the Environment Mapping effect.

Inheritance diagram for FEMEnvironmentMapPass:



Collaboration diagram for FEMEnvironmentMapPass:



Public Member Functions

- FEMEnvironmentMapPass (Root *mRoot, unsigned int width, unsigned int height)
- ~FEMEnvironmentMapPass (void)

Protected Member Functions

- virtual void entityMaterialSetup ()
- virtual void entityMaterialCleanup ()
- virtual void onRenderStart (NameValuePairList *namedParams=0)
- virtual void onRenderEnd (NameValuePairList *namedParams=0)

4.20.1 Detailed Description

Performs the actual pre-processing steps for the Environment Mapping effect.

SuperClass ManagedOgreRenderTexturePass

Class FEMEnvironmentMapPass

4.20.2 Constructor & Destructor Documentation

4.20.2.1 FEMEnvironmentMapPass::FEMEnvironmentMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor

Parameters:

mRoot Pointer to the Ogre Root object

width The width of the environment cube-map

height The height of the environment cube-map

Remarks:

The width and height parameters must be equal and the power of 2.

4.20.2.2 FEMEnvironmentMapPass::~FEMEnvironmentMapPass (void)

Destructor

4.20.3 Member Function Documentation

4.20.3.1 virtual void FEMEnvironmentMapPass::entityMaterialCleanup () [protected, virtual]

Cleans up the material of the rendered entity. Can be overriden, if different functionality is desired.

Reimplemented from ManagedOgreRenderTexturePass.

4.20.3.2 virtual void FEMEnvironmentMapPass::entityMaterialSetup () [protected, virtual]

Sets up the material of the rendered entity. Can be overriden, if different functionality is desired.

Reimplemented from ManagedOgreRenderTexturePass.

4.20.3.3 virtual void FEMEnvironmentMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.20.3.4 virtual void FEMEnvironmentMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:

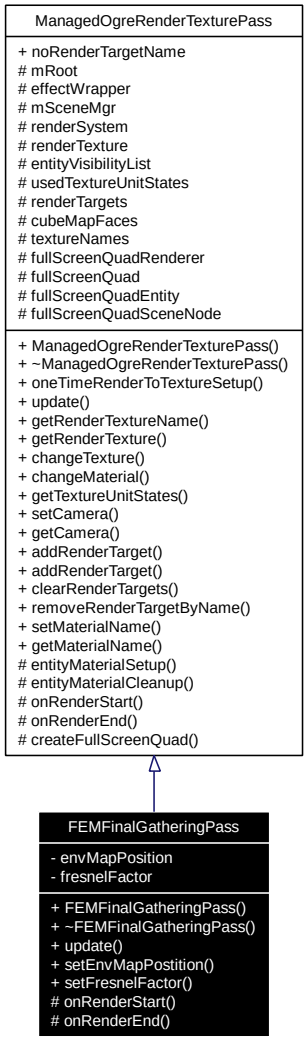
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

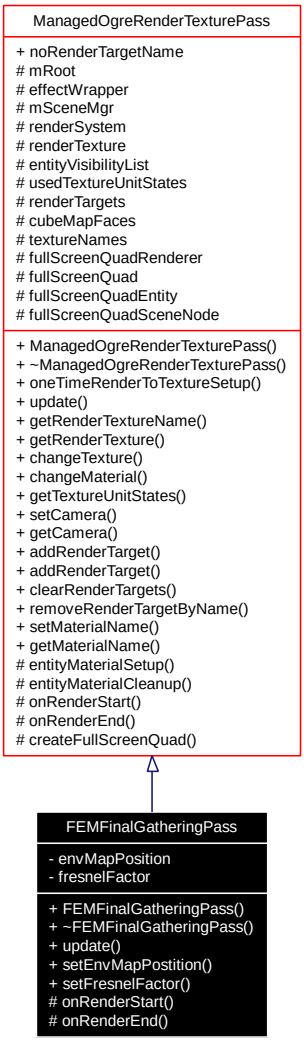
4.21 FEMFinalGatheringPass Class Reference

Performs the actual rendering of the Environment Mapping Effect.

Inheritance diagram for FEMFinalGatheringPass:



Collaboration diagram for FEMFinalGatheringPass:



Public Member Functions

- FEMFinalGatheringPass (Root *mRoot)
- ~FEMFinalGatheringPass (void)
- void update (void)
- void setEnvMapPostition (Vector3 envMapPosition)
- void setFresnelFactor (float fresnelFactor)

Protected Member Functions

- virtual void onRenderStart (NameValuePairList *namedParams=0)
- virtual void onRenderEnd (NameValuePairList *namedParams=0)

4.21.1 Detailed Description

Performs the actual rendering of the Environment Mapping Effect.

SuperClass ManagedOgreRenderTexturePass

Class FEMFinalGatheringPass

4.21.2 Constructor & Destructor Documentation

4.21.2.1 FEMFinalGatheringPass::FEMFinalGatheringPass (Root * *mRoot*)

Constructor

Parameters:
mRoot The Ogre Root object

4.21.2.2 FEMFinalGatheringPass::~FEMFinalGatheringPass (void)

Destructor

4.21.3 Member Function Documentation

4.21.3.1 virtual void FEMFinalGatheringPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.21.3.2 virtual void FEMFinalGatheringPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.21.3.3 void FEMFinalGatheringPass::setEnvMapPostition (Vector3 *envMapPosition*)

Sets the world-space position of the environment cube-map for the EnvMap shader. This is used, because the cubemap is not regenerated in every frame.

Parameters:
envMapPosition The position vector

4.21.3.4 void FEMFinalGatheringPass::setFresnelFactor (float *fresnelFactor*)

Sets the material's Fresnel factor for the EnvMap shader.

Parameters:
fresnelFactor The Fresnel factor. 0.0f means only a small reflection in narrow angles, 1.0f complete reflection to every direction.

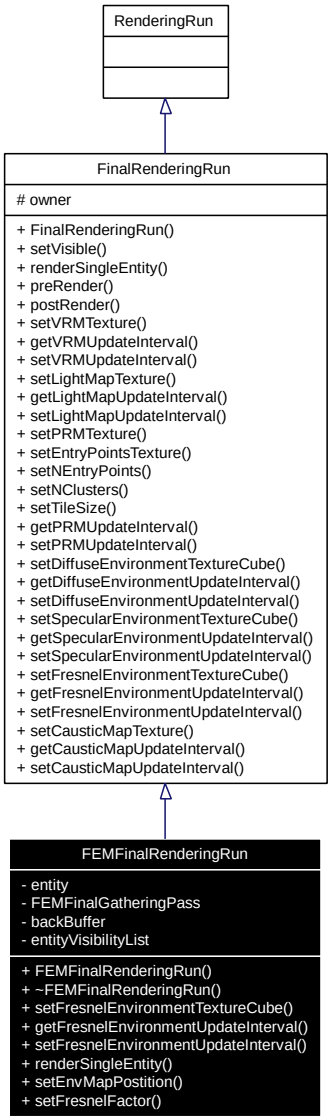
4.21.3.5 void FEMFinalGatheringPass::update (void)

Performs the rendering

4.22 FEMFinalRenderingRun Class Reference

Controlls the rendering of the Environment Mapping effect.

Inheritance diagram for FEMFinalRenderingRun:



Collaboration diagram for FEMFinalRenderingRun:

Public Member Functions

- FEMFinalRenderingRun (Entity *ent)
- ~FEMFinalRenderingRun (void)
- virtual void setFresnelEnvironmentTextureCube (const String &fresnelEnvironmentTextureCube-Name)

  Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getFresnelEnvironmentUpdateInterval ()
- virtual void setFresnelEnvironmentUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the FEM udate interval desired for the owner entity. If FEM is not used, the method should have no effect.

- virtual void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_-FACE_POSITIVE_X)

  Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

- void setEnvMapPostition (Vector3 envMapPosition)

- void setFresnelFactor (float fresnelFactor)

4.22.1 Detailed Description

Controlls the rendering of the Environment Mapping effect.

SuperClass FinalRenderingRun

Class FEMFinalRenderingRun

4.22.2 Constructor & Destructor Documentation

4.22.2.1 FEMFinalRenderingRun::FEMFinalRenderingRun (Entity * *ent*)

Constructor

Parameters:
ent Owner entity.

4.22.2.2 FEMFinalRenderingRun::~FEMFinalRenderingRun (void)

Destructor

4.22.3 Member Function Documentation

4.22.3.1 virtual unsigned int FEMFinalRenderingRun::getFresnelEnvironmentUpdateInterval () [inline, virtual]

Returns:
0 if Fresnel Enviroment Map is not used, the desired length of the update interval otherwise.

Reimplemented from FinalRenderingRun.

4.22.3.2 virtual void FEMFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

This method is supposed to reproduce the behaviour of rendering an object using the standard OGRE pipeline. Thus, it is forbidden to commit any of the following:

- clear the color, depth or stencil of the backbuffer
- alter the depth testing, stencil testing, alpha blending render state without restoring it
- render with altered depth testing, stencil testing, alpha blending to the backbuffer
- alter entity or billboard visibilities without restoring them

Parameters:

backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Implements FinalRenderingRun.

4.22.3.3 void FEMFinalRenderingRun::setEnvMapPostition (Vector3 *envMapPosition*)

Sets the world-space position of the environment cube-map for the EnvMap shader. This is used, because the cubemap is not regenerated in every frame.

Parameters:

envMapPosition The position vector

4.22.3.4 virtual void FEMFinalRenderingRun::setFresnelEnvironmentTextureCube (const String & *fresnelEnvironmentTextureCubeName*) [virtual]

Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

fresnelEnvironmentTextureCubeName The precomputed Fresnel Environment Map texture's name, as returned by FresnelEnvironmentRenderingRun::getResultTextureName().

Reimplemented from FinalRenderingRun.

4.22.3.5 virtual void FEMFinalRenderingRun::setFresnelEnvironmentUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [virtual]

Set the FEM udate interval desired for the owner entity. If FEM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the FEM.

Reimplemented from FinalRenderingRun.

4.22.3.6 void FEMFinalRenderingRun::setFresnelFactor (float *fresnelFactor*)

Sets the material's Fresnel factor for the EnvMap shader.

Parameters:

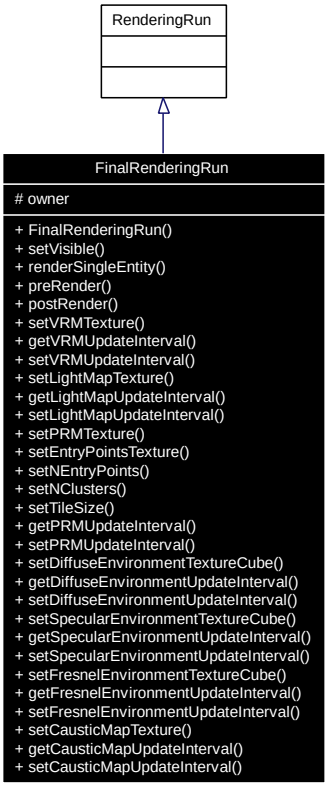
fresnelFactor The Fresnel factor. 0.0f means only a small reflection in narrow angles, 1.0f complete reflection to every direction.

4.23 FinalRenderingRun Class Reference

This class and the EntityRenderingObject class encapsulate the complete illumination model implemented in the illumination workpackage.

Inheritance diagram for FinalRenderingRun:

Collaboration diagram for FinalRenderingRun:



Public Member Functions

- FinalRenderingRun (Entity *owner)

  Constructor.

- void setVisible (bool visible)

  Calls Entity::setVisible.

- virtual void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)=0

  Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

- virtual void preRender (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)

  Called before renderSingleEntity would be called for any entity. Could be useful e.g. for pre-rendering depth. See FinalRenderingRun::renderSingleEntity for usage guidelines. However, it is encouraged to apply custom render states in this method.

- virtual void postRender (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)

  Called after renderSingleEntity has been called for all entities. This is where additional effects can be blend-added to the final image. See FinalRenderingRun::renderSingleEntity for usage guidelines. However, it is encouraged to apply custom render states in this method.

- virtual void setVRMTexture (const String &vrmTextureName)

  Set the entity's Visibility Ratio Map (soft shadow map). Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getVRMUpdateInterval ()
- virtual void setVRMUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the VRM udate interval desired for the owner entity. If VRM is not used, the method should have no effect.

- virtual void setLightMapTexture (const String &lightMapTextureName)

  Set the entity's Light Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getLightMapUpdateInterval ()
- virtual void setLightMapUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the Light Map udate interval desired for the owner entity. If Light Map is not used, the method should have no effect.

- virtual void setPRMTexture (const String &prmTextureName)

  Set the entity's Precomputed Radiance Map. Resources possibly re-computed later must be passed by reference or name.

- virtual void setEntryPointsTexture (const String &entryPointsTextureName)
- virtual void setNEntryPoints (unsigned int &nEntryPoints)
- virtual void setNClusters (unsigned int &nClusters)
- virtual void setTileSize (unsigned int &tileSize)
- virtual unsigned int getPRMUpdateInterval ()
- virtual void setPRMUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the PRM udate interval desired for the owner entity. If PRM is not used, the method should have no effect.

- virtual void setDiffuseEnvironmentTextureCube (const String &diffuseEnvironmentTextureCube-Name)

  Set the entity's Diffuse Environment Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getDiffuseEnvironmentUpdateInterval ()
- virtual void setDiffuseEnvironmentUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the DEM udate interval desired for the owner entity. If DEM is not used, the method should have no effect.

- virtual void setSpecularEnvironmentTextureCube (const String &specularEnvironmentTextureCube-Name)

  Set the entity's Specular Environment Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getSpecularEnvironmentUpdateInterval ()
- virtual void setSpecularEnvironmentUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the SEM udate interval desired for the owner entity. If SEM is not used, the method should have no effect.

- virtual void setFresnelEnvironmentTextureCube (const String &fresnelEnvironmentTextureCube-Name)

  Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getFresnelEnvironmentUpdateInterval ()
- virtual void setFresnelEnvironmentUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the FEM udate interval desired for the owner entity. If FEM is not used, the method should have no effect.

- virtual void setCausticMapTexture (const String &causticMapTextureName)

  Set the entity's Caustic Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getCausticMapUpdateInterval ()
- virtual void setCausticMapUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the Caustic Map udate interval desired for the owner entity. If Caustic Map is not used, the method should have no effect.

Protected Attributes

- Entity * owner

4.23.1 Detailed Description

This class and the EntityRenderingObject class encapsulate the complete illumination model implemented in the illumination workpackage.

A FinalRenderingRun instance is stored with all Entities. What preprocessing is necessary for the final rendering is coded into classes derived from FinalRenderingRun. How often (in how many frames) those preprocessing runs are to be performed can be set.

Typically, a FinalRenderingRun-derived class has a number of static ManagedOgreRenderTexturePass instances for performing intermediate computations, and a non-static ManagedOgreRenderTexturePass member that renders to the frame buffer.

Data flow between runs

A RenderingRun gathers its input from the following sources:

- static resources in the IlluminationManager
- data linked to the entity
  - via the FinalRenderingRun interface, the references to standard input resources are passed
  - through the 'Entity* owner' member
    - the entity's Ogre::Material can be accessed (colour, textures, etc.)
    - the entity's EntityRenderingObject can be accessed (properties of internal runs can be queried directly)

4.23.2 Constructor & Destructor Documentation

4.23.2.1 FinalRenderingRun::FinalRenderingRun (Entity * *owner*) [inline]

Constructor.

Parameters:
owner The entity the FinalRenderingRun is linked to.

4.23.3 Member Function Documentation

4.23.3.1 virtual unsigned int FinalRenderingRun::getCausticMapUpdateInterval () [inline, virtual]

Returns:
0 if Caustic Map is not used, the desired length of the update interval otherwise.

Reimplemented in CausticFinalRenderingRun.

4.23.3.2 virtual unsigned int FinalRenderingRun::getDiffuseEnvironmentUpdateInterval () [inline, virtual]

Returns:
0 if Diffuse Enviroment Map is not used, the desired length of the update interval otherwise.

4.23.3.3 virtual unsigned int FinalRenderingRun::getFresnelEnvironmentUpdateInterval () [inline, virtual]

Returns:
0 if Fresnel Enviroment Map is not used, the desired length of the update interval otherwise.

Reimplemented in DEMFinalRenderingRun, FEMFinalRenderingRun, and SEMFinalRenderingRun.

4.23.3.4 virtual unsigned int FinalRenderingRun::getLightMapUpdateInterval () [inline, virtual]

Returns:
0 if Light Map is not used, the desired length of the update interval otherwise.

Reimplemented in LightMapFinalRenderingRun.

4.23.3.5 virtual unsigned int FinalRenderingRun::getPRMUpdateInterval () [inline, virtual]

Returns:
0 if PRM is not used, the desired length of the update interval otherwise.

4.23.3.6 virtual unsigned int FinalRenderingRun::getSpecularEnvironmentUpdateInterval () [inline, virtual]

Returns:
0 if Specular Enviroment Map is not used, the desired length of the update interval otherwise.

4.23.3.7 virtual unsigned int FinalRenderingRun::getVRMUpdateInterval () [inline, virtual]

Returns:
0 if VRM is not used, the desired length of the update interval otherwise.

4.23.3.8 virtual void FinalRenderingRun::postRender (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [inline, virtual]

Called after renderSingleEntity has been called for all entities. This is where additional effects can be blend-added to the final image. See FinalRenderingRun::renderSingleEntity for usage guidelines. However, it is encouraged to apply custom render states in this method.

Parameters:
backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Reimplemented in ImageBasedLightingFinalRenderingRun.

4.23.3.9 virtual void FinalRenderingRun::preRender (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [inline, virtual]

Called before renderSingleEntity would be called for any entity. Could be useful e.g. for pre-rendering depth. See FinalRenderingRun::renderSingleEntity for usage guidelines. However, it is encouraged to apply custom render states in this method.

Parameters:
backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

4.23.3.10 virtual void FinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [pure virtual]

Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update,

after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

This method is supposed to reproduce the behaviour of rendering an object using the standard OGRE pipeline. Thus, it is forbidden to commit any of the following:

- clear the color, depth or stencil of the backbuffer
- alter the depth testing, stencil testing, alpha blending render state without restoring it
- render with altered depth testing, stencil testing, alpha blending to the backbuffer
- alter entity or billboard visibilities without restoring them

Parameters:

backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Implemented in CausticFinalRenderingRun, DEMFinalRenderingRun, FEMFinalRenderingRun, ImageBasedLightingFinalRenderingRun, LightMapFinalRenderingRun, SEMFinalRenderingRun, and SoftShadowFinalRenderingRun.

4.23.3.11 virtual void FinalRenderingRun::setCausticMapTexture (const String & *causticMapTextureName*) [inline, virtual]

Set the entity's Caustic Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

causticMapTextureName The precomputed Caustic Map texture's name, as returned by CausticMapRenderingRun::getResultTextureName().

4.23.3.12 virtual void FinalRenderingRun::setCausticMapUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the Caustic Map udate interval desired for the owner entity. If Caustic Map is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the Caustic Map.

Reimplemented in CausticFinalRenderingRun.

4.23.3.13 virtual void FinalRenderingRun::setDiffuseEnvironmentTextureCube (const String & *diffuseEnvironmentTextureCubeName*) [inline, virtual]

Set the entity's Diffuse Environment Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

diffuseEnvironmentTextureCubeName The precomputed Diffuse Environment Map texture's name, as returned by DiffuseEnvironmentRenderingRun::getResultTextureName().

4.23.3.14 virtual void FinalRenderingRun::setDiffuseEnvironmentUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the DEM udate interval desired for the owner entity. If DEM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the DEM.

4.23.3.15 virtual void FinalRenderingRun::setEntryPointsTexture (const String & *entryPointsTextureName*) [inline, virtual]

4.23.3.16 virtual void FinalRenderingRun::setFresnelEnvironmentTextureCube (const String & *fresnelEnvironmentTextureCubeName*) [inline, virtual]

Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

fresnelEnvironmentTextureCubeName The precomputed Fresnel Environment Map texture's name, as returned by FresnelEnvironmentRenderingRun::getResultTextureName().

Reimplemented in DEMFinalRenderingRun, FEMFinalRenderingRun, and SEMFinalRenderingRun.

4.23.3.17 virtual void FinalRenderingRun::setFresnelEnvironmentUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the FEM udate interval desired for the owner entity. If FEM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the FEM.

Reimplemented in DEMFinalRenderingRun, FEMFinalRenderingRun, and SEMFinalRenderingRun.

4.23.3.18 virtual void FinalRenderingRun::setLightMapTexture (const String & *lightMapTextureName*) [inline, virtual]

Set the entity's Light Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

lightMapTextureName The precomputed Light Map texture's name, as returned by LightMapRenderingRun::getResultTextureName().

Reimplemented in LightMapFinalRenderingRun.

4.23.3.19 virtual void FinalRenderingRun::setLightMapUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the Light Map udate interval desired for the owner entity. If Light Map is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the Light Map.

Reimplemented in LightMapFinalRenderingRun.

4.23.3.20 virtual void FinalRenderingRun::setNClusters (unsigned int & *nClusters*) [inline, virtual]

4.23.3.21 virtual void FinalRenderingRun::setNEntryPoints (unsigned int & *nEntryPoints*) [inline, virtual]

4.23.3.22 virtual void FinalRenderingRun::setPRMTexture (const String & *prmTextureName*) [inline, virtual]

Set the entity's Precomputed Radiance Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

prmTextureName The precomputed Light Map texture's name, as returned by PRMRenderingRun::getResultTextureName().

4.23.3.23 virtual void FinalRenderingRun::setPRMUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the PRM udate interval desired for the owner entity. If PRM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the PRM.

4.23.3.24 virtual void FinalRenderingRun::setSpecularEnvironmentTextureCube (const String & *specularEnvironmentTextureCubeName*) [inline, virtual]

Set the entity's Specular Environment Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

specularEnvironmentTextureCubeName The precomputed Specular Environment Map texture's name, as returned by SpecularEnvironmentRenderingRun::getResultTextureName().

4.23.3.25 virtual void FinalRenderingRun::setSpecularEnvironmentUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the SEM udate interval desired for the owner entity. If SEM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the SEM.

4.23.3.26 virtual void FinalRenderingRun::setTileSize (unsigned int & *tileSize*) [inline, virtual]

4.23.3.27 void FinalRenderingRun::setVisible (bool *visible*) [inline]

Calls Entity::setVisible.

Parameters:

visible True if the entity should be made visible, false if it should be hidden.

4.23.3.28 virtual void FinalRenderingRun::setVRMTexture (const String & *vrmTextureName*) [inline, virtual]

Set the entity's Visibility Ratio Map (soft shadow map). Resources possibly re-computed later must be passed by reference or name.

Parameters:

vrmTextureName The precomputed VRM texture's name, as returned by VRMRenderingRun::getResultTextureName().

4.23.3.29 virtual void FinalRenderingRun::setVRMUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Set the VRM udate interval desired for the owner entity. If VRM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the VRM.

4.23.4 Member Data Documentation

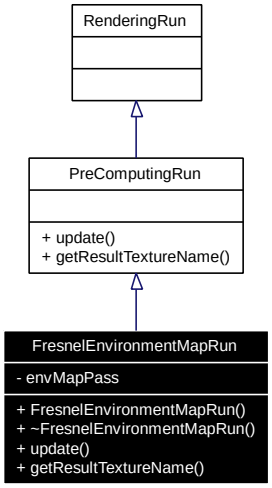
4.23.4.1 Entity* FinalRenderingRun::owner [protected]

The owner entity of this FinalRenderingRun instance.

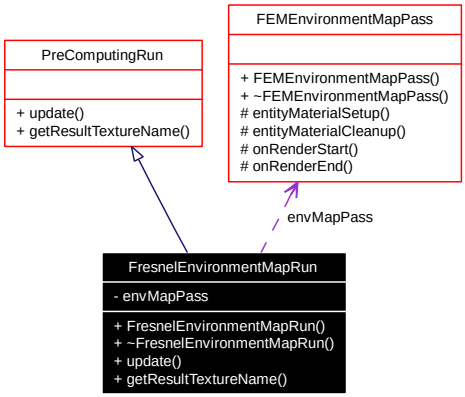
4.24 FresnelEnvironmentMapRun Class Reference

Controlls the actual pre-processing steps for the Environment Mapping effect.

Inheritance diagram for FresnelEnvironmentMapRun:



Collaboration diagram for FresnelEnvironmentMapRun:



Public Member Functions

- FresnelEnvironmentMapRun (Entity *entity, unsigned int resolution)
- ~FresnelEnvironmentMapRun (void)
- virtual void update (void)
- virtual const String & getResultTextureName ()

  This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

4.24.1 Detailed Description

Controlls the actual pre-processing steps for the Environment Mapping effect.

SuperClass PreComputingRun Class FresnelEnvironmentRun

4.24.2 Constructor & Destructor Documentation

4.24.2.1 FresnelEnvironmentMapRun::FresnelEnvironmentMapRun (Entity * *entity*, unsigned int *resolution*)

Constructor

Parameters:

entity The owner entity.

resolution The resolution of the texture.

Remarks:

The resolution parameter must be equal and the power of 2.

4.24.2.2 FresnelEnvironmentMapRun::~FresnelEnvironmentMapRun (void)

Destructor

4.24.3 Member Function Documentation

4.24.3.1 virtual const String& FresnelEnvironmentMapRun::getResultTextureName () [virtual]

This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Returns:

the main result texture's name

Reimplemented from PreComputingRun.

4.24.3.2 virtual void FresnelEnvironmentMapRun::update (void) [virtual]

Performs the update of the environment cube-map. LOD can be implemented by varying the frequency of calls.

Implements PreComputingRun.

4.25 HdriSampler Class Reference

High dynamic range image sampler. Generates directional light samples.

Public Member Functions

- void calculateRadii ()

  Recompute approximate Voronoi radii.

- HdriSampler (void)

  Constructor.

- ~HdriSampler (void)

  Destructor.

- bool loadHdrFile (char *filename)

  Load hdri image from file.

- void generatePoints (int nSamples)

  Generate samples.

- void relax (int nSteps)

  Apply Lloyd's relaxation.

Public Attributes

- VoronoiGenerator voro

  Voronoi mesh.

- std::vector< Vector > samplePoints

  Sampled directions.

- std::vector< Vector > powers

  Light sample powers.

- std::vector< float > voronoiRadii

  Voronoi radii.

4.25.1 Detailed Description

High dynamic range image sampler. Generates directional light samples.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 HdriSampler::HdriSampler (void)

Constructor.

4.25.2.2 HdriSampler::~HdriSampler (void)

Destructor.

4.25.3 Member Function Documentation

4.25.3.1 void HdriSampler::calculateRadii ()

Recompute approximate Voronoi radii.

4.25.3.2 void HdriSampler::generatePoints (int *nSamples*)

Generate samples.

Parameters:
nSamples Samples to generate. Minimum 4.

4.25.3.3 bool HdriSampler::loadHdrFile (char * *filename*)

Load hdri image from file.

Parameters:
filename hdr File name.

4.25.3.4 void HdriSampler::relax (int *nSteps*)

Apply Lloyd's relaxation.

Parameters:
nSteps Number of iterations.

4.25.4 Member Data Documentation

4.25.4.1 std::vector<Vector> HdriSampler::powers

Light sample powers.

4.25.4.2 std::vector<Vector> HdriSampler::samplePoints

Sampled directions.

4.25.4.3 VoronoiGenerator HdriSampler::voro

Voronoi mesh.

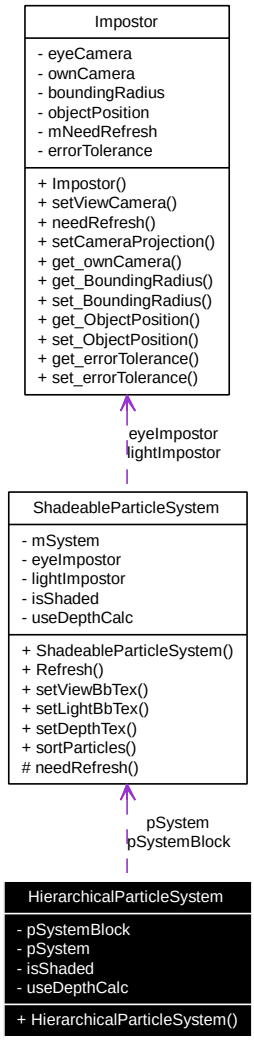
4.25.4.4 std::vector<float> HdriSampler::voronoiRadii

Voronoi radii.

4.26 HierarchicalParticleSystem Class Reference

Hierarchially built particle system This class implements a particle system which is build of particle system blocks.

Collaboration diagram for HierarchicalParticleSystem:



Public Member Functions

- HierarchicalParticleSystem (String name, String blockpscript, String pscript, String textureNameFront="", String textureNameBack="")

  Constructor.

4.26.1 Detailed Description

Hierarchially built particle system This class implements a particle system which is build of particle system blocks.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 HierarchicalParticleSystem::HierarchicalParticleSystem (String *name*, String *blockpscript*, String *pscript*, String *textureNameFront* = "", String *textureNameBack* = "")

Constructor.

The input arguments are the particle scripts for the blocks and for the entire system. These scripts describe the motion and life of the particles.,

Parameters:

name the name of the hierarchical system. The particle system names (which oge uses) will be generaed from this name.

blockpscript the name of the particle script describing the characteristics of the blocks

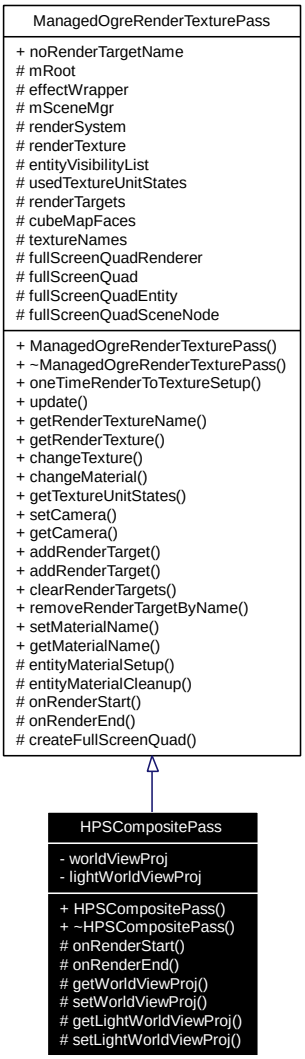
pscript the name of the particle script describing the characteristics of the system build of particle system blocks

textureNameFront the name of the front depth texture used when rendering the particles of the blocks (optional: this information can be coded in the colortexture of the particle material

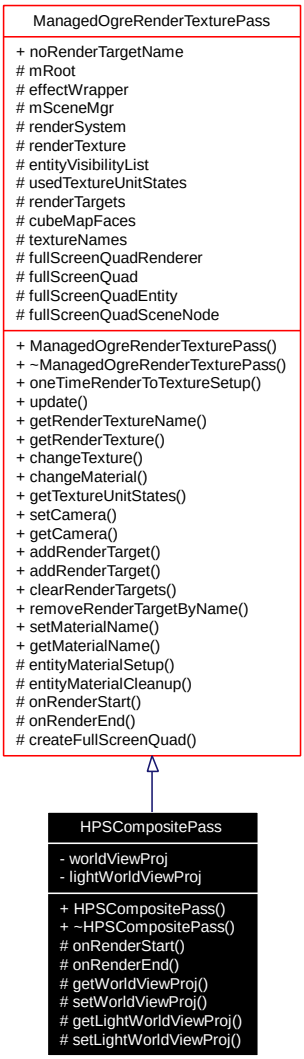
textureNameBack the name of the back depth texture used when rendering the particles of the blocks (optional: this information can be coded in the colortexture of the particle material

4.27 HPSCompositePass Class Reference

Inheritance diagram for HPSCompositePass:



Collaboration diagram for HPSCompositePass:



Public Member Functions

- HPSCompositePass (Root *mRoot)

  Constructor.

- ~HPSCompositePass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated.

- Matrix4 getWorldViewProj ()
- void setWorldViewProj (Matrix4 matrix4)

  Sets the transformation matrix (world-view-proj).

- Matrix4 getLightWorldViewProj ()

  Returns the transformation matrix (world-view-proj) from the lightsource.

- void setLightWorldViewProj (Matrix4 matrix4)

  Sets the transformation matrix (world-view-proj) from the light source.

4.27.1 Detailed Description

Rendering pass that computes composit textures for a particle system.

The composite texture stores the front and back depth information of a particle system, and also stores the accumulated density from the viewpoint.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 HPSCompositePass::HPSCompositePass (Root * *mRoot*)

Constructor.

4.27.2.2 HPSCompositePass::~HPSCompositePass () [inline]

Destructor.

4.27.3 Member Function Documentation

4.27.3.1 Matrix4 HPSCompositePass::getLightWorldViewProj () [protected]

Returns the transformation matrix (world-view-proj) from the lightsource.

4.27.3.2 Matrix4 HPSCompositePass::getWorldViewProj () [protected]

Returns the transformation matrix (world-view-proj)

4.27.3.3 void HPSCompositePass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated.

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.27.3.4 void HPSCompositePass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated.

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.27.3.5 void HPSCompositePass::setLightWorldViewProj (Matrix4 *matrix4*) [protected]

Sets the transformation matrix (world-view-proj) from the light source.

Parameters:
matrix4 the new value of the light transformation matrix.

4.27.3.6 void HPSCompositePass::setWorldViewProj (Matrix4 *matrix4*) [protected]

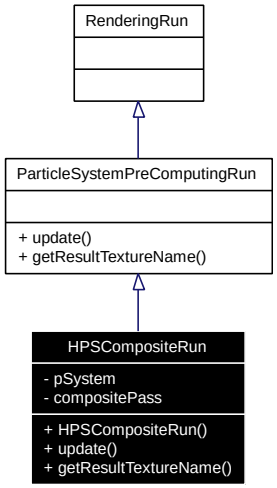
Sets the transformation matrix (world-view-proj).

Parameters:
matrix4 the new value of the transformation matrix.

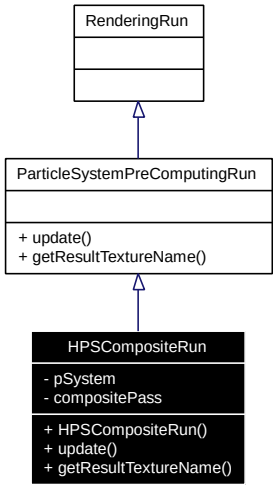
4.28 HPSCompositeRun Class Reference

Computes the composite texture for a given system and view camera.

Inheritance diagram for HPSCompositeRun:



Collaboration diagram for HPSCompositeRun:



Public Member Functions

- HPSCompositeRun (MovableObject *system)

  Constructor.

- virtual void update ()
- virtual const String & getResultTextureName ()

Returns the name of the computed composite texture.

4.28.1 Detailed Description

Computes the composite texture for a given system and view camera.

The composite texture stores the front and back depth information of a particle system, and also stores the accumulated density from the viewpoint.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 HPSCompositeRun::HPSCompositeRun (MovableObject * *system*) [inline]

Constructor.

Parameters:

system particle system, the owner object of a particle system based precomputing run.

4.28.3 Member Function Documentation

4.28.3.1 virtual const String& HPSCompositeRun::getResultTextureName () [inline, virtual]

Returns the name of the computed composite texture.

Reimplemented from ParticleSystemPreComputingRun.

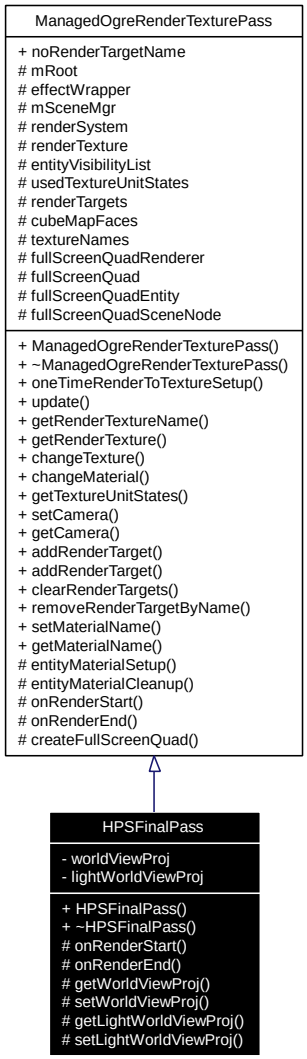
4.28.3.2 virtual void HPSCompositeRun::update (void) [inline, virtual]

Implements ParticleSystemPreComputingRun.

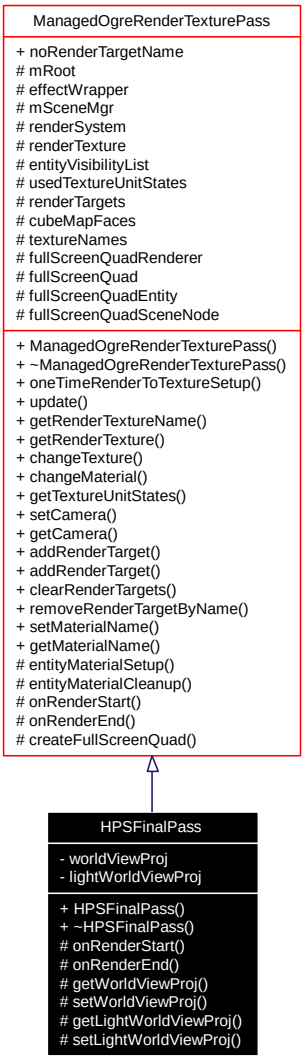
4.29 HPSFinalPass Class Reference

Rendering pass that renderes the particle system, with shading and without visual artifacts.

Inheritance diagram for HPSFinalPass:



Collaboration diagram for HPSFinalPass:



Public Member Functions

- HPSFinalPass (Root *mRoot)

  Constructor.

- ~HPSFinalPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated.

- Matrix4 getWorldViewProj ()

  Returns the transformation matrix (world-view-proj).

- void setWorldViewProj (Matrix4 matrix4)

  Sets the transformation matrix (world-view-proj).

- Matrix4 getLightWorldViewProj ()

  Returns the transformation matrix (world-view-proj) from the lightsource.

- void setLightWorldViewProj (Matrix4 matrix4)

  Sets the transformation matrix (world-view-proj) from the light source.

4.29.1 Detailed Description

Rendering pass that renderes the particle system, with shading and without visual artifacts.

The final rendering is made with the estimating of real lighting conditions, with a use of a light illumination texture calculated previously. The billboard clipping artifacts are elliminated with the use of depth and density information data also calculated previously in other passes.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 HPSFinalPass::HPSFinalPass (Root * *mRoot*)

Constructor.

4.29.2.2 HPSFinalPass::~HPSFinalPass () [inline]

Destructor.

4.29.3 Member Function Documentation

4.29.3.1 Matrix4 HPSFinalPass::getLightWorldViewProj () [protected]

Returns the transformation matrix (world-view-proj) from the lightsource.

4.29.3.2 Matrix4 HPSFinalPass::getWorldViewProj () [protected]

Returns the transformation matrix (world-view-proj).

4.29.3.3 void HPSFinalPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated.

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.29.3.4 void HPSFinalPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated.

Runs before the render-texture object is updated place all shader setup here.

Parameters:

namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.29.3.5 void HPSFinalPass::setLightWorldViewProj (Matrix4 *matrix4*) [protected]

Sets the transformation matrix (world-view-proj) from the light source.

Parameters:

matrix4 the new value of the light transformation matrix.

4.29.3.6 void HPSFinalPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

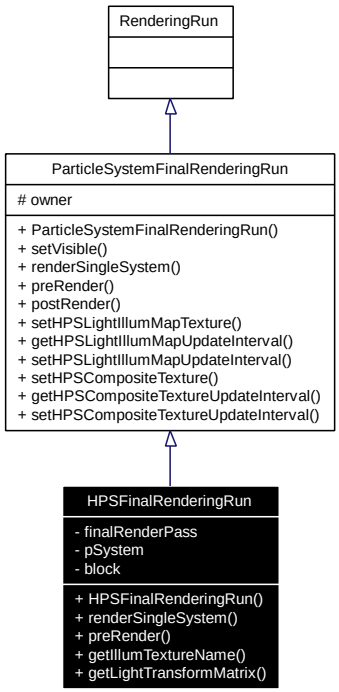
Sets the transformation matrix (world-view-proj).

Parameters:

matrix4 the new value of the transformation matrix.

4.30 HPSFinalRenderingRun Class Reference

Inheritance diagram for HPSFinalRenderingRun:



Collaboration diagram for HPSFinalRenderingRun:

Public Member Functions

- HPSFinalRenderingRun (ParticleSystem *system, bool isblock)

  Constructor.

- renderSingleSystem (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_-POSITIVE_X)

  Render the particle system to the frame buffer.

- virtual preRender (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_-X)

  Render object's depth in camera space.

- const String & getIllumTextureName ()

  Returns the system's illumination map name (can be used for shadows).

- const Matrix4 & getLightTransformMatrix ()

  Returns the transformation matrix (world-view-projection) of the light source illuminating the system (can be used for shadows).

4.30.1 Detailed Description

Final rendering run for displaying a shaded particle system.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 HPSFinalRenderingRun::HPSFinalRenderingRun (ParticleSystem * *system*, bool *isblock*) [inline]

Constructor.

Parameters:

system movable object (particle system), the owner object of a particle system based precomputing run.

isblock bool must be true if this system is a block of a bigger system.

4.30.3 Member Function Documentation

4.30.3.1 const String& HPSFinalRenderingRun::getIllumTextureName ()

Returns the system's illumination map name (can be used for shadows).

4.30.3.2 const Matrix4& HPSFinalRenderingRun::getLightTransformMatrix ()

Returns the transformation matrix (world-view-projection) of the light source illuminating the system (can be used for shadows).

4.30.3.3 virtual HPSFinalRenderingRun::preRender (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Render object's depth in camera space.

Reimplemented from ParticleSystemFinalRenderingRun.

4.30.3.4 HPSFinalRenderingRun::renderSingleSystem (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

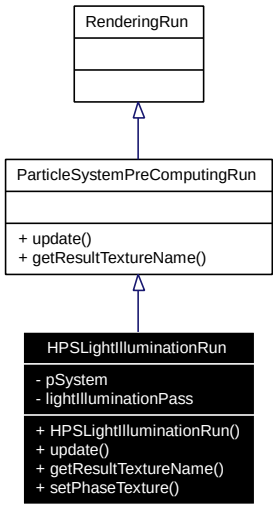
Render the particle system to the frame buffer.

Implements ParticleSystemFinalRenderingRun.

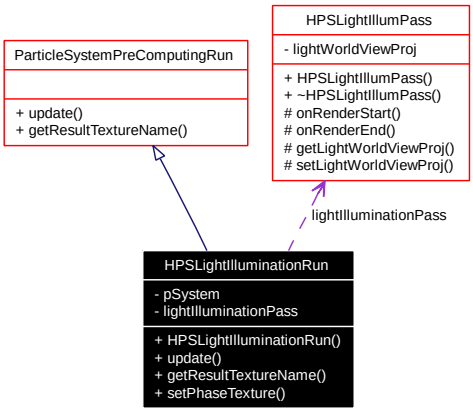
4.31 HPSLightIlluminationRun Class Reference

Computes light illumination map for a given system and light source.

Inheritance diagram for HPSLightIlluminationRun:



Collaboration diagram for HPSLightIlluminationRun:



Public Member Functions

- HPSLightIlluminationRun (MovableObject *system)

  Constructor.

- virtual void update ()
- virtual const String & getResultTextureName ()

  Returns the name of the computed illumination map.

- void setPhaseTexture (String &texturename)

  Sets the name of the texture containing phase function values. This texture is used to replace function computation with texture read.

4.31.1 Detailed Description

Computes light illumination map for a given system and light source.

A light illumination texture stores the extinct light intensity information in different depths.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 HPSLightIlluminationRun::HPSLightIlluminationRun (MovableObject * *system*) [inline]

Constructor.

Parameters:

system particle system, the owner object of a particle system based precomputing run.

4.31.3 Member Function Documentation

4.31.3.1 virtual const String& HPSLightIlluminationRun::getResultTextureName () [inline, virtual]

Returns the name of the computed illumination map.

Reimplemented from ParticleSystemPreComputingRun.

4.31.3.2 void HPSLightIlluminationRun::setPhaseTexture (String & *texturename*)

Sets the name of the texture containing phase function values. This texture is used to replace function computation with texture read.

Parameters:

texturename the name of the texture

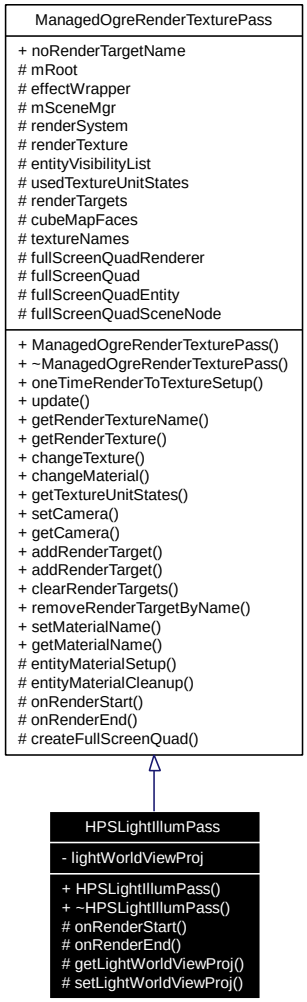
4.31.3.3 virtual void HPSLightIlluminationRun::update (void) [inline, virtual]

Implements ParticleSystemPreComputingRun.

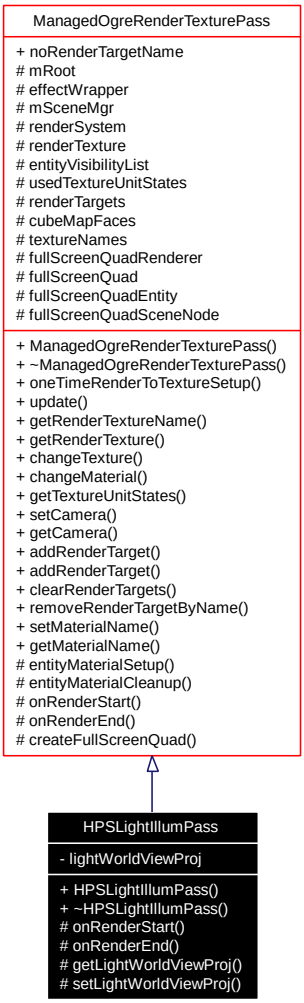
4.32 HPSLightIllumPass Class Reference

Rendering pass that calculates light illumination map for a particle system.

Inheritance diagram for HPSLightIllumPass:



Collaboration diagram for HPSLightIllumPass:



Public Member Functions

- HPSLightIllumPass (Root *mRoot)

  Constructor.

- ~HPSLightIllumPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated.

- Matrix4 getLightWorldViewProj ()

Returns the transformation matrix (world-view-proj) from the lightsource.

- void setLightWorldViewProj (Matrix4 matrix4)

  Sets the transformation matrix (world-view-proj) from the light source.

4.32.1 Detailed Description

Rendering pass that calculates light illumination map for a particle system.

A light illumination texture stores the extinct light intensity information in different depths. It is created with rendering the particles from the lightsource, and using a shader to store the extinction in different dephs in the separate color channels.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 HPSLightIllumPass::HPSLightIllumPass (Root * *mRoot*)

Constructor.

4.32.2.2 HPSLightIllumPass::~HPSLightIllumPass () [inline]

Destructor.

4.32.3 Member Function Documentation

4.32.3.1 Matrix4 HPSLightIllumPass::getLightWorldViewProj () [protected]

Returns the transformation matrix (world-view-proj) from the lightsource.

4.32.3.2 void HPSLightIllumPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated.

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.32.3.3 void HPSLightIllumPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated.

Runs before the render-texture object is updated place all shader setup here.

Parameters:

namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.32.3.4 void HPSLightIllumPass::setLightWorldViewProj (Matrix4 *matrix4*) [protected]

Sets the transformation matrix (world-view-proj) from the light source.

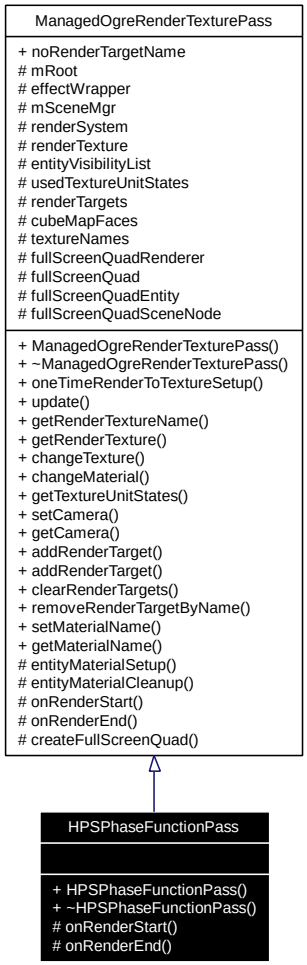
Parameters:

matrix4 the new value of the light transformation matrix.

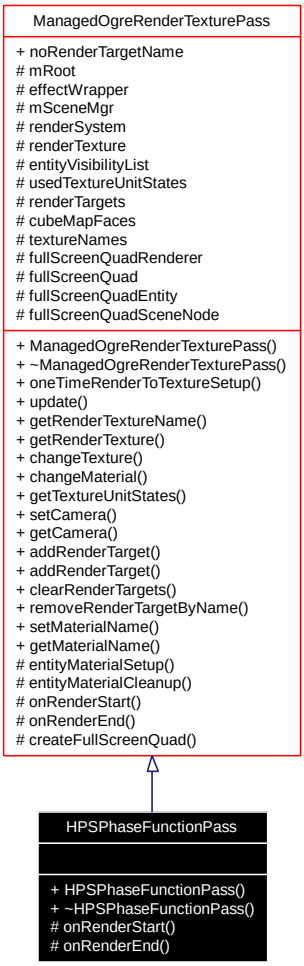
4.33 HPSPhaseFunctionPass Class Reference

Rendering pass that creates a texture containing phase function values.

Inheritance diagram for HPSPhaseFunctionPass:



Collaboration diagram for HPSPhaseFunctionPass:



Public Member Functions

- HPSPhaseFunctionPass (Root *mRoot)

  Constructor.

- ~HPSPhaseFunctionPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated.

4.33.1 Detailed Description

Rendering pass that creates a texture containing phase function values.

This texture is used to replace function computation with a texture read. It uses a Mie scattering function. The u coordinate represent the symmetry, while the v coordinate represents the cos of the angle between the incoming and the outgoing directions.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 HPSPhaseFunctionPass::HPSPhaseFunctionPass (Root * *mRoot*)

Constructor.

4.33.2.2 HPSPhaseFunctionPass::~HPSPhaseFunctionPass () [inline]

Destructor.

4.33.3 Member Function Documentation

4.33.3.1 void HPSPhaseFunctionPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated.

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.33.3.2 void HPSPhaseFunctionPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated.

Runs before the render-texture object is updated place all shader setup here.

Parameters:

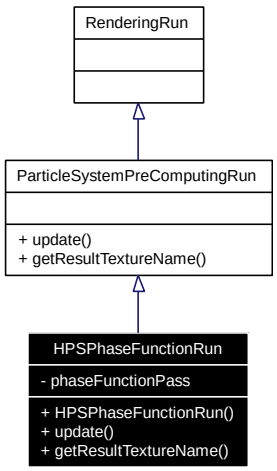
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

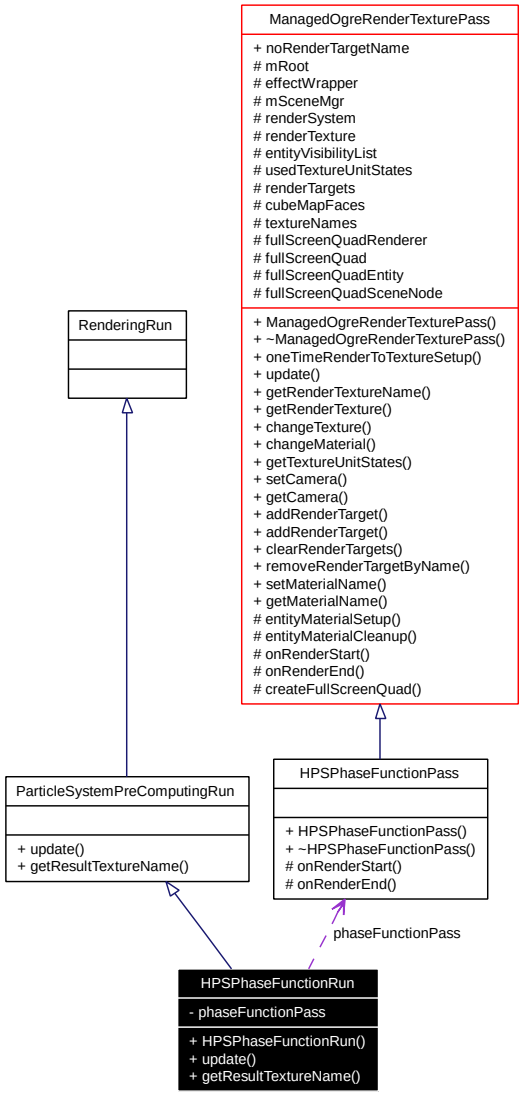
4.34 HPSPhaseFunctionRun Class Reference

Computes phace function values.

Inheritance diagram for HPSPhaseFunctionRun:



Collaboration diagram for HPSPhaseFunctionRun:



Public Member Functions

- HPSPhaseFunctionRun ()
- virtual void update ()
- virtual const String & getResultTextureName ()

  Returns the computed phase texture.

4.34.1 Detailed Description

Computes phace function values.

This texture is used to replace function computation with a texture read. It uses a Mie scattering function. The u coordinate represent the symmetry, while the v coordinate represents the cos of the angle between the incoming and the outgoing directions.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 HPSPhaseFunctionRun::HPSPhaseFunctionRun () [inline]

Constructor.

4.34.3 Member Function Documentation

4.34.3.1 virtual const String& HPSPhaseFunctionRun::getResultTextureName () [virtual]

Returns the computed phase texture.

Reimplemented from ParticleSystemPreComputingRun.

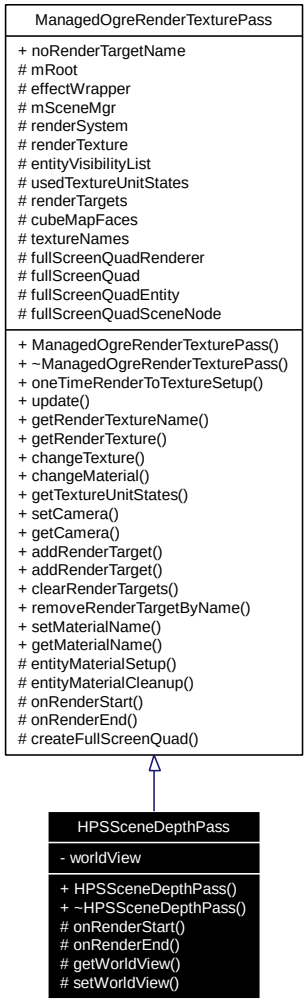
4.34.3.2 virtual void HPSPhaseFunctionRun::update () [virtual]

Implements ParticleSystemPreComputingRun.

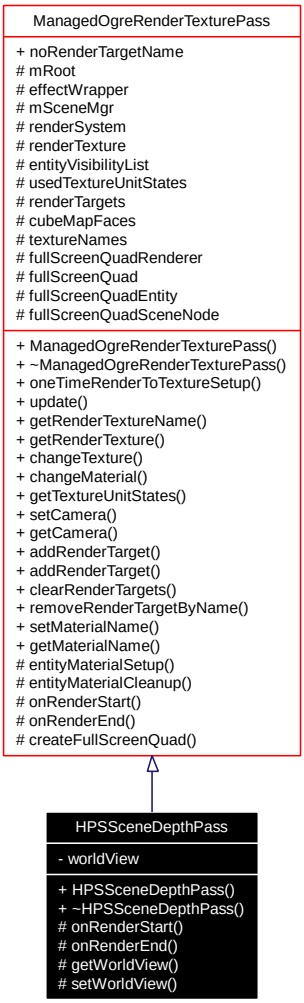
4.35 HPSSceneDepthPass Class Reference

A pass that renders the whole scene's depth map in camera space.

Inheritance diagram for HPSSceneDepthPass:



Collaboration diagram for HPSSceneDepthPass:



Public Member Functions

- HPSSceneDepthPass (Root *mRoot)

  Constructor.

- ~HPSSceneDepthPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated.

- Matrix4 getWorldView ()

Returns the transformation matrix (world-view-proj).

- void setWorldView (Matrix4 matrix4)

  Sets the transformation matrix (world-view).

4.35.1 Detailed Description

A pass that renders the whole scene's depth map in camera space.

This texture will be used for final rendering of particle systems if depth calculation is used. It has to be calculated once per frame from the camera's viewpoint.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 HPSSceneDepthPass::HPSSceneDepthPass (Root * *mRoot*)

Constructor.

4.35.2.2 HPSSceneDepthPass::~HPSSceneDepthPass () [inline]

Destructor.

4.35.3 Member Function Documentation

4.35.3.1 Matrix4 HPSSceneDepthPass::getWorldView () [protected]

Returns the transformation matrix (world-view-proj).

4.35.3.2 void HPSSceneDepthPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated.

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.35.3.3 void HPSSceneDepthPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated.

Runs before the render-texture object is updated place all shader setup here.

Parameters:

namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.35.3.4 void HPSSceneDepthPass::setWorldView (Matrix4 *matrix4*) [protected]

Sets the transformation matrix (world-view).

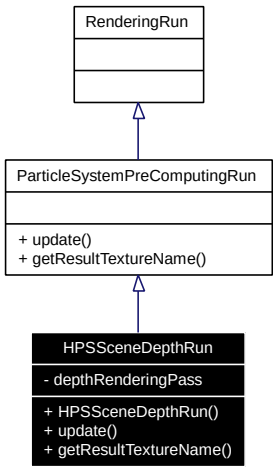
Parameters:

matrix4 the new value of the transformation matrix.

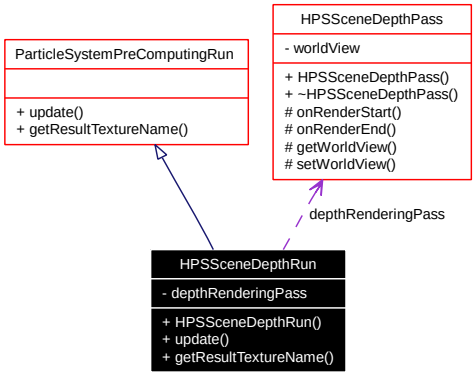
4.36 HPSSceneDepthRun Class Reference

Renders the whole scene's depth map in camera space.

Inheritance diagram for HPSSceneDepthRun:



Collaboration diagram for HPSSceneDepthRun:



Public Member Functions

- HPSSceneDepthRun ()
- virtual void update ()
- virtual const String & getResultTextureName ()

  Returns the depth texture.

4.36.1 Detailed Description

Renders the whole scene's depth map in camera space.

This texture will be used for final rendering of particle systems if depth calculation is used. It has to be calculated once per frame from the camera's viewpoint.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 HPSSceneDepthRun::HPSSceneDepthRun () [inline]

Constructor.

4.36.3 Member Function Documentation

4.36.3.1 virtual const String& HPSSceneDepthRun::getResultTextureName () [virtual]

Returns the depth texture.

Reimplemented from ParticleSystemPreComputingRun.

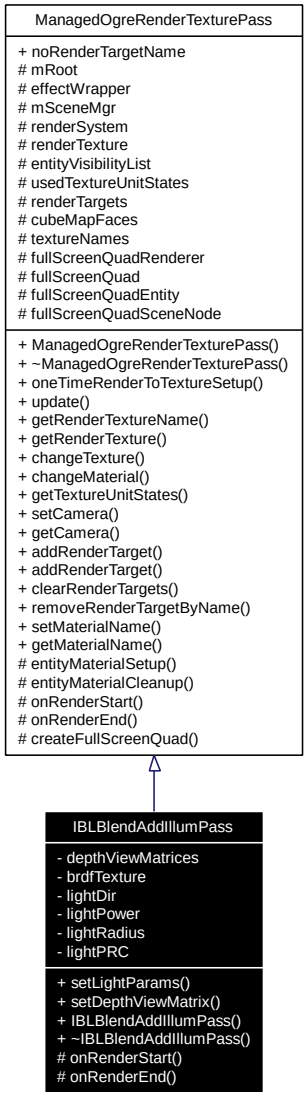
4.36.3.2 virtual void HPSSceneDepthRun::update () [virtual]

Implements ParticleSystemPreComputingRun.

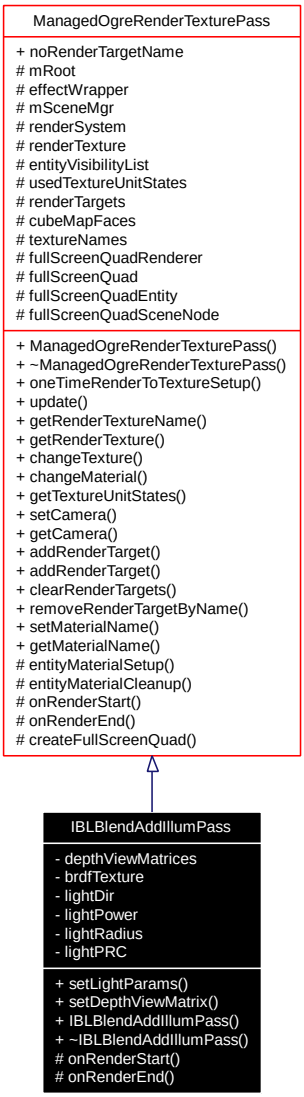
4.37 IBLBlendAddIllumPass Class Reference

Blend-adds illumination for four directional light samples.

Inheritance diagram for IBLBlendAddIllumPass:



Collaboration diagram for IBLBlendAddIllumPass:



Public Member Functions

- void setLightParams (Vector3 ∗dirs, Vector3 ∗powers, float ∗radii, float ∗prc)
- void setDepthViewMatrix (unsigned int index, const Matrix4 &m)
- IBLBlendAddIllumPass ()
- ∼IBLBlendAddIllumPass ()

Protected Member Functions

- void onRenderStart (NameValuePairList ∗namedParams=0)

  Runs before the render-texture object is updated place all shader setup here.

- void onRenderEnd (NameValuePairList ∗namedParams=0)

  Runs after the render-texture object is updated place all cleanup code here.

4.37.1 Detailed Description

Blend-adds illumination for four directional light samples.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 IBLBlendAddIllumPass::IBLBlendAddIllumPass ()

4.37.2.2 IBLBlendAddIllumPass::~IBLBlendAddIllumPass ()

4.37.3 Member Function Documentation

4.37.3.1 void IBLBlendAddIllumPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.37.3.2 void IBLBlendAddIllumPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

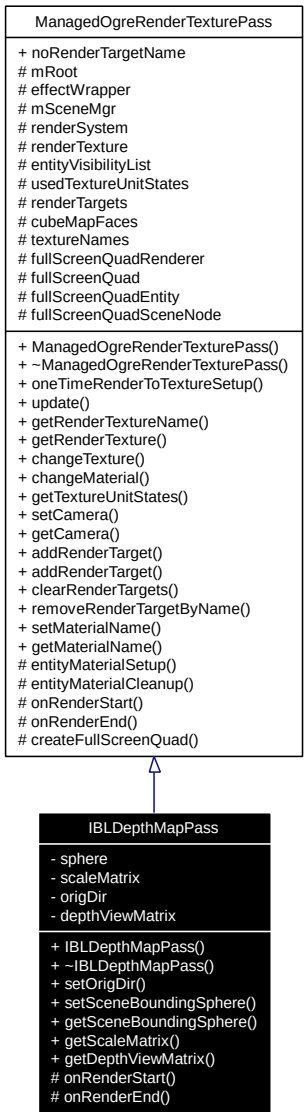
4.37.3.3 void IBLBlendAddIllumPass::setDepthViewMatrix (unsigned int *index*, const Matrix4 & *m*) [inline]

4.37.3.4 void IBLBlendAddIllumPass::setLightParams (Vector3 * *dirs*, Vector3 * *powers*, float * *radii*, float * *prc*) [inline]

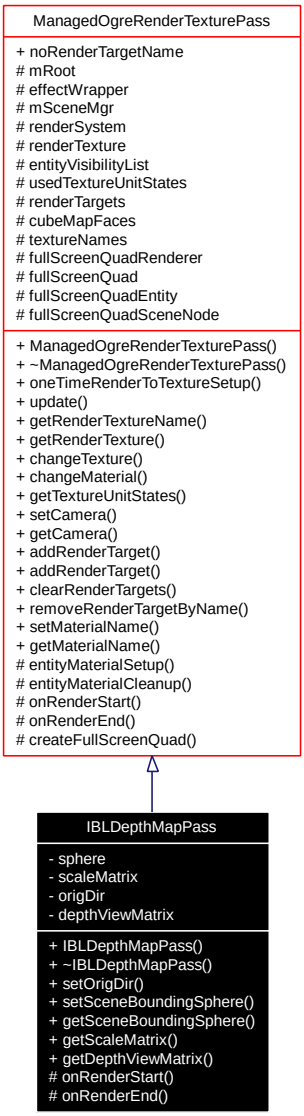
4.38 IBLDepthMapPass Class Reference

Renders a depth map for a directional light sample.

Inheritance diagram for IBLDepthMapPass:



Collaboration diagram for IBLDepthMapPass:



Public Member Functions

- IBLDepthMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D)
- ~IBLDepthMapPass ()
- void setOrigDir (const Vector3 &origDir)
- void setSceneBoundingSphere (const Sphere &sphere)
- Sphere getSceneBoundingSphere ()
- const Matrix4 & getScaleMatrix ()
- const Matrix4 & getDepthViewMatrix ()

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated place all shader setup here.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated place all cleanup code here.

4.38.1 Detailed Description

Renders a depth map for a directional light sample.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 IBLDepthMapPass::IBLDepthMapPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = TEX_TYPE_2D)

4.38.2.2 IBLDepthMapPass::~IBLDepthMapPass ()

4.38.3 Member Function Documentation

4.38.3.1 const Matrix4& IBLDepthMapPass::getDepthViewMatrix () [inline]

4.38.3.2 const Matrix4& IBLDepthMapPass::getScaleMatrix () [inline]

4.38.3.3 Sphere IBLDepthMapPass::getSceneBoundingSphere () [inline]

4.38.3.4 void IBLDepthMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.38.3.5 void IBLDepthMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:

namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

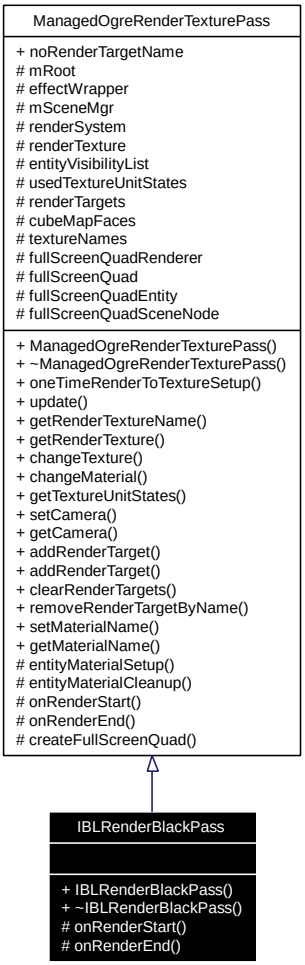
4.38.3.6 void IBLDepthMapPass::setOrigDir (const Vector3 & *origDir*) [inline]

4.38.3.7 void IBLDepthMapPass::setSceneBoundingSphere (const Sphere & *sphere*) [inline]

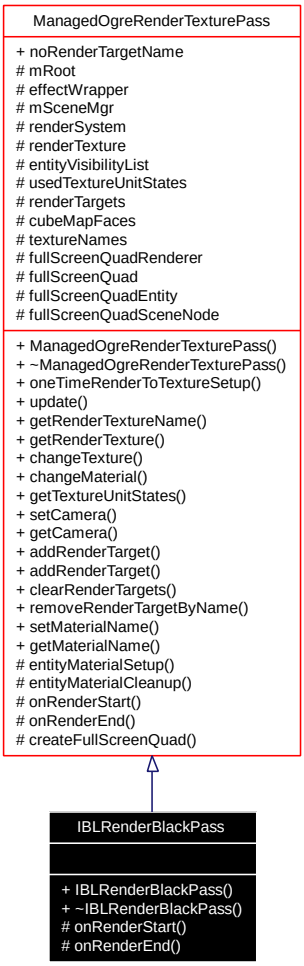
4.39 IBLRenderBlackPass Class Reference

Renders depth.

Inheritance diagram for IBLRenderBlackPass:



Collaboration diagram for IBLRenderBlackPass:



Public Member Functions

- IBLRenderBlackPass ()
- ~IBLRenderBlackPass ()

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated place all shader setup here.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated place all cleanup code here.

4.39.1 Detailed Description

Renders depth.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 IBLRenderBlackPass::IBLRenderBlackPass ()

4.39.2.2 IBLRenderBlackPass::~IBLRenderBlackPass ()

4.39.3 Member Function Documentation

4.39.3.1 void IBLRenderBlackPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.39.3.2 void IBLRenderBlackPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:

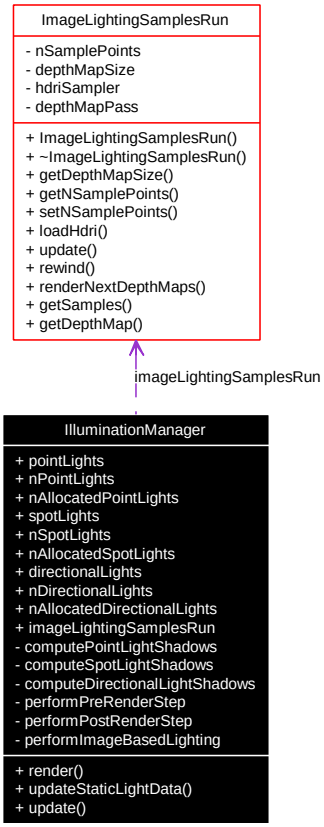
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.40 IlluminationManager Class Reference

A wrapper class for the illumination module data and functions.

Collaboration diagram for IlluminationManager:



Static Public Member Functions

- static void render (RenderTarget *rt, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)

  Render the scene as described for method update, but without executing any preprocessing. This method is typically called to execute a full final rendering on a render target different from the frame buffer. This is necessary e.g. for textured plane mirrors, or enviroment map faces.

- static void updateStaticLightData ()

  Scans the RenderSystem and re-allocates rendering runs. Updates the PreComputingRuns associated to lights.

- static void update (unsigned long frameNumber, RenderTarget *rt, CubeMapFaces cf=CUBEMAP_-FACE_POSITIVE_X)

Static Public Attributes

- static PointLightPreProcData * pointLights [32]
- static unsigned int nPointLights
- static unsigned int nAllocatedPointLights
- static SpotLightPreProcData * spotLights [32]
- static unsigned int nSpotLights
- static unsigned int nAllocatedSpotLights
- static DirectionalLightPreProcData * directionalLights [32]
- static unsigned int nDirectionalLights
- static unsigned int nAllocatedDirectionalLights
- static ImageLightingSamplesRun imageLightingSamplesRun

Classes

- class **DirectionalLightPreProcData**

  An Ogre directional light with a depth map.

- class **LightPreProcData**

  Base structure for illumination module light sources, augmenting Ogre::Light data, and separating Light types. Subclasses contain PreProcessingRuns for per light preprocessing data, or extended light information like soft shadow light radius.

- class **PointLightPreProcData**

  An Ogre point light with a depth cube.

- class **SpotLightPreProcData**

  An Ogre spot light with a depth map.

4.40.1 Detailed Description

A wrapper class for the illumination module data and functions.

4.40.2 Member Function Documentation

4.40.2.1 static void IlluminationManager::render (RenderTarget * *rt*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [static]

Render the scene as described for method update, but without executing any preprocessing. This method is typically called to execute a full final rendering on a render target different from the frame buffer. This is necessary e.g. for textured plane mirrors, or enviroment map faces.

Parameters:

rt The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

4.40.2.2 static void IlluminationManager::update (unsigned long *frameNumber*, RenderTarget * *rt*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [static]

This method replaces the standard OGRE rendering pipeline. As opposed to the standard pipeline, entities do not only have a single assigned material, but a FinalRenderingRun of arbitrary complexity, including multiple passes and switching render targets between them. FinalRenderingRuns also define what preprocessing is to be done in a frame. Before rendering all the entities using their associated FinalRenderingRun's FinalRenderingRun::renderSingleEntity, the necessary precomputed data is manufactured for every entity using its EntityRenderingObject's EntityRenderingObject::update.

Non entity-bound preprocessing (like creating spot light depth maps) is performed regularly, independent of entities within the scene.

Parameters:

frameNumber The current frame number. This is used to determine which preprocessing runs should be executed to update preprocessed data.

rt The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

4.40.2.3 static void IlluminationManager::updateStaticLightData () [static]

Scans the RenderSystem and re-allocates rendering runs. Updates the PreComputingRuns associated to lights.

4.40.3 Member Data Documentation

4.40.3.1 DirectionalLightPreProcData* IlluminationManager::directionalLights[32] [static]

The array of directional lights.

4.40.3.2 ImageLightingSamplesRun IlluminationManager::imageLightingSamplesRun [static]

4.40.3.3 unsigned int IlluminationManager::nAllocatedDirectionalLights [static]

The number of allocated items in the array of directional lights.

4.40.3.4 unsigned int IlluminationManager::nAllocatedPointLights [static]

The number of allocated items in the array of point lights.

4.40.3.5 unsigned int IlluminationManager::nAllocatedSpotLights [static]

The number of allocated items in the array of spot lights.

4.40.3.6 unsigned int IlluminationManager::nDirectionalLights [static]

The number of actual lights in the array of directional lights.

4.40.3.7 unsigned int IlluminationManager::nPointLights [static]

The number of actual lights in the array of point lights.

4.40.3.8 unsigned int IlluminationManager::nSpotLights [static]

The number of actual lights in the array of spot lights.

4.40.3.9 PointLightPreProcData* IlluminationManager::pointLights[32] [static]

The array of point lights.

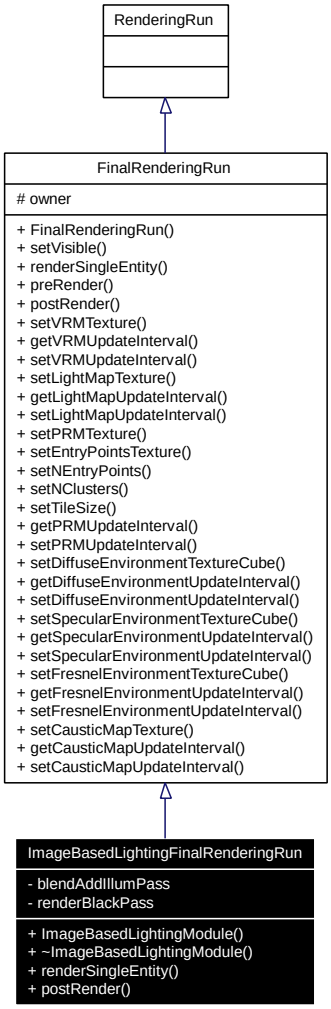
4.40.3.10 SpotLightPreProcData* IlluminationManager::spotLights[32] [static]

The array of spot lights.

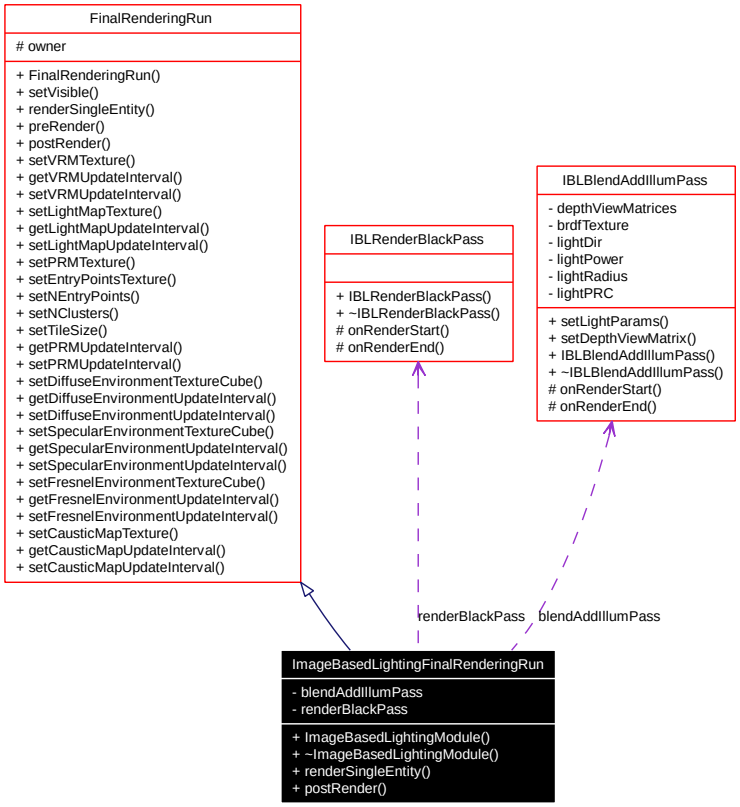
4.41 ImageBasedLightingFinalRenderingRun Class Reference

Renders image based lighting.

Inheritance diagram for ImageBasedLightingFinalRenderingRun:



Collaboration diagram for ImageBasedLightingFinalRenderingRun:



Public Member Functions

- ImageBasedLightingModule (Entity *owner)
- ~ImageBasedLightingModule ()
- renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_-POSITIVE_X)

  Render everything in black.

- postRender (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)

  Blend-add illumination for all environment samples. Depth maps are generated on-the-fly.

4.41.1 Detailed Description

Renders image based lighting.

Uses results from IlluminationModule::imageLightingSamplesRun. There are two steps:

- renderSingleEntity only renders depth (and black colour).
- postRender is called for every (quad of) directional light sample and every entity by IlluminationManager::render.

4.41.2 Constructor & Destructor Documentation

4.41.2.1 ImageBasedLightingFinalRenderingRun::~ImageBasedLightingModule ()

Destructor.

4.41.3 Member Function Documentation

4.41.3.1 ImageBasedLightingFinalRenderingRun::ImageBasedLightingModule (Entity * *owner*)

Constructor.

Parameters:

owner Owner entity.

4.41.3.2 ImageBasedLightingFinalRenderingRun::postRender (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Blend-add illumination for all environment samples. Depth maps are generated on-the-fly.

Parameters:

backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Reimplemented from FinalRenderingRun.

4.41.3.3 ImageBasedLightingFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Render everything in black.

Parameters:

backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

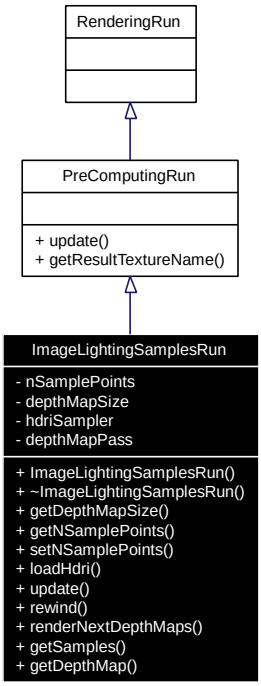
cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Implements FinalRenderingRun.

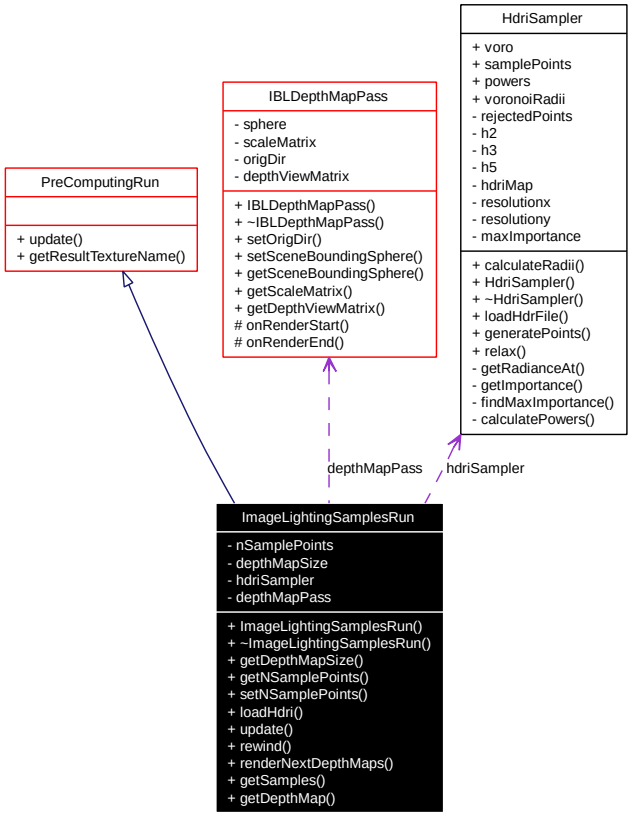
4.42 ImageLightingSamplesRun Class Reference

Precomputing run that computes environment lighting samples.

Inheritance diagram for ImageLightingSamplesRun:



Collaboration diagram for ImageLightingSamplesRun:



Public Member Functions

- ImageLightingSamplesRun (unsigned int depthMapSize, const String &environHdriPath, unsigned int nSamplePoints)

  Contructor.

- ~ImageLightingSamplesRun ()

  Destructor.

- unsigned int getDepthMapSize ()
- unsigned int getNSamplePoints ()
- void setNSamplePoints (unsigned int nSamplePoints)
- void loadHdri (const String &environHdriPath)
- void update ()
- void rewind ()
- bool renderNextDepthMaps ()
- HdriSampler * getSamples ()
- const String & getDepthMap (unsigned int index)

4.42.1 Detailed Description

Precomputing run that computes environment lighting samples.

Precomputing run that computes

- samples of the image illumination using importance sampling, Voronoi tesselation and Lloyd's relaxation.
- depth maps for the samples However, only four depth maps are stored at a time (they must be frequently updated and would take up too much resources.) Method renderNextDepthMaps must be used to render depth maps of the next quad of samples. This is handled by IlluminationManager::render.

4.42.2 Constructor & Destructor Documentation

4.42.2.1 ImageLightingSamplesRun::ImageLightingSamplesRun (unsigned int *depthMapSize*, const String & *environHdriPath*, unsigned int *nSamplePoints*)

Contructor.

Parameters:

depthMapSize The size of the depth map.

environHdriPath Path to the High Dynamic Range Image file used for illumination.

nSamplePoints Number of directional samples to be generated.

4.42.2.2 ImageLightingSamplesRun::~ImageLightingSamplesRun ()

Destructor.

4.42.3 Member Function Documentation

4.42.3.1 const String& ImageLightingSamplesRun::getDepthMap (unsigned int *index*) [inline]

Parameters:

index The index of the depth map texture to be retrieved. There are 4 textures. renderNextDepthMaps will update them with the next 4 values.

Returns:

The depth map texture specified by index (0..3).

4.42.3.2 unsigned int ImageLightingSamplesRun::getDepthMapSize () [inline]

Returns:

Depth map resolution.

4.42.3.3 unsigned int ImageLightingSamplesRun::getNSamplePoints () [inline]

Returns:
Number of directional samples generated.

4.42.3.4 HdriSampler* ImageLightingSamplesRun::getSamples () [inline]

Returns:
The High Dynamic Range Image sampler holding the directional samples.

4.42.3.5 void ImageLightingSamplesRun::loadHdri (const String & *environHdriPath*)

Parameters:
environHdriPath Path to the High Dynamic Range Image file used for illumination.

4.42.3.6 bool ImageLightingSamplesRun::renderNextDepthMaps ()

Render depth maps of next quadruple of samples into by depthsMapPass[0..3].

Returns:
true if successful, false if no samples are left.

4.42.3.7 void ImageLightingSamplesRun::rewind ()

Next renderNextDepthMaps will render the depth maps of the first quadruple of samples.

4.42.3.8 void ImageLightingSamplesRun::setNSamplePoints (unsigned int *nSamplePoints*) [inline]

Parameters:
nSamplePoints Number of directional samples to be generated.

4.42.3.9 void ImageLightingSamplesRun::update () [virtual]

Inherited from PreComputingRun. Recomputes samples.

Implements PreComputingRun.

4.43 Impostor Class Reference

Impostor class. An impostor is suitable to replace an object with its image. This Impostor Class implements a dinamically generated impostor, which refreshes the object's image only when the camera moved "enough".

Public Member Functions

- Impostor (String name)
- void setViewCamera (Camera *camera)

  Sets the viewer's camera.

- bool needRefresh ()

  Returns weather the impostor should be refreshed.

- void setCameraProjection ()

  Sets the view and projection matrices of the impostor's own camera for rendering.

- Camera * get_ownCamera ()

  Returns the impostor's own camera.

- const Real get_BoundingRadius () const

  Returns the bounding radius of the represented object.

- void set_BoundingRadius (Real value)

  Sets the bounding radius of the represented object.

- const Vector3 get_ObjectPosition () const

  Returns the position of the represented object.

- void set_ObjectPosition (Vector3 value)

  Sets the position of the represented object.

- const Real get_errorTolerance () const

  Returns the error tolarance of the impostor. Error tolarance describes the amount of move can be made by the camera without refresh.

- void set_errorTolerance (Real value)

  Sets the error tolarance of the impostor. Error tolarance describes the amount of move can be made by the camera without refresh.

4.43.1 Detailed Description

Impostor class. An impostor is suitable to replace an object with its image. This Impostor Class implements a dinamically generated impostor, which refreshes the object's image only when the camera moved "enough".

4.43.2 Constructor & Destructor Documentation

4.43.2.1 Impostor::Impostor (String *name*) [inline]

4.43.3 Member Function Documentation

4.43.3.1 const Real Impostor::get_BoundingRadius () const [inline]

Returns the bounding radius of the represented object.

The bounding radius of the represented object is needed when determining the need of refresh, and setting own camera projection.

4.43.3.2 const Real Impostor::get_errorTolerance () const [inline]

Returns the error tolarance of the impostor. Error tolarance describes the amount of move can be made by the camera without refresh.

4.43.3.3 const Vector3 Impostor::get_ObjectPosition () const [inline]

Returns the position of the represented object.

4.43.3.4 Camera* Impostor::get_ownCamera () [inline]

Returns the impostor's own camera.

The eye camera and the impostor's own camera is not the same, as the own camera always looks toward the represented object. The impostor should only be refreshed if the camera moves, not when it turns. Thus the eye camera and the own camera will always have the same position, but not the same direction and projection settings.

Returns:
The Systems own camera.

4.43.3.5 bool Impostor::needRefresh ()

Returns weather the impostor should be refreshed.

4.43.3.6 void Impostor::set_BoundingRadius (Real *value*) [inline]

Sets the bounding radius of the represented object.

The bounding radius of the represented object is needed when determining the need of refresh, and setting own camera projection.

Parameters:

value Float number, sets the represented object's bounding radius.

4.43.3.7 void Impostor::set_errorTolerance (Real *value*) [inline]

Sets the error tolarance of the impostor. Error tolarance describes the amount of move can be made by the camera without refresh.

4.43.3.8 void Impostor::set_ObjectPosition (Vector3 *value*) [inline]

Sets the position of the represented object.

4.43.3.9 void Impostor::setCameraProjection ()

Sets the view and projection matrices of the impostor's own camera for rendering.

4.43.3.10 void Impostor::setViewCamera (Camera * *camera*) [inline]

Sets the viewer's camera.

4.44 InfoPlane Struct Reference

Public Attributes

- Vector3 topLeft
- Vector3 topRight
- Vector3 bottomRight
- Vector3 bottomLeft
- Vector3 normal
- Real d

4.44.1 Member Data Documentation

4.44.1.1 Vector3 InfoPlane::bottomLeft

4.44.1.2 Vector3 InfoPlane::bottomRight

4.44.1.3 Real InfoPlane::d

4.44.1.4 Vector3 InfoPlane::normal

4.44.1.5 Vector3 InfoPlane::topLeft

4.44.1.6 Vector3 InfoPlane::topRight

4.45 Leaf Struct Reference

Public Attributes

- unsigned int idLeaf
- Vector3 point
- vector< unsigned int > vFaces
- Vector4 coord
- vector< unsigned int > vFaces
- Real factorU
- Real factorV
- vector< unsigned int > vFaces
- vector< unsigned int > vFaces

4.45.1 Member Data Documentation

4.45.1.1 Vector4 Leaf::coord

4.45.1.2 Real Leaf::factorU

4.45.1.3 Real Leaf::factorV

4.45.1.4 unsigned int Leaf::idLeaf

4.45.1.5 Vector3 Leaf::point

4.45.1.6 vector<unsigned int> Leaf::vFaces

4.45.1.7 vector<unsigned int> Leaf::vFaces

4.45.1.8 vector<unsigned int> Leaf::vFaces

4.45.1.9 vector<unsigned int> Leaf::vFaces

4.46 LeafNormals Struct Reference

Public Attributes

- vector< Vector3 > normals
- vector< Vector3 > normals

4.46.1 Member Data Documentation

4.46.1.1 vector<Vector3> LeafNormals::normals

4.46.1.2 vector<Vector3> LeafNormals::normals

4.47 LeavesGenerator Class Reference

This class recieve as input meshes that contains leaves vertexs and faces and do a preprocessing step for trying to guess which faces are used to define each leaf of the tree.

Public Member Functions

- LeavesGenerator (void)

  Constructor method of the class `LeavesGenerator`.

- ~LeavesGenerator (void)

  Destructor method of the class `LeavesGenerator`.

- LeavesGenerator * getSingletonPtr (void)

  Return pointer to singleton LeavesGenerator object.

- LeavesGenerator & getSingleton (void)

  Return singleton LeavesGenerator object.

- void generateLeaves (char *filenames[ ])

  This method check the mesh information to guess which faces define each leaf.

- LeavesGenerator (void)

  Constructor method of the class `LeavesGenerator`.

- ~LeavesGenerator (void)

  Destructor method of the class `LeavesGenerator`.

- LeavesGenerator * getSingletonPtr (void)

  Return pointer to singleton LeavesGenerator object.

- LeavesGenerator & getSingleton (void)

  Return singleton LeavesGenerator object.

- void generateLeaves (char *filenames[ ])

  This method check the mesh information to guess which faces define each leaf.

Protected Types

- typedef set< int, greater< int > > ID_FACESet
- typedef set< int, greater< int > > ID_FACESet

Protected Attributes

- Vector3 vertex
- TiXmlNode * node
- TiXmlNode * nodeIt
- TiXmlNode * nodeIt2
- char * cNomFitxer
- TIXML_STRING * nomFitxer
- vector< Vector3 > vFaces
- vector< Vector3 > vVertexs
- vector< ID_FACESet > vLeaves
- ID_FACESet::iterator iLeaves
- vector< Vector3 >::iterator iFaces01
- vector< Vector3 >::iterator iFaces02
- vector< Vector3 >::iterator iVertexs
- vector< Real > vD
- vector< Real > vNx
- vector< Real > vNy
- vector< Real > vNz
- vector< Vector3 > vLeavesPoint
- vector< LeafNormals > vLeavesNormals
- vector< Vector3 > vNormals
- unsigned int v1
- unsigned int v2
- unsigned int v3
- unsigned int nvertexs
- unsigned int iface
- unsigned int ivertex
- bool loadOkay
- double x1
- double y1
- double z1
- unsigned int iV101
- unsigned int iV201
- unsigned int iV301
- unsigned int iV102
- unsigned int iV202
- unsigned int iV302
- Vector3 v101
- Vector3 v201
- Vector3 v301
- Vector3 v102
- Vector3 v202
- Vector3 v302
- unsigned int iface01
- unsigned int iface02
- time_t timer
- tm * initTime
- TIXML_STRING * outputFilename
- TiXmlNode * rootNode

- TiXmlNode * dNode
- TiXmlNode * nXNode
- TiXmlNode * nYNode
- TiXmlNode * nZNode
- TiXmlNode * leavesNode
- TiXmlNode * leafNode
- TiXmlNode * coord4dNode
- TiXmlNode * pointNode
- TiXmlNode * facesNode
- TiXmlNode * faceNode
- unsigned int posNorm
- unsigned int negNorm
- TiXmlDocument outputFile
- TiXmlNode * node
- TiXmlNode * nodeIt
- char * cNomFitxer
- TIXML_STRING * nomFitxer
- vector< Vector3 > vFaces
- vector< Vector3 > vVertexs
- vector< ID_FACESet > vLeaves
- vector< Vector3 >::iterator iFaces01
- vector< Vector3 >::iterator iFaces02
- vector< Vector3 >::iterator iVertexs
- vector< Real > vD
- vector< Real > vNx
- vector< Real > vNy
- vector< Real > vNz
- vector< Vector3 > vLeavesPoint
- vector< LeafNormals > vLeavesNormals
- vector< Vector3 > vNormals
- tm * initTime
- TIXML_STRING * outputFilename
- TiXmlNode * rootNode
- TiXmlNode * dNode
- TiXmlNode * nXNode
- TiXmlNode * nYNode
- TiXmlNode * nZNode
- TiXmlNode * leavesNode
- TiXmlNode * leafNode
- TiXmlNode * coord4dNode
- TiXmlNode * pointNode
- TiXmlNode * facesNode
- TiXmlNode * faceNode

4.47.1 Detailed Description

This class recieve as input meshes that contains leaves vertexs and faces and do a preprocessing step for trying to guess which faces are used to define each leaf of the tree.

The main tasks of the class `LeavesGenerator` are:

- Load the mesh file that contains all the faces of the leaves.
- Generate sets of faces that should be in the same leaf.
- Store a xml file that contains the number of leaves found, and the list of faces that each leaf contain.

4.47.2 Member Typedef Documentation

4.47.2.1 typedef set<int,greater<int> > LeavesGenerator::ID_FACESet [protected]

4.47.2.2 typedef set<int,greater<int> > LeavesGenerator::ID_FACESet [protected]

4.47.3 Constructor & Destructor Documentation

4.47.3.1 LeavesGenerator::LeavesGenerator (void) [inline]

Constructor method of the class `LeavesGenerator`.

4.47.3.2 LeavesGenerator::~LeavesGenerator (void) [inline]

Destructor method of the class `LeavesGenerator`.

4.47.3.3 LeavesGenerator::LeavesGenerator (void) [inline]

Constructor method of the class `LeavesGenerator`.

4.47.3.4 LeavesGenerator::~LeavesGenerator (void) [inline]

Destructor method of the class `LeavesGenerator`.

4.47.4 Member Function Documentation

4.47.4.1 void LeavesGenerator::generateLeaves (char * *filenames*[ ]) [inline]

This method check the mesh information to guess which faces define each leaf.

Parameters:
filenames Strings that contains all the filenames needed

4.47.4.2 void LeavesGenerator::generateLeaves (char * *filenames*[ ]) [inline]

This method check the mesh information to guess which faces define each leaf.

Parameters:
filenames Strings that contains all the filenames needed

4.47.4.3 LeavesGenerator& LeavesGenerator::getSingleton (void) [inline]

Return singleton LeavesGenerator object.

Returns:
Singleton LeavesGenerator object

4.47.4.4 LeavesGenerator& LeavesGenerator::getSingleton (void) [inline]

Return singleton LeavesGenerator object.

Returns:
Singleton LeavesGenerator object

4.47.4.5 LeavesGenerator* LeavesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton LeavesGenerator object.

Returns:
Pointer to singleton LeavesGenerator object

4.47.4.6 LeavesGenerator* LeavesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton LeavesGenerator object.

Returns:
Pointer to singleton LeavesGenerator object

4.47.5 Member Data Documentation

4.47.5.1 char* LeavesGenerator::cNomFitxer [protected]

4.47.5.2 char* LeavesGenerator::cNomFitxer [protected]

4.47.5.3 TiXmlNode* LeavesGenerator::coord4dNode [protected]

4.47.5.4 TiXmlNode* LeavesGenerator::coord4dNode [protected]

4.47.5.5 TiXmlNode* LeavesGenerator::dNode [protected]

4.47.5.6 TiXmlNode* LeavesGenerator::dNode [protected]

4.47.5.7 TiXmlNode* LeavesGenerator::faceNode [protected]

4.47.5.8 TiXmlNode* LeavesGenerator::faceNode [protected]

4.47.5.9 TiXmlNode* LeavesGenerator::facesNode [protected]

4.47.5.10 TiXmlNode* LeavesGenerator::facesNode [protected]

4.47.5.11 unsigned int LeavesGenerator::iface [protected]

4.47.5.12 unsigned int LeavesGenerator::iface01 [protected]

4.47.5.13 unsigned int LeavesGenerator::iface02 [protected]

4.47.5.14 vector<Vector3>::iterator LeavesGenerator::iFaces01 [protected]

4.47.5.15 vector<Vector3>::iterator LeavesGenerator::iFaces01 [protected]

4.47.5.16 vector<Vector3>::iterator LeavesGenerator::iFaces02 [protected]

4.47.5.17 vector<Vector3>::iterator LeavesGenerator::iFaces02 [protected]

4.47.5.18 ID_FACESet::iterator LeavesGenerator::iLeaves [protected]

4.47.5.19 struct tm* LeavesGenerator::initTime [protected]

4.47.5.20 struct tm* LeavesGenerator::initTime [protected]

4.47.5.21 unsigned int LeavesGenerator::iV101 [protected]

4.47.5.22 unsigned int LeavesGenerator::iV102 [protected]

4.47.5.23 unsigned int LeavesGenerator::iV201 [protected]

4.47.5.24 unsigned int LeavesGenerator::iV202 [protected]

4.47.5.25 unsigned int LeavesGenerator::iV301 [protected]

4.47.5.26 unsigned int LeavesGenerator::iV302 [protected]

4.47.5.27 unsigned int LeavesGenerator::ivertex [protected]

4.47.5.28 vector<Vector3>::iterator LeavesGenerator::iVertexs [protected]

4.47.5.29 vector<Vector3>::iterator LeavesGenerator::iVertexs [protected]

4.47.5.30 TiXmlNode* LeavesGenerator::leafNode [protected]

4.47.5.31 TiXmlNode* LeavesGenerator::leafNode [protected]

4.47.5.32 TiXmlNode* LeavesGenerator::leavesNode [protected]

4.47.5.33 TiXmlNode* LeavesGenerator::leavesNode [protected]

4.47.5.34 bool LeavesGenerator::loadOkay [protected]

4.47.5.35 unsigned int LeavesGenerator::negNorm [protected]

4.47.5.36 TiXmlNode* LeavesGenerator::node [protected]

4.47.5.37 TiXmlNode* LeavesGenerator::node [protected]

4.47.5.38 TiXmlNode* LeavesGenerator::nodeIt [protected]

4.47.5.39 TiXmlNode* LeavesGenerator::nodeIt [protected]

4.47.5.40 TiXmlNode * LeavesGenerator::nodeIt2 [protected]

4.47.5.41 TIXML_STRING* LeavesGenerator::nomFitxer [protected]

4.47.5.42 TIXML_STRING* LeavesGenerator::nomFitxer [protected]

4.47.5.43 unsigned int LeavesGenerator::nvertexs [protected]

4.47.5.44 TiXmlNode* LeavesGenerator::nXNode [protected]

4.47.5.45 TiXmlNode* LeavesGenerator::nXNode [protected]

4.47.5.46 TiXmlNode* LeavesGenerator::nYNode [protected]

4.47.5.47 TiXmlNode* LeavesGenerator::nYNode [protected]

4.47.5.48 TiXmlNode* LeavesGenerator::nZNode [protected]

4.47.5.49 TiXmlNode* LeavesGenerator::nZNode [protected]

4.47.5.50 TiXmlDocument LeavesGenerator::outputFile [protected]

4.47.5.51 TIXML_STRING* LeavesGenerator::outputFilename [protected]

4.47.5.52 TIXML_STRING* LeavesGenerator::outputFilename [protected]

4.47.5.53 TiXmlNode* LeavesGenerator::pointNode [protected]

4.47.5.54 TiXmlNode* LeavesGenerator::pointNode [protected]

4.47.5.55 unsigned int LeavesGenerator::posNorm [protected]

4.47.5.56 TiXmlNode* LeavesGenerator::rootNode [protected]

4.47.5.57 TiXmlNode* LeavesGenerator::rootNode [protected]

4.47.5.58 time_t LeavesGenerator::timer [protected]

4.47.5.59 unsigned int LeavesGenerator::v1 [protected]

4.47.5.60 Vector3 LeavesGenerator::v101 [protected]

4.47.5.61 Vector3 LeavesGenerator::v102 [protected]

4.47.5.62 unsigned int LeavesGenerator::v2 [protected]

4.47.5.63 Vector3 LeavesGenerator::v201 [protected]

4.47.5.64 Vector3 LeavesGenerator::v202 [protected]

4.47.5.65 unsigned int LeavesGenerator::v3 [protected]

4.47.5.66 Vector3 LeavesGenerator::v301 [protected]

4.47.5.67 Vector3 LeavesGenerator::v302 [protected]

4.47.5.68 vector<Real> LeavesGenerator::vD [protected]

4.47.5.69 vector<Real> LeavesGenerator::vD [protected]

4.47.5.70 Vector3 LeavesGenerator::vertex [protected]

4.47.5.71 vector<Vector3> LeavesGenerator::vFaces [protected]

4.47.5.72 vector<Vector3> LeavesGenerator::vFaces [protected]

4.47.5.73 vector<ID_FACESet> LeavesGenerator::vLeaves [protected]

4.47.5.74 vector<ID_FACESet> LeavesGenerator::vLeaves [protected]

4.47.5.75 vector<LeafNormals> LeavesGenerator::vLeavesNormals [protected]

4.47.5.76 vector<LeafNormals> LeavesGenerator::vLeavesNormals [protected]

4.47.5.77 vector<Vector3> LeavesGenerator::vLeavesPoint [protected]

4.47.5.78 vector<Vector3> LeavesGenerator::vLeavesPoint [protected]

4.47.5.79 vector<Vector3> LeavesGenerator::vNormals [protected]

4.47.5.80 vector<Vector3> LeavesGenerator::vNormals [protected]

4.47.5.81 vector<Real> LeavesGenerator::vNx [protected]

4.47.5.82 vector<Real> LeavesGenerator::vNx [protected]

4.47.5.83 vector<Real> LeavesGenerator::vNy [protected]

4.47.5.84 vector<Real> LeavesGenerator::vNy [protected]

4.47.5.85 vector<Real> LeavesGenerator::vNz [protected]

4.47.5.86 vector<Real> LeavesGenerator::vNz [protected]

4.47.5.87 vector<Vector3> LeavesGenerator::vVertexs [protected]

4.47.5.88 vector<Vector3> LeavesGenerator::vVertexs [protected]

4.47.5.89 double LeavesGenerator::x1 [protected]

4.47.5.90 double LeavesGenerator::y1 [protected]

4.47.5.91 double LeavesGenerator::z1 [protected]

4.48 LeavesInfo Struct Reference

Public Attributes

- vector< Leaf > vLeavesInfo
- vector< Leaf >::iterator iLeavesInfoBegin
- vector< Leaf >::iterator iLeavesInfoEnd
- vector< Leaf > vLeavesInfo
- vector< Leaf >::iterator iLeavesInfoBegin
- vector< Leaf >::iterator iLeavesInfoEnd
- vector< Leaf > vLeavesInfo
- vector< Leaf >::iterator iLeavesInfoBegin
- vector< Leaf >::iterator iLeavesInfoEnd
- vector< Leaf > vLeavesInfo
- vector< Leaf >::iterator iLeavesInfoBegin
- vector< Leaf >::iterator iLeavesInfoEnd

4.48.1 Member Data Documentation

4.48.1.1 vector<Leaf>::iterator LeavesInfo::iLeavesInfoBegin

4.48.1.2 vector<Leaf>::iterator LeavesInfo::iLeavesInfoBegin

4.48.1.3 vector<Leaf>::iterator LeavesInfo::iLeavesInfoBegin

4.48.1.4 vector<Leaf>::iterator LeavesInfo::iLeavesInfoBegin

4.48.1.5 vector<Leaf>::iterator LeavesInfo::iLeavesInfoEnd

4.48.1.6 vector<Leaf>::iterator LeavesInfo::iLeavesInfoEnd

4.48.1.7 vector<Leaf>::iterator LeavesInfo::iLeavesInfoEnd

4.48.1.8 vector<Leaf>::iterator LeavesInfo::iLeavesInfoEnd

4.48.1.9 vector<Leaf> LeavesInfo::vLeavesInfo

4.48.1.10 vector<Leaf> LeavesInfo::vLeavesInfo

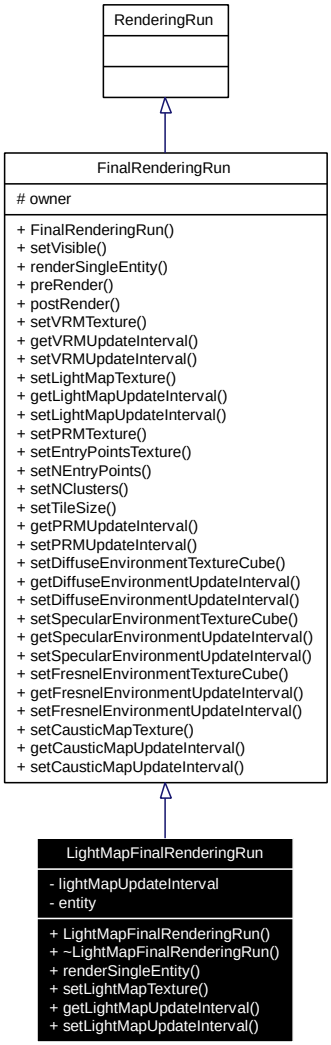
4.48.1.11 vector<Leaf> LeavesInfo::vLeavesInfo

4.48.1.12 vector<Leaf> LeavesInfo::vLeavesInfo

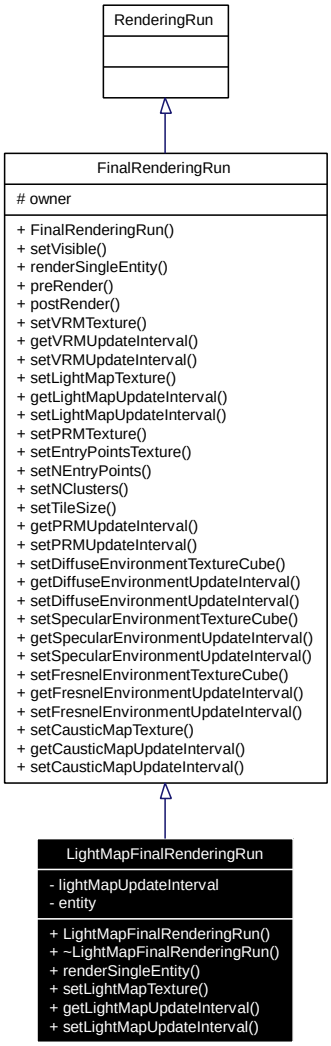
4.49 LightMapFinalRenderingRun Class Reference

Renders the final image from the light map.

Inheritance diagram for LightMapFinalRenderingRun:



Collaboration diagram for LightMapFinalRenderingRun:



Public Member Functions

- LightMapFinalRenderingRun (Entity *entity)

  Constructor.

- ~LightMapFinalRenderingRun ()

  Destructor.

- virtual void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_X)
- virtual void setLightMapTexture (const String &lightMapTextureName)

  Setter for light map texture.

- virtual unsigned int getLightMapUpdateInterval ()

  Getter for light map update interval.

- virtual void setLightMapUpdateInterval (unsigned int updateIntervalNumOfFrames)

Setter for light map update interval.

4.49.1 Detailed Description

Renders the final image from the light map.

4.49.2 Constructor & Destructor Documentation

4.49.2.1 LightMapFinalRenderingRun::LightMapFinalRenderingRun (Entity * *entity*)

Constructor.

Parameters:
entity The entity to render to.

4.49.2.2 LightMapFinalRenderingRun::~LightMapFinalRenderingRun ()

Destructor.

4.49.3 Member Function Documentation

4.49.3.1 virtual unsigned int LightMapFinalRenderingRun::getLightMapUpdateInterval () [inline, virtual]

Getter for light map update interval.

Returns:
The light map update interval.

Reimplemented from FinalRenderingRun.

4.49.3.2 virtual void LightMapFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [inline, virtual]

See also:
FinalRenderingPass::renderSingleEntity()

Implements FinalRenderingRun.

4.49.3.3 virtual void LightMapFinalRenderingRun::setLightMapTexture (const String & *lightMapTextureName*) [inline, virtual]

Setter for light map texture.

Parameters:
lightMapTextureName The name of the light map texture.

Reimplemented from FinalRenderingRun.

4.49.3.4 virtual void LightMapFinalRenderingRun::setLightMapUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

Setter for light map update interval.

Parameters:
updateIntervalNumOfFrames The light map update interval.

Reimplemented from FinalRenderingRun.

4.50 LightMapRenderingRun Class Reference

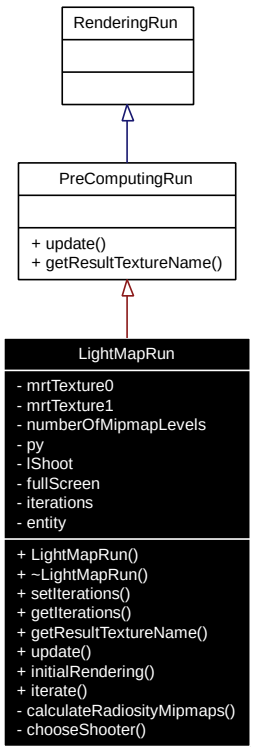
Computes the indirect diffuse illumination for an entity (the level geometry, typically).

4.50.1 Detailed Description

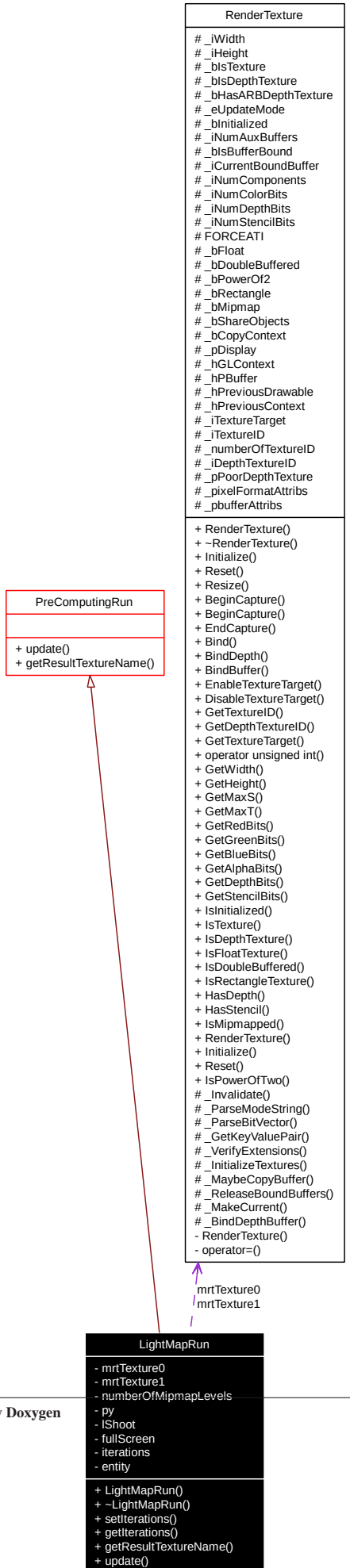
Computes the indirect diffuse illumination for an entity (the level geometry, typically).

4.51 LightMapRun Class Reference

Inheritance diagram for LightMapRun:



Collaboration diagram for LightMapRun:


RenderTexture
_iWidth
_iHeight
_bIsTexture
_bIsDepthTexture
_bHasARBDepthTexture
_eUpdateMode
_bInitialized
_iNumAuxBuffers
_bIsBufferBound
_iCurrentBoundBuffer
_iNumComponents
_iNumColorBits
_iNumDepthBits
_iNumStencilBits
FORCEATI
_bFloat
_bDoubleBuffered
_bPowerOf2
_bRectangle
_bMipmap
_bShareObjects
_bCopyContext
_pDisplay
_hGLContext
_hPBuffer
_hPreviousDrawable
_hPreviousContext
_iTextureTarget
_iTextureID
_numberOfTextureID
_iDepthTextureID
_pPoorDepthTexture
_pixelFormatAttribs
_pbufferAttribs
+ RenderTexture()
+ ~RenderTexture()
+ Initialize()
+ Reset()
+ Resize()
+ BeginCapture()
+ BeginCapture()
+ EndCapture()
+ Bind()
+ BindDepth()
+ BindBuffer()
+ EnableTextureTarget()
+ DisableTextureTarget()
+ GetTextureID()
+ GetDepthTextureID()
+ GetTextureTarget()
+ operator unsigned int()
+ GetWidth()
+ GetHeight()
+ GetMaxS()
+ GetMaxT()
+ GetRedBits()
+ GetGreenBits()
+ GetBlueBits()
+ GetAlphaBits()
+ GetDepthBits()
+ GetStencilBits()
+ IsInitialized()
+ IsTexture()
+ IsDepthTexture()
+ IsFloatTexture()
+ IsDoubleBuffered()
+ IsRectangleTexture()
+ HasDepth()
+ HasStencil()
+ IsMipmapped()
+ RenderTexture()
+ Initialize()
+ Reset()
+ IsPowerOfTwo()
# _Invalidate()
# _ParseModeString()
# _ParseBitVector()
# _GetKeyValuePair()
# _VerifyExtensions()
# _InitializeTextures()
# _MaybeCopyBuffer()
# _ReleaseBoundBuffers()
# _MakeCurrent()
# _BindDepthBuffer()
- RenderTexture()
- operator=()
PreComputingRun
+ update()
+ getResultTextureName()
mrtTexture0
mrtTexture1
LightMapRun
- mrtTexture0
- mrtTexture1
- numberOfMipmapLevels
- py
- lShoot
- fullScreen
- iterations
- entity
+ LightMapRun()
+ ~LightMapRun()
+ setIterations()
+ getIterations()
+ getResultTextureName()
+ update()

Public Member Functions

- LightMapRun (unsigned int width, unsigned int height, unsigned char numberOfMipmapLevels, unsigned int lod, Entity *entity)
- ~LightMapRun ()
- void setIterations (unsigned int iterations)

  Sets the number of iterations.

- unsigned int getIterations (void)

  Sets the number of iterations.

- const String & getResultTextureName ()
- void update ()
- void initialRendering ()
- void iterate ()

4.51.1 Constructor & Destructor Documentation

4.51.1.1 LightMapRun::LightMapRun (unsigned int *width*, unsigned int *height*, unsigned char *numberOfMipmapLevels*, unsigned int *lod*, Entity * *entity*)

Constructor.

Parameters:

width The width of render texture objects.

height The height of render texture objects.

numberOfMipmapLevels The number of mip map levels for radiosity texture objects.

lod The level of detail factor.

entity The entity to render.

4.51.1.2 LightMapRun::~LightMapRun ()

Destructor.

4.51.2 Member Function Documentation

4.51.2.1 unsigned int LightMapRun::getIterations (void) [inline]

Sets the number of iterations.

Returns:

The number of iterations to do.

4.51.2.2 const String& LightMapRun::getResultTextureName () [virtual]

See also:
PreComputingRun::getResultTextureName();

Reimplemented from PreComputingRun.

4.51.2.3 void LightMapRun::initialRendering ()

Renders initial visibility and emission into their respective textures.

4.51.2.4 void LightMapRun::iterate ()

Iteration to calculate lightmap. Generates mipmaps, chooses a shooter then calculates visibility and radiosity for all 5 shooter hemicube sides. Finally it averages the current radiosity with the one calculated from the shooter.

4.51.2.5 void LightMapRun::setIterations (unsigned int *iterations*) [inline]

Sets the number of iterations.

Parameters:
iterations The number of iterations to do.

4.51.2.6 void LightMapRun::update () [virtual]

See also:
PreComputingRun::update();

Implements PreComputingRun.

4.52 Listid Class Reference

Public Member Functions

- Listid ()
- std::vector< unsigned int > * GetList ()
- int GetSize () const
- unsigned int Get (unsigned int i)
- bool Insert (unsigned int id)
- void Print ()

4.52.1 Detailed Description

A list of id (integers) for patch and polygons indices

4.52.2 Constructor & Destructor Documentation

4.52.2.1 Listid::Listid () [inline]

4.52.3 Member Function Documentation

4.52.3.1 unsigned int Listid::Get (unsigned int *i*)

Returns pointer to patch(i)

4.52.3.2 std::vector<unsigned int>* Listid::GetList () [inline]

Returns the pointer to the list of voxels.

4.52.3.3 int Listid::GetSize () const [inline]

Returns the number of elements of list

4.52.3.4 bool Listid::Insert (unsigned int *id*)

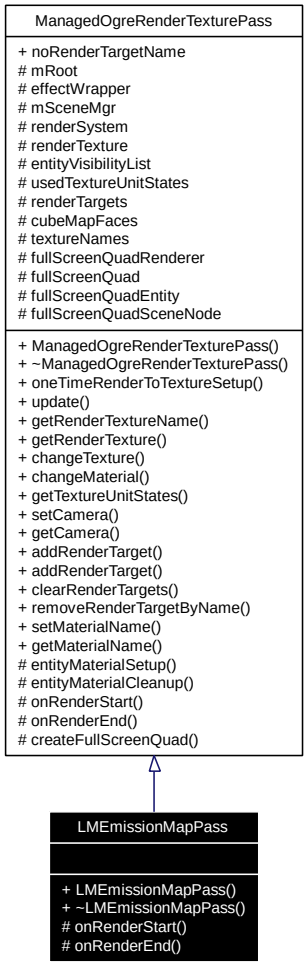
4.52.3.5 void Listid::Print ()

print stats

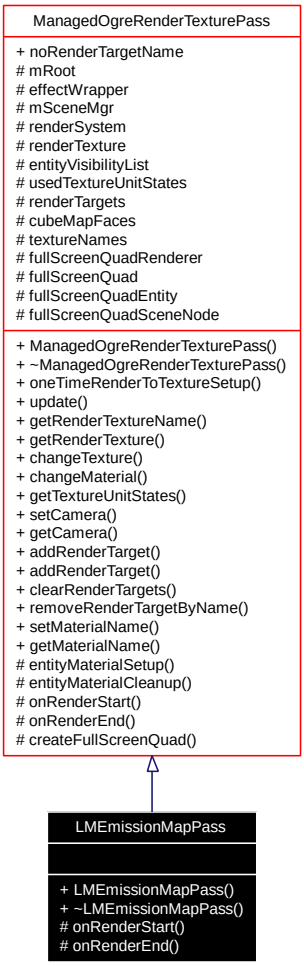
4.53 LMEmissionMapPass Class Reference

Renders emission map to a texture.

Inheritance diagram for LMEmissionMapPass:



Collaboration diagram for LMEmissionMapPass:



Public Member Functions

- LMEmissionMapPass (Root *mRoot, unsigned int width, unsigned int height,)

  Constructor.

- ~LMEmissionMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.53.1 Detailed Description

Renders emission map to a texture.

4.53.2 Constructor & Destructor Documentation

4.53.2.1 LMEmissionMapPass::LMEmissionMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor.

Parameters:

mRoot The current ogre root object.

width The width of the render texture instance

height The height of the render texture instance

4.53.2.2 LMEmissionMapPass::~LMEmissionMapPass ()

Destructor.

4.53.3 Member Function Documentation

4.53.3.1 void LMEmissionMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.53.3.2 void LMEmissionMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

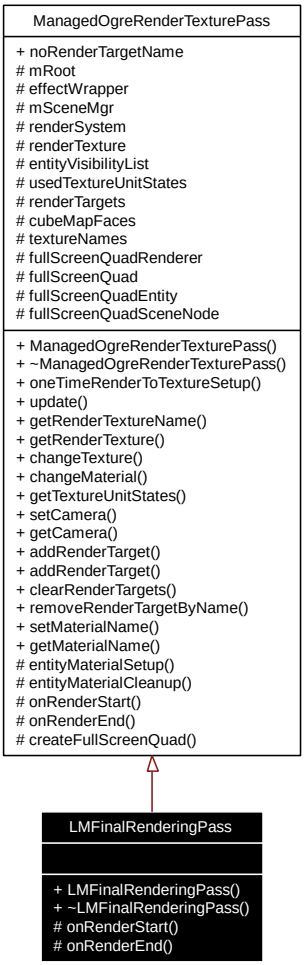
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

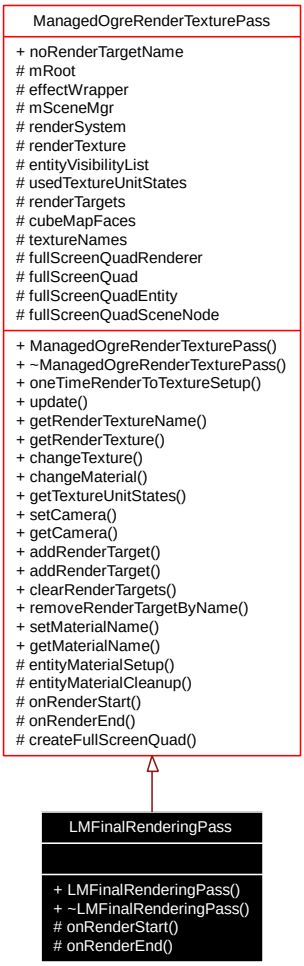
4.54 LMFinalRenderingPass Class Reference

Final rendering pass for light map rendering.

Inheritance diagram for LMFinalRenderingPass:



Collaboration diagram for LMFinalRenderingPass:



Public Member Functions

- LMFinalRenderingPass (Root *mRoot)

  Constructor.

- ~LMFinalRenderingPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.54.1 Detailed Description

Final rendering pass for light map rendering.

4.54.2 Constructor & Destructor Documentation

4.54.2.1 LMFinalRenderingPass::LMFinalRenderingPass (Root * *mRoot*)

Constructor.

Parameters:
mRoot The current ogre root object.

4.54.2.2 LMFinalRenderingPass::~LMFinalRenderingPass ()

Destructor.

4.54.3 Member Function Documentation

4.54.3.1 void LMFinalRenderingPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.54.3.2 void LMFinalRenderingPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

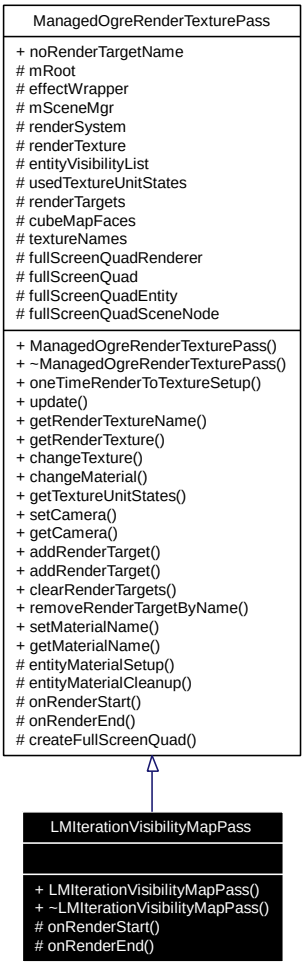
See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

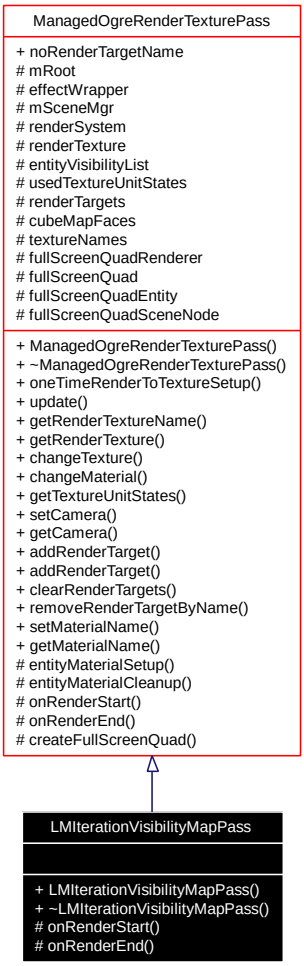
4.55 LMIterationVisibilityMapPass Class Reference

Renders visibility informations to a RGBF16 texture.

Inheritance diagram for LMIterationVisibilityMapPass:



Collaboration diagram for LMIterationVisibilityMapPass:



Public Member Functions

- LMIterationVisibilityMapPass (Root *mRoot, unsigned int width, unsigned int height)

  Constructor.

- ~LMIterationVisibilityMapPass ()

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.55.1 Detailed Description

Renders visibility informations to a RGBF16 texture.

4.55.2 Constructor & Destructor Documentation

4.55.2.1 LMIterationVisibilityMapPass::LMIterationVisibilityMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor.

Parameters:

mRoot The current ogre root object.

width The width of the render texture instance

height The height of the render texture instance

4.55.2.2 LMIterationVisibilityMapPass::~LMIterationVisibilityMapPass ()

Destructor.

4.55.3 Member Function Documentation

4.55.3.1 void LMIterationVisibilityMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.55.3.2 void LMIterationVisibilityMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

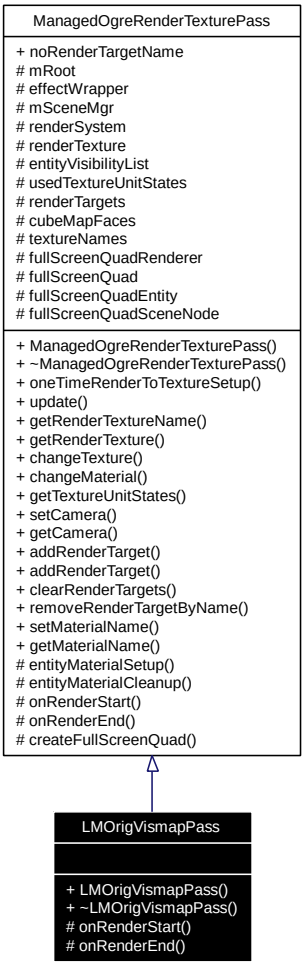
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

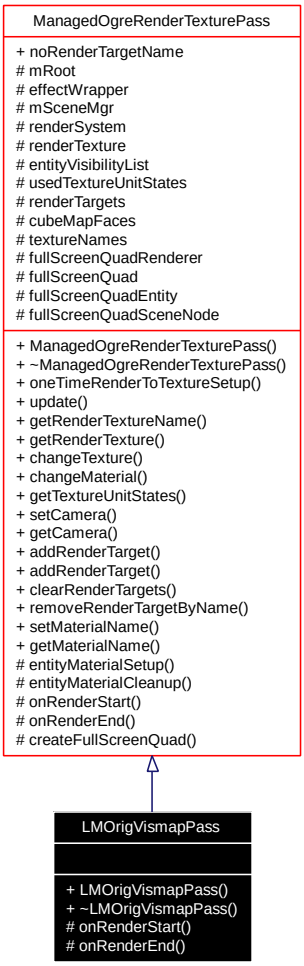
4.56 LMOrigVismapPass Class Reference

Renders patch index to a RGB16 texture.

Inheritance diagram for LMOrigVismapPass:



Collaboration diagram for LMOrigVismapPass:



Public Member Functions

- LMOrigVismapPass (Root *mRoot, unsigned int width, unsigned int height)

  Constructor.

- ~LMOrigVismapPass ()

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.56.1 Detailed Description

Renders patch index to a RGB16 texture.

4.56.2 Constructor & Destructor Documentation

4.56.2.1 LMOrigVismapPass::LMOrigVismapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor.

Parameters:

mRoot The current ogre root object.

width The width of the render texture instance

height The height of the render texture instance

4.56.2.2 LMOrigVismapPass::~LMOrigVismapPass ()

Destructor.

4.56.3 Member Function Documentation

4.56.3.1 void LMOrigVismapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.56.3.2 void LMOrigVismapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

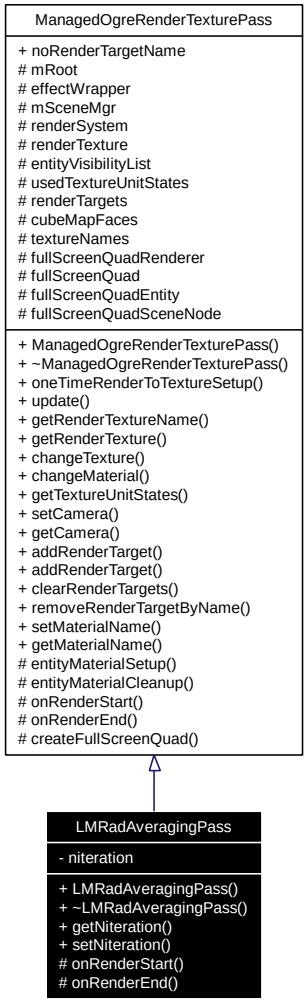
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

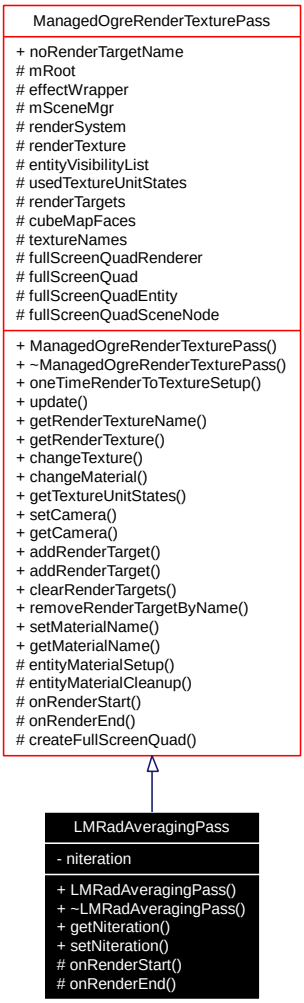
4.57 LMRadAveragingPass Class Reference

Averages the actual radiosity information texture with the results of the previous passes.

Inheritance diagram for LMRadAveragingPass:



Collaboration diagram for LMRadAveragingPass:



Public Member Functions

- LMRadAveragingPass (Root *mRoot)

  Constructor.

- ~LMRadAveragingPass ()
- float getNiteration ()

  Returns the current iteration count.

- void setNiteration (float niteration)

  Setter for current iteration count.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.57.1 Detailed Description

Averages the actual radiosity information texture with the results of the previous passes.

4.57.2 Constructor & Destructor Documentation

4.57.2.1 LMRadAveragingPass::LMRadAveragingPass (Root * *mRoot*)

Constructor.

Parameters:
mRoot The current ogre root object.

4.57.2.2 LMRadAveragingPass::~LMRadAveragingPass ()

Destructor.

4.57.3 Member Function Documentation

4.57.3.1 float LMRadAveragingPass::getNiteration () [inline]

Returns the current iteration count.

Returns:
iteration count.

4.57.3.2 void LMRadAveragingPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.57.3.3 void LMRadAveragingPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

4.57.3.4 void LMRadAveragingPass::setNiteration (float *niteration*) [inline]

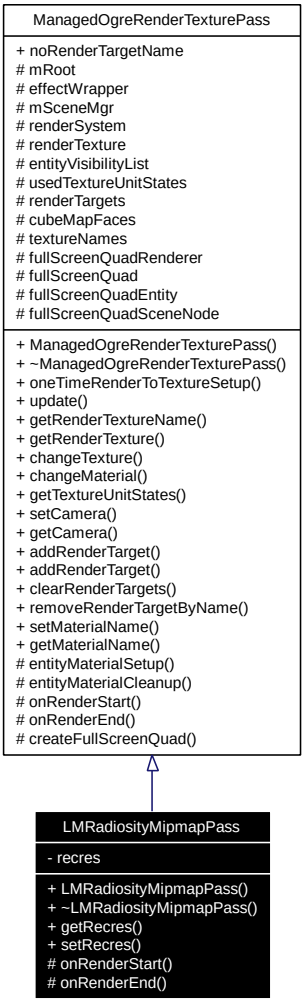
Setter for current iteration count.

Parameters:
niteration Iteration count.

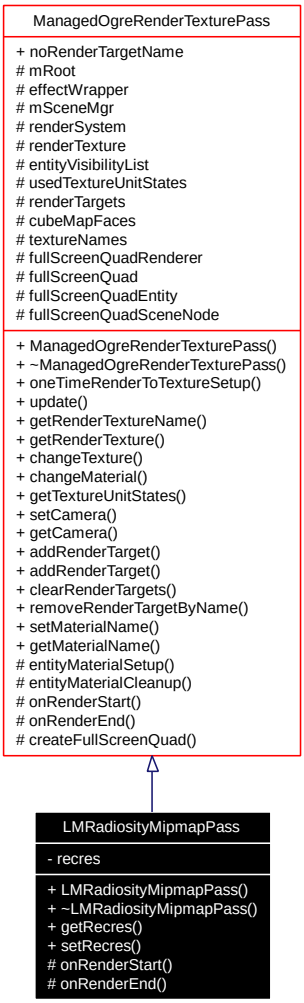
4.58 LMRadiosityMipmapPass Class Reference

Renders radiosity mipmap to a floating point 64 bit RGBA texture. Renders to a full screen quad.

Inheritance diagram for LMRadiosityMipmapPass:



Collaboration diagram for LMRadiosityMipmapPass:



Public Member Functions

- LMRadiosityMipmapPass (Root *mRoot, unsigned int width, unsigned int height, String renderTextureName)

  Constructor.

- ~LMRadiosityMipmapPass ()
- float getRecres ()

  Gets the reciproc of the resolution.

- void setRecres (float recres)

  Sets the reciproc of the resolution.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.58.1 Detailed Description

Renders radiosity mipmap to a floating point 64 bit RGBA texture. Renders to a full screen quad.

4.58.2 Constructor & Destructor Documentation

4.58.2.1 LMRadiosityMipmapPass::LMRadiosityMipmapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*, String *renderTextureName*)

Constructor.

Parameters:

mRoot The current ogre root object.

renderTextureName The name of the render texture instance to create. Set to ManagedOgreRenderTexturePass::

width The width of the render texture instance

height The height of the render texture instance

4.58.2.2 LMRadiosityMipmapPass::~LMRadiosityMipmapPass ()

Destructor.

4.58.3 Member Function Documentation

4.58.3.1 float LMRadiosityMipmapPass::getRecres () [inline]

Gets the reciproc of the resolution.

Returns:

The reciproc of the resolution.

4.58.3.2 void LMRadiosityMipmapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:

ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.58.3.3 void LMRadiosityMipmapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

4.58.3.4 void LMRadiosityMipmapPass::setRecres (float *recres*) [inline]

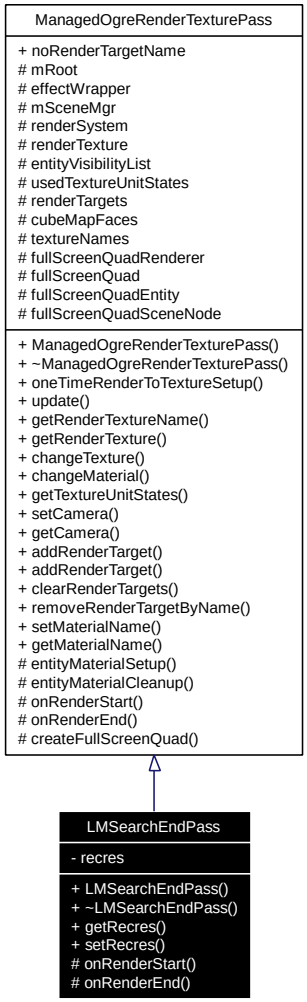
Sets the reciproc of the resolution.

Parameters:
recres The reciproc of the resolution.

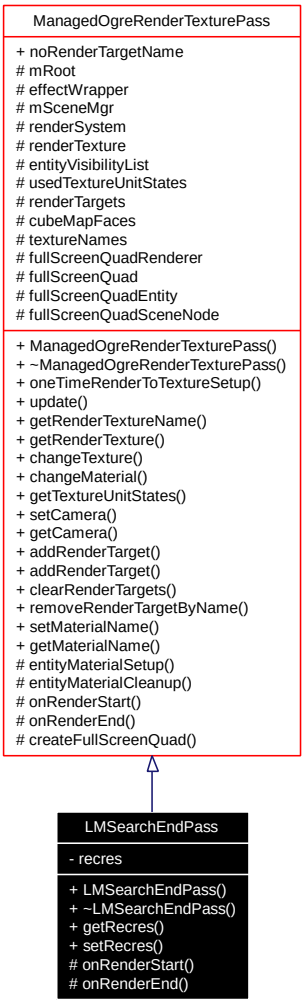
4.59 LMSearchEndPass Class Reference

Searches to the bottom of radiosity mipmaps.

Inheritance diagram for LMSearchEndPass:



Collaboration diagram for LMSearchEndPass:



Public Member Functions

- LMSearchEndPass (Root *mRoot)

  Constructor.

- ~LMSearchEndPass ()
- float getRecres ()

  Gets the reciproc of the resolution.

- void setRecres (float recres)

  Sets the reciproc of the resolution.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.59.1 Detailed Description

Searches to the bottom of radiosity mipmaps.

4.59.2 Constructor & Destructor Documentation

4.59.2.1 LMSearchEndPass::LMSearchEndPass (Root * *mRoot*)

Constructor.

Parameters:
mRoot The current ogre root object.

4.59.2.2 LMSearchEndPass::~LMSearchEndPass ()

Destructor.

4.59.3 Member Function Documentation

4.59.3.1 float LMSearchEndPass::getRecres () [inline]

Gets the reciproc of the resolution.

Returns:
The reciproc of the resolution.

4.59.3.2 void LMSearchEndPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.59.3.3 void LMSearchEndPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

4.59.3.4 void LMSearchEndPass::setRecres (float *recres*) [inline]

Sets the reciproc of the resolution.

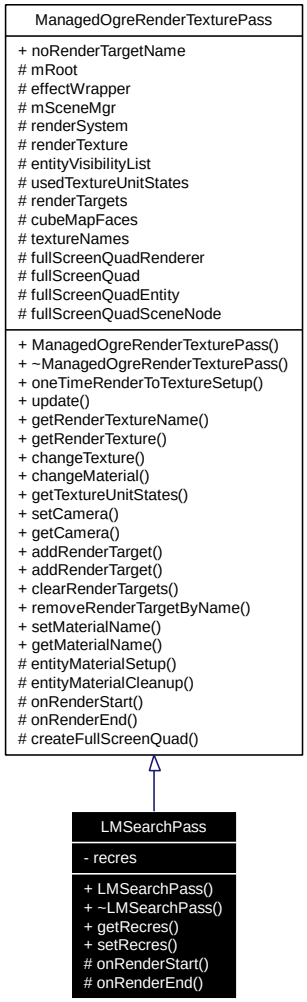
Parameters:

recres The reciproc of the resolution.

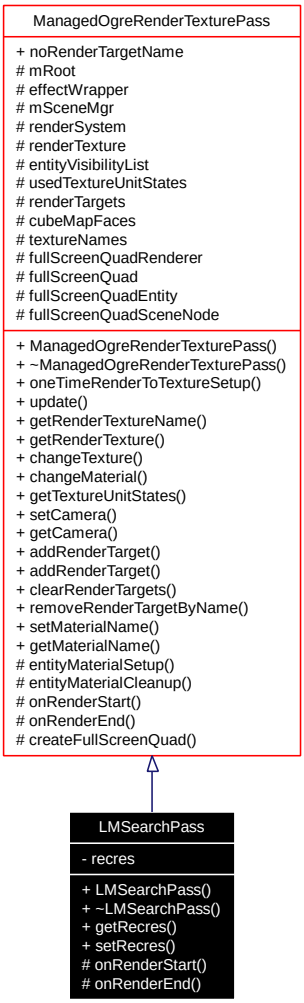
4.60 LMSearchPass Class Reference

Searches to the bottom of radiosity mipmaps.

Inheritance diagram for LMSearchPass:



Collaboration diagram for LMSearchPass:



Public Member Functions

- LMSearchPass (Root *mRoot)

  Constructor.

- ~LMSearchPass ()
- float getRecres ()

  Gets the reciproc of the resolution.

- void setRecres (float recres)

  Sets the reciproc of the resolution.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.60.1 Detailed Description

Searches to the bottom of radiosity mipmaps.

4.60.2 Constructor & Destructor Documentation

4.60.2.1 LMSearchPass::LMSearchPass (Root * *mRoot*)

Constructor.

Parameters:
mRoot The current ogre root object.

4.60.2.2 LMSearchPass::~LMSearchPass ()

Destructor.

4.60.3 Member Function Documentation

4.60.3.1 float LMSearchPass::getRecres () [inline]

Gets the reciproc of the resolution.

Returns:
The reciproc of the resolution.

4.60.3.2 void LMSearchPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.60.3.3 void LMSearchPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

4.60.3.4 void LMSearchPass::setRecres (float *recres*) [inline]

Sets the reciproc of the resolution.

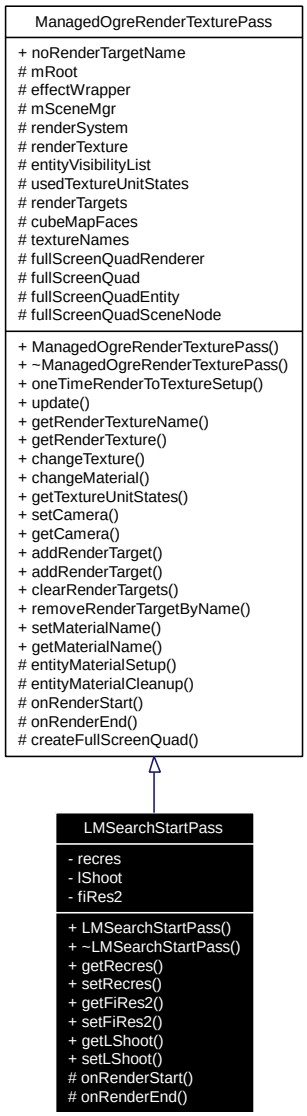
Parameters:

recres The reciproc of the resolution.

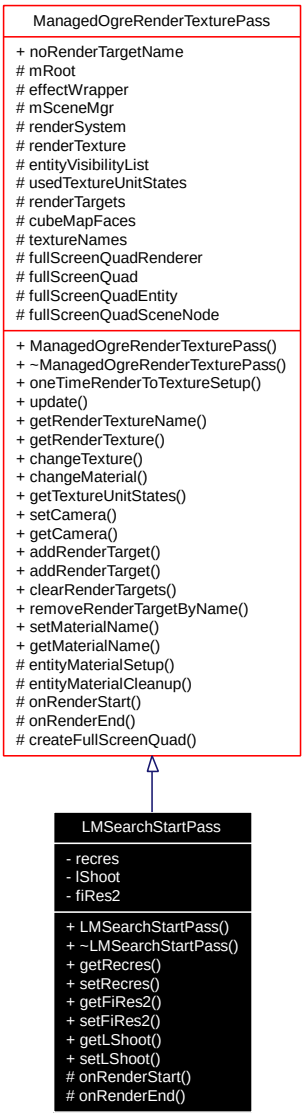
4.61 LMSearchStartPass Class Reference

Searches the top level of the radiosity mipmaps.

Inheritance diagram for LMSearchStartPass:



Collaboration diagram for LMSearchStartPass:



Public Member Functions

- LMSearchStartPass (Root *mRoot)

  Constructor.

- ~LMSearchStartPass ()
- float getRecres ()

  Gets the reciproc of the resolution.

- void setRecres (float recres)

  Sets the reciproc of the resolution.

- float getFiRes2 ()
- void setFiRes2 (float fiRes2)
- Vector4 & getLShoot ()
- void setLShoot (Vector4 &lShoot)

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.61.1 Detailed Description

Searches the top level of the radiosity mipmaps.

4.61.2 Constructor & Destructor Documentation

4.61.2.1 LMSearchStartPass::LMSearchStartPass (Root * *mRoot*)

Constructor.

Parameters:
mRoot The current ogre root object.

4.61.2.2 LMSearchStartPass::~LMSearchStartPass ()

Destructor.

4.61.3 Member Function Documentation

4.61.3.1 float LMSearchStartPass::getFiRes2 () [inline]

4.61.3.2 Vector4& LMSearchStartPass::getLShoot () [inline]

4.61.3.3 float LMSearchStartPass::getRecres () [inline]

Gets the reciproc of the resolution.

Returns:
The reciproc of the resolution.

4.61.3.4 void LMSearchStartPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.61.3.5 void LMSearchStartPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

4.61.3.6 void LMSearchStartPass::setFiRes2 (float *fiRes2*) [inline]

4.61.3.7 void LMSearchStartPass::setLShoot (Vector4 & *lShoot*) [inline]

4.61.3.8 void LMSearchStartPass::setRecres (float *recres*) [inline]

Sets the reciproc of the resolution.

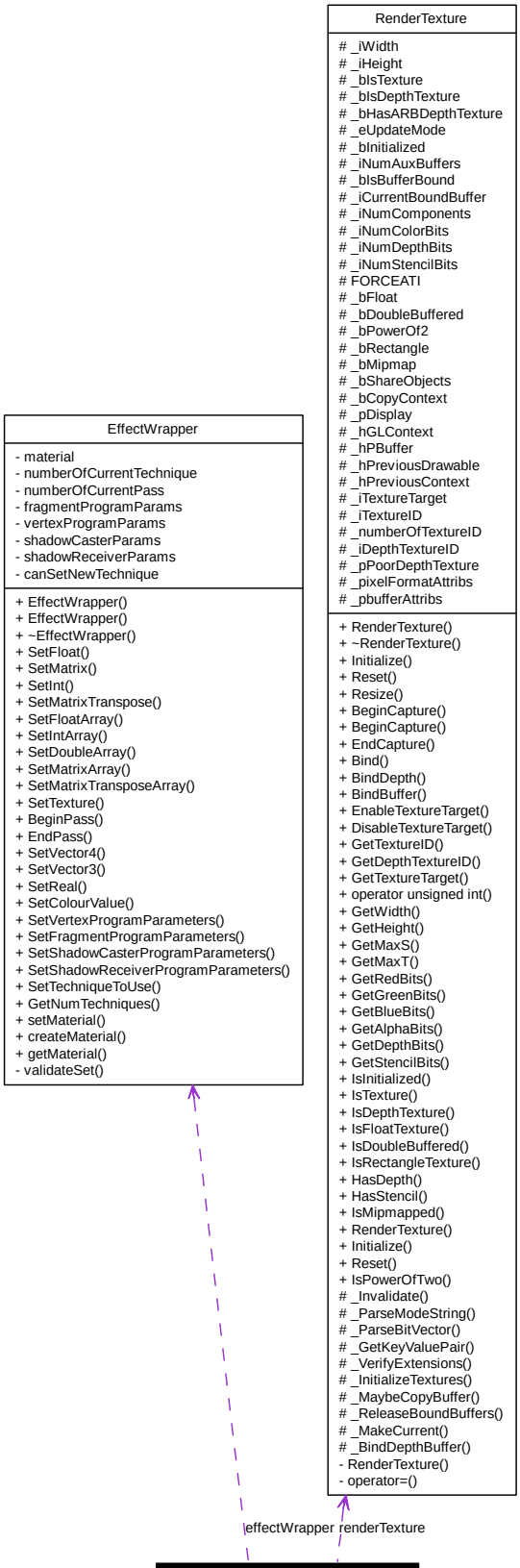
Parameters:
recres The reciproc of the resolution.

4.62 ManagedOgreRenderTexturePass Class Reference

ManagedOgreRenderTexturePass encapsulates a hardware accelerated GPU driven render-to-texture pass.

Inheritance diagram for ManagedOgreRenderTexturePass:

Collaboration diagram for ManagedOgreRenderTexturePass:


RenderTexture
_iWidth
_iHeight
_bIsTexture
_bIsDepthTexture
_bHasARBDepthTexture
_eUpdateMode
_bInitialized
_iNumAuxBuffers
_bIsBufferBound
_iCurrentBoundBuffer
_iNumComponents
_iNumColorBits
_iNumDepthBits
_iNumStencilBits
FORCEATI
_bFloat
_bDoubleBuffered
_bPowerOf2
_bRectangle
_bMipmap
_bShareObjects
_bCopyContext
_pDisplay
_hGLContext
_hPBuffer
_hPreviousDrawable
_hPreviousContext
_iTextureTarget
_iTextureID
_numberOfTextureID
_iDepthTextureID
_pPoorDepthTexture
_pixelFormatAttribs
_pbufferAttribs
+ RenderTexture()
+ ~RenderTexture()
+ Initialize()
+ Reset()
+ Resize()
+ BeginCapture()
+ BeginCapture()
+ EndCapture()
+ Bind()
+ BindDepth()
+ BindBuffer()
+ EnableTextureTarget()
+ DisableTextureTarget()
+ GetTextureID()
+ GetDepthTextureID()
+ GetTextureTarget()
+ operator unsigned int()
+ GetWidth()
+ GetHeight()
+ GetMaxS()
+ GetMaxT()
+ GetRedBits()
+ GetGreenBits()
+ GetBlueBits()
+ GetAlphaBits()
+ GetDepthBits()
+ GetStencilBits()
+ IsInitialized()
+ IsTexture()
+ IsDepthTexture()
+ IsFloatTexture()
+ IsDoubleBuffered()
+ IsRectangleTexture()
+ HasDepth()
+ HasStencil()
+ IsMipmapped()
+ RenderTexture()
+ Initialize()
+ Reset()
+ IsPowerOfTwo()
# _Invalidate()
# _ParseModeString()
# _ParseBitVector()
# _GetKeyValuePair()
# _VerifyExtensions()
# _InitializeTextures()
# _MaybeCopyBuffer()
# _ReleaseBoundBuffers()
# _MakeCurrent()
# _BindDepthBuffer()
- RenderTexture()
- operator=()
EffectWrapper
- material
- numberOfCurrentTechnique
- numberOfCurrentPass
- fragmentProgramParams
- vertexProgramParams
- shadowCasterParams
- shadowReceiverParams
- canSetNewTechnique
+ EffectWrapper()
+ EffectWrapper()
+ ~EffectWrapper()
+ SetFloat()
+ SetMatrix()
+ SetInt()
+ SetMatrixTranspose()
+ SetFloatArray()
+ SetIntArray()
+ SetDoubleArray()
+ SetMatrixArray()
+ SetMatrixTransposeArray()
+ SetTexture()
+ BeginPass()
+ EndPass()
+ SetVector4()
+ SetVector3()
+ SetReal()
+ SetColourValue()
+ SetVertexProgramParameters()
+ SetFragmentProgramParameters()
+ SetShadowCasterProgramParameters()
+ SetShadowReceiverProgramParameters()
+ SetTechniqueToUse()
+ GetNumTechniques()
+ setMaterial()
+ createMaterial()
+ getMaterial()
- validateSet()
effectWrapper renderTexture
ManagedOgreRenderTexturePass
+ noRenderTargetName
# mRoot
# effectWrapper
# mSceneMgr
# renderSystem
# renderTexture
# entityVisibilityList
# usedTextureUnitStates
# renderTargets
# cubeMapFaces
# textureNames
# fullScreenQuadRenderer
# fullScreenQuad
# fullScreenQuadEntity
# fullScreenQuadSceneNode
+ ManagedOgreRenderTexturePass()
+ ~ManagedOgreRenderTexturePass()
+ oneTimeRenderToTextureSetup()

Public Member Functions

- ManagedOgreRenderTexturePass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- virtual ~ManagedOgreRenderTexturePass ()
- virtual void oneTimeRenderToTextureSetup (String &materialName, StringVector &textureNames)

  Creates render to texture objects.

- virtual void update (NameValuePairList *startParams=0, NameValuePairList *endParams=0)

  Updates the render-texture.

- const String & getRenderTextureName ()
- RenderTexture * getRenderTexture ()
- bool changeTexture (const String &textureName, unsigned short textureUnit)
- bool changeMaterial (String &materialName, StringVector &newTextureNames, bool useOldTextureNames=false)
- std::vector< TextureUnitState * > * getTextureUnitStates ()
- void setCamera (Camera *camera)
- Camera * getCamera ()
- void addRenderTarget (RenderTarget *renderTarget)

  Adds a render target to the render target queue. For multiple render targets.

- void addRenderTarget (RenderTarget *renderTarget, CubeMapFaces cubeMapFace)

  Adds a render target to the render target queue. For multiple render targets. For cube map texture render targets.

- void clearRenderTargets ()

  Clears the render target list.

- void removeRenderTargetByName (const String &renderTargetName)

  Removes a render target from the list of render targets.

- virtual void setMaterialName (const String &materialName)

  Changes the material of the pass.

- virtual const String & getMaterialName ()

  Retrieves the name of the material used in the pass.

Static Public Attributes

- static const char * noRenderTargetName

Protected Member Functions

- virtual void entityMaterialSetup ()

  Controls the setup of entities. It is called from update after onRenderStart.

- virtual void entityMaterialCleanup ()

  Controls the cleanup of entities. It is called from update before onRenderEnd.

- virtual void onRenderStart (NameValuePairList *namedParams=0)

  Runs before the render-texture object is updated place all shader setup here.

- virtual void onRenderEnd (NameValuePairList *namedParams=0)

  Runs after the render-texture object is updated place all cleanup code here.

- virtual void createFullScreenQuad ()

  Creates a full screen quad for FSQ rendering.

Protected Attributes

- Root * mRoot

  Pointer to the current ogre root.

- EffectWrapper * effectWrapper

  EffectWrapper for convinient shader setup.

- SceneManager * mSceneMgr

  Pointer to the current Ogre SceneManager.

- RenderSystem * renderSystem

  Pointer to the current Ogre RenderSystem.

- RenderTexture * renderTexture

  The pointer to the pass's own render texture object.

- std::vector< bool > entityVisibilityList

  A list to hold all visibility information for entities.

- std::vector< TextureUnitState * > usedTextureUnitStates

  List of texture units.

- std::vector< RenderTarget * > renderTargets

  A list of render targets to use in the pass. For MRT-s.

- std::vector< CubeMapFaces > cubeMapFaces

  A list of cube map faces to use for Cube map RT-s.

- StringVector textureNames

  List of texture names used in the pass.

- bool fullScreenQuadRenderer
  Shows if we are rendering a full screen quad.

Static Protected Attributes

- static MovablePlane * fullScreenQuad
  MovablePlane instance for full screen quad.
- static Entity * fullScreenQuadEntity
  Entity to hold the full screen quad.
- static SceneNode * fullScreenQuadSceneNode
  SceneNode instance for full screen quad.

4.62.1 Detailed Description

ManagedOgreRenderTexturePass encapsulates a hardware accelerated GPU driven render-to-texture pass.

4.62.2 Constructor & Destructor Documentation

4.62.2.1 ManagedOgreRenderTexturePass::ManagedOgreRenderTexturePass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Parameters:

mRoot The current ogre root object.

renderTextureName The name of the render texture instance to create. Set to ManagedOgreRenderTexturePass::

width The width of the render texture instance

height The height of the render texture instance

texType The texture type of the render texture instance

internalFormat The pixel format of the render texture instance

miscParams All non standard parameters of the render texture instance

fullScreenQuadRenderer Defines full screen quad rendering.

4.62.2.2 virtual ManagedOgreRenderTexturePass::~ManagedOgreRenderTexturePass () `[inline, virtual]`

Destructor.

4.62.3 Member Function Documentation

4.62.3.1 void ManagedOgreRenderTexturePass::addRenderTarget (RenderTarget * *renderTarget*, CubeMapFaces *cubeMapFace*) [inline]

Adds a render target to the render target queue. For multiple render targets. For cube map texture render targets.

Parameters:

renderTarget Pointer to the render target to add.

cubeMapFace The cube map face for cube map render targets.

4.62.3.2 void ManagedOgreRenderTexturePass::addRenderTarget (RenderTarget * *renderTarget*) [inline]

Adds a render target to the render target queue. For multiple render targets.

Parameters:

renderTarget Pointer to the render target to add.

4.62.3.3 bool ManagedOgreRenderTexturePass::changeMaterial (String & *materialName*, StringVector & *newTextureNames*, bool *useOldTextureNames* = false)

Parameters:

materialName Name of the specific material to create.

newTextureNames A list of texture names to set.

useOldTextureNames Shows whether to use old texture names or not.

Returns:

true if texture could be set for the specific texture unit.

4.62.3.4 bool ManagedOgreRenderTexturePass::changeTexture (const String & *textureName*, unsigned short *textureUnit*)

Parameters:

textureName Name of the specific texture to set.

textureUnit Number of the texture unit to set.

Returns:

true if texture could be set for the specific texture unit.

4.62.3.5 void ManagedOgreRenderTexturePass::clearRenderTargets () [inline]

Clears the render target list.

4.62.3.6 virtual void ManagedOgreRenderTexturePass::createFullScreenQuad () [protected, virtual]

Creates a full screen quad for FSQ rendering.

4.62.3.7 virtual void ManagedOgreRenderTexturePass::entityMaterialCleanup () [protected, virtual]

Controls the cleanup of entities. It is called from update before onRenderEnd.

Reimplemented in DEMEnvironmentMapPass, FEMEnvironmentMapPass, and SEMEnvironmentMapPass.

4.62.3.8 virtual void ManagedOgreRenderTexturePass::entityMaterialSetup () [protected, virtual]

Controls the setup of entities. It is called from update after onRenderStart.

Reimplemented in DEMEnvironmentMapPass, FEMEnvironmentMapPass, and SEMEnvironmentMapPass.

4.62.3.9 Camera* ManagedOgreRenderTexturePass::getCamera () [inline]

Returns:
Current camera.

4.62.3.10 virtual const String& ManagedOgreRenderTexturePass::getMaterialName () [inline, virtual]

Retrieves the name of the material used in the pass.

Returns:
The name of the material in use.

4.62.3.11 RenderTexture* ManagedOgreRenderTexturePass::getRenderTexture () [inline]

Returns:
Returns render-to-texture object pointer.

4.62.3.12 const String& ManagedOgreRenderTexturePass::getRenderTextureName ()

`[inline]`

Returns:
Returns render-to-texture object name.

4.62.3.13 std::vector<TextureUnitState*>* ManagedOgreRenderTexturePass::getTextureUnitStates ()

`[inline]`

Returns:
A list of texture unit states.

4.62.3.14 virtual void ManagedOgreRenderTexturePass::oneTimeRenderToTextureSetup (String & *materialName*, StringVector & *textureNames*)

`[virtual]`

Creates render to texture objects.

Parameters:
materialName Name of material to create
textureNames The names of the textures to use.

4.62.3.15 virtual void ManagedOgreRenderTexturePass::onRenderEnd (NameValuePairList * *namedParams* = 0)

`[inline, protected, virtual]`

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented in CAURenderColorDistanceCubeMapPass, CAURenderPhotonHitMapPass, CAURenderPhotonUVMapPass, CAURenderRefractObjectMapPass, CAURenderUmbraPass, CAURenderUVCubeMapPass, DEMEnvironmentMapPass, FEMEnvironmentMapPass, HPSCompositePass, HPSLightIllumPass, HPSPhaseFunctionPass, HPSSceneDepthPass, LMEmissionMapPass, LMIterationVisibilityMapPass, LMOrigVismapPass, LMRadAveragingPass, RadiosityMapPass, LMRadiosityMipmapPass, RenderGeometryPass, LMSearchEndPass, LMSearchPass, LMSearchStartPass, PMFFilteringPass, PMFNormalMapPass, SEMEnvironmentMapPass, VRMFilteringMapPass, VRMRenderDepthBufferMapPass, VRMRenderGeometryMapPass, CAURenderFinalPass, DEMFinalGatheringPass, FEMFinalGatheringPass, HPSFinalPass, IBLBlendAddIllumPass, IBLDepthMapPass, IBLRenderBlackPass, LMFinalRenderingPass, SEMFinalGatheringPass, and VRMRenderFinalPass.

4.62.3.16 virtual void ManagedOgreRenderTexturePass::onRenderStart (NameValuePairList * *namedParams* = 0)

`[inline, protected, virtual]`

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented in CAURenderColorDistanceCubeMapPass, CAURenderPhotonHitMapPass, CAURenderPhotonUVMapPass, CAURenderRefractObjectMapPass, CAURenderUmbraPass, CAURenderUVCubeMapPass, DEMEnvironmentMapPass, FEMEnvironmentMapPass, HPSCompositePass, HPSLightIllumPass, HPSPhaseFunctionPass, HPSSceneDepthPass, LMEmissionMapPass, LMIterationVisibilityMapPass, LMOrigVismapPass, LMRadAveragingPass, RadiosityMapPass, LMRadiosityMipmapPass, RenderGeometryPass, LMSearchEndPass, LMSearchPass, LMSearchStartPass, PMFFilteringPass, PMFNormalMapPass, SEMEnvironmentMapPass, VRMFilteringMapPass, VRMRenderDepthBufferMapPass, VRMRenderGeometryMapPass, CAURenderFinalPass, DEMFinalGatheringPass, FEMFinalGatheringPass, HPSFinalPass, IBLBlendAddIllumPass, IBLDepthMapPass, IBLRenderBlackPass, LMFinalRenderingPass, SEMFinalGatheringPass, and VRMRenderFinalPass.

4.62.3.17 void ManagedOgreRenderTexturePass::removeRenderTargetByName (const String & *renderTargetName*) [inline]

Removes a render target from the list of render targets.

Parameters:
renderTargetName The name of the render target to remove.

4.62.3.18 void ManagedOgreRenderTexturePass::setCamera (Camera * *camera*) [inline]

Parameters:
camera Camera camera to set.

4.62.3.19 virtual void ManagedOgreRenderTexturePass::setMaterialName (const String & *materialName*) [inline, virtual]

Changes the material of the pass.

Parameters:
materialName The name of the material to use.

4.62.3.20 virtual void ManagedOgreRenderTexturePass::update (NameValuePairList * *startParams* = 0, NameValuePairList * *endParams* = 0) [virtual]

Updates the render-texture.

Parameters:
startParams Starting parameters for the rendering pass.
endParams Ending parameters for the rendering pass.

4.62.4 Member Data Documentation

4.62.4.1 std::vector<CubeMapFaces> ManagedOgreRenderTexturePass::cubeMapFaces [protected]

A list of cube map faces to use for Cube map RT-s.

4.62.4.2 EffectWrapper* ManagedOgreRenderTexturePass::effectWrapper [protected]

EffectWrapper for convinient shader setup.

4.62.4.3 std::vector<bool> ManagedOgreRenderTexturePass::entityVisibilityList [protected]

A list to hold all visibility information for entities.

4.62.4.4 MovablePlane* ManagedOgreRenderTexturePass::fullScreenQuad [static, protected]

MovablePlane instance for full screen quad.

4.62.4.5 Entity* ManagedOgreRenderTexturePass::fullScreenQuadEntity [static, protected]

Entity to hold the full screen quad.

4.62.4.6 bool ManagedOgreRenderTexturePass::fullScreenQuadRenderer [protected]

Shows if we are rendering a full screen quad.

4.62.4.7 SceneNode* ManagedOgreRenderTexturePass::fullScreenQuadSceneNode [static, protected]

SceneNode instance for full screen quad.

4.62.4.8 Root* ManagedOgreRenderTexturePass::mRoot [protected]

Pointer to the current ogre root.

4.62.4.9 SceneManager* ManagedOgreRenderTexturePass::mSceneMgr [protected]

Pointer to the current Ogre SceneManager.

4.62.4.10 const char* ManagedOgreRenderTexturePass::noRenderTargetName [static]

If the name of the render target equals this String, then no render target is created.

4.62.4.11 RenderSystem* ManagedOgreRenderTexturePass::renderSystem [protected]

Pointer to the current Ogre RenderSystem.

4.62.4.12 std::vector<RenderTarget*> ManagedOgreRenderTexturePass::renderTargets [protected]

A list of render targets to use in the pass. For MRT-s.

4.62.4.13 RenderTexture* ManagedOgreRenderTexturePass::renderTexture [protected]

The pointer to the pass's own render texture object.

4.62.4.14 StringVector ManagedOgreRenderTexturePass::textureNames [protected]

List of texture names used in the pass.

4.62.4.15 std::vector<TextureUnitState*> ManagedOgreRenderTexturePass::usedTextureUnitStates [protected]

List of texture units.

4.63 MultipleUserDefinedObject Class Reference

Hashmap for multiple user defined objects.

Public Member Functions

- long getTypeID (void) const
  Inherited from Ogre::UserDefinedObject.
- const String & getTypeName (void) const
  Inherited from Ogre::UserDefinedObject.
- UserDefinedObject * getUserObjectByType (long typeId) const
  Retrieves a UserDefinedObject of specific type.
- void attachUserObject (UserDefinedObject *object)
  Inserts a new UserDefinedObject. Loses reference to existing object of the same type, if it already exists.
- UserDefinedObject * detachUserObject (long typeId)
  Removes the UserDefinedObject of the specified type ID.

Static Public Attributes

- static const long multipleUserDefinedObjectTypeId
- static const String multipleUserDefinedObjectTypeName

4.63.1 Detailed Description

Hashmap for multiple user defined objects.

4.63.2 Member Function Documentation

4.63.2.1 void MultipleUserDefinedObject::attachUserObject (UserDefinedObject * *object*) [inline]

Inserts a new UserDefinedObject. Loses reference to existing object of the same type, if it already exists.

Parameters:
object The object to be inserted.

4.63.2.2 UserDefinedObject* MultipleUserDefinedObject::detachUserObject (long *typeId*) [inline]

Removes the UserDefinedObject of the specified type ID.

Parameters:
typeId The type ID the object to be removed returns on getTypeID.

Returns:
A pointer to the removed object, or NULL if not found.

4.63.2.3 long MultipleUserDefinedObject::getTypeID (void) const [inline]

Inherted from Ogre::UserDefinedObject.

Returns:
The UserDefinedObject subclass ID, for type reflection.

4.63.2.4 const String& MultipleUserDefinedObject::getTypeName (void) const [inline]

Inherted from Ogre::UserDefinedObject.

Returns:
The UserDefinedObject subclass name, for type reflection.

4.63.2.5 UserDefinedObject* MultipleUserDefinedObject::getUserObjectByType (long *typeId*) const [inline]

Retrieves a UserDefinedObject of specific type.

Parameters:
typeId The type ID the object returns on getTypeID.

Returns:
The UserDefinedObject of the given type, or NULL, if not found.

4.63.3 Member Data Documentation

4.63.3.1 const long MultipleUserDefinedObject::multipleUserDefinedObjectTypeId [static]

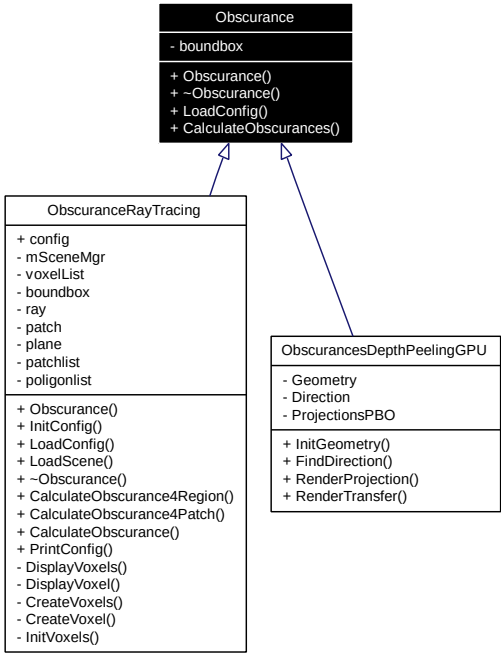
Ogre::UserDefinedObject subclass ID, for type reflection.

4.63.3.2 const String MultipleUserDefinedObject::multipleUserDefinedObjectTypeName [static]

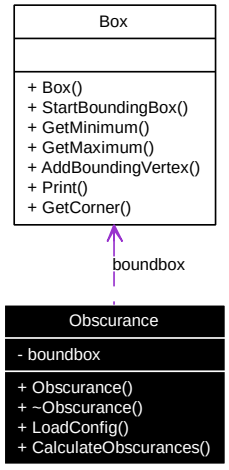
Ogre::UserDefinedObject subclass name, for type reflection.

4.64 Obscurance Class Reference

Inheritance diagram for Obscurance:



Collaboration diagram for Obscurance:



Public Member Functions

- Obscurance ()
- ~Obscurance ()

- bool LoadConfig (const char *file)
- virtual ObscuranceMap * CalculateObscurances ()

4.64.1 Detailed Description

An obscurance super class containing the common methods for all obscurance tecniques.

4.64.2 Constructor & Destructor Documentation

4.64.2.1 Obscurance::Obscurance ()

Constructor.

4.64.2.2 Obscurance::~Obscurance ()

Destructor.

Reimplemented in ObscuranceRayTracing.

4.64.3 Member Function Documentation

4.64.3.1 virtual ObscuranceMap* Obscurance::CalculateObscurances () [virtual]

Calculates obscurances.

Returns:
The calculated obscurance map.

4.64.3.2 bool Obscurance::LoadConfig (const char * *file*)

Loads the configuration.

Parameters:
file The name of the file of the configuration.

Reimplemented in ObscuranceRayTracing.

4.65 ObscuranceMap Class Reference

Public Member Functions

- ObscuranceMap ()
- ∼ObscuranceMap ()
- bool SetObscuranceMap (Texture ∗tex, unsigned int w, unsigned int h)

4.65.1 Detailed Description

A class to encapsulate a ObscuranceMap.

4.65.2 Constructor & Destructor Documentation

4.65.2.1 ObscuranceMap::ObscuranceMap ()

Constructor.

4.65.2.2 ObscuranceMap::∼ObscuranceMap ()

Destructor.

4.65.3 Member Function Documentation

4.65.3.1 bool ObscuranceMap::SetObscuranceMap (Texture ∗ *tex*, unsigned int *w*, unsigned int *h*)

Sets the Obscurance map.

Parameters:

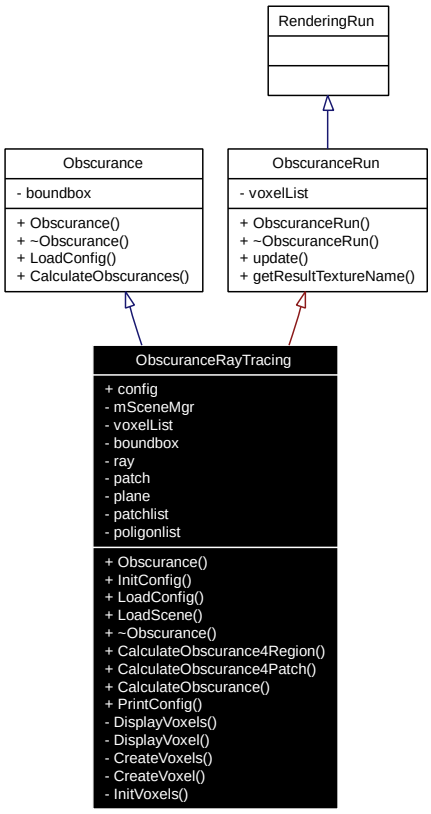
tex A pointer to the texture.

w The width of the texture.

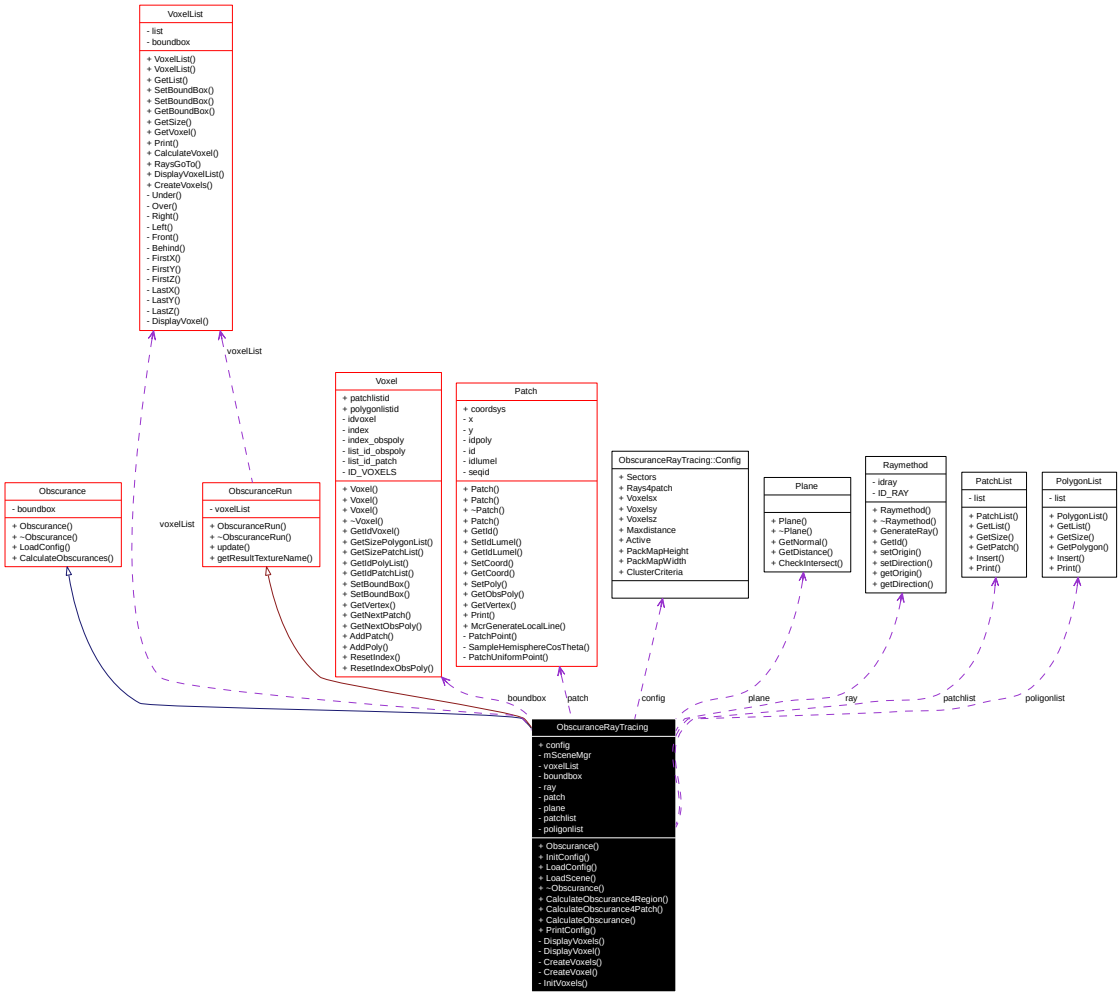
h The height of the texture.

4.66 ObscuranceRayTracing Class Reference

Inheritance diagram for ObscuranceRayTracing:



Collaboration diagram for ObscuranceRayTracing:



Public Member Functions

- Obscurance (Ogre::SceneManager *scenemanager=NULL)
- void InitConfig ()
- bool LoadConfig (const char *file)
- bool LoadScene ()
- ~Obscurance ()
- bool CalculateObscurance4Region ()
- bool CalculateObscurance4Patch ()
- ObscuranceMap * CalculateObscurance ()

Static Public Member Functions

- static bool PrintConfig ()

Public Attributes

- Config config

Classes

- struct Config

4.66.1 Detailed Description

This class reptesents the calculation of the Obscurances with ray tracing. Superclass: Obscurance, ObscuranceRun Calss: ObscuranceRayTracing

4.66.2 Constructor & Destructor Documentation

4.66.2.1 ObscuranceRayTracing::~Obscurance ()

Destructor.

Reimplemented from Obscurance.

4.66.3 Member Function Documentation

4.66.3.1 ObscuranceMap* ObscuranceRayTracing::CalculateObscurance ()

Calculates the obscurances for scene.

4.66.3.2 bool ObscuranceRayTracing::CalculateObscurance4Patch ()

Calculates Obscurance for patch.

Returns:
Returns with true if it is needed.

4.66.3.3 bool ObscuranceRayTracing::CalculateObscurance4Region ()

Calculates Obscurance for region.

Returns:
Returns with true if it is needed.

4.66.3.4 void ObscuranceRayTracing::InitConfig ()

Initializes the config to 0.

4.66.3.5 bool ObscuranceRayTracing::LoadConfig (const char * *file*)

Initializes the config with the file.

Parameters:
file The name of the config file.

Reimplemented from Obscurance.

4.66.3.6 bool ObscuranceRayTracing::LoadScene ()

Loads the scene.

4.66.3.7 ObscuranceRayTracing::Obscurance (Ogre::SceneManager * *scenemanager* = NULL)

Constructor.

Parameters:
scenemanager The scene manager of the Ogre.

4.66.3.8 static bool ObscuranceRayTracing::PrintConfig () [static]

Prints the config.

4.66.4 Member Data Documentation

4.66.4.1 Config ObscuranceRayTracing::config

the config values

4.67 ObscuranceRayTracing::Config Struct Reference

Static Public Attributes

- static unsigned int Sectors
- static unsigned int Rays4patch
- static unsigned int Voxelsx
- static unsigned int Voxelsy
- static unsigned int Voxelsz
- static float Maxdistance
- static bool Active
- static unsigned int PackMapHeight
- static unsigned int PackMapWidth
- static unsigned int ClusterCriteria

4.67.1 Detailed Description

configuration parametres read from a obscurance.cfg

4.67.2 Member Data Documentation

4.67.2.1 bool ObscuranceRayTracing::Config::Active [static]

4.67.2.2 unsigned int ObscuranceRayTracing::Config::ClusterCriteria [static]

4.67.2.3 float ObscuranceRayTracing::Config::Maxdistance [static]

4.67.2.4 unsigned int ObscuranceRayTracing::Config::PackMapHeight [static]

4.67.2.5 unsigned int ObscuranceRayTracing::Config::PackMapWidth [static]

4.67.2.6 unsigned int ObscuranceRayTracing::Config::Rays4patch [static]

4.67.2.7 unsigned int ObscuranceRayTracing::Config::Sectors [static]

4.67.2.8 unsigned int ObscuranceRayTracing::Config::Voxelsx [static]

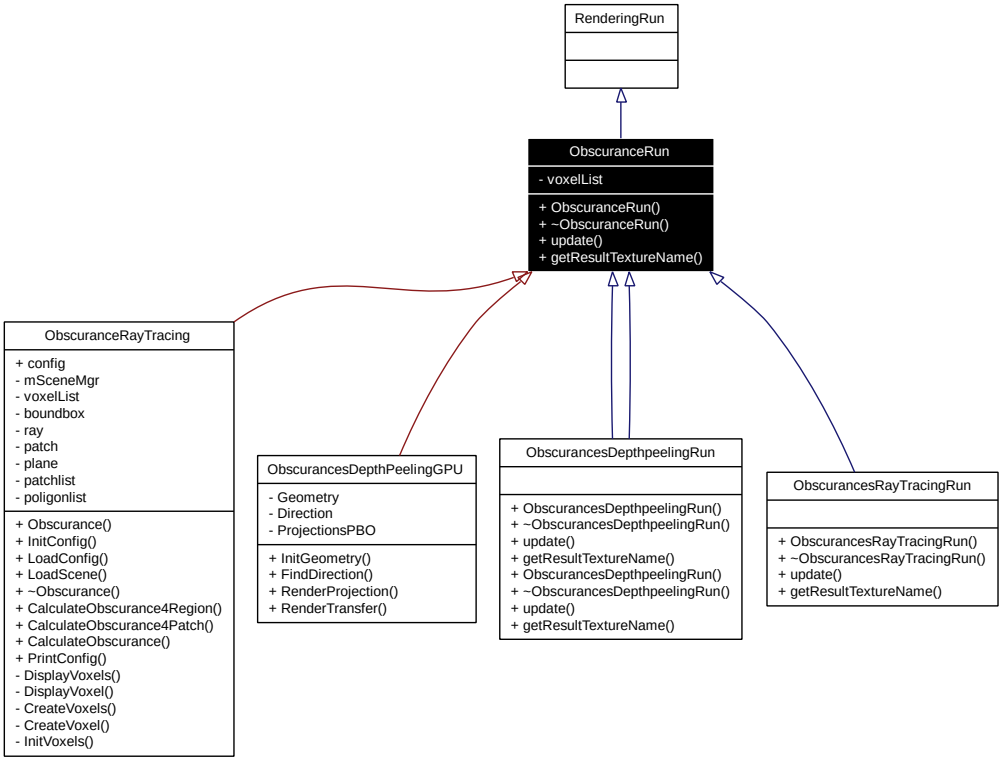
4.67.2.9 unsigned int ObscuranceRayTracing::Config::Voxelsy [static]

4.67.2.10 unsigned int ObscuranceRayTracing::Config::Voxelsz [static]

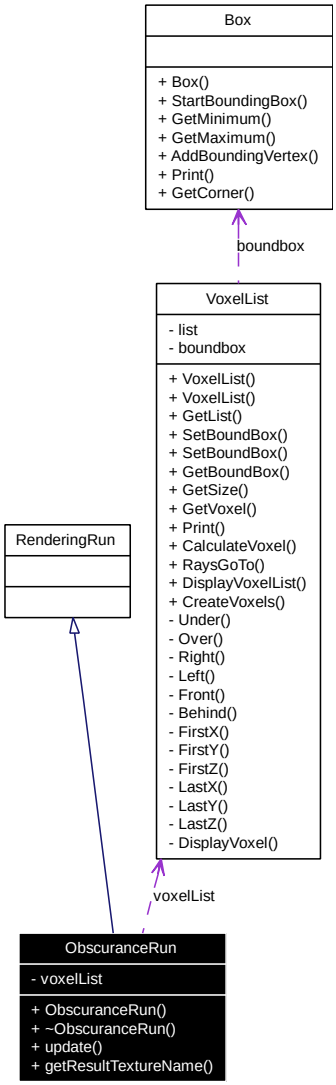
4.68 ObscuranceRun Class Reference

This run computes the obscurances.

Inheritance diagram for ObscuranceRun:



Collaboration diagram for ObscuranceRun:



Public Member Functions

- ObscuranceRun ()
- ~ObscuranceRun ()
- virtual void update ()=0
- virtual const String & getResultTextureName ()

4.68.1 Detailed Description

This run computes the obscurances.

SuperClass: RenderingRun Class: ObscuranceRun

4.68.2 Constructor & Destructor Documentation

4.68.2.1 ObscuranceRun::ObscuranceRun ()

Constructor

4.68.2.2 ObscuranceRun::~ObscuranceRun ()

Destructor

4.68.3 Member Function Documentation

4.68.3.1 virtual const String& ObscuranceRun::getResultTextureName () [inline, virtual]

Returns the main result texture's name. This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Reimplemented in ObscurancesDepthpeelingRun, ObscurancesRayTracingRun, and ObscurancesDepthpeelingRun.

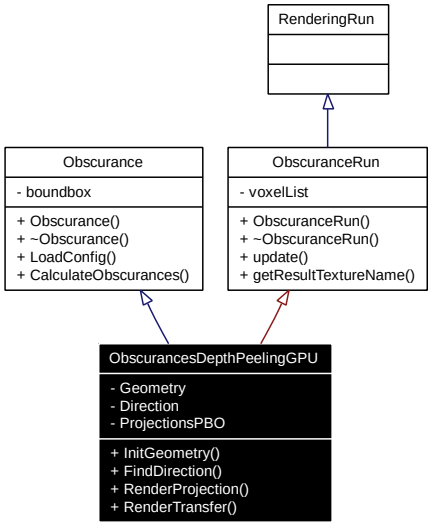
4.68.3.2 virtual void ObscuranceRun::update () [pure virtual]

Recomputs the texture.

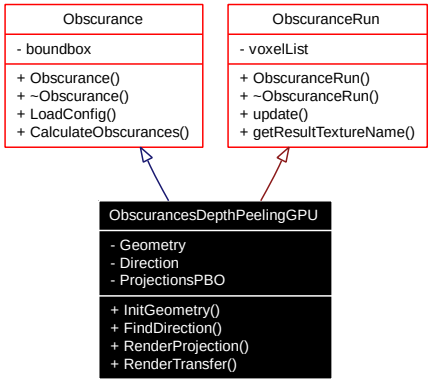
Implemented in ObscurancesDepthpeelingRun, ObscurancesRayTracingRun, and ObscurancesDepthpeelingRun.

4.69 ObscurancesDepthPeelingGPU Class Reference

Inheritance diagram for ObscurancesDepthPeelingGPU:



Collaboration diagram for ObscurancesDepthPeelingGPU:



Public Member Functions

- void InitGeometry (void)
- void FindDirection (void)
- void RenderProjection (void)
- void RenderTransfer (void)

4.69.1 Detailed Description

This class reptesents the calculation of the Obscurances with depth peeling on GPU. Superclass: Obscurance, ObscuranceRun Calss: ObscurancesDepthPeelingGPU

4.69.2 Member Function Documentation

4.69.2.1 void ObscurancesDepthPeelingGPU::FindDirection (void)

Method to find a random direction.

4.69.2.2 void ObscurancesDepthPeelingGPU::InitGeometry (void)

Method to Initialize the geometric data.

4.69.2.3 void ObscurancesDepthPeelingGPU::RenderProjection (void)

Method to depth-peel the scene in this direction.

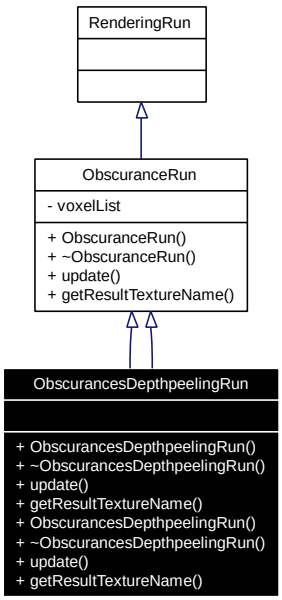
4.69.2.4 void ObscurancesDepthPeelingGPU::RenderTransfer (void)

Method to calculate the obscurances transfer between the layers.

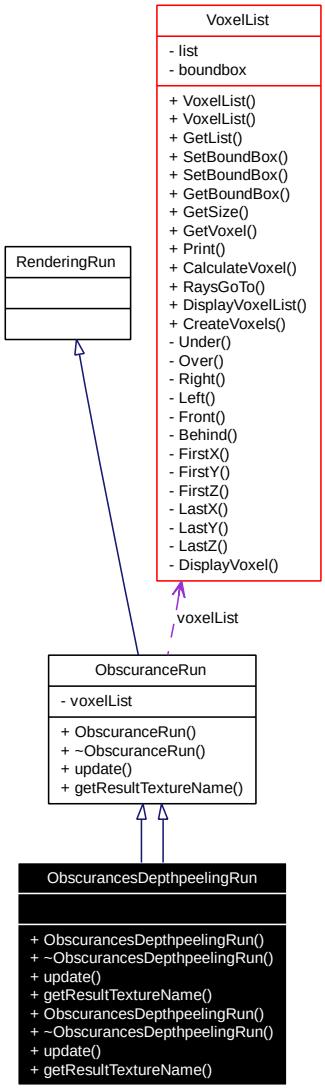
4.70 ObscurancesDepthpeelingRun Class Reference

This run computes the obscurance with depth peeling.

Inheritance diagram for ObscurancesDepthpeelingRun:



Collaboration diagram for ObscurancesDepthpeelingRun:



Public Member Functions

- ObscurancesDepthpeelingRun ()
- ~ObscurancesDepthpeelingRun ()
- virtual void update ()=0
- virtual const String & getResultTextureName ()
- ObscurancesDepthpeelingRun ()
- ~ObscurancesDepthpeelingRun ()
- virtual void update ()=0
- virtual const String & getResultTextureName ()

4.70.1 Detailed Description

This run computes the obscurance with depth peeling.

SuperClass: ObscuranceRun Class: ObscurancesDepthpeelingRun

4.70.2 Constructor & Destructor Documentation

4.70.2.1 ObscurancesDepthpeelingRun::ObscurancesDepthpeelingRun ()

Constructor

4.70.2.2 ObscurancesDepthpeelingRun::~ObscurancesDepthpeelingRun ()

Destructor

4.70.2.3 ObscurancesDepthpeelingRun::ObscurancesDepthpeelingRun ()

Constructor

4.70.2.4 ObscurancesDepthpeelingRun::~ObscurancesDepthpeelingRun ()

Destructor

4.70.3 Member Function Documentation

4.70.3.1 virtual const String& ObscurancesDepthpeelingRun::getResultTextureName () [inline, virtual]

Returns the main result texture's name. This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Reimplemented from ObscuranceRun.

4.70.3.2 virtual const String& ObscurancesDepthpeelingRun::getResultTextureName () [inline, virtual]

Returns the main result texture's name. This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Reimplemented from ObscuranceRun.

4.70.3.3 virtual void ObscurancesDepthpeelingRun::update () [pure virtual]

Recomputs the texture.

Implements ObscuranceRun.

4.70.3.4 virtual void ObscurancesDepthpeelingRun::update () [pure virtual]

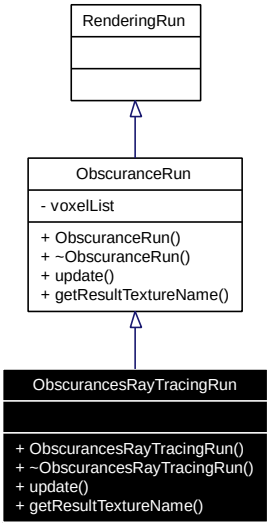
Recomputs the texture.

Implements ObscuranceRun.

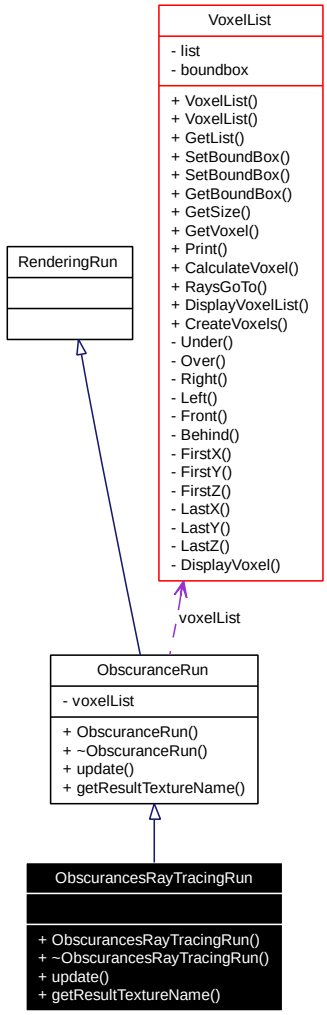
4.71 ObscurancesRayTracingRun Class Reference

This run computes the obscurance with ray tracing.

Inheritance diagram for ObscurancesRayTracingRun:



Collaboration diagram for ObscurancesRayTracingRun:



Public Member Functions

- ObscurancesRayTracingRun ()
- ~ObscurancesRayTracingRun ()
- virtual void update ()=0
- virtual const String & getResultTextureName ()

4.71.1 Detailed Description

This run computes the obscurance with ray tracing.

SuperClass: ObscuranceRun Class: ObscurancesRayTracingRun

4.71.2 Constructor & Destructor Documentation

4.71.2.1 ObscurancesRayTracingRun::ObscurancesRayTracingRun ()

Constructor

4.71.2.2 ObscurancesRayTracingRun::~ObscurancesRayTracingRun ()

Destructor

4.71.3 Member Function Documentation

4.71.3.1 virtual const String& ObscurancesRayTracingRun::getResultTextureName () [inline, virtual]

Returns the main result texture's name. This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Reimplemented from ObscuranceRun.

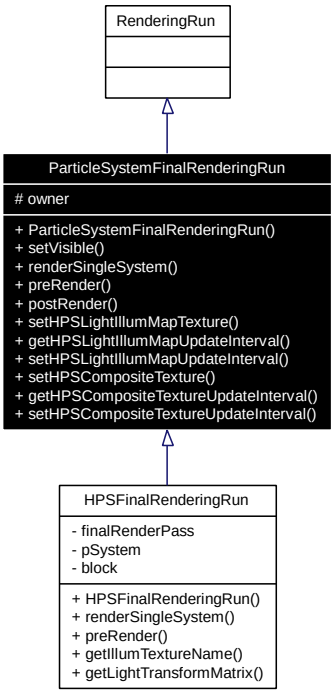
4.71.3.2 virtual void ObscurancesRayTracingRun::update () [pure virtual]

Recomputs the texture.

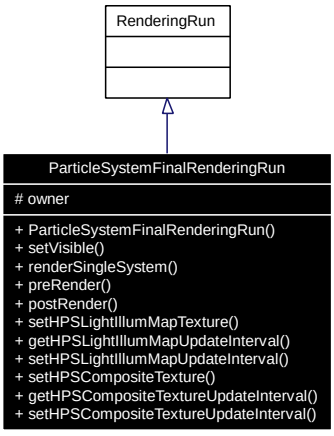
Implements ObscuranceRun.

4.72 ParticleSystemFinalRenderingRun Class Reference

Inheritance diagram for ParticleSystemFinalRenderingRun:



Collaboration diagram for ParticleSystemFinalRenderingRun:



Public Member Functions

- ParticleSystemFinalRenderingRun (ParticleSystem *owner)
- void setVisible (bool visible)

- virtual renderSingleSystem (RenderTarget ∗backBuffer, CubeMapFaces cf=CUBEMAP_FACE_-POSITIVE_X)=0
- virtual preRender (RenderTarget ∗backBuffer, CubeMapFaces cf=CUBEMAP_FACE_POSITIVE_-X)
- virtual postRender (RenderTarget ∗backBuffer, CubeMapFaces cf=CUBEMAP_FACE_-POSITIVE_X)
- virtual void setHPSLightIllumMapTexture (const String &LightillumMapTextureName)
- virtual unsigned int getHPSLightIllumMapUpdateInterval ()
- virtual void setHPSLightIllumMapUpdateInterval (unsigned int updateIntervalNumOfFrames)
- virtual void setHPSCompositeTexture (const String &CompositeTextureName)
- virtual unsigned int getHPSCompositeTextureUpdateInterval ()
- virtual void setHPSCompositeTextureUpdateInterval (unsigned int updateIntervalNumOfFrames)

Protected Attributes

- ParticleSystem ∗ owner

4.72.1 Detailed Description

This class and the EntityRenderingObject class encapsulate the complete illumination model implemented in the illumination workpackage. A FinalRenderingRun instance is stored with all ParticleSystems. What preprocessing for the final rendering is necessary is coded into extending classes. How often (in how many frames) those preprocessing runs are to be performed can be set.

Typically, a FinalRenderingRun-derived class has a number of static ManagedOgreRenderTexturePass instances for performing intermediate computations, and a non-static ManagedOgreRenderTexturePass member that renders to the frame buffer.

Implementations of the virtual functions set the appropriate parameters of the terminal ManagedRender-TargetPass member.

4.72.2 Constructor & Destructor Documentation

4.72.2.1 ParticleSystemFinalRenderingRun::ParticleSystemFinalRenderingRun (ParticleSystem ∗ *owner*) [inline]

4.72.3 Member Function Documentation

4.72.3.1 virtual unsigned int ParticleSystemFinalRenderingRun::getHPSCompositeTexture-UpdateInterval () [inline, virtual]

4.72.3.2 virtual unsigned int ParticleSystemFinalRenderingRun::getHPSLightIllumMapUpdateInterval () [inline, virtual]

4.72.3.3 virtual ParticleSystemFinalRenderingRun::postRender (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [inline, virtual]

4.72.3.4 virtual ParticleSystemFinalRenderingRun::preRender (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [inline, virtual]

Reimplemented in HPSFinalRenderingRun.

4.72.3.5 virtual ParticleSystemFinalRenderingRun::renderSingleSystem (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [pure virtual]

Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names)that had been passed to the virtual set<anything> functions reference the updated results.

This method is supposed to reproduce the behaviour of rendering an object using the standard OGRE pipeline. Thus, it is forbidden to commit any of the following:

- clear the color, depth or stencil of the backbuffer
- alter the depth testing, stencil testing, alpha blending render state without restoring it
- render with altered depth testing, stencil testing, alpha blending to the backbuffer

Implemented in HPSFinalRenderingRun.

4.72.3.6 virtual void ParticleSystemFinalRenderingRun::setHPSCompositeTexture (const String & *CompositeTextureName*) [inline, virtual]

4.72.3.7 virtual void ParticleSystemFinalRenderingRun::setHPSCompositeTextureUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

4.72.3.8 virtual void ParticleSystemFinalRenderingRun::setHPSLightIllumMapTexture (const String & *LightillumMapTextureName*) [inline, virtual]

4.72.3.9 virtual void ParticleSystemFinalRenderingRun::setHPSLightIllumMapUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [inline, virtual]

4.72.3.10 void ParticleSystemFinalRenderingRun::setVisible (bool *visible*) [inline]

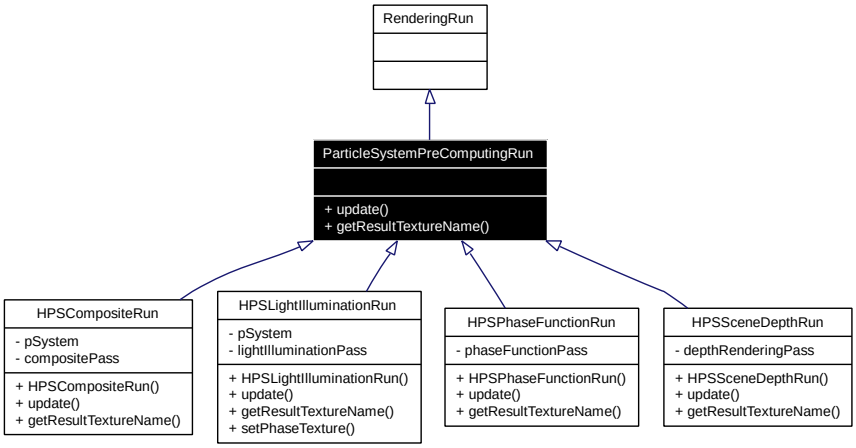
4.72.4 Member Data Documentation

4.72.4.1 ParticleSystem* ParticleSystemFinalRenderingRun::owner [protected]

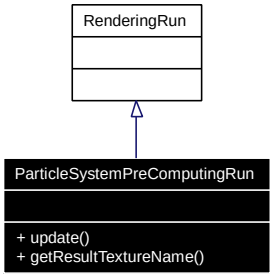
the owner particle system of this rendering descriptor instance

4.73 ParticleSystemPreComputingRun Class Reference

Inheritance diagram for ParticleSystemPreComputingRun:



Collaboration diagram for ParticleSystemPreComputingRun:



Public Member Functions

- virtual void update ()=0
- virtual const String & getResultTextureName ()

4.73.1 Member Function Documentation

4.73.1.1 virtual const String& ParticleSystemPreComputingRun::getResultTextureName () [inline, virtual]

return the main result texture's name This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Reimplemented in HPSCompositeRun, HPSLightIlluminationRun, HPSPhaseFunctionRun, and HPSSceneDepthRun.

4.73.1.2 virtual void ParticleSystemPreComputingRun::update () [pure virtual]

re-compute contained data

Implemented in HPSCompositeRun, HPSLightIlluminationRun, HPSPhaseFunctionRun, and HPSSceneDepthRun.

4.74 ParticleSystemRenderingObject Class Reference

Collaboration diagram for ParticleSystemRenderingObject:

Public Member Functions

- long getTypeID (void) const
- const String & getTypeName (void) const
- FinalRenderingRun ∗ getFinalRenderingRun ()
- ParticleSystemRenderingObject (Entity ∗owner, unsigned long startFrame, const RenderingType &renderingType)
- void updateHPSLightIllumMap (long framecount)
- void updateHPSCompositeTexture (long framecount)
- ∼ParticleSystemRenderingObject (void)

Static Public Attributes

- static const long particleSystemRenderingObjectTypeId

- static const String particleSystemRenderingObjectTypeName

4.74.1 Constructor & Destructor Documentation

4.74.1.1 ParticleSystemRenderingObject::ParticleSystemRenderingObject (Entity * *owner*, unsigned long *startFrame*, const RenderingType & *renderingType*)

4.74.1.2 ParticleSystemRenderingObject::~ParticleSystemRenderingObject (void)

4.74.2 Member Function Documentation

4.74.2.1 FinalRenderingRun* ParticleSystemRenderingObject::getFinalRenderingRun () [inline]

4.74.2.2 long ParticleSystemRenderingObject::getTypeID (void) const [inline]

4.74.2.3 const String& ParticleSystemRenderingObject::getTypeName (void) const [inline]

4.74.2.4 void ParticleSystemRenderingObject::updateHPSCompositeTexture (long *framecount*)

4.74.2.5 void ParticleSystemRenderingObject::updateHPSLightIllumMap (long *framecount*)

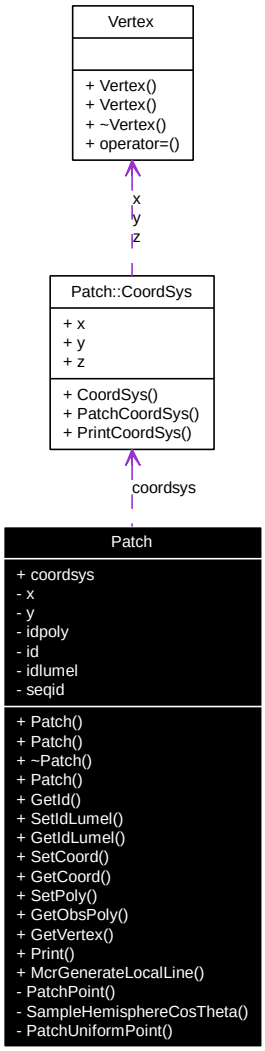
4.74.3 Member Data Documentation

4.74.3.1 const long ParticleSystemRenderingObject::particleSystemRenderingObjectTypeId [static]

4.74.3.2 const String ParticleSystemRenderingObject::particleSystemRenderingObjectTypeName [static]

4.75 Patch Class Reference

Collaboration diagram for Patch:



Public Member Functions

- Patch ()
- Patch (int, int)
- ~Patch ()
- Patch (const Patch &)
- unsigned int GetId () const
- void SetIdLumel (unsigned int i)
- unsigned int GetIdLumel () const
- void SetCoord (unsigned int xx, unsigned int yy)

- void GetCoord (unsigned int &xx, unsigned int &yy)
- void SetPoly (unsigned int)
- unsigned int GetObsPoly ()
- Vertex * GetVertex (int)
- void Print ()
- Ray * McrGenerateLocalLine (double *xi)

Public Attributes

- CoordSys coordsys

Classes

- class CoordSys

4.75.1 Detailed Description

This class represents a patch of one polygon/plane.

4.75.2 Constructor & Destructor Documentation

4.75.2.1 Patch::Patch ()

Constructor.

4.75.2.2 Patch::Patch (int, int)

Constructor.

4.75.2.3 Patch::~Patch ()

4.75.2.4 Patch::Patch (const Patch &)

4.75.3 Member Function Documentation

4.75.3.1 void Patch::GetCoord (unsigned int & *xx*, unsigned int & *yy*) [inline]

Gets the coords of the lumel in x and y.

4.75.3.2 unsigned int Patch::GetId () const [inline]

4.75.3.3 unsigned int Patch::GetIdLumel () const [inline]

Returns the id lumel of patch relative to polygon's lightmap

4.75.3.4 unsigned int Patch::GetObsPoly () [inline]

4.75.3.5 Vertex* Patch::GetVertex (int)

4.75.3.6 Ray* Patch::McrGenerateLocalLine (double * *xi*)

4.75.3.7 void Patch::Print ()

4.75.3.8 void Patch::SetCoord (unsigned int *xx*, unsigned int *yy*) [inline]

Sets the coords of the lumel in x and y.

4.75.3.9 void Patch::SetIdLumel (unsigned int *i*) [inline]

Sets the id lumel relative to polygon's lightmap

4.75.3.10 void Patch::SetPoly (unsigned *int*)

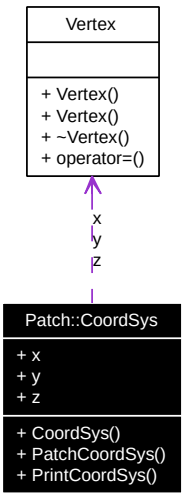
4.75.4 Member Data Documentation

4.75.4.1 CoordSys Patch::coordsys

coordinate system with Z along the normal

4.76 Patch::CoordSys Class Reference

Collaboration diagram for Patch::CoordSys:



Public Member Functions

- CoordSys ()
- void PatchCoordSys (Patch ∗P)
- void PrintCoordSys ()

Public Attributes

- Vertex x
- Vertex y
- Vertex z

4.76.1 Detailed Description

Coordsys

4.76.2 Constructor & Destructor Documentation

4.76.2.1 Patch::CoordSys::CoordSys () [inline]

4.76.3 Member Function Documentation

4.76.3.1 void Patch::CoordSys::PatchCoordSys (Patch * *P*)

Creates a coordinate system on the patch P with Z direction along the normal.

4.76.3.2 void Patch::CoordSys::PrintCoordSys ()

4.76.4 Member Data Documentation

4.76.4.1 Vertex Patch::CoordSys::x

4.76.4.2 Vertex Patch::CoordSys::y

4.76.4.3 Vertex Patch::CoordSys::z

4.77 PatchList Class Reference

Public Member Functions

- PatchList ()
- std::vector< Patch > * GetList ()
- int GetSize () const
- Patch * GetPatch (unsigned int i)
- bool Insert (Patch *patch)
- void Print ()

4.77.1 Detailed Description

PatchList

4.77.2 Constructor & Destructor Documentation

4.77.2.1 PatchList::PatchList () [inline]

4.77.3 Member Function Documentation

4.77.3.1 std::vector<Patch>* PatchList::GetList () [inline]

Returns the pointer to the list of voxels

4.77.3.2 Patch* PatchList::GetPatch (unsigned int *i*)

Returns pointer to patch(i)

4.77.3.3 int PatchList::GetSize () const [inline]

Returns the number of elements of list

4.77.3.4 bool PatchList::Insert (Patch * *patch*)

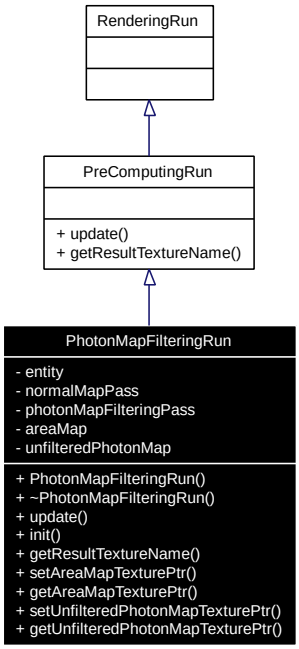
4.77.3.5 void PatchList::Print ()

print stats

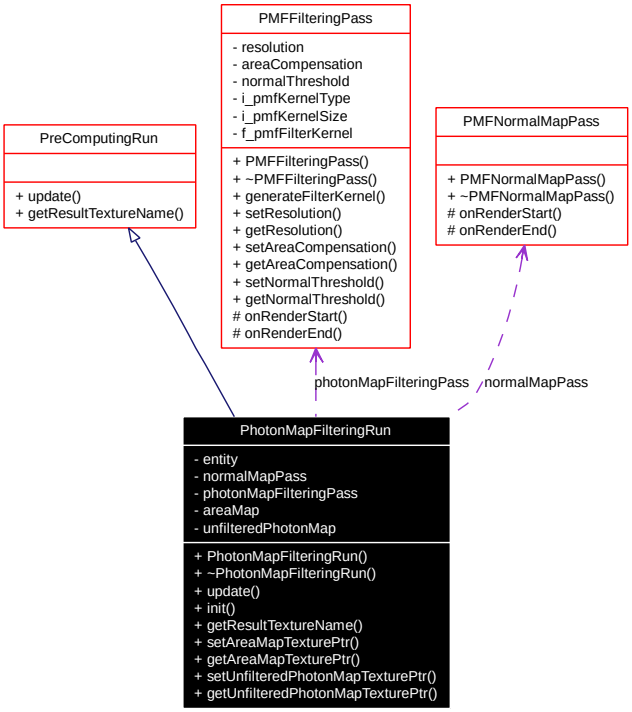
4.78 PhotonMapFilteringRun Class Reference

Computes a filtered photon map texture for an entity.

Inheritance diagram for PhotonMapFilteringRun:



Collaboration diagram for PhotonMapFilteringRun:



Public Member Functions

- PhotonMapFilteringRun (Entity *entity, const String &areaMapName, const String &unfiltered-PhotonMapName)
- ~PhotonMapFilteringRun ()
- virtual void update ()

  Updates the filtered photon map texture corresponding to the actual parameters.

- void init (float areaCompensation=-1.0, float normalThreshold=-1.0)

  Initializes the uniform rendering parameters.

- virtual const String & getResultTextureName ()

  Getter method to obtain the name of the filtered photon map texture object.

- void setAreaMapTexturePtr (TexturePtr areaMap)

  Setting up the area map texture.

- TexturePtr getAreaMapTexturePtr ()

  Obtaining the area map.

- void setUnfilteredPhotonMapTexturePtr (TexturePtr unfilteredPhotonMap)

  Setting up the unfiltered photon map texture.

- TexturePtr getUnfilteredPhotonMapTexturePtr ()

  Obtaining the unfiltered photon map texture.

4.78.1 Detailed Description

Computes a filtered photon map texture for an entity.

4.78.2 Constructor & Destructor Documentation

4.78.2.1 PhotonMapFilteringRun::PhotonMapFilteringRun (Entity * *entity*, const String & *areaMapName*, const String & *unfilteredPhotonMapName*)

Constructor.

Parameters:

entity The owner entity of an entity-bound precomputing run.

areaMapName The name of a texture file, which contains the area map.

unfilteredPhotonMapName The name of a texture file, which contains the unfiltered photon map.

4.78.2.2 PhotonMapFilteringRun::~PhotonMapFilteringRun ()

Destructor.

4.78.3 Member Function Documentation

4.78.3.1 TexturePtr PhotonMapFilteringRun::getAreaMapTexturePtr () [inline]

Obtaining the area map.

Returns:

TexturePtr object, which represents the the actual area map texture.

A getter method to obtain the area map texture object of the application.

4.78.3.2 virtual const String& PhotonMapFilteringRun::getResultTextureName () [virtual]

Getter method to obtain the name of the filtered photon map texture object.

Returns:

a reference to a String object, which represents the name of the filtered photon map texture object.

Reimplemented from PreComputingRun.

4.78.3.3 TexturePtr PhotonMapFilteringRun::getUnfilteredPhotonMapTexturePtr () [inline]

Obtaining the unfiltered photon map texture.

Returns:
TexturePtr object, which represents the the actual unfiltered photon map texture.

A getter method to obtain the unfiltered photon map texture object of the application.

4.78.3.4 void PhotonMapFilteringRun::init (float *areaCompensation* = -1.0, float *normalThreshold* = -1.0)

Initializes the uniform rendering parameters.

Calling this method the area compensation and the normal threshold parameter of the filtering algorithm will be initialized. If the input parameters are -1.0 the corresponding member variables will not be changed.

4.78.3.5 void PhotonMapFilteringRun::setAreaMapTexturePtr (TexturePtr *areaMap*) [inline]

Setting up the area map texture.

Parameters:
areaMap TexturePtr object, which represents the area map texture.

A setter method to define the area map texture object of the application.

4.78.3.6 void PhotonMapFilteringRun::setUnfilteredPhotonMapTexturePtr (TexturePtr *unfilteredPhotonMap*) [inline]

Setting up the unfiltered photon map texture.

Parameters:
unfilteredPhotonMap TexturePtr object, which represents the unfiltered photon map texture.

A setter method to define the unfiltered photon map texture object of the application.

4.78.3.7 virtual void PhotonMapFilteringRun::update () [virtual]

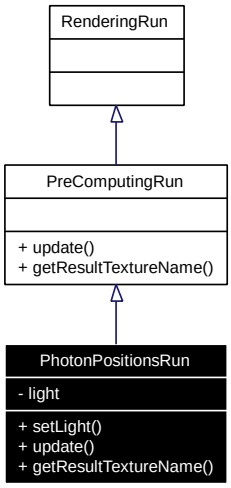
Updates the filtered photon map texture corresponding to the actual parameters.

This method calls the update method of the normal map pass and the photon map filtering pass.

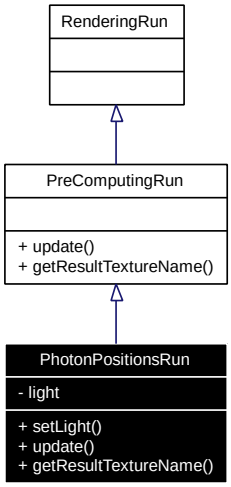
Implements PreComputingRun.

4.79 PhotonPositionsRun Class Reference

Inheritance diagram for PhotonPositionsRun:



Collaboration diagram for PhotonPositionsRun:



Public Member Functions

- void setLight (Light *light)
- virtual void update ()
- virtual const String & getResultTextureName ()

4.79.1 Detailed Description

Computes caustic photon positions for a spot light

4.79.2 Member Function Documentation

4.79.2.1 virtual const String& PhotonPositionsRun::getResultTextureName () [inline, virtual]

Reimplemented from PreComputingRun.

4.79.2.2 void PhotonPositionsRun::setLight (Light * *light*) [inline]

Parameters:

light The owner light of an light-bound precomputing run.

4.79.2.3 virtual void PhotonPositionsRun::update (void) [inline, virtual]

Implements PreComputingRun.

4.80 Plane Class Reference

Public Member Functions

- Plane ()
- ∼Plane ()
- Vertex ∗ GetNormal ()
- float GetDistance () const
- bool CheckIntersect (Ray ∗)

4.80.1 Detailed Description

This class represents a plane

4.80.2 Constructor & Destructor Documentation

4.80.2.1 Plane::Plane ()

4.80.2.2 Plane::∼Plane ()

4.80.3 Member Function Documentation

4.80.3.1 bool Plane::CheckIntersect (Ray ∗)

4.80.3.2 float Plane::GetDistance () const [inline]

4.80.3.3 Vertex∗ Plane::GetNormal ()

4.81 PlanesCorrector Class Reference

This class recieve as input all the leaves submeshes associate to each cluster plane generated by the class `SubMeshesLeavesGenerator` and the planes information generated by the class `PlaneGenerator` to fit all the leaves of each cluster plane in the smaller quad for generating the impostor texture in a later step.

Public Member Functions

- PlanesCorrector (void)

  Constructor method of the class `PlanesCorrector`.

- ~PlanesCorrector (void)

  Destructor method of the class `PlanesCorrector`.

- PlanesCorrector * getSingletonPtr (void)

  Return pointer to singleton PlanesCorrector object.

- PlanesCorrector & getSingleton (void)

  Return singleton PlanesCorrector object.

- void correctPlanes (char *filenames[ ])

  This method loads the leaves submeshes information, the information of the cluster planes and project each leaf in it,s associated cluster plane for compute the smaller quad that fit all the projected leaves in each plane.

- PlanesCorrector (void)

  Constructor method of the class `PlanesCorrector`.

- ~PlanesCorrector (void)

  Destructor method of the class `PlanesCorrector`.

- PlanesCorrector * getSingletonPtr (void)

  Return pointer to singleton PlanesCorrector object.

- PlanesCorrector & getSingleton (void)

  Return singleton PlanesCorrector object.

- void correctPlanes (char *filenames[ ])

  This method loads the leaves submeshes information, the information of the cluster planes and project each leaf in it,s associated cluster plane for compute the smaller quad that fit all the projected leaves in each plane.

Protected Attributes

- char * cFileName01
- char * cFileName03
- TIXML_STRING * fileName03
- unsigned int numPlanes03
- TiXmlNode * node03
- TiXmlNode * planeNode03
- TiXmlNode * coord4dNode03
- time_t timer
- tm * initTime
- TiXmlDocument doc03
- bool loadOkay03
- TIXML_STRING * outputFilename
- vector< Vector4 > vPlanes
- TiXmlDocument outputFile
- TiXmlNode * rootNode
- TiXmlNode * planeNode
- TiXmlNode * coord4dNode
- TiXmlNode * iNode
- TiXmlNode * jNode
- TiXmlNode * kNode
- TiXmlNode * vTopLeftNode
- TiXmlNode * vTopRightNode
- TiXmlNode * vBottomRightNode
- TiXmlNode * vBottomLeftNode
- tm * endTime
- char * cFileName01
- char * cFileName03
- TIXML_STRING * fileName03
- TiXmlNode * node03
- TiXmlNode * planeNode03
- TiXmlNode * coord4dNode03
- tm * initTime
- TIXML_STRING * outputFilename
- vector< Vector4 > vPlanes
- TiXmlNode * rootNode
- TiXmlNode * planeNode
- TiXmlNode * coord4dNode
- TiXmlNode * iNode
- TiXmlNode * jNode
- TiXmlNode * kNode
- TiXmlNode * vTopLeftNode
- TiXmlNode * vTopRightNode
- TiXmlNode * vBottomRightNode
- TiXmlNode * vBottomLeftNode
- tm * endTime

4.81.1 Detailed Description

This class recieve as input all the leaves submeshes associate to each cluster plane generated by the class SubMeshesLeavesGenerator and the planes information generated by the class PlaneGenerator to fit all the leaves of each cluster plane in the smaller quad for generating the impostor texture in a later step.

The main tasks of the class PlanesCorrector are:

- Load the submeshes files that contains all the faces of the leaves associated to each cluster plane.
- Project all the leaves of each submesh to each associated cluster plane.
- Compute the smaller quad that fits all the projected leaves.
- Store a xml file that contains the number of planes, and the vertexs of the smaller plane generated for each cluster plane.

4.81.2 Constructor & Destructor Documentation

4.81.2.1 PlanesCorrector::PlanesCorrector (void) [inline]

Constructor method of the class PlanesCorrector.

4.81.2.2 PlanesCorrector::~PlanesCorrector (void) [inline]

Destructor method of the class PlanesCorrector.

4.81.2.3 PlanesCorrector::PlanesCorrector (void) [inline]

Constructor method of the class PlanesCorrector.

4.81.2.4 PlanesCorrector::~PlanesCorrector (void) [inline]

Destructor method of the class PlanesCorrector.

4.81.3 Member Function Documentation

4.81.3.1 void PlanesCorrector::correctPlanes (char * *filenames*[ ]) [inline]

This method loads the leaves submeshes information, the information of the cluster planes and project each leaf in it,s associated cluster plane for compute the smaller quad that fit all the projected leaves in each plane.

Parameters:
filenames Strings that contains all the filenames needed

4.81.3.2 void PlanesCorrector::correctPlanes (char * *filenames*[ ]) [inline]

This method loads the leaves submeshes information, the information of the cluster planes and project each leaf in it,s associated cluster plane for compute the smaller quad that fit all the projected leaves in each plane.

Parameters:
filenames Strings that contains all the filenames needed

4.81.3.3 PlanesCorrector& PlanesCorrector::getSingleton (void) [inline]

Return singleton PlanesCorrector object.

Returns:
Singleton PlanesCorrector object

4.81.3.4 PlanesCorrector& PlanesCorrector::getSingleton (void) [inline]

Return singleton PlanesCorrector object.

Returns:
Singleton PlanesCorrector object

4.81.3.5 PlanesCorrector* PlanesCorrector::getSingletonPtr (void) [inline]

Return pointer to singleton PlanesCorrector object.

Returns:
Pointer to singleton PlanesCorrector object

4.81.3.6 PlanesCorrector* PlanesCorrector::getSingletonPtr (void) [inline]

Return pointer to singleton PlanesCorrector object.

Returns:
Pointer to singleton PlanesCorrector object

4.81.4 Member Data Documentation

4.81.4.1 char* PlanesCorrector::cFileName01 [protected]

4.81.4.2 char* PlanesCorrector::cFileName01 [protected]

4.81.4.3 char* PlanesCorrector::cFileName03 [protected]

4.81.4.4 char* PlanesCorrector::cFileName03 [protected]

4.81.4.5 TiXmlNode* PlanesCorrector::coord4dNode [protected]

4.81.4.6 TiXmlNode* PlanesCorrector::coord4dNode [protected]

4.81.4.7 TiXmlNode* PlanesCorrector::coord4dNode03 [protected]

4.81.4.8 TiXmlNode* PlanesCorrector::coord4dNode03 [protected]

4.81.4.9 TiXmlDocument PlanesCorrector::doc03 [protected]

4.81.4.10 struct tm* PlanesCorrector::endTime [protected]

4.81.4.11 struct tm* PlanesCorrector::endTime [protected]

4.81.4.12 TIXML_STRING* PlanesCorrector::fileName03 [protected]

4.81.4.13 TIXML_STRING* PlanesCorrector::fileName03 [protected]

4.81.4.14 struct tm* PlanesCorrector::initTime [protected]

4.81.4.15 struct tm* PlanesCorrector::initTime [protected]

4.81.4.16 TiXmlNode* PlanesCorrector::iNode [protected]

4.81.4.17 TiXmlNode* PlanesCorrector::iNode [protected]

4.81.4.18 TiXmlNode* PlanesCorrector::jNode [protected]

4.81.4.19 TiXmlNode* PlanesCorrector::jNode [protected]

4.81.4.20 TiXmlNode* PlanesCorrector::kNode [protected]

4.81.4.21 TiXmlNode* PlanesCorrector::kNode [protected]

4.81.4.22 bool PlanesCorrector::loadOkay03 [protected]

4.81.4.23 TiXmlNode* PlanesCorrector::node03 [protected]

4.81.4.24 TiXmlNode* PlanesCorrector::node03 [protected]

4.81.4.25 unsigned int PlanesCorrector::numPlanes03 [protected]

4.81.4.26 TiXmlDocument PlanesCorrector::outputFile [protected]

4.81.4.27 TIXML_STRING* PlanesCorrector::outputFilename [protected]

4.81.4.28 TIXML_STRING* PlanesCorrector::outputFilename [protected]

4.81.4.29 TiXmlNode* PlanesCorrector::planeNode [protected]

4.81.4.30 TiXmlNode* PlanesCorrector::planeNode [protected]

4.81.4.31 TiXmlNode* PlanesCorrector::planeNode03 [protected]

4.81.4.32 TiXmlNode* PlanesCorrector::planeNode03 [protected]

4.81.4.33 TiXmlNode* PlanesCorrector::rootNode [protected]

4.81.4.34 TiXmlNode* PlanesCorrector::rootNode [protected]

4.81.4.35 time_t PlanesCorrector::timer [protected]

4.81.4.36 TiXmlNode* PlanesCorrector::vBottomLeftNode [protected]

4.81.4.37 TiXmlNode* PlanesCorrector::vBottomLeftNode [protected]

4.81.4.38 TiXmlNode* PlanesCorrector::vBottomRightNode [protected]

4.81.4.39 TiXmlNode* PlanesCorrector::vBottomRightNode [protected]

4.81.4.40 vector<Vector4> PlanesCorrector::vPlanes [protected]

4.81.4.41 vector<Vector4> PlanesCorrector::vPlanes [protected]

4.81.4.42 TiXmlNode* PlanesCorrector::vTopLeftNode [protected]

4.81.4.43 TiXmlNode* PlanesCorrector::vTopLeftNode [protected]

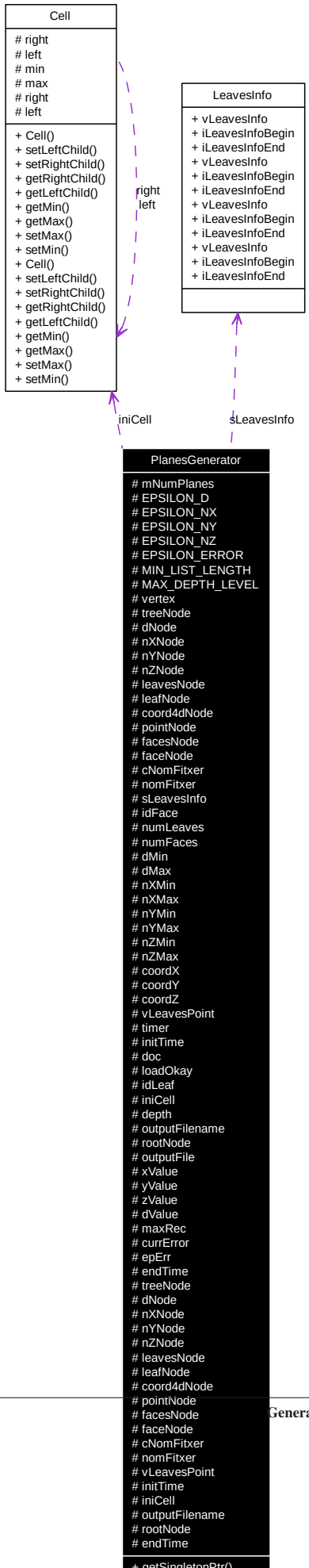
4.81.4.44 TiXmlNode* PlanesCorrector::vTopRightNode [protected]

4.81.4.45 TiXmlNode* PlanesCorrector::vTopRightNode [protected]

4.82 PlanesGenerator Class Reference

This class implements a clustering algorithm and identify N = 10-100 cluster planes. These clusters will be planes to which the leaf members are close and stores the [nx, ny, nz, d] cluster parameters of the planes generated. Uses the leaves information generated by the class `LeavesGenerator`.

Collaboration diagram for PlanesGenerator:


Cell
right
left
min
max
right
left
+ Cell()
+ setLeftChild()
+ setRightChild()
+ getRightChild()
+ getLeftChild()
+ getMin()
+ getMax()
+ setMax()
+ setMin()
+ Cell()
+ setLeftChild()
+ setRightChild()
+ getRightChild()
+ getLeftChild()
+ getMin()
+ getMax()
+ setMax()
+ setMin()
right
left
LeavesInfo
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
iniCell
sLeavesInfo
PlanesGenerator
mNumPlanes
EPSILON_D
EPSILON_NX
EPSILON_NY
EPSILON_NZ
EPSILON_ERROR
MIN_LIST_LENGTH
MAX_DEPTH_LEVEL
vertex
treeNode
dNode
nXNode
nYNode
nZNode
leavesNode
leafNode
coord4dNode
pointNode
facesNode
faceNode
cNomFitxer
nomFitxer
sLeavesInfo
idFace
numLeaves
numFaces
dMin
dMax
nXMin
nXMax
nYMin
nYMax
nZMin
nZMax
coordX
coordY
coordZ
vLeavesPoint
timer
initTime
doc
loadOkay
idLeaf
iniCell
depth
outputFilename
rootNode
outputFile
xValue
yValue
zValue
dValue
maxRec
currError
epErr
endTime
treeNode
dNode
nXNode
nYNode
nZNode
leavesNode
leafNode
coord4dNode
pointNode
facesNode
faceNode
cNomFitxer
nomFitxer
vLeavesPoint
initTime
iniCell
outputFilename
rootNode
endTime

Public Member Functions

- PlanesGenerator * getSingletonPtr (void)

  Return pointer to singleton PlanesGenerator object.

- PlanesGenerator & getSingleton (void)

  Return singleton PlanesGenerator object.

- void init (char *filenames[ ])

  This method loads the xml files with the leaves mesh, and the leaves information needed to apply the plane clustering algorithm.

- PlanesGenerator (void)

  Constructor method of the class `PlanesGenerator`.

- ~PlanesGenerator (void)

  Destructor method of the class `PlanesGenerator`.

- void setEpsilonD (Real epD)

  This method set the epsilon d *value that will be used to stop the clustering generation when the difference between the max and min value of the* d *parameter is smaller than the* epsilon.

- void setEpsilonNX (Real epNX)

  This method set the epsilon nx *(x coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the* NX *parameter is smaller than the* epsilon.

- void setEpsilonNY (Real epNY)

  This method set the epsilon ny *(y coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the* NY *parameter is smaller than the* epsilon.

- void setEpsilonNZ (Real epNZ)

  This method set the epsilon nz *(z coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the* NZ *parameter is smaller than the* epsilon.

- void setEpsilonERROR (Real epErr)

  This method set the epsilon d error *value that will be used to stop the clustering generation when the difference between d parameter of the current cluster plane and the d parameter computed with the normal plane and one point of each leaf contained in the current cluster plane is smaller than this* epsilon.

- void setMinListLength (Real minLL)

  This method set min length of the list of leaves that can be associated to each plane cluster. This parameter is used to stop the plane clustering algorithm when the current plane leaves list is smaller or equal to the min size.

- void setMaxDepthLevel (Real maxD)

  This method set max depth recursive level that the plane clustering algorithm can achieve.

- unsigned int getNumPlanes ()

  This method get the number of cluster planes that have been generated.

- void kdTreeBuild (Cell *cell, Real currError, unsigned int depth, unsigned int currCoord, LeavesInfo &sLeavesInfo, TiXmlNode *currNode)

  This method construct the kd-tree used to cluster the leaves in different planes.

- PlanesGenerator * getSingletonPtr (void)

  Return pointer to singleton PlanesGenerator object.

- PlanesGenerator & getSingleton (void)

  Return singleton PlanesGenerator object.

- void init (char *filenames[ ])

  This method loads the xml files with the leaves mesh, and the leaves information needed to apply the plane clustering algorithm.

- PlanesGenerator (void)

  Constructor method of the class `PlanesGenerator`.

- ~PlanesGenerator (void)

  Destructor method of the class `PlanesGenerator`.

- void setEpsilonD (Real epD)

  This method set the epsilon d *value that will be used to stop the clustering generation when the difference between the max and min value of the* d *parameter is smaller than the* epsilon.

- void setEpsilonNX (Real epNX)

  This method set the epsilon nx *(x coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the* NX *parameter is smaller than the* epsilon.

- void setEpsilonNY (Real epNY)

  This method set the epsilon ny *(y coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the* NY *parameter is smaller than the* epsilon.

- void setEpsilonNZ (Real epNZ)

  This method set the epsilon nz *(z coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the* NZ *parameter is smaller than the* epsilon.

- void setEpsilonERROR (Real epErr)

  This method set the epsilon d error *value that will be used to stop the clustering generation when the difference between d parameter of the current cluster plane and the d parameter computed with the normal plane and one point of each leaf contained in the current cluster plane is smaller than this* epsilon.

- void setMinListLength (Real minLL)

  This method set min length of the list of leaves that can be associated to each plane cluster. This parameter is used to stop the plane clustering algorithm when the current plane leaves list is smaller or equal to the min size.

- void setMaxDepthLevel (Real maxD)

  This method set max depth recursive level that the plane clustering algorithm can achieve.

- unsigned int getNumPlanes ()

  This method get the number of cluster planes that have been generated.

- void kdTreeBuild (Cell *cell, Real currError, unsigned int depth, unsigned int currCoord, LeavesInfo &sLeavesInfo, TiXmlNode *currNode)

  This method construct the kd-tree used to cluster the leaves in different planes.

Protected Attributes

- unsigned int mNumPlanes
- Real EPSILON_D
- Real EPSILON_NX
- Real EPSILON_NY
- Real EPSILON_NZ
- Real EPSILON_ERROR
- Real MIN_LIST_LENGTH
- Real MAX_DEPTH_LEVEL
- Vector3 vertex
- TiXmlNode * treeNode
- TiXmlNode * dNode
- TiXmlNode * nXNode
- TiXmlNode * nYNode
- TiXmlNode * nZNode
- TiXmlNode * leavesNode
- TiXmlNode * leafNode
- TiXmlNode * coord4dNode
- TiXmlNode * pointNode
- TiXmlNode * facesNode
- TiXmlNode * faceNode
- char * cNomFitxer
- TIXML_STRING * nomFitxer
- LeavesInfo sLeavesInfo
- unsigned int idFace
- unsigned int numLeaves
- unsigned int numFaces
- Real dMin
- Real dMax
- Real nXMin
- Real nXMax
- Real nYMin
- Real nYMax
- Real nZMin
- Real nZMax
- Real coordX
- Real coordY
- Real coordZ
- vector< Vector3 > vLeavesPoint
- time_t timer

- tm * initTime
- TiXmlDocument doc
- bool loadOkay
- unsigned int idLeaf
- Cell * iniCell
- unsigned int depth
- TIXML_STRING * outputFilename
- TiXmlNode * rootNode
- TiXmlDocument outputFile
- Real xValue
- Real yValue
- Real zValue
- Real dValue
- Real maxRec
- Real currError
- Real epErr
- tm * endTime
- TiXmlNode * treeNode
- TiXmlNode * dNode
- TiXmlNode * nXNode
- TiXmlNode * nYNode
- TiXmlNode * nZNode
- TiXmlNode * leavesNode
- TiXmlNode * leafNode
- TiXmlNode * coord4dNode
- TiXmlNode * pointNode
- TiXmlNode * facesNode
- TiXmlNode * faceNode
- char * cNomFitxer
- TIXML_STRING * nomFitxer
- vector< Vector3 > vLeavesPoint
- tm * initTime
- Cell * iniCell
- TIXML_STRING * outputFilename
- TiXmlNode * rootNode
- tm * endTime

Classes

- struct lessCoord

4.82.1 Detailed Description

This class implements a clustering algorithm and identify N = 10-100 cluster planes. These clusters will be planes to which the leaf members are close and stores the [nx, ny, nz, d] cluster parameters of the planes generated. Uses the leaves information generated by the class `LeavesGenerator`.

The main tasks of the class `PlanesGenerator` are:

- Load the mesh file that contains all the faces of the leaves.

- Load the leaves information generated by the LeavesGenerator class.
- Apply the clustering algorithm to all the leaves.
- Store a xml file that contains the number of cluster planes generated, and the list of leaves that each plane cluster contain.

4.82.2 Constructor & Destructor Documentation

4.82.2.1 PlanesGenerator::PlanesGenerator (void) [inline]

Constructor method of the class PlanesGenerator.

4.82.2.2 PlanesGenerator::~PlanesGenerator (void) [inline]

Destructor method of the class PlanesGenerator.

4.82.2.3 PlanesGenerator::PlanesGenerator (void) [inline]

Constructor method of the class PlanesGenerator.

4.82.2.4 PlanesGenerator::~PlanesGenerator (void) [inline]

Destructor method of the class PlanesGenerator.

4.82.3 Member Function Documentation

4.82.3.1 unsigned int PlanesGenerator::getNumPlanes () [inline]

This method get the number of cluster planes that have been generated.

Returns:
The number of cluster planes generated.

4.82.3.2 unsigned int PlanesGenerator::getNumPlanes () [inline]

This method get the number of cluster planes that have been generated.

Returns:
The number of cluster planes generated.

4.82.3.3 PlanesGenerator& PlanesGenerator::getSingleton (void) [inline]

Return singleton PlanesGenerator object.

Returns:
Singleton PlanesGenerator object

4.82.3.4 PlanesGenerator& PlanesGenerator::getSingleton (void) [inline]

Return singleton PlanesGenerator object.

Returns:
Singleton PlanesGenerator object

4.82.3.5 PlanesGenerator* PlanesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton PlanesGenerator object.

Returns:
Pointer to singleton PlanesGenerator object

4.82.3.6 PlanesGenerator* PlanesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton PlanesGenerator object.

Returns:
Pointer to singleton PlanesGenerator object

4.82.3.7 void PlanesGenerator::init (char * *filenames*[ ]) [inline]

This method loads the xml files with the leaves mesh, and the leaves information needed to apply the plane clustering algorithm.

Parameters:

filenames Strings that contains all the filenames needed

4.82.3.8 void PlanesGenerator::init (char * *filenames*[ ]) [inline]

This method loads the xml files with the leaves mesh, and the leaves information needed to apply the plane clustering algorithm.

Parameters:

filenames Strings that contains all the filenames needed

4.82.3.9 void PlanesGenerator::kdTreeBuild (Cell * *cell*, Real *currError*, unsigned int *depth*, unsigned int *currCoord*, LeavesInfo & *sLeavesInfo*, TiXmlNode * *currNode*) [inline]

This method construct the kd-tree used to cluster the leaves in different planes.

Parameters:

cell The current cell that should be clustered.

currError The *d* parameter error used to stop the clustering process.

depth The recursive level.

currCoord The current coordinate of the 4d (normal x,y,z and d) struct of the plane used to cluster the leaves.

sLeavesInfo Info.

currNode The current Node.

4.82.3.10 void PlanesGenerator::kdTreeBuild (Cell * *cell*, Real *currError*, unsigned int *depth*, unsigned int *currCoord*, LeavesInfo & *sLeavesInfo*, TiXmlNode * *currNode*) [inline]

This method construct the kd-tree used to cluster the leaves in different planes.

Parameters:

cell The current cell that should be clustered.

currError The *d* parameter error used to stop the clustering process.

depth The recursive level.

currCoord The current coordinate of the 4d (normal x,y,z and d) struct of the plane used to cluster the leaves.

sLeavesInfo Info.

currNode The current Node.

4.82.3.11 void PlanesGenerator::setEpsilonD (Real *epD*) [inline]

This method set the *epsilon d* value that will be used to stop the clustering generation when the difference between the max and min value of the *d* parameter is smaller than the *epsilon*.

Parameters:
epD The *epsilon d* value to set.

4.82.3.12 void PlanesGenerator::setEpsilonD (Real *epD*) [inline]

This method set the *epsilon d* value that will be used to stop the clustering generation when the difference between the max and min value of the *d* parameter is smaller than the *epsilon*.

Parameters:
epD The *epsilon d* value to set.

4.82.3.13 void PlanesGenerator::setEpsilonERROR (Real *epErr*) [inline]

This method set the *epsilon d error* value that will be used to stop the clustering generation when the difference between d parameter of the current cluster plane and the d parameter computed with the normal plane and one point of each leaf contained in the current cluster plane is smaller than this *epsilon*.

Parameters:
epErr The *epsilon d error* value to set.

4.82.3.14 void PlanesGenerator::setEpsilonERROR (Real *epErr*) [inline]

This method set the *epsilon d error* value that will be used to stop the clustering generation when the difference between d parameter of the current cluster plane and the d parameter computed with the normal plane and one point of each leaf contained in the current cluster plane is smaller than this *epsilon*.

Parameters:
epErr The *epsilon d error* value to set.

4.82.3.15 void PlanesGenerator::setEpsilonNX (Real *epNX*) [inline]

This method set the *epsilon nx* (x coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the *NX* parameter is smaller than the *epsilon*.

Parameters:
epNX The *epsilon nx* value to set.

4.82.3.16 void PlanesGenerator::setEpsilonNX (Real *epNX*) [inline]

This method set the *epsilon nx* (x coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the *NX* parameter is smaller than the *epsilon*.

Parameters:
epNX The *epsilon nx* value to set.

4.82.3.17 void PlanesGenerator::setEpsilonNY (Real *epNY*) [inline]

This method set the *epsilon ny* (y coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the *NY* parameter is smaller than the *epsilon*.

Parameters:
epNY The *epsilon ny* value to set.

4.82.3.18 void PlanesGenerator::setEpsilonNY (Real *epNY*) [inline]

This method set the *epsilon ny* (y coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the *NY* parameter is smaller than the *epsilon*.

Parameters:
epNY The *epsilon ny* value to set.

4.82.3.19 void PlanesGenerator::setEpsilonNZ (Real *epNZ*) [inline]

This method set the *epsilon nz* (z coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the *NZ* parameter is smaller than the *epsilon*.

Parameters:
epNZ The *epsilon nz* value to set.

4.82.3.20 void PlanesGenerator::setEpsilonNZ (Real *epNZ*) [inline]

This method set the *epsilon nz* (z coordinate of the plane normal) value that will be used to stop the clustering generation when the difference between the max and min value of the *NZ* parameter is smaller than the *epsilon*.

Parameters:
epNZ The *epsilon nz* value to set.

4.82.3.21 void PlanesGenerator::setMaxDepthLevel (Real *maxD*) [inline]

This method set max depth recursive level that the plane clustering algorithm can achieve.

Parameters:
maxD The maximum depth recursive level.

4.82.3.22 void PlanesGenerator::setMaxDepthLevel (Real *maxD*) [inline]

This method set max depth recursive level that the plane clustering algorithm can achieve.

Parameters:
maxD The maximum depth recursive level.

4.82.3.23 void PlanesGenerator::setMinListLength (Real *minLL*) [inline]

This method set min length of the list of leaves that can be associated to each plane cluster. This parameter is used to stop the plane clustering algorithm when the current plane leaves list is smaller or equal to the min size.

Parameters:
minLL The minimum number of leaves that we want in every cluster plane.

4.82.3.24 void PlanesGenerator::setMinListLength (Real *minLL*) [inline]

This method set min length of the list of leaves that can be associated to each plane cluster. This parameter is used to stop the plane clustering algorithm when the current plane leaves list is smaller or equal to the min size.

Parameters:
minLL The minimum number of leaves that we want in every cluster plane.

4.82.4 Member Data Documentation

4.82.4.1 char* PlanesGenerator::cNomFitxer [protected]

4.82.4.2 char* PlanesGenerator::cNomFitxer [protected]

4.82.4.3 TiXmlNode* PlanesGenerator::coord4dNode [protected]

4.82.4.4 TiXmlNode* PlanesGenerator::coord4dNode [protected]

4.82.4.5 Real PlanesGenerator::coordX [protected]

4.82.4.6 Real PlanesGenerator::coordY [protected]

4.82.4.7 Real PlanesGenerator::coordZ [protected]

4.82.4.8 Real PlanesGenerator::currError [protected]

4.82.4.9 unsigned int PlanesGenerator::depth [protected]

4.82.4.10 Real PlanesGenerator::dMax [protected]

4.82.4.11 Real PlanesGenerator::dMin [protected]

4.82.4.12 TiXmlNode* PlanesGenerator::dNode [protected]

4.82.4.13 TiXmlNode* PlanesGenerator::dNode [protected]

4.82.4.14 TiXmlDocument PlanesGenerator::doc [protected]

4.82.4.15 Real PlanesGenerator::dValue [protected]

4.82.4.16 struct tm* PlanesGenerator::endTime [protected]

4.82.4.17 struct tm* PlanesGenerator::endTime [protected]

4.82.4.18 Real PlanesGenerator::epErr [protected]

4.82.4.19 Real PlanesGenerator::EPSILON_D [protected]

4.82.4.20 Real PlanesGenerator::EPSILON_ERROR [protected]

4.82.4.21 Real PlanesGenerator::EPSILON_NX [protected]

4.82.4.22 Real PlanesGenerator::EPSILON_NY [protected]

4.82.4.23 Real PlanesGenerator::EPSILON_NZ [protected]

4.82.4.24 TiXmlNode* PlanesGenerator::faceNode [protected]

4.82.4.25 TiXmlNode* PlanesGenerator::faceNode [protected]

4.82.4.26 TiXmlNode* PlanesGenerator::facesNode [protected]

4.82.4.27 TiXmlNode* PlanesGenerator::facesNode [protected]

4.82.4.28 unsigned int PlanesGenerator::idFace [protected]

4.82.4.29 unsigned int PlanesGenerator::idLeaf [protected]

4.82.4.30 Cell* PlanesGenerator::iniCell [protected]

4.82.4.31 Cell* PlanesGenerator::iniCell [protected]

4.82.4.32 struct tm* PlanesGenerator::initTime [protected]

4.82.4.33 struct tm* PlanesGenerator::initTime [protected]

4.82.4.34 TiXmlNode* PlanesGenerator::leafNode [protected]

4.82.4.35 TiXmlNode* PlanesGenerator::leafNode [protected]

4.82.4.36 TiXmlNode* PlanesGenerator::leavesNode [protected]

4.82.4.37 TiXmlNode* PlanesGenerator::leavesNode [protected]

4.82.4.38 bool PlanesGenerator::loadOkay [protected]

4.82.4.39 Real PlanesGenerator::MAX_DEPTH_LEVEL [protected]

4.82.4.40 Real PlanesGenerator::maxRec [protected]

4.82.4.41 Real PlanesGenerator::MIN_LIST_LENGTH [protected]

4.82.4.42 unsigned int PlanesGenerator::mNumPlanes [protected]

4.82.4.43 TIXML_STRING* PlanesGenerator::nomFitxer [protected]

4.82.4.44 TIXML_STRING* PlanesGenerator::nomFitxer [protected]

4.82.4.45 unsigned int PlanesGenerator::numFaces [protected]

4.82.4.46 unsigned int PlanesGenerator::numLeaves [protected]

4.82.4.47 Real PlanesGenerator::nXMax [protected]

4.82.4.48 Real PlanesGenerator::nXMin [protected]

4.82.4.49 TiXmlNode* PlanesGenerator::nXNode [protected]

4.82.4.50 TiXmlNode* PlanesGenerator::nXNode [protected]

4.82.4.51 Real PlanesGenerator::nYMax [protected]

4.82.4.52 Real PlanesGenerator::nYMin [protected]

4.82.4.53 TiXmlNode* PlanesGenerator::nYNode [protected]

4.82.4.54 TiXmlNode* PlanesGenerator::nYNode [protected]

4.82.4.55 Real PlanesGenerator::nZMax [protected]

4.82.4.56 Real PlanesGenerator::nZMin [protected]

4.82.4.57 TiXmlNode* PlanesGenerator::nZNode [protected]

4.82.4.58 TiXmlNode* PlanesGenerator::nZNode [protected]

4.82.4.59 TiXmlDocument PlanesGenerator::outputFile [protected]

4.82.4.60 TIXML_STRING* PlanesGenerator::outputFilename [protected]

4.82.4.61 TIXML_STRING* PlanesGenerator::outputFilename [protected]

4.82.4.62 TiXmlNode* PlanesGenerator::pointNode [protected]

4.82.4.63 TiXmlNode* PlanesGenerator::pointNode [protected]

4.82.4.64 TiXmlNode* PlanesGenerator::rootNode [protected]

4.82.4.65 TiXmlNode* PlanesGenerator::rootNode [protected]

4.82.4.66 LeavesInfo PlanesGenerator::sLeavesInfo [protected]

4.82.4.67 time_t PlanesGenerator::timer [protected]

4.82.4.68 TiXmlNode* PlanesGenerator::treeNode [protected]

4.82.4.69 TiXmlNode* PlanesGenerator::treeNode [protected]

4.82.4.70 Vector3 PlanesGenerator::vertex [protected]

4.82.4.71 vector<Vector3> PlanesGenerator::vLeavesPoint [protected]

4.82.4.72 vector<Vector3> PlanesGenerator::vLeavesPoint [protected]

4.82.4.73 Real PlanesGenerator::xValue [protected]

4.82.4.74 Real PlanesGenerator::yValue [protected]

4.82.4.75 Real PlanesGenerator::zValue [protected]

4.83 PlanesGenerator::lessCoord Struct Reference

Public Member Functions

- bool operator() (const Leaf l1, const Leaf l2) const
- bool operator() (const Leaf l1, const Leaf l2) const

4.83.1 Member Function Documentation

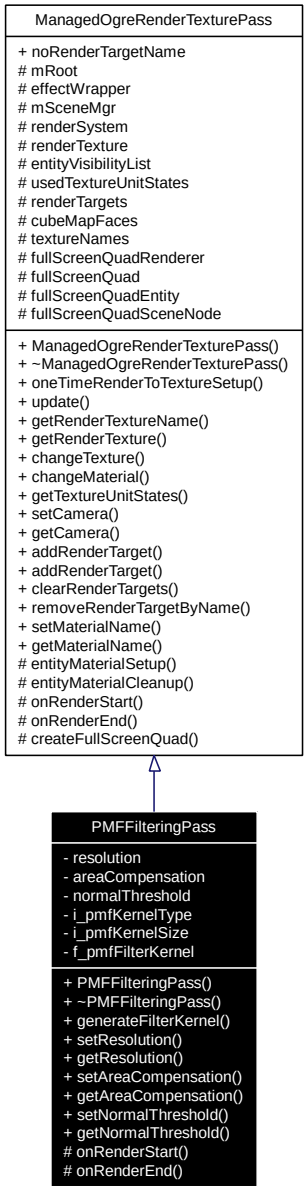
4.83.1.1 bool PlanesGenerator::lessCoord::operator() (const Leaf *l1*, const Leaf *l2*) const [inline]

4.83.1.2 bool PlanesGenerator::lessCoord::operator() (const Leaf *l1*, const Leaf *l2*) const [inline]

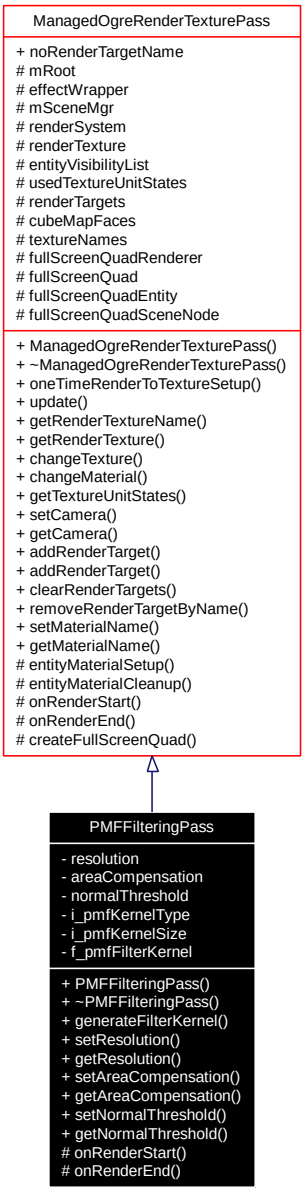
4.84 PMFFilteringPass Class Reference

Filters the incoming unfiltered photon map texture using an area map and a normal map.

Inheritance diagram for PMFFilteringPass:



Collaboration diagram for PMFFilteringPass:



Public Member Functions

- PMFFilteringPass (Root *mRoot, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_FLOAT16_RGBA, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false, String renderTextureName="FilteredPhotonMapTexture")

  : Construtor.

- ~PMFFilteringPass ()

  : Destructor.

- void generateFilterKernel (t_pmfKernelType kt, int kernelSize)

  Function to generate a filter kernel according to the given shape and size.

- void setResolution (float resolution)

  Sets the reciprocal value of the texture resolution.

- float getResolution ()

  Gets the reciprocal value of the resolution of the unfiltered photon map.

- void setAreaCompensation (float ac)

  Setter method to define area compensation.

- float getAreaCompensation ()

  Getter method to obtain area compensation.

- void setNormalThreshold (float nt)

  Setter method to define normal threshold.

- float getNormalThreshold ()

  Getter method to obtain normal threshold.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

  Pre rendering method.

- void onRenderEnd (NameValuePairList *namedParams=0)

  Post rendering method.

4.84.1 Detailed Description

Filters the incoming unfiltered photon map texture using an area map and a normal map.

The instances of this class are to generate a filtered photon map of the owner entity. The shader programs can access to the originally unfiltered photon map and the area map through the texture units of this textures. This association is done by the constructor of the PhotonMapFilteringRenderingRun class. The type of the resulting texture is PF_FLOAT16_RGBA.

4.84.2 Constructor & Destructor Documentation

4.84.2.1 PMFFilteringPass::PMFFilteringPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_FLOAT16_RGBA`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`, String *renderTextureName* = `"FilteredPhotonMapTexture"`)

: Construtor.

Parameters:

mRoot pointer to the Root object of the graphics engine.

width the width of render texture in pixels.

height the height of render texture in pixels.

texType the type of render texture. Its default value is TEX_TYPE_2D.

internalFormat the pixel format of the render texture. Its default value is PF_FLOAT16_RGBA.

miscParams a pointer to miscelaneous parameters. Its default value is 0.

fullScreenQuadRenderer a bool flag to decide on full screen rendering on a quad. Its default value is false.

renderTextureName a unique name for the render texture. Its default value is "FilteredPhotonMapTexture".

4.84.2.2 PMFFilteringPass::~PMFFilteringPass ()

: Destructor.

4.84.3 Member Function Documentation

4.84.3.1 void PMFFilteringPass::generateFilterKernel (t_pmfKernelType *kt*, int *kernelSize*)

Function to generate a filter kernel according to the given shape and size.

Parameters:

kt an enumerated type to specify the kernel shape.

kernelSize an integer to define the half of the kernel size in pixels.

4.84.3.2 float PMFFilteringPass::getAreaCompensation ()

Getter method to obtain area compensation.

Returns:

the multiplier of the area compensation.

4.84.3.3 float PMFFilteringPass::getNormalThreshold ()

Getter method to obtain normal threshold.

Returns:

the normal threshold.

4.84.3.4 float PMFFilteringPass::getResolution ()

Gets the reciprocal value of the resolution of the unfiltered photon map.

Returns:
the reciprocal value of the resolution of the unfiltered photon map.

4.84.3.5 void PMFFilteringPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Post rendering method.

It runs after the render-texture object is updated. Place all cleanup code here.

Parameters:
namedParams Parameter list.

Reimplemented from ManagedOgreRenderTexturePass.

4.84.3.6 void PMFFilteringPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Pre rendering method.

It runs before the render-texture object is updated. Place all shader setup here.

Parameters:
namedParams Parameter list.

Reimplemented from ManagedOgreRenderTexturePass.

4.84.3.7 void PMFFilteringPass::setAreaCompensation (float *ac*)

Setter method to define area compensation.

Parameters:
ac a multiplier of the area compensation.

4.84.3.8 void PMFFilteringPass::setNormalThreshold (float *nt*)

Setter method to define normal threshold.

Parameters:
nt a multiplier of the normal threshold.

4.84.3.9 void PMFFilteringPass::setResolution (float *resolution*)

Sets the reciprocal value of the texture resolution.

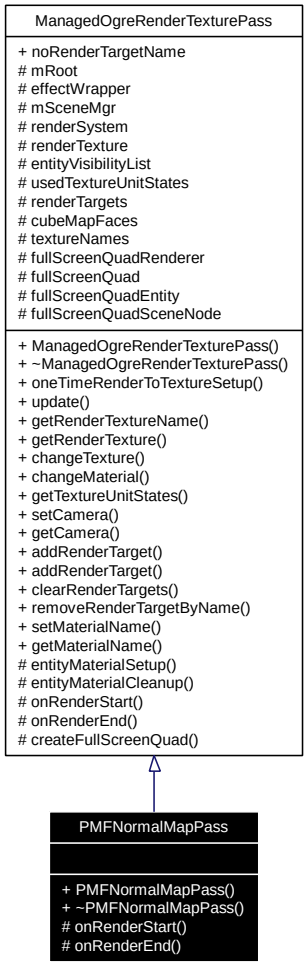
Parameters:

resolution a float to set the reciprocal value of the resolution of the unfiltered photon map.

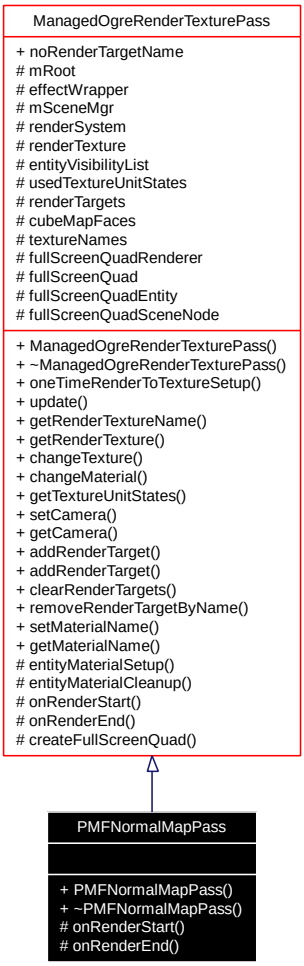
4.85 PMFNormalMapPass Class Reference

Generates a texture of the surface normals.

Inheritance diagram for PMFNormalMapPass:



Collaboration diagram for PMFNormalMapPass:



Public Member Functions

- PMFNormalMapPass (Root ∗mRoot, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_FLOAT32_RGBA, const NameValuePairList ∗miscParams=0, bool fullScreenQuadRenderer=false, String renderTextureName="NormapMapTexture")

  : Construtor.

- ∼PMFNormalMapPass ()

  : Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList ∗namedParams=0)

  Pre rendering method.

- void onRenderEnd (NameValuePairList ∗namedParams=0)

  Post rendering method.

4.85.1 Detailed Description

Generates a texture of the surface normals.

The instances of this class are to generate a texture of the surface normals of the owner entity. The resulting texture is a PF_FLOAT32_RGBA type texture, where the X, Y and Z coordinates of the surface normals will be stored in the R, G and B coordinates of the render texture of this class.

4.85.2 Constructor & Destructor Documentation

4.85.2.1 PMFNormalMapPass::PMFNormalMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_FLOAT32_RGBA`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`, String *renderTextureName* = `"NormapMapTexture"`)

: Construtor.

Parameters:

mRoot pointer to the Root object of the graphics engine.

width the width of render texture in pixels.

height the height of render texture in pixels.

texType the type of render texture. Its default value is TEX_TYPE_2D.

internalFormat the pixel format of the render texture. Its default value is PF_FLOAT32_RGBA.

miscParams a pointer to miscelaneous parameters. Its default value is 0.

fullScreenQuadRenderer a bool flag to decide on full screen rendering on a quad. Its default value is false.

renderTextureName a unique name for the render texture. Its default value is "NormapMapTexture".

4.85.2.2 PMFNormalMapPass::~PMFNormalMapPass ()

: Destructor.

4.85.3 Member Function Documentation

4.85.3.1 void PMFNormalMapPass::onRenderEnd (NameValuePairList * *namedParams* = `0`) `[protected, virtual]`

Post rendering method.

It runs after the render-texture object is updated. Place all cleanup code here.

Parameters:

namedParams Parameter list.

Reimplemented from ManagedOgreRenderTexturePass.

4.85.3.2 void PMFNormalMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Pre rendering method.

It runs before the render-texture object is updated. Place all shader setup here.

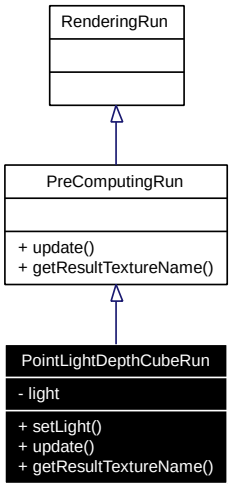
Parameters:

namedParams Parameter list.

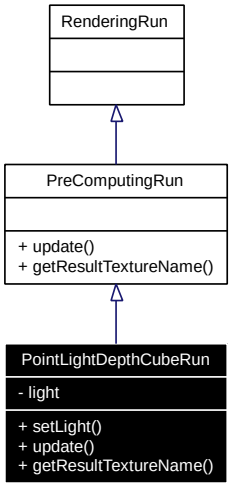
Reimplemented from ManagedOgreRenderTexturePass.

4.86 PointLightDepthCubeRun Class Reference

Inheritance diagram for PointLightDepthCubeRun:



Collaboration diagram for PointLightDepthCubeRun:



Public Member Functions

- void setLight (Light *light)
- virtual void update ()
- virtual const String & getResultTextureName ()

4.86.1 Detailed Description

Computes a depth cube for an omnidirectional point light

4.86.2 Member Function Documentation

4.86.2.1 virtual const String& PointLightDepthCubeRun::getResultTextureName () [inline, virtual]

Reimplemented from PreComputingRun.

4.86.2.2 void PointLightDepthCubeRun::setLight (Light * *light*) [inline]

Parameters:

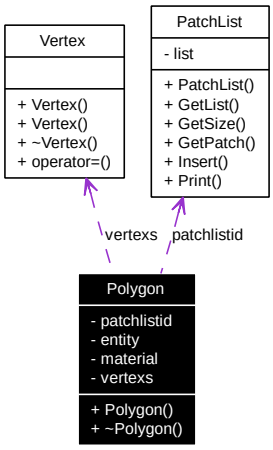
light The owner light of an light-bound precomputing run.

4.86.2.3 virtual void PointLightDepthCubeRun::update (void) [inline, virtual]

Implements PreComputingRun.

4.87 Polygon Class Reference

Collaboration diagram for Polygon:



Public Member Functions

- Polygon ()
- ~Polygon ()

4.87.1 Detailed Description

Polygon

4.87.2 Constructor & Destructor Documentation

4.87.2.1 Polygon::Polygon ()

4.87.2.2 Polygon::~Polygon ()

4.88 PolygonList Class Reference

Public Member Functions

- PolygonList ()
- std::vector< Polygon > * GetList ()
- int GetSize () const
- Polygon * GetPolygon (unsigned int i)
- bool Insert (Polygon *p)
- void Print ()

4.88.1 Detailed Description

PolygonList

4.88.2 Constructor & Destructor Documentation

4.88.2.1 PolygonList::PolygonList () [inline]

4.88.3 Member Function Documentation

4.88.3.1 std::vector<Polygon>* PolygonList::GetList () [inline]

Returns the pointer to the list of voxels

4.88.3.2 Polygon* PolygonList::GetPolygon (unsigned int *i*)

Returns pointer to patch(i)

4.88.3.3 int PolygonList::GetSize () const [inline]

Returns the number of elements of list

4.88.3.4 bool PolygonList::Insert (Polygon * *p*)

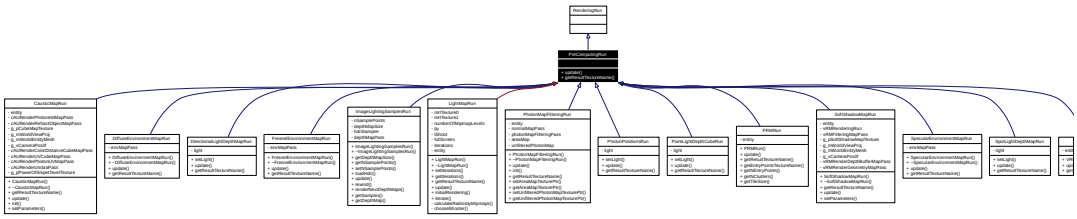
4.88.3.5 void PolygonList::Print ()

Prints stats

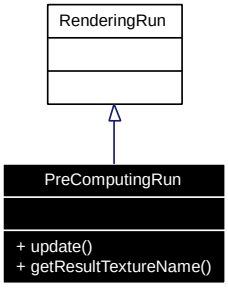
4.89 PreComputingRun Class Reference

Precomputing run superclass. Subclasses store and update precompution results.

Inheritance diagram for PreComputingRun:



Collaboration diagram for PreComputingRun:



Public Member Functions

- virtual void update ()=0
- virtual const String & getResultTextureName ()

  This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

4.89.1 Detailed Description

Precomputing run superclass. Subclasses store and update precompution results.

Entity-contained PreComputingRun subclasses should receive (and store) an 'Entity∗ owner' pointer as a constructor parameter. This provides an convenient way to directly access Ogre::Material data and textures, or results of previous PreComputingRuns.

4.89.2 Member Function Documentation

4.89.2.1 virtual const String& PreComputingRun::getResultTextureName () [inline, virtual]

This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Returns:
the main result texture's name

Reimplemented in CausticMapRun, DiffuseEnvironmentMapRun, DirectionalLightDepthMapRun, FresnelEnvironmentMapRun, LightMapRun, PhotonMapFilteringRun, PhotonPositionsRun, PointLightDepthCubeRun, PRMRun, SoftShadowMapRun, SpecularEnvironmentMapRun, SpotLightDepthMapRun, and VRMRun.

4.89.2.2 virtual void PreComputingRun::update () [pure virtual]

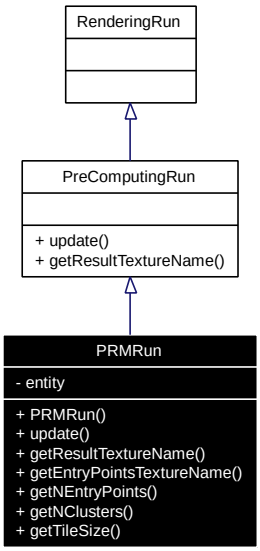
re-compute contained data

Implemented in CausticMapRun, DiffuseEnvironmentMapRun, DirectionalLightDepthMapRun, FresnelEnvironmentMapRun, ImageLightingSamplesRun, LightMapRun, PhotonMapFilteringRun, PhotonPositionsRun, PointLightDepthCubeRun, PRMRun, SoftShadowMapRun, SpecularEnvironmentMapRun, SpotLightDepthMapRun, and VRMRun.

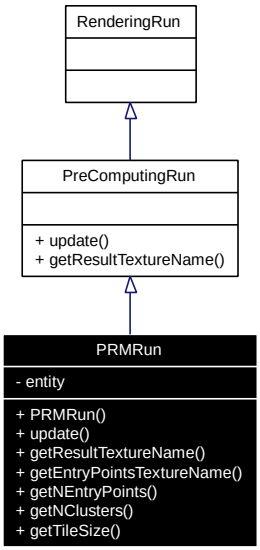
4.90 PRMRun Class Reference

Precomputing run that computes entry points and PRM.

Inheritance diagram for PRMRun:



Collaboration diagram for PRMRun:



Public Member Functions

- PRMRun (Entity *entity, unsigned int prmResolution, unsigned int prmNEntryPoints, unsigned int prmNClusters, unsigned int prmTileSize)

Contructor.

- void update ()
- const String & getResultTextureName ()
- const String & getEntryPointsTextureName ()
- unsigned int & getNEntryPoints ()
- unsigned int & getNClusters ()
- unsigned int & getTileSize ()

4.90.1 Detailed Description

Precomputing run that computes entry points and PRM.

4.90.2 Constructor & Destructor Documentation

4.90.2.1 PRMRun::PRMRun (Entity * *entity*, unsigned int *prmResolution*, unsigned int *prmNEntryPoints*, unsigned int *prmNClusters*, unsigned int *prmTileSize*) [inline]

Contructor.

Parameters:

entity The owner entity of an entity-bound precomputing run.

prmResolution PRM texture size.

prmNEntryPoints The number of entry points.

prmNClusters The number of entry point clusters.

prmTileSize The resolution of a PRM pane.

4.90.3 Member Function Documentation

4.90.3.1 const String& PRMRun::getEntryPointsTextureName ()

Returns:

Entry points' texture texture name.

4.90.3.2 unsigned int& PRMRun::getNClusters ()

Returns:

Number of entry point clusters.

4.90.3.3 unsigned int& PRMRun::getNEntryPoints ()

Returns:
Number of entry points.

4.90.3.4 const String& PRMRun::getResultTextureName () [virtual]

Returns:
PRM texture name.

Reimplemented from PreComputingRun.

4.90.3.5 unsigned int& PRMRun::getTileSize ()

Returns:
RPM tile size.

4.90.3.6 void PRMRun::update () [virtual]

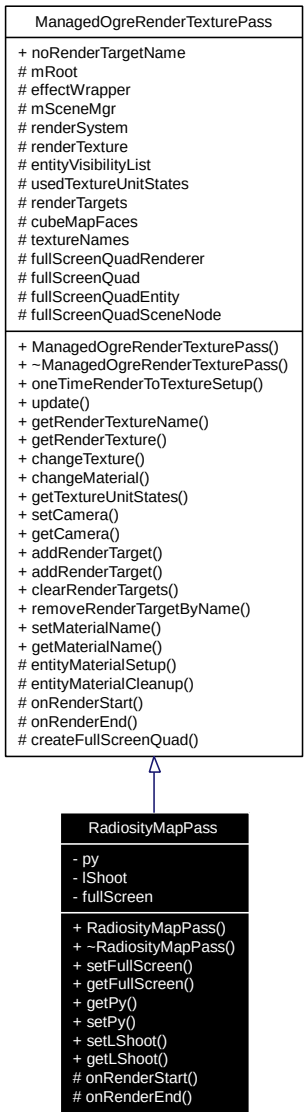
Inherited. Creates entry points. Create virtual light source bushes. Renders PRM.

Implements PreComputingRun.

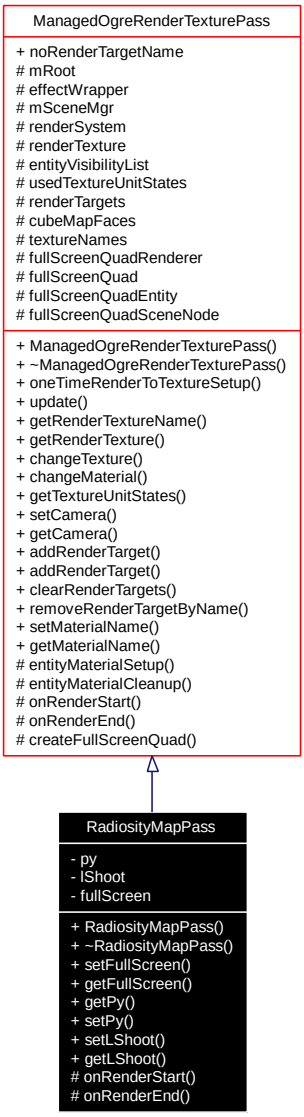
4.91 RadiosityMapPass Class Reference

Renders radiosity informations to an RGBA128F texture.

Inheritance diagram for RadiosityMapPass:



Collaboration diagram for RadiosityMapPass:



Public Member Functions

- RadiosityMapPass (Root *mRoot, unsigned int width, unsigned int height)

  Constructor.

- ~RadiosityMapPass ()
- void setFullScreen (bool fullScreen)

  Sets whether render to full or half screen. For hemicube shooting.

- bool getFullScreen ()

  Gets whether render to full or half screen. For hemicube shooting.

- float getPy ()

  Gets the shooter luminance per texture size.

- void setPy (float py)

  Sets the shooter luminance per texture size.

- void setLShoot (Vector4 lShoot)

  Sets the current shooter radiance. Alpha is the area of the uniform disc to shoot from.

- Vector4 getLShoot ()

  Gets the current shooter radiance. Alpha is the area of the uniform disc to shoot from.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

4.91.1 Detailed Description

Renders radiosity informations to an RGBA128F texture.

SuperClass: ManagedOgreRenderTexturePass Class: RadiosityMapPass

4.91.2 Constructor & Destructor Documentation

4.91.2.1 RadiosityMapPass::RadiosityMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor.

Parameters:

mRoot The current ogre root object.

width The width of the render texture instance

height The height of the render texture instance

4.91.2.2 RadiosityMapPass::~RadiosityMapPass ()

Destructor.

4.91.3 Member Function Documentation

4.91.3.1 bool RadiosityMapPass::getFullScreen () [inline]

Gets whether render to full or half screen. For hemicube shooting.

Returns:
Render to full screen if true. Render to half screen if false.

4.91.3.2 Vector4 RadiosityMapPass::getLShoot () [inline]

Gets the current shooter radiance. Alpha is the area of the uniform disc to shoot from.

Returns:
The radiance of the current shooter. Alpha is the area of the uniform disc to shoot from.

4.91.3.3 float RadiosityMapPass::getPy () [inline]

Gets the shooter luminance per texture size.

Returns:
The shooter luminance per texture size.

4.91.3.4 void RadiosityMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderEnd()

Reimplemented from ManagedOgreRenderTexturePass.

4.91.3.5 void RadiosityMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

See also:
ManagedOgreRenderTexturePass::onRenderStart()

Reimplemented from ManagedOgreRenderTexturePass.

4.91.3.6 void RadiosityMapPass::setFullScreen (bool *fullScreen*) [inline]

Sets whether render to full or half screen. For hemicube shooting.

Parameters:
fullScreen Render to full screen if true. Render to half screen if false.

4.91.3.7 void RadiosityMapPass::setLShoot (Vector4 *lShoot*) [inline]

Sets the current shooter radiance. Alpha is the area of the uniform disc to shoot from.

Parameters:

lShoot The radiance. Alpha is the area of the uniform disc to shoot from.

4.91.3.8 void RadiosityMapPass::setPy (float *py*) [inline]

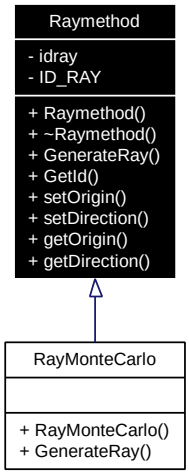
Sets the shooter luminance per texture size.

Parameters:

py The shooter luminance per texture size.

4.92 Raymethod Class Reference

Inheritance diagram for Raymethod:



Public Member Functions

- Raymethod ()
- ~Raymethod ()
- virtual Ray * GenerateRay ()
- unsigned int GetId () const
- void setOrigin (float x, float y, float z)
- void setDirection (float x, float y, float z)
- Vertex getOrigin ()
- Vertex getDirection ()

4.92.1 Detailed Description

Super class interface for generate a random ray to test in the scene

4.92.2 Constructor & Destructor Documentation

4.92.2.1 Raymethod::Raymethod ()

4.92.2.2 Raymethod::~Raymethod ()

4.92.3 Member Function Documentation

4.92.3.1 virtual Ray* Raymethod::GenerateRay () [virtual]

Reimplemented in RayMonteCarlo.

4.92.3.2 Vertex Raymethod::getDirection ()

4.92.3.3 unsigned int Raymethod::GetId () const [inline]

4.92.3.4 Vertex Raymethod::getOrigin ()

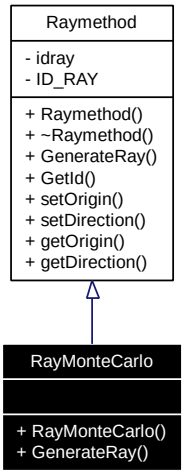
4.92.3.5 void Raymethod::setDirection (float *x*, float *y*, float *z*)

4.92.3.6 void Raymethod::setOrigin (float *x*, float *y*, float *z*)

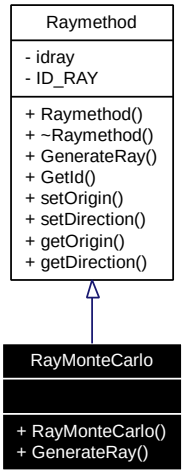
Functions to replace when we utilize another engine (now is ogre dependant)

4.93 RayMonteCarlo Class Reference

Inheritance diagram for RayMonteCarlo:



Collaboration diagram for RayMonteCarlo:



Public Member Functions

- RayMonteCarlo ()
- Ray ∗ GenerateRay ()

4.93.1 Detailed Description

class for generate a random ray with the montecarlo method

4.93.2 Constructor & Destructor Documentation

4.93.2.1 RayMonteCarlo::RayMonteCarlo ()

4.93.3 Member Function Documentation

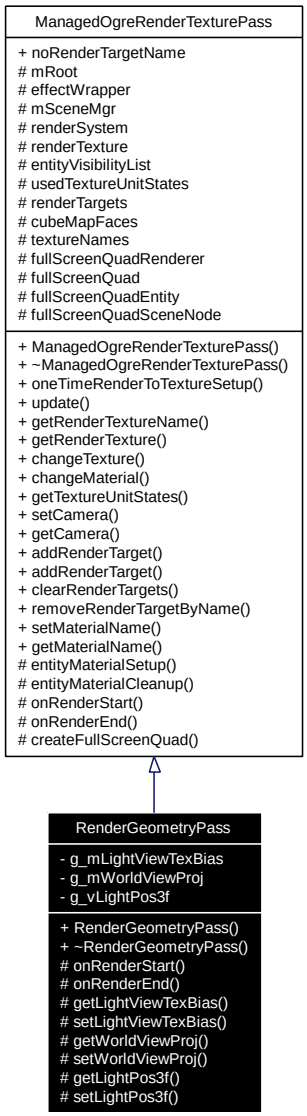
4.93.3.1 Ray* RayMonteCarlo::GenerateRay () [virtual]

Reimplemented from Raymethod.

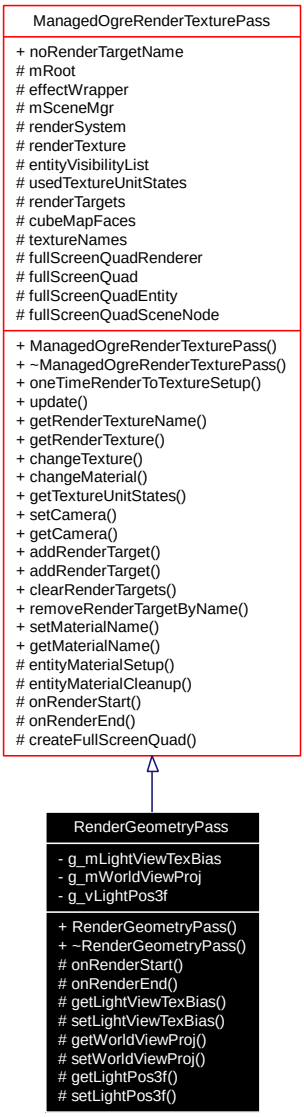
4.94 RenderGeometryPass Class Reference

Creates a texture to store geometry information from the entity.

Inheritance diagram for RenderGeometryPass:



Collaboration diagram for RenderGeometryPass:



Public Member Functions

- RenderGeometryPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~RenderGeometryPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)

- Matrix4 getLightViewTexBias ()
  Returns the value of g_mLightViewTexBias matrix.
- void setLightViewTexBias (Matrix4 matrix4)
  Sets the value of g_mLightViewTexBias matrix.
- Matrix4 getWorldViewProj ()
  Returns the value of g_mWorldViewProj matrix.
- void setWorldViewProj (Matrix4 matrix4)
  Sets the value of g_mWorldViewProj matrix.
- float * getLightPos3f ()
  Returns the value of g_vLightPos3f.
- void setLightPos3f (float *vector3)
  Sets the value of g_vLightPos3f.

4.94.1 Detailed Description

Creates a texture to store geometry information from the entity.

SuperClass: ManagedOgreRenderTexturePass Class: RenderGeometryPass The resulting texture is a PF_-FLOAT32_RGBA type texture

4.94.2 Constructor & Destructor Documentation

4.94.2.1 RenderGeometryPass::RenderGeometryPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.94.2.2 RenderGeometryPass::~RenderGeometryPass () [inline]

Destructor.

4.94.3 Member Function Documentation

4.94.3.1 float* RenderGeometryPass::getLightPos3f () [protected]

Returns the value of g_vLightPos3f.

4.94.3.2 Matrix4 RenderGeometryPass::getLightViewTexBias () [protected]

Returns the value of g_mLightViewTexBias matrix.

4.94.3.3 Matrix4 RenderGeometryPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.94.3.4 void RenderGeometryPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.94.3.5 void RenderGeometryPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.94.3.6 void RenderGeometryPass::setLightPos3f (float * *vector3*) [protected]

Sets the value of g_vLightPos3f.

Parameters:
vector3 Contains the new value of g_vLightPos3f.

4.94.3.7 void RenderGeometryPass::setLightViewTexBias (Matrix4 *matrix4*) [protected]

Sets the value of g_mLightViewTexBias matrix.

Parameters:
matrix4 Contains the new value of g_mLightViewTexBias matrix.

4.94.3.8 void RenderGeometryPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

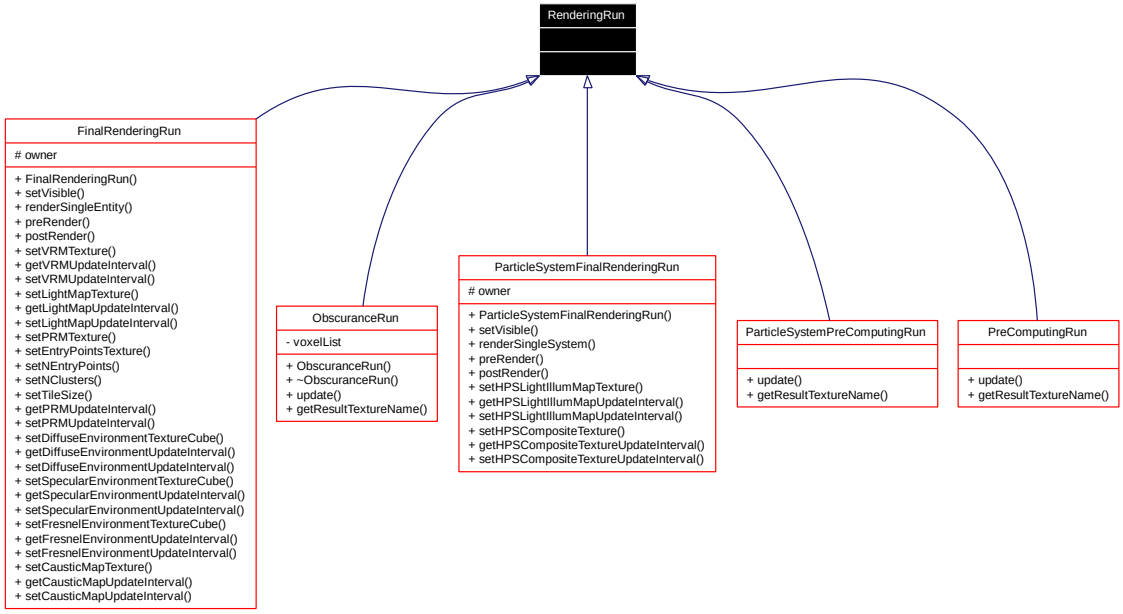
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

4.95 RenderingRun Class Reference

Base class for computation modules of the illumination workpackage.

Inheritance diagram for RenderingRun:



4.95.1 Detailed Description

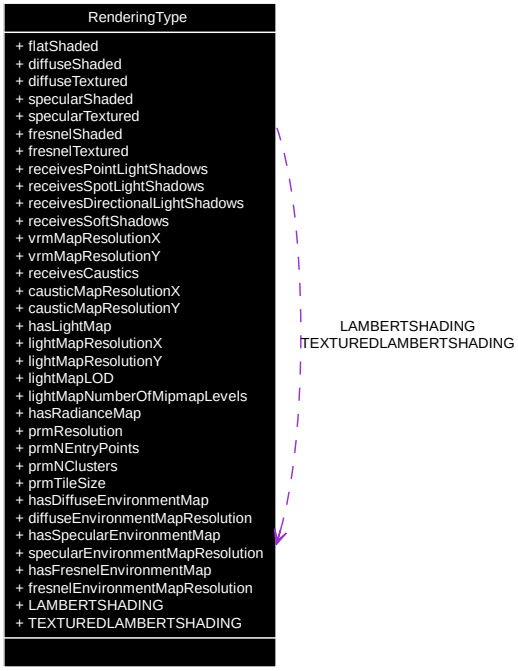
Base class for computation modules of the illumination workpackage.

A run typically - but not necessarily oe exclusively - consists of a series of rendering passes derived from ManagedOgreRenderTexturePass. Instances of RenderingRun-derived objects are typically stored with entities, or globally in the IlluminationModule class. A RenderingRun stores the result of the computation (in a PreComputingRun subclass), or renders to the frame buffer (in a FinalRun subclass). Intermediate computation data should not be stored with the instances. When allocating and freeing the resources associated with temporary data are costly (typically rendertextures), then they should be declared static and shared between the instances. Thus, a typical RenderingRun class will contain a number of static ManagedOgreRenderTexturePass instances and a few ManagedOgreRenderTexturePass members containing persistent computation results.

4.96 RenderingType Class Reference

A class capable of describing all implemented illumination modes that could be applied when rendering an entity.

Collaboration diagram for RenderingType:



Public Attributes

- bool flatShaded
- bool diffuseShaded
- bool diffuseTextured
- bool specularShaded
- bool specularTextured
- bool fresnelShaded
- bool fresnelTextured
- bool receivesPointLightShadows
- bool receivesSpotLightShadows
- bool receivesDirectionalLightShadows
- bool receivesSoftShadows
- unsigned int vrmMapResolutionX
- unsigned int vrmMapResolutionY
- bool receivesCaustics
- unsigned int causticMapResolutionX
- unsigned int causticMapResolutionY
- bool hasLightMap

- unsigned int lightMapResolutionX
- unsigned int lightMapResolutionY
- unsigned int lightMapLOD
- unsigned char lightMapNumberOfMipmapLevels
- bool hasRadianceMap
- unsigned int prmResolution
- unsigned int prmNEntryPoints
- unsigned int prmNClusters
- unsigned int prmTileSize
- bool hasDiffuseEnvironmentMap
- unsigned int diffuseEnvironmentMapResolution
- bool hasSpecularEnvironmentMap
- unsigned int specularEnvironmentMapResolution
- bool hasFresnelEnvironmentMap
- unsigned int fresnelEnvironmentMapResolution

Static Public Attributes

- static const RenderingType LAMBERTSHADING
- static const RenderingType TEXTUREDLAMBERTSHADING

4.96.1 Detailed Description

A class capable of describing all implemented illumination modes that could be applied when rendering an entity.

EntityRenderingObject::EntityRenderingObject will take a RenderingType as a parameter, and instantiate an appropriate FinalRenderingRun class.

4.96.2 Member Data Documentation

4.96.2.1 unsigned int RenderingType::causticMapResolutionX

4.96.2.2 unsigned int RenderingType::causticMapResolutionY

4.96.2.3 unsigned int RenderingType::diffuseEnvironmentMapResolution

4.96.2.4 bool RenderingType::diffuseShaded

4.96.2.5 bool RenderingType::diffuseTextured

4.96.2.6 bool RenderingType::flatShaded

4.96.2.7 unsigned int RenderingType::fresnelEnvironmentMapResolution

4.96.2.8 bool RenderingType::fresnelShaded

4.96.2.9 bool RenderingType::fresnelTextured

4.96.2.10 bool RenderingType::hasDiffuseEnvironmentMap

4.96.2.11 bool RenderingType::hasFresnelEnvironmentMap

4.96.2.12 bool RenderingType::hasLightMap

4.96.2.13 bool RenderingType::hasRadianceMap

4.96.2.14 bool RenderingType::hasSpecularEnvironmentMap

4.96.2.15 const RenderingType RenderingType::LAMBERTSHADING [static]

Basic Lambert shading of diffuse materials, without shadows or any other effects.

4.96.2.16 unsigned int RenderingType::lightMapLOD

4.96.2.17 unsigned char RenderingType::lightMapNumberOfMipmapLevels

4.96.2.18 unsigned int RenderingType::lightMapResolutionX

4.96.2.19 unsigned int RenderingType::lightMapResolutionY

4.96.2.20 unsigned int RenderingType::prmNClusters

4.96.2.21 unsigned int RenderingType::prmNEntryPoints

4.96.2.22 unsigned int RenderingType::prmResolution

4.96.2.23 unsigned int RenderingType::prmTileSize

4.96.2.24 bool RenderingType::receivesCaustics

4.96.2.25 bool RenderingType::receivesDirectionalLightShadows

4.96.2.26 bool RenderingType::receivesPointLightShadows

4.96.2.27 bool RenderingType::receivesSoftShadows

4.96.2.28 bool RenderingType::receivesSpotLightShadows

4.96.2.29 unsigned int RenderingType::specularEnvironmentMapResolution

4.96.2.30 bool RenderingType::specularShaded

4.96.2.31 bool RenderingType::specularTextured

4.96.2.32 const RenderingType RenderingType::TEXTUREDLAMBERTSHADING [static]

Basic Lambert shading using the primary Material texture as a diffuse BRDF map.

4.96.2.33 unsigned int RenderingType::vrmMapResolutionX

4.96.2.34 unsigned int RenderingType::vrmMapResolutionY

4.97 RenderTexture Class Reference

Public Types

- enum UpdateMode { RT_RENDER_TO_TEXTURE, RT_COPY_TO_TEXTURE }

Public Member Functions

- RenderTexture (const char *strMode="rgb tex2D")
- ~RenderTexture ()
- bool Initialize (int width, int height, bool shareObjects=true, bool copyContext=false)
- bool Reset (const char *strMode,...)
- bool Resize (int width, int height)
- bool BeginCapture ()
- bool BeginCapture (RenderTexture *current)
- bool EndCapture ()
- void Bind () const
- void BindDepth () const
- bool BindBuffer (int iBuffer, int textureIDIndex=0)
- void EnableTextureTarget () const

  Enables the texture target appropriate for this render texture.

- void DisableTextureTarget () const

  Disables the texture target appropriate for this render texture.

- unsigned int GetTextureID (int textureIDIndex=0) const

  Returns the texture ID. Useful in Cg applications.

- unsigned int GetDepthTextureID () const

  Returns the depth texture ID. Useful in Cg applications.

- unsigned int GetTextureTarget () const

  Returns the texture target this texture is bound to.

- operator unsigned int () const

  Conversion operator allows RenderTexture to be passed to GL calls.

- int GetWidth () const

  Returns the width of the offscreen buffer.

- int GetHeight () const

  Returns the width of the offscreen buffer.

- int GetMaxS () const

  Returns the maximum S texture coordinate.

- int GetMaxT () const

  Returns the maximum T texture coordinate.

- int GetRedBits () const

  Returns the number of red bits allocated.

- int GetGreenBits () const

  Returns the number of green bits allocated.

- int GetBlueBits () const

  Returns the number of blue bits allocated.

- int GetAlphaBits () const

  Returns the number of alpha bits allocated.

- int GetDepthBits () const

  Returns the number of depth bits allocated.

- int GetStencilBits () const

  Returns the number of stencil bits allocated.

- bool IsInitialized () const

  True if this RenderTexture has been properly initialized.

- bool IsTexture () const

  True if this is a texture and not just an offscreen buffer.

- bool IsDepthTexture () const

  True if this is a depth texture and not just an offscreen buffer.

- bool IsFloatTexture () const

  True if this is a floating point buffer / texture.

- bool IsDoubleBuffered () const

  True if this is a double-buffered pbuffer.

- bool IsRectangleTexture () const

  True if this texture has non-power-of-two dimensions.

- bool HasDepth () const
- bool HasStencil () const

  True if this pbuffer has a stencil buffer.

- bool IsMipmapped () const

  True if this texture has mipmaps.

- RenderTexture (int width, int height, bool bIsTexture=true, bool bIsDepthTexture=false)
- bool Initialize (bool bShare=true, bool bDepth=false, bool bStencil=false, bool bMipmap=false, bool bAnisoFilter=false, unsigned int iRBits=8, unsigned int iGBits=8, unsigned int iBBits=8, unsigned int iABits=8, UpdateMode updateMode=RT_COPY_TO_TEXTURE)
- bool Reset (int iWidth, int iHeight)

Static Public Member Functions

- static bool IsPowerOfTwo (int n)

  Returns true if /param n is an integer power of 2.

Protected Types

- typedef std::pair< std::string, std::string > KeyVal

Protected Member Functions

- bool _Invalidate ()
- void _ParseModeString (const char *modeString, std::vector< int > &pixelFormatAttribs, std::vector< int > &pbufferAttribs)
- std::vector< int > _ParseBitVector (std::string bitVector)
- KeyVal _GetKeyValuePair (std::string token)
- bool _VerifyExtensions ()
- bool _InitializeTextures ()
- void _MaybeCopyBuffer ()
- bool _ReleaseBoundBuffers ()
- bool _MakeCurrent ()
- bool _BindDepthBuffer () const

Protected Attributes

- int _iWidth
- int _iHeight
- bool _bIsTexture
- bool _bIsDepthTexture
- bool _bHasARBDepthTexture
- UpdateMode _eUpdateMode
- bool _bInitialized
- unsigned int _iNumAuxBuffers
- bool * _bIsBufferBound
- int * _iCurrentBoundBuffer
- unsigned int _iNumComponents
- unsigned int _iNumColorBits [4]
- unsigned int _iNumDepthBits
- unsigned int _iNumStencilBits
- bool FORCEATI
- bool _bFloat
- bool _bDoubleBuffered
- bool _bPowerOf2
- bool _bRectangle
- bool _bMipmap
- bool _bShareObjects
- bool _bCopyContext
- Display * _pDisplay

- GLXContext _hGLContext
- GLXPbuffer _hPBuffer
- GLXDrawable _hPreviousDrawable
- GLXContext _hPreviousContext
- GLenum _iTextureTarget
- unsigned int * _iTextureID
- unsigned int _numberOfTextureID
- unsigned int _iDepthTextureID
- unsigned short * _pPoorDepthTexture
- std::vector< int > _pixelFormatAttribs
- std::vector< int > _pbufferAttribs

4.97.1 Member Typedef Documentation

4.97.1.1 typedef std::pair<std::string, std::string> RenderTexture::KeyVal [protected]

4.97.2 Member Enumeration Documentation

4.97.2.1 enum RenderTexture::UpdateMode

Enumeration values:

RT_RENDER_TO_TEXTURE

RT_COPY_TO_TEXTURE

4.97.3 Constructor & Destructor Documentation

4.97.3.1 RenderTexture::RenderTexture (const char * *strMode* = "rgb tex2D")

4.97.3.2 RenderTexture::~RenderTexture ()

4.97.3.3 RenderTexture::RenderTexture (int *width*, int *height*, bool *bIsTexture* = true, bool *bIsDepthTexture* = false)

4.97.4 Member Function Documentation

4.97.4.1 bool RenderTexture::_BindDepthBuffer () const [protected]

4.97.4.2 KeyVal RenderTexture::_GetKeyValuePair (std::string *token*) [protected]

4.97.4.3 bool RenderTexture::_InitializeTextures () [protected]

4.97.4.4 bool RenderTexture::_Invalidate () [protected]

4.97.4.5 bool RenderTexture::_MakeCurrent () [protected]

4.97.4.6 void RenderTexture::_MaybeCopyBuffer () [protected]

4.97.4.7 std::vector<int> RenderTexture::_ParseBitVector (std::string *bitVector*) [protected]

4.97.4.8 void RenderTexture::_ParseModeString (const char * *modeString*, std::vector< int > & *pixelFormatAttribs*, std::vector< int > & *pbufferAttribs*) [protected]

4.97.4.9 bool RenderTexture::_ReleaseBoundBuffers () [protected]

4.97.4.10 bool RenderTexture::_VerifyExtensions () [protected]

4.97.4.11 bool RenderTexture::BeginCapture (RenderTexture * *current*)

4.97.4.12 bool RenderTexture::BeginCapture ()

4.97.4.13 void RenderTexture::Bind () const

4.97.4.14 bool RenderTexture::BindBuffer (int *iBuffer*, int *textureIDIndex* = 0)

4.97.4.15 void RenderTexture::BindDepth () const

4.97.4.16 void RenderTexture::DisableTextureTarget () const [inline]

Disables the texture target appropriate for this render texture.

4.97.4.17 void RenderTexture::EnableTextureTarget () const [inline]

Enables the texture target appropriate for this render texture.

4.97.4.18 bool RenderTexture::EndCapture ()

4.97.4.19 int RenderTexture::GetAlphaBits () const [inline]

Returns the number of alpha bits allocated.

4.97.4.20 int RenderTexture::GetBlueBits () const [inline]

Returns the number of blue bits allocated.

4.97.4.21 int RenderTexture::GetDepthBits () const [inline]

Returns the number of depth bits allocated.

4.97.4.22 unsigned int RenderTexture::GetDepthTextureID () const [inline]

Returns the depth texture ID. Useful in Cg applications.

4.97.4.23 int RenderTexture::GetGreenBits () const [inline]

Returns the number of green bits allocated.

4.97.4.24 int RenderTexture::GetHeight () const [inline]

Returns the width of the offscreen buffer.

4.97.4.25 int RenderTexture::GetMaxS () const [inline]

Returns the maximum S texture coordinate.

4.97.4.26 int RenderTexture::GetMaxT () const [inline]

Returns the maximum T texture coordinate.

4.97.4.27 int RenderTexture::GetRedBits () const [inline]

Returns the number of red bits allocated.

4.97.4.28 int RenderTexture::GetStencilBits () const [inline]

Returns the number of stencil bits allocated.

4.97.4.29 unsigned int RenderTexture::GetTextureID (int *textureIDIndex* = 0) const [inline]

Returns the texture ID. Useful in Cg applications.

4.97.4.30 unsigned int RenderTexture::GetTextureTarget () const [inline]

Returns the texture target this texture is bound to.

4.97.4.31 int RenderTexture::GetWidth () const [inline]

Returns the width of the offscreen buffer.

4.97.4.32 bool RenderTexture::HasDepth () const [inline]

True if this texture has non-power-of-two dimensions. True if this pbuffer has a depth buffer.

4.97.4.33 bool RenderTexture::HasStencil () const [inline]

True if this pbuffer has a stencil buffer.

4.97.4.34 bool RenderTexture::Initialize (bool *bShare* = true, bool *bDepth* = false, bool *bStencil* = false, bool *bMipmap* = false, bool *bAnisoFilter* = false, unsigned int *iRBits* = 8, unsigned int *iGBits* = 8, unsigned int *iBBits* = 8, unsigned int *iABits* = 8, UpdateMode *updateMode* = RT_COPY_TO_TEXTURE)

4.97.4.35 bool RenderTexture::Initialize (int *width*, int *height*, bool *shareObjects* = true, bool *copyContext* = false)

Call this once before use. Set bShare to true to share lists, textures, and program objects between the render texture context and the current active GL context.

4.97.4.36 bool RenderTexture::IsDepthTexture () const [inline]

True if this is a depth texture and not just an offscreen buffer.

4.97.4.37 bool RenderTexture::IsDoubleBuffered () const [inline]

True if this is a double-buffered pbuffer.

4.97.4.38 bool RenderTexture::IsFloatTexture () const [inline]

True if this is a floating point buffer / texture.

4.97.4.39 bool RenderTexture::IsInitialized () const [inline]

True if this RenderTexture has been properly initialized.

4.97.4.40 bool RenderTexture::IsMipmapped () const [inline]

True if this texture has mipmaps.

4.97.4.41 RenderTexture::IsPowerOfTwo (int *n*) [inline, static]

Returns true if /param n is an integer power of 2.

Taken from Steve Baker's Cute Code Collection. http://www.sjbaker.org/steve/software/cute_code.html

4.97.4.42 bool RenderTexture::IsRectangleTexture () const [inline]

True if this texture has non-power-of-two dimensions.

4.97.4.43 bool RenderTexture::IsTexture () const [inline]

True if this is a texture and not just an offscreen buffer.

4.97.4.44 RenderTexture::operator unsigned int () const [inline]

Conversion operator allows RenderTexture to be passed to GL calls.

4.97.4.45 bool RenderTexture::Reset (int *iWidth*, int *iHeight*)

4.97.4.46 bool RenderTexture::Reset (const char * *strMode*, ...)

4.97.4.47 bool RenderTexture::Resize (int *width*, int *height*)

4.97.5 Member Data Documentation

4.97.5.1 bool RenderTexture::_bCopyContext [protected]

4.97.5.2 bool RenderTexture::_bDoubleBuffered [protected]

4.97.5.3 bool RenderTexture::_bFloat [protected]

4.97.5.4 bool RenderTexture::_bHasARBDepthTexture [protected]

4.97.5.5 bool RenderTexture::_bInitialized [protected]

4.97.5.6 bool* RenderTexture::_bIsBufferBound [protected]

4.97.5.7 bool RenderTexture::_bIsDepthTexture [protected]

4.97.5.8 bool RenderTexture::_bIsTexture [protected]

4.97.5.9 bool RenderTexture::_bMipmap [protected]

4.97.5.10 bool RenderTexture::_bPowerOf2 [protected]

4.97.5.11 bool RenderTexture::_bRectangle [protected]

4.97.5.12 bool RenderTexture::_bShareObjects [protected]

4.97.5.13 UpdateMode RenderTexture::_eUpdateMode [protected]

4.97.5.14 GLXContext RenderTexture::_hGLContext [protected]

4.97.5.15 GLXPbuffer RenderTexture::_hPBuffer [protected]

4.97.5.16 GLXContext RenderTexture::_hPreviousContext [protected]

4.97.5.17 GLXDrawable RenderTexture::_hPreviousDrawable [protected]

4.97.5.18 int* RenderTexture::_iCurrentBoundBuffer [protected]

4.97.5.19 unsigned int RenderTexture::_iDepthTextureID [protected]

4.97.5.20 int RenderTexture::_iHeight [protected]

4.97.5.21 unsigned int RenderTexture::_iNumAuxBuffers [protected]

4.97.5.22 unsigned int RenderTexture::_iNumColorBits[4] [protected]

4.97.5.23 unsigned int RenderTexture::_iNumComponents [protected]

4.97.5.24 unsigned int RenderTexture::_iNumDepthBits [protected]

4.97.5.25 unsigned int RenderTexture::_iNumStencilBits [protected]

4.97.5.26 unsigned int* RenderTexture::_iTextureID [protected]

4.97.5.27 GLenum RenderTexture::_iTextureTarget [protected]

4.97.5.28 int RenderTexture::_iWidth [protected]

4.97.5.29 unsigned int RenderTexture::_numberOfTextureID [protected]

4.97.5.30 std::vector<int> RenderTexture::_pbufferAttribs [protected]

4.97.5.31 Display* RenderTexture::_pDisplay [protected]

4.97.5.32 std::vector<int> RenderTexture::_pixelFormatAttribs [protected]

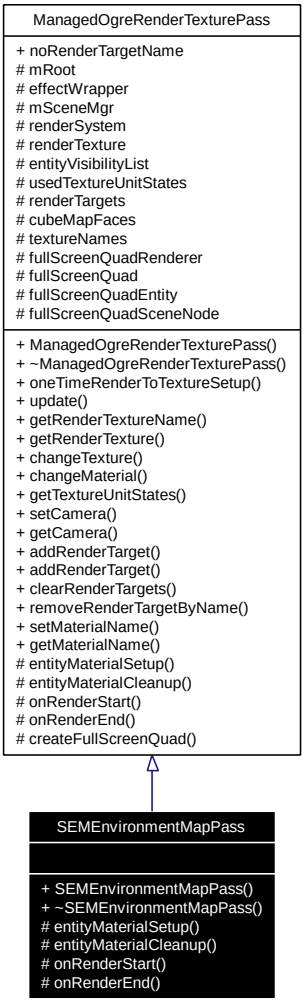
4.97.5.33 unsigned short* RenderTexture::_pPoorDepthTexture [protected]

4.97.5.34 bool RenderTexture::FORCEATI [protected]

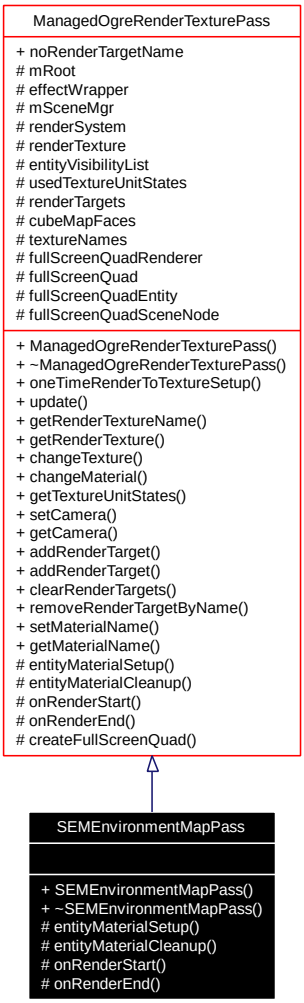
4.98 SEMEnvironmentMapPass Class Reference

Performs the actual pre-processing steps for the Environment Mapping effect.

Inheritance diagram for SEMEnvironmentMapPass:



Collaboration diagram for SEMEnvironmentMapPass:



Public Member Functions

- SEMEnvironmentMapPass (Root *mRoot, unsigned int width, unsigned int height)
- ~SEMEnvironmentMapPass (void)

Protected Member Functions

- virtual void entityMaterialSetup ()
- virtual void entityMaterialCleanup ()
- virtual void onRenderStart (NameValuePairList *namedParams=0)
- virtual void onRenderEnd (NameValuePairList *namedParams=0)

4.98.1 Detailed Description

Performs the actual pre-processing steps for the Environment Mapping effect.

SuperClass ManagedOgreRenderTexturePass

Class SEMEnvironmentMapPass

4.98.2 Constructor & Destructor Documentation

4.98.2.1 SEMEnvironmentMapPass::SEMEnvironmentMapPass (Root * *mRoot*, unsigned int *width*, unsigned int *height*)

Constructor

Parameters:

mRoot Pointer to the Ogre Root object

width The width of the environment cube-map

height The height of the environment cube-map

Remarks:

The width and height parameters must be equal and the power of 2.

4.98.2.2 SEMEnvironmentMapPass::~SEMEnvironmentMapPass (void)

Destructor

4.98.3 Member Function Documentation

4.98.3.1 virtual void SEMEnvironmentMapPass::entityMaterialCleanup () [protected, virtual]

Cleans up the material of the rendered entity. Can be overriden, if different functionality is desired.

Reimplemented from ManagedOgreRenderTexturePass.

4.98.3.2 virtual void SEMEnvironmentMapPass::entityMaterialSetup () [protected, virtual]

Sets up the material of the rendered entity. Can be overriden, if different functionality is desired.

Reimplemented from ManagedOgreRenderTexturePass.

4.98.3.3 virtual void SEMEnvironmentMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:

namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.98.3.4 virtual void SEMEnvironmentMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:

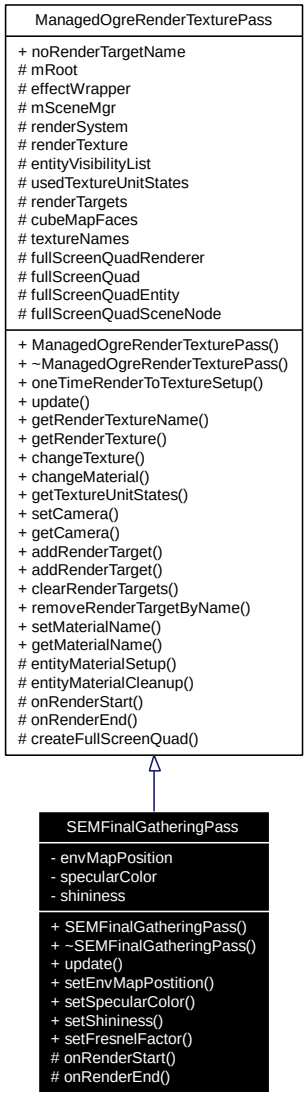
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

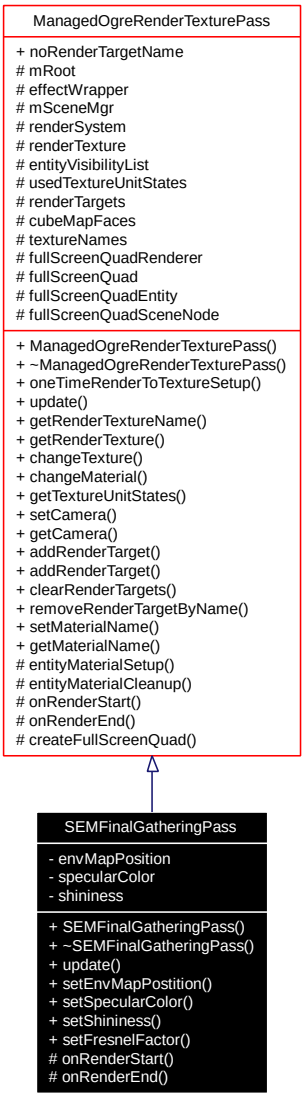
4.99 SEMFinalGatheringPass Class Reference

Performs the actual rendering of the Environment Mapping Effect.

Inheritance diagram for SEMFinalGatheringPass:



Collaboration diagram for SEMFinalGatheringPass:



Public Member Functions

- SEMFinalGatheringPass (Root ∗mRoot)
- ∼SEMFinalGatheringPass (void)
- void update (void)
- void setEnvMapPostition (Vector3 envMapPosition)
- void setSpecularColor (Vector3 specularColor)
- void setShininess (float shininess)
- void setFresnelFactor (float fresnelFactor)

Protected Member Functions

- virtual void onRenderStart (NameValuePairList ∗namedParams=0)
- virtual void onRenderEnd (NameValuePairList ∗namedParams=0)

4.99.1 Detailed Description

Performs the actual rendering of the Environment Mapping Effect.

SuperClass ManagedOgreRenderTexturePass

Class SEMFinalGatheringPass

4.99.2 Constructor & Destructor Documentation

4.99.2.1 SEMFinalGatheringPass::SEMFinalGatheringPass (Root * *mRoot*)

Constructor

Parameters:
mRoot The Ogre Root object

4.99.2.2 SEMFinalGatheringPass::~SEMFinalGatheringPass (void)

Destructor

4.99.3 Member Function Documentation

4.99.3.1 virtual void SEMFinalGatheringPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.99.3.2 virtual void SEMFinalGatheringPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.99.3.3 void SEMFinalGatheringPass::setEnvMapPostition (Vector3 *envMapPosition*)

Sets the world-space position of the environment cube-map for the EnvMap shader. This is used, because the cubemap is not regenerated in every frame.

Parameters:

envMapPosition The position vector

4.99.3.4 void SEMFinalGatheringPass::setFresnelFactor (float *fresnelFactor*)

Sets the material's Fresnel factor for the EnvMap shader.

Parameters:

fresnelFactor The Fresnel factor. 0.0f means only a small reflection in narrow angles, 1.0f complete reflection to every direction.

4.99.3.5 void SEMFinalGatheringPass::setShininess (float *shininess*)

4.99.3.6 void SEMFinalGatheringPass::setSpecularColor (Vector3 *specularColor*)

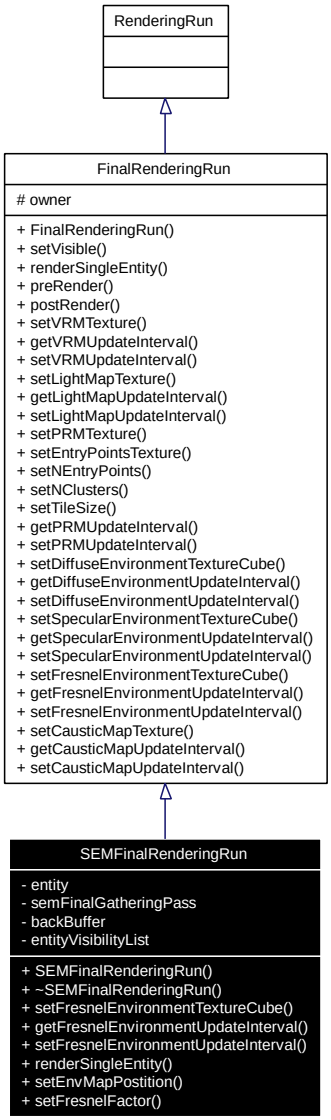
4.99.3.7 void SEMFinalGatheringPass::update (void)

Performs the rendering

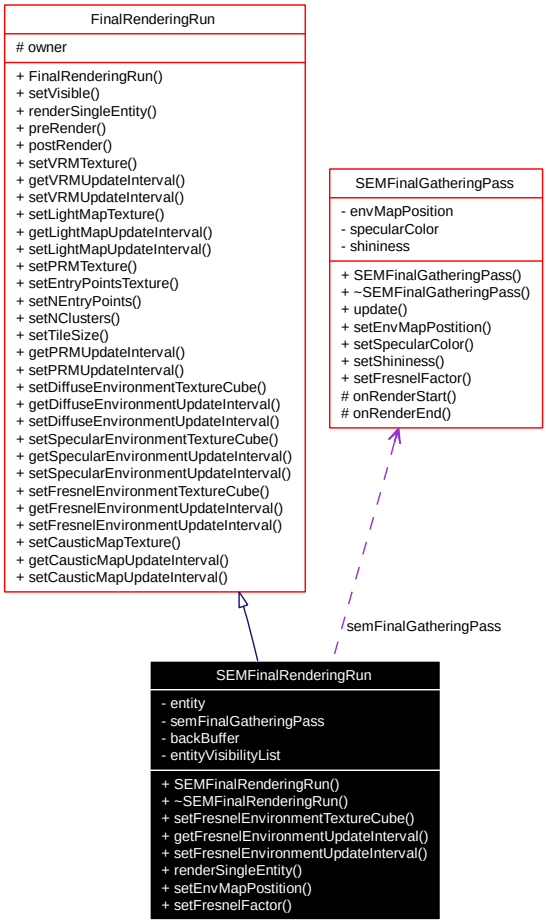
4.100 SEMFinalRenderingRun Class Reference

Controlls the rendering of the Environment Mapping effect.

Inheritance diagram for SEMFinalRenderingRun:



Collaboration diagram for SEMFinalRenderingRun:



Public Member Functions

- SEMFinalRenderingRun (Entity *ent)
- ~SEMFinalRenderingRun (void)
- virtual void setFresnelEnvironmentTextureCube (const String &fresnelEnvironmentTextureCube-Name)

  Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

- virtual unsigned int getFresnelEnvironmentUpdateInterval ()
- virtual void setFresnelEnvironmentUpdateInterval (unsigned int updateIntervalNumOfFrames)

  Set the SEM udate interval desired for the owner entity. If SEM is not used, the method should have no effect.

- virtual void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_-FACE_POSITIVE_X)

  Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

- void setEnvMapPostition (Vector3 envMapPosition)

- void setFresnelFactor (float fresnelFactor)

4.100.1 Detailed Description

Controlls the rendering of the Environment Mapping effect.

SuperClass FinalRenderingRun

Class SEMFinalRenderingRun

4.100.2 Constructor & Destructor Documentation

4.100.2.1 SEMFinalRenderingRun::SEMFinalRenderingRun (Entity * *ent*)

Constructor

Parameters:
ent Owner entity.

4.100.2.2 SEMFinalRenderingRun::~SEMFinalRenderingRun (void)

Destructor

4.100.3 Member Function Documentation

4.100.3.1 virtual unsigned int SEMFinalRenderingRun::getFresnelEnvironmentUpdateInterval () [inline, virtual]

Returns:
0 if Fresnel Enviroment Map is not used, the desired length of the update interval otherwise.

Reimplemented from FinalRenderingRun.

4.100.3.2 virtual void SEMFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Perform the passes necessary to render the entity to the frame buffer, with all the illumination effects the implementing FinalRenderingRun-subclass supports. This method is called by IlluminationModule::update, after all the necessary preprocessing steps have been executed. Thus, the references (or names) that had been set via the virtual set<anything> functions reference the updated results.

This method is supposed to reproduce the behaviour of rendering an object using the standard OGRE pipeline. Thus, it is forbidden to commit any of the following:

- clear the color, depth or stencil of the backbuffer
- alter the depth testing, stencil testing, alpha blending render state without restoring it
- render with altered depth testing, stencil testing, alpha blending to the backbuffer
- alter entity or billboard visibilities without restoring them

Parameters:

backBuffer The render target to be rendered to. While this is typically the frame buffer, 'final' rendering can be performed for a texture output, e.g. when rendering an environment map.

cf Meaningful if the render target is a cube map. Identifies the face to be rendered to.

Implements FinalRenderingRun.

4.100.3.3 void SEMFinalRenderingRun::setEnvMapPostition (Vector3 *envMapPosition*)

Sets the world-space position of the environment cube-map for the EnvMap shader. This is used, because the cubemap is not regenerated in every frame.

Parameters:

envMapPosition The position vector

4.100.3.4 virtual void SEMFinalRenderingRun::setFresnelEnvironmentTextureCube (const String & *fresnelEnvironmentTextureCubeName*) [virtual]

Set the entity's Fresnel Environment Map. Resources possibly re-computed later must be passed by reference or name.

Parameters:

fresnelEnvironmentTextureCubeName The precomputed Fresnel Environment Map texture's name, as returned by FresnelEnvironmentRenderingRun::getResultTextureName().

Reimplemented from FinalRenderingRun.

4.100.3.5 virtual void SEMFinalRenderingRun::setFresnelEnvironmentUpdateInterval (unsigned int *updateIntervalNumOfFrames*) [virtual]

Set the SEM udate interval desired for the owner entity. If SEM is not used, the method should have no effect.

Parameters:

updateIntervalNumOfFrames After how many frames should the preprocesing step be repeated to update the SEM.

Reimplemented from FinalRenderingRun.

4.100.3.6 void SEMFinalRenderingRun::setFresnelFactor (float *fresnelFactor*)

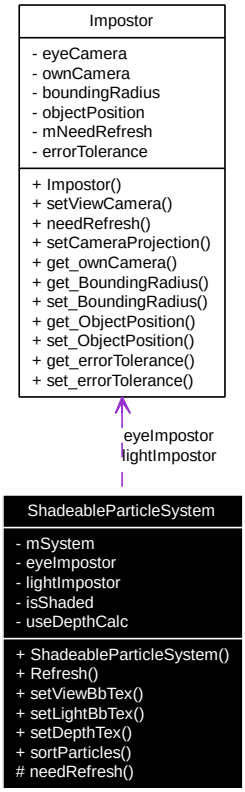
Sets the material's Fresnel factor for the EnvMap shader.

Parameters:

fresnelFactor The Fresnel factor. 0.0f means only a small reflection in narrow angles, 1.0f complete reflection to every direction.

4.101 ShadeableParticleSystem Class Reference

Collaboration diagram for ShadeableParticleSystem:



Public Member Functions

- ShadeableParticleSystem (const String &name, const String &scriptname, SceneNode *node)

  Constructor.

- void Refresh ()

  Refreshes the system.

- void setViewBbTex (String texname)

  Sets the composite texture for rendering.

- void setLightBbTex (String texname)

  Sets the light illumination texture for rendering.

- void setDepthTex (String texname)

  Sets the depth texture (containing scene depth information in camera space for rendering.).

- *void sortParticles (BILLBOARD_SORTORDER SortOrder, Vector3 ViewPoint)

Protected Member Functions

- bool needRefresh ()

  This function decides if the illumination or the depth information need to be refreshed.

4.101.1 Constructor & Destructor Documentation

4.101.1.1 ShadeableParticleSystem::ShadeableParticleSystem (const String & *name*, const String & *scriptname*, SceneNode * *node*)

Constructor.

Parameters:

name the name of the hierarchical system. The particle system names (which oge uses) will be generaed from this name.

scriptname the name of the particle script describing the characteristics of the system.

node the scene node to attach the particle system to.

4.101.2 Member Function Documentation

4.101.2.1 bool ShadeableParticleSystem::needRefresh () [protected]

This function decides if the illumination or the depth information need to be refreshed.

4.101.2.2 void ShadeableParticleSystem::Refresh ()

Refreshes the system.

4.101.2.3 void ShadeableParticleSystem::setDepthTex (String *texname*) [inline]

Sets the depth texture (containing scene depth information in camera space for rendering.).

4.101.2.4 void ShadeableParticleSystem::setLightBbTex (String *texname*) [inline]

Sets the light illumination texture for rendering.

4.101.2.5 void ShadeableParticleSystem::setViewBbTex (String *texname*) [inline]

Sets the composite texture for rendering.

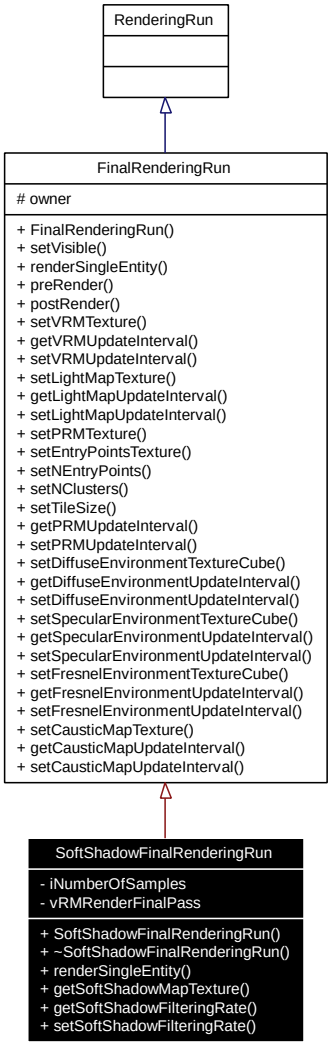
4.101.2.6 ∗ void ShadeableParticleSystem::sortParticles (BILLBOARD_SORTORDER *SortOrder*, Vector3 *ViewPoint*)

Sorts particles. Sorts particles from the given point (Vector3 viewpoint) either in incrementing or in decrementing order (bool increment)

4.102 SoftShadowFinalRenderingRun Class Reference

Render the entity into the backbuffer with its final texture.

Inheritance diagram for SoftShadowFinalRenderingRun:



Collaboration diagram for SoftShadowFinalRenderingRun:

Public Member Functions

- SoftShadowFinalRenderingRun (Entity *entity)
- ~SoftShadowFinalRenderingRun ()

  Destructor.

- void renderSingleEntity (RenderTarget *backBuffer, CubeMapFaces cf=CUBEMAP_FACE_-POSITIVE_X)

  Renders a sinle entity into the screen.

- const String & getSoftShadowMapTexture ()

  Returns the softShadowMapTexture.

- unsigned int getSoftShadowFilteringRate ()

  Gets the number of the samples used in the filtering.

- void setSoftShadowFilteringRate (unsigned int usedNumberOfSamples)

  Sets the number of the samples used in the filtering.

4.102.1 Detailed Description

Render the entity into the backbuffer with its final texture.

SuperClass: FinalRenderingRun Class: SoftShadowFinalRenderingRun

4.102.2 Constructor & Destructor Documentation

4.102.2.1 SoftShadowFinalRenderingRun::SoftShadowFinalRenderingRun (Entity * *entity*)

Constructor.

Parameters:
entity The owner entity of the Run.

4.102.2.2 SoftShadowFinalRenderingRun::~SoftShadowFinalRenderingRun ()

Destructor.

4.102.3 Member Function Documentation

4.102.3.1 unsigned int SoftShadowFinalRenderingRun::getSoftShadowFilteringRate ()

Gets the number of the samples used in the filtering.

Returns:
The number of the samples used in the filtering..

4.102.3.2 const String& SoftShadowFinalRenderingRun::getSoftShadowMapTexture ()

Returns the softShadowMapTexture.

4.102.3.3 void SoftShadowFinalRenderingRun::renderSingleEntity (RenderTarget * *backBuffer*, CubeMapFaces *cf* = CUBEMAP_FACE_POSITIVE_X) [virtual]

Renders a sinle entity into the screen.

Parameters:
backBuffer RenderTarget, The screen.
cf CubeMapFaces, A CubeMap face.

Implements FinalRenderingRun.

4.102.3.4 void SoftShadowFinalRenderingRun::setSoftShadowFilteringRate (unsigned int *usedNumberOfSamples*)

Sets the number of the samples used in the filtering.

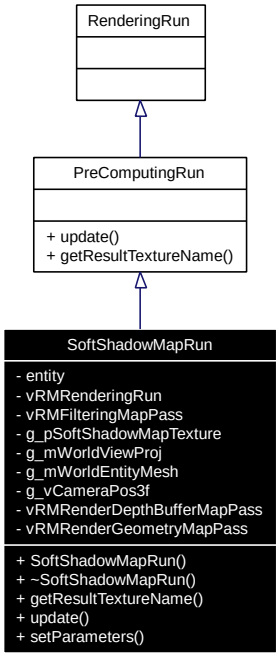
Parameters:

usedNumberOfSamples int, The number of the samples used in the filtering.

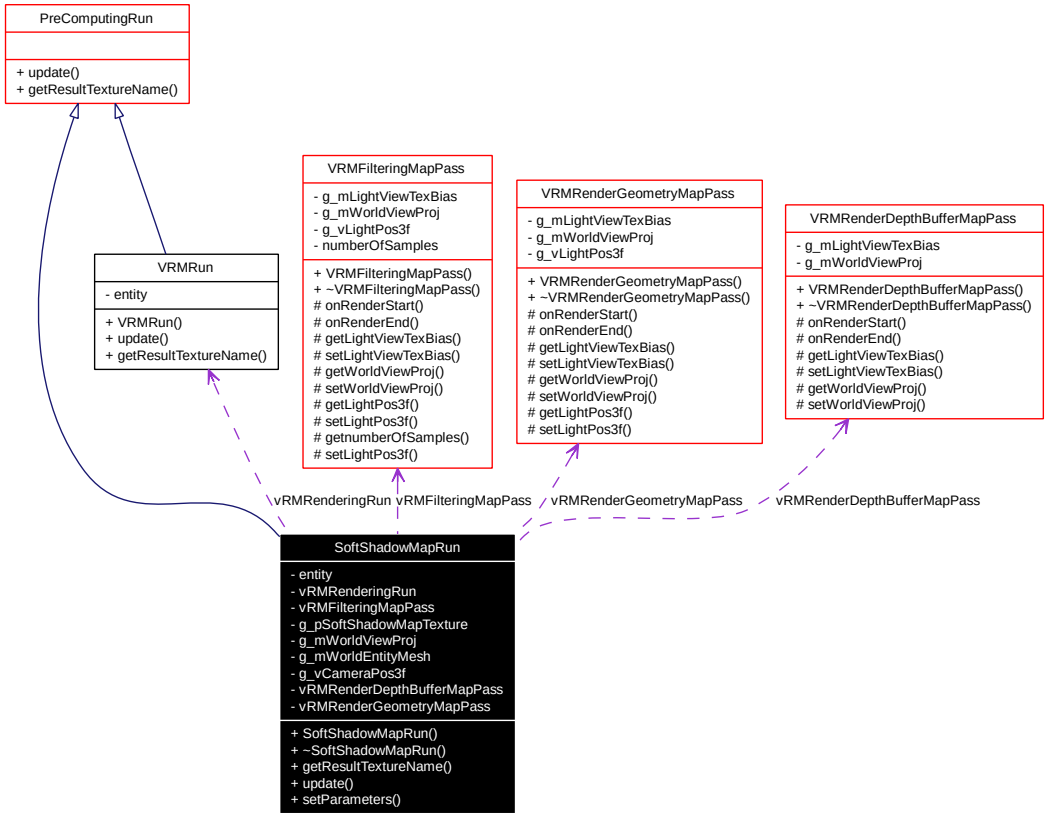
4.103 SoftShadowMapRun Class Reference

Generates soft shadow effect.

Inheritance diagram for SoftShadowMapRun:



Collaboration diagram for SoftShadowMapRun:



Public Member Functions

- SoftShadowMapRun (Entity *entity)

  Constructor.

- ~SoftShadowMapRun ()

  Destructor.

- const String & getResultTextureName ()

  Returns with the created texture.

- void update ()

  Recalculates the passes.

- void setParameters ()

  Changes the value of the parameters.

4.103.1 Detailed Description

Generates soft shadow effect.

SuperClass: PreComputingRun Class: SoftShadowMapRun The instances of this class are to generate softshadow effects for the entity. The resulting texture is a PF_FLOAT32_RGBA type texture

4.103.2 Constructor & Destructor Documentation

4.103.2.1 SoftShadowMapRun::SoftShadowMapRun (Entity * *entity*)

Constructor.

Constructor.

Parameters:

entity The owner entity of an entity-bound precomputing run.

4.103.2.2 SoftShadowMapRun::~SoftShadowMapRun ()

Destructor.

4.103.3 Member Function Documentation

4.103.3.1 const String& SoftShadowMapRun::getResultTextureName () `[virtual]`

Returns with the created texture.

Reimplemented from PreComputingRun.

4.103.3.2 void SoftShadowMapRun::setParameters ()

Changes the value of the parameters.

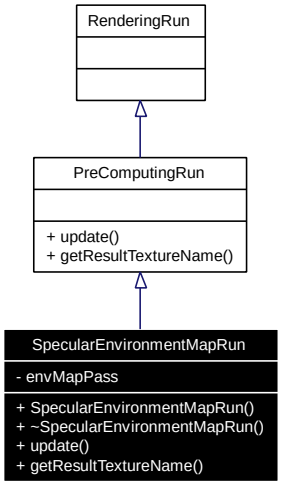
4.103.3.3 void SoftShadowMapRun::update () `[virtual]`

Recalculates the passes.

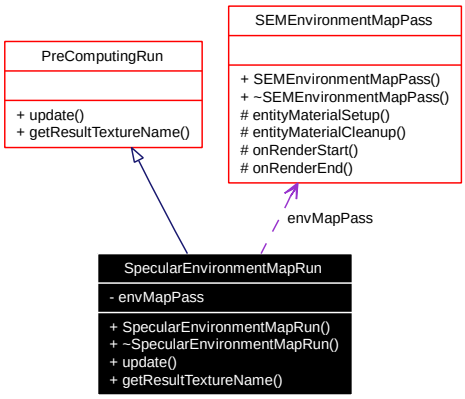
Implements PreComputingRun.

4.104 SpecularEnvironmentMapRun Class Reference

Inheritance diagram for SpecularEnvironmentMapRun:



Collaboration diagram for SpecularEnvironmentMapRun:



Public Member Functions

- SpecularEnvironmentMapRun (Entity ∗entity, unsigned int resolution)
- ∼SpecularEnvironmentMapRun (void)
- virtual void update (void)
- virtual const String & getResultTextureName ()

  This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

4.104.1 Constructor & Destructor Documentation

4.104.1.1 SpecularEnvironmentMapRun::SpecularEnvironmentMapRun (Entity * *entity*, unsigned int *resolution*)

Constructor

Parameters:

entity The owner entity.

resolution The resolution of the environment cube-map.

Remarks:

The resolution parameter must be equal and the power of 2.

4.104.1.2 SpecularEnvironmentMapRun::~SpecularEnvironmentMapRun (void)

Destructor

4.104.2 Member Function Documentation

4.104.2.1 virtual const String& SpecularEnvironmentMapRun::getResultTextureName () [virtual]

This method is provided for naming consistence. Special PreComputingRuns, if any, where the result is not a texture, may ignore this method. Further methods may be added to retrieve additional texture names or references non-texture results.

Returns:

the main result texture's name

Reimplemented from PreComputingRun.

4.104.2.2 virtual void SpecularEnvironmentMapRun::update (void) [virtual]

Performs the update of the environment cube-map. LOD can be implemented by varying the frequency of calls.

Implements PreComputingRun.

4.105 SPlane Struct Reference

Public Attributes

- Vector3 topLeft
- Vector3 topRight
- Vector3 bottomLeft
- Vector3 bottomRight
- Vector3 normal

4.105.1 Member Data Documentation

4.105.1.1 Vector3 SPlane::bottomLeft

4.105.1.2 Vector3 SPlane::bottomRight

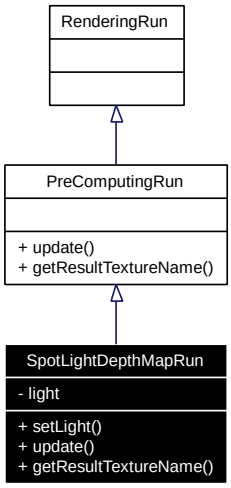
4.105.1.3 Vector3 SPlane::normal

4.105.1.4 Vector3 SPlane::topLeft

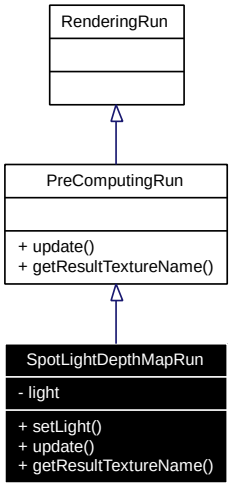
4.105.1.5 Vector3 SPlane::topRight

4.106 SpotLightDepthMapRun Class Reference

Inheritance diagram for SpotLightDepthMapRun:



Collaboration diagram for SpotLightDepthMapRun:



Public Member Functions

- void setLight (Light *light)
- virtual void update ()
- virtual const String & getResultTextureName ()

4.106.1 Detailed Description

Computes a depth map for a spot light

4.106.2 Member Function Documentation

4.106.2.1 virtual const String& SpotLightDepthMapRun::getResultTextureName () [inline, virtual]

Reimplemented from PreComputingRun.

4.106.2.2 void SpotLightDepthMapRun::setLight (Light * *light*) [inline]

Parameters:

light The owner light of an light-bound precomputing run.

4.106.2.3 virtual void SpotLightDepthMapRun::update (void) [inline, virtual]

Implements PreComputingRun.

4.107 SubMeshesLeavesGenerator Class Reference

This module generates the submeshes of all the leaves associated to each cluster plane generated by the class PlanesGenerator.

Collaboration diagram for SubMeshesLeavesGenerator:


LeavesInfo
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
+ vLeavesInfo
+ iLeavesInfoBegin
+ iLeavesInfoEnd
SubMeshesLeavesGenerator
vertex
node
nodeIt
nodeIt2
cFileName01
cFileName02
cFileName03
fileName01
vFaces
vVertexs
vNormals
vTexCoords
vLeaves
iLeaves
iFaces01
iFaces02
iVertexs
v1
v2
v3
nvertexs
iface
ivertex
nTextures
timer
initTime
doc
loadOkay
x1
y1
z1
uc1
vc1
vertex02
treeNode02
leavesNode02
leafNode02
coord4dNode02
facesNode02
faceNode02
fileName02
sLeavesInfo02
idFace02
numLeaves02
numFaces02
doc02
loadOkay02
fileName03
numPlanes03
numSubMeshes
node03
planeNode03
leafNode03
doc03
loadOkay03
vFacesSubMesh
vVertexsSubMesh
vNormalsSubMesh
vTexCoordsSubMesh
endTime
node
nodeIt
cFileName01
cFileName02
cFileName03
fileName01
vFaces
vVertexs
vNormals
vTexCoords
vLeaves
iFaces01
iFaces02
iVertexs
initTime
treeNode02
leavesNode02
leafNode02
coord4dNode02
facesNode02
faceNode02

Public Member Functions

- SubMeshesLeavesGenerator (void)

  Constructor method of the class `SubMeshesLeavesGenerator`.

- ~SubMeshesLeavesGenerator (void)

  Destructor method of the class `SubMeshesLeavesGenerator`.

- SubMeshesLeavesGenerator * getSingletonPtr (void)

  Return pointer to singleton SubMeshesLeavesGenerator object.

- SubMeshesLeavesGenerator & getSingleton (void)

  Return singleton LeavesGenerator object.

- void generateSubMeshesLeaves (char *filenames[ ])

  This method load the leaves mesh file, the leaves information file and the information about the cluster planes and generate all the leaves submeshes associated to each cluster planes.

- SubMeshesLeavesGenerator (void)

  Constructor method of the class `SubMeshesLeavesGenerator`.

- ~SubMeshesLeavesGenerator (void)

  Destructor method of the class `SubMeshesLeavesGenerator`.

- SubMeshesLeavesGenerator * getSingletonPtr (void)

  Return pointer to singleton SubMeshesLeavesGenerator object.

- SubMeshesLeavesGenerator & getSingleton (void)

  Return singleton LeavesGenerator object.

- void generateSubMeshesLeaves (char *filenames[ ])

  This method load the leaves mesh file, the leaves information file and the information about the cluster planes and generate all the leaves submeshes associated to each cluster planes.

Protected Types

- typedef set< int, greater< int > > ID_FACESet
- typedef set< int, greater< int > > ID_FACESet

Protected Attributes

- Vector3 vertex
- TiXmlNode * node
- TiXmlNode * nodeIt
- TiXmlNode * nodeIt2
- char * cFileName01
- char * cFileName02
- char * cFileName03
- TIXML_STRING * fileName01

- vector< Vector3 > vFaces
- vector< Vector3 > vVertexs
- vector< Vector3 > vNormals
- vector< Vector2 > vTexCoords
- vector< ID_FACESet > vLeaves
- ID_FACESet::iterator iLeaves
- vector< Vector3 >::iterator iFaces01
- vector< Vector3 >::iterator iFaces02
- vector< Vector3 >::iterator iVertexs
- unsigned int v1
- unsigned int v2
- unsigned int v3
- unsigned int nvertexs
- unsigned int iface
- unsigned int ivertex
- float nTextures
- time_t timer
- tm * initTime
- TiXmlDocument doc
- bool loadOkay
- double x1
- double y1
- double z1
- double uc1
- double vc1
- Vector3 vertex02
- TiXmlNode * treeNode02
- TiXmlNode * leavesNode02
- TiXmlNode * leafNode02
- TiXmlNode * coord4dNode02
- TiXmlNode * facesNode02
- TiXmlNode * faceNode02
- TIXML_STRING * fileName02
- LeavesInfo sLeavesInfo02
- unsigned int idFace02
- unsigned int numLeaves02
- unsigned int numFaces02
- TiXmlDocument doc02
- bool loadOkay02
- TIXML_STRING * fileName03
- unsigned int numPlanes03
- unsigned int numSubMeshes
- TiXmlNode * node03
- TiXmlNode * planeNode03
- TiXmlNode * leafNode03
- TiXmlDocument doc03
- bool loadOkay03
- vector< Vector3 > vFacesSubMesh
- vector< Vector3 > vVertexsSubMesh
- vector< Vector3 > vNormalsSubMesh

- vector< Vector2 > vTexCoordsSubMesh
- tm * endTime
- TiXmlNode * node
- TiXmlNode * nodeIt
- char * cFileName01
- char * cFileName02
- char * cFileName03
- TIXML_STRING * fileName01
- vector< Vector3 > vFaces
- vector< Vector3 > vVertexs
- vector< Vector3 > vNormals
- vector< Vector2 > vTexCoords
- vector< ID_FACESet > vLeaves
- vector< Vector3 >::iterator iFaces01
- vector< Vector3 >::iterator iFaces02
- vector< Vector3 >::iterator iVertexs
- tm * initTime
- TiXmlNode * treeNode02
- TiXmlNode * leavesNode02
- TiXmlNode * leafNode02
- TiXmlNode * coord4dNode02
- TiXmlNode * facesNode02
- TiXmlNode * faceNode02
- TIXML_STRING * fileName02
- TIXML_STRING * fileName03
- TiXmlNode * node03
- TiXmlNode * planeNode03
- TiXmlNode * leafNode03
- vector< Vector3 > vFacesSubMesh
- vector< Vector3 > vVertexsSubMesh
- vector< Vector3 > vNormalsSubMesh
- vector< Vector2 > vTexCoordsSubMesh
- tm * endTime

4.107.1 Detailed Description

This module generates the submeshes of all the leaves associated to each cluster plane generated by the class `PlanesGenerator`.

The main tasks of the class `SubMeshesLeavesGenerator` are:

- Load the mesh file that contains all the faces of the leaves.
- Load the xml file that contains the leaves information, which faces of the original mesh are from the same leaf.
- Load the xml file that contains the cluster planes generated by the `PlanesGenerator`, with the information of which leaves are associated to each cluster plane.
- Generate all the submeshes that contains all the leaves associated to each cluster plane.
- Store a xml files with the submeshes of all the leaves associated to each cluster plane.

4.107.2 Member Typedef Documentation

4.107.2.1 typedef set<int,greater<int> > SubMeshesLeavesGenerator::ID_FACESet [protected]

4.107.2.2 typedef set<int,greater<int> > SubMeshesLeavesGenerator::ID_FACESet [protected]

4.107.3 Constructor & Destructor Documentation

4.107.3.1 SubMeshesLeavesGenerator::SubMeshesLeavesGenerator (void) [inline]

Constructor method of the class SubMeshesLeavesGenerator.

4.107.3.2 SubMeshesLeavesGenerator::~SubMeshesLeavesGenerator (void) [inline]

Destructor method of the class SubMeshesLeavesGenerator.

4.107.3.3 SubMeshesLeavesGenerator::SubMeshesLeavesGenerator (void) [inline]

Constructor method of the class SubMeshesLeavesGenerator.

4.107.3.4 SubMeshesLeavesGenerator::~SubMeshesLeavesGenerator (void) [inline]

Destructor method of the class SubMeshesLeavesGenerator.

4.107.4 Member Function Documentation

4.107.4.1 void SubMeshesLeavesGenerator::generateSubMeshesLeaves (char * *filenames*[ ]) [inline]

This method load the leaves mesh file, the leaves information file and the information about the cluster planes and generate all the leaves submeshes associated to each cluster planes.

Parameters:
filenames Strings that contains all the filenames needed

4.107.4.2 void SubMeshesLeavesGenerator::generateSubMeshesLeaves (char * *filenames*[ ]) [inline]

This method load the leaves mesh file, the leaves information file and the information about the cluster planes and generate all the leaves submeshes associated to each cluster planes.

Parameters:
filenames Strings that contains all the filenames needed

4.107.4.3 SubMeshesLeavesGenerator& SubMeshesLeavesGenerator::getSingleton (void) [inline]

Return singleton LeavesGenerator object.

Returns:
Singleton LeavesGenerator object

4.107.4.4 SubMeshesLeavesGenerator& SubMeshesLeavesGenerator::getSingleton (void) [inline]

Return singleton LeavesGenerator object.

Returns:
Singleton LeavesGenerator object

4.107.4.5 SubMeshesLeavesGenerator* SubMeshesLeavesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton SubMeshesLeavesGenerator object.

Returns:
Pointer to singleton SubMeshesLeavesGenerator object

4.107.4.6 SubMeshesLeavesGenerator* SubMeshesLeavesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton SubMeshesLeavesGenerator object.

Returns:
Pointer to singleton SubMeshesLeavesGenerator object

4.107.5 Member Data Documentation

4.107.5.1 char* SubMeshesLeavesGenerator::cFileName01 [protected]

4.107.5.2 char* SubMeshesLeavesGenerator::cFileName01 [protected]

4.107.5.3 char* SubMeshesLeavesGenerator::cFileName02 [protected]

4.107.5.4 char* SubMeshesLeavesGenerator::cFileName02 [protected]

4.107.5.5 char* SubMeshesLeavesGenerator::cFileName03 [protected]

4.107.5.6 char* SubMeshesLeavesGenerator::cFileName03 [protected]

4.107.5.7 TiXmlNode* SubMeshesLeavesGenerator::coord4dNode02 [protected]

4.107.5.8 TiXmlNode* SubMeshesLeavesGenerator::coord4dNode02 [protected]

4.107.5.9 TiXmlDocument SubMeshesLeavesGenerator::doc [protected]

4.107.5.10 TiXmlDocument SubMeshesLeavesGenerator::doc02 [protected]

4.107.5.11 TiXmlDocument SubMeshesLeavesGenerator::doc03 [protected]

4.107.5.12 struct tm* SubMeshesLeavesGenerator::endTime [protected]

4.107.5.13 struct tm* SubMeshesLeavesGenerator::endTime [protected]

4.107.5.14 TiXmlNode* SubMeshesLeavesGenerator::faceNode02 [protected]

4.107.5.15 TiXmlNode* SubMeshesLeavesGenerator::faceNode02 [protected]

4.107.5.16 TiXmlNode* SubMeshesLeavesGenerator::facesNode02 [protected]

4.107.5.17 TiXmlNode* SubMeshesLeavesGenerator::facesNode02 [protected]

4.107.5.18 TIXML_STRING* SubMeshesLeavesGenerator::fileName01 [protected]

4.107.5.19 TIXML_STRING* SubMeshesLeavesGenerator::fileName01 [protected]

4.107.5.20 TIXML_STRING* SubMeshesLeavesGenerator::fileName02 [protected]

4.107.5.21 TIXML_STRING* SubMeshesLeavesGenerator::fileName02 [protected]

4.107.5.22 TIXML_STRING* SubMeshesLeavesGenerator::fileName03 [protected]

4.107.5.23 TIXML_STRING* SubMeshesLeavesGenerator::fileName03 [protected]

4.107.5.24 unsigned int SubMeshesLeavesGenerator::idFace02 [protected]

4.107.5.25 unsigned int SubMeshesLeavesGenerator::iface [protected]

4.107.5.26 vector<Vector3>::iterator SubMeshesLeavesGenerator::iFaces01 [protected]

4.107.5.27 vector<Vector3>::iterator SubMeshesLeavesGenerator::iFaces01 [protected]

4.107.5.28 vector<Vector3>::iterator SubMeshesLeavesGenerator::iFaces02 [protected]

4.107.5.29 vector<Vector3>::iterator SubMeshesLeavesGenerator::iFaces02 [protected]

4.107.5.30 ID_FACESet::iterator SubMeshesLeavesGenerator::iLeaves [protected]

4.107.5.31 struct tm* SubMeshesLeavesGenerator::initTime [protected]

4.107.5.32 struct tm* SubMeshesLeavesGenerator::initTime [protected]

4.107.5.33 unsigned int SubMeshesLeavesGenerator::ivertex [protected]

4.107.5.34 vector<Vector3>::iterator SubMeshesLeavesGenerator::iVertexs [protected]

4.107.5.35 vector<Vector3>::iterator SubMeshesLeavesGenerator::iVertexs [protected]

4.107.5.36 TiXmlNode* SubMeshesLeavesGenerator::leafNode02 [protected]

4.107.5.37 TiXmlNode* SubMeshesLeavesGenerator::leafNode02 [protected]

4.107.5.38 TiXmlNode* SubMeshesLeavesGenerator::leafNode03 [protected]

4.107.5.39 TiXmlNode* SubMeshesLeavesGenerator::leafNode03 [protected]

4.107.5.40 TiXmlNode* SubMeshesLeavesGenerator::leavesNode02 [protected]

4.107.5.41 TiXmlNode* SubMeshesLeavesGenerator::leavesNode02 [protected]

4.107.5.42 bool SubMeshesLeavesGenerator::loadOkay [protected]

4.107.5.43 bool SubMeshesLeavesGenerator::loadOkay02 [protected]

4.107.5.44 bool SubMeshesLeavesGenerator::loadOkay03 [protected]

4.107.5.45 TiXmlNode* SubMeshesLeavesGenerator::node [protected]

4.107.5.46 TiXmlNode* SubMeshesLeavesGenerator::node [protected]

4.107.5.47 TiXmlNode* SubMeshesLeavesGenerator::node03 [protected]

4.107.5.48 TiXmlNode* SubMeshesLeavesGenerator::node03 [protected]

4.107.5.49 TiXmlNode* SubMeshesLeavesGenerator::nodeIt [protected]

4.107.5.50 TiXmlNode* SubMeshesLeavesGenerator::nodeIt [protected]

4.107.5.51 TiXmlNode * SubMeshesLeavesGenerator::nodeIt2 [protected]

4.107.5.52 float SubMeshesLeavesGenerator::nTextures [protected]

4.107.5.53 unsigned int SubMeshesLeavesGenerator::numFaces02 [protected]

4.107.5.54 unsigned int SubMeshesLeavesGenerator::numLeaves02 [protected]

4.107.5.55 unsigned int SubMeshesLeavesGenerator::numPlanes03 [protected]

4.107.5.56 unsigned int SubMeshesLeavesGenerator::numSubMeshes [protected]

4.107.5.57 unsigned int SubMeshesLeavesGenerator::nvertexs [protected]

4.107.5.58 TiXmlNode* SubMeshesLeavesGenerator::planeNode03 [protected]

4.107.5.59 TiXmlNode* SubMeshesLeavesGenerator::planeNode03 [protected]

4.107.5.60 LeavesInfo SubMeshesLeavesGenerator::sLeavesInfo02 [protected]

4.107.5.61 time_t SubMeshesLeavesGenerator::timer [protected]

4.107.5.62 TiXmlNode* SubMeshesLeavesGenerator::treeNode02 [protected]

4.107.5.63 TiXmlNode* SubMeshesLeavesGenerator::treeNode02 [protected]

4.107.5.64 double SubMeshesLeavesGenerator::uc1 [protected]

4.107.5.65 unsigned int SubMeshesLeavesGenerator::v1 [protected]

4.107.5.66 unsigned int SubMeshesLeavesGenerator::v2 [protected]

4.107.5.67 unsigned int SubMeshesLeavesGenerator::v3 [protected]

4.107.5.68 double SubMeshesLeavesGenerator::vc1 [protected]

4.107.5.69 Vector3 SubMeshesLeavesGenerator::vertex [protected]

4.107.5.70 Vector3 SubMeshesLeavesGenerator::vertex02 [protected]

4.107.5.71 vector<Vector3> SubMeshesLeavesGenerator::vFaces [protected]

4.107.5.72 vector<Vector3> SubMeshesLeavesGenerator::vFaces [protected]

4.107.5.73 vector<Vector3> SubMeshesLeavesGenerator::vFacesSubMesh [protected]

4.107.5.74 vector<Vector3> SubMeshesLeavesGenerator::vFacesSubMesh [protected]

4.107.5.75 vector<ID_FACESet> SubMeshesLeavesGenerator::vLeaves [protected]

4.107.5.76 vector<ID_FACESet> SubMeshesLeavesGenerator::vLeaves [protected]

4.107.5.77 vector<Vector3> SubMeshesLeavesGenerator::vNormals [protected]

4.107.5.78 vector<Vector3> SubMeshesLeavesGenerator::vNormals [protected]

4.107.5.79 vector<Vector3> SubMeshesLeavesGenerator::vNormalsSubMesh [protected]

4.107.5.80 vector<Vector3> SubMeshesLeavesGenerator::vNormalsSubMesh [protected]

4.107.5.81 vector<Vector2> SubMeshesLeavesGenerator::vTexCoords [protected]

4.107.5.82 vector<Vector2> SubMeshesLeavesGenerator::vTexCoords [protected]

4.107.5.83 vector<Vector2> SubMeshesLeavesGenerator::vTexCoordsSubMesh [protected]

4.107.5.84 vector<Vector2> SubMeshesLeavesGenerator::vTexCoordsSubMesh [protected]

4.107.5.85 vector<Vector3> SubMeshesLeavesGenerator::vVertexs [protected]

4.107.5.86 vector<Vector3> SubMeshesLeavesGenerator::vVertexs [protected]

4.107.5.87 vector<Vector3> SubMeshesLeavesGenerator::vVertexsSubMesh [protected]

4.107.5.88 vector<Vector3> SubMeshesLeavesGenerator::vVertexsSubMesh [protected]

4.107.5.89 double SubMeshesLeavesGenerator::x1 [protected]

4.107.5.90 double SubMeshesLeavesGenerator::y1 [protected]

4.107.5.91 double SubMeshesLeavesGenerator::z1 [protected]

4.108 SubMeshesPlanesGenerator Class Reference

This class generate a compact mesh with all the cluster planes that will be used during the visualization later step. Recieve the planes information generated by the `PlanesCorrector`, (the vertexs of each smaller quad associated to the cluster plane that fits all the leaves clustered for each plane) to generate the impostor leaves mesh.

Public Member Functions

- SubMeshesPlanesGenerator (void)

  Constructor method of the class `SubMeshesPlanesGenerator`.

- ~SubMeshesPlanesGenerator (void)

  Destructor method of the class `SubMeshesPlanesGenerator`.

- SubMeshesPlanesGenerator * getSingletonPtr (void)

  Return pointer to singleton SubMeshesPlanesGenerator object.

- SubMeshesPlanesGenerator & getSingleton (void)

  Return singleton SubMeshesPlanesGenerator object.

- void generatePlanesSubMeshes (char *filenames[ ])

  This load the information of the smaller quad associated to each plane and generate the impostor leaves mesh.

- SubMeshesPlanesGenerator (void)

  Constructor method of the class `SubMeshesPlanesGenerator`.

- ~SubMeshesPlanesGenerator (void)

  Destructor method of the class `SubMeshesPlanesGenerator`.

- SubMeshesPlanesGenerator * getSingletonPtr (void)

  Return pointer to singleton SubMeshesPlanesGenerator object.

- SubMeshesPlanesGenerator & getSingleton (void)

  Return singleton SubMeshesPlanesGenerator object.

- void generatePlanesSubMeshes (char *filenames[ ])

  This load the information of the smaller quad associated to each plane and generate the impostor leaves mesh.

Protected Attributes

- TiXmlNode * node
- TiXmlNode * nodeIt

- char * cFileName01
- TIXML_STRING * fileName01
- vector< SPlane > vPlanesInfo
- unsigned int nPlanes
- time_t timer
- tm * initTime
- bool loadOkay
- TIXML_STRING * outputFilename
- TiXmlDocument doc
- TiXmlDocument outputFile
- TiXmlNode * rootNode
- TiXmlNode * submeshesNode
- TiXmlNode * submeshNode
- TiXmlNode * facesNode
- TiXmlNode * faceNode
- TiXmlNode * geometryNode
- TiXmlNode * vertexbufferNode
- TiXmlNode * vertexNode
- TiXmlNode * positionNode
- TiXmlNode * normalNode
- TiXmlNode * texcoordNode
- tm * endTime
- ofstream materialOutputFile
- TiXmlNode * node
- TiXmlNode * nodeIt
- char * cFileName01
- TIXML_STRING * fileName01
- vector< SPlane > vPlanesInfo
- tm * initTime
- TIXML_STRING * outputFilename
- TiXmlNode * rootNode
- TiXmlNode * geometryNode
- tm * endTime

4.108.1 Detailed Description

This class generate a compact mesh with all the cluster planes that will be used during the visualization later step. Recieve the planes information generated by the `PlanesCorrector`, (the vertexs of each smaller quad associated to the cluster plane that fits all the leaves clustered for each plane) to generate the impostor leaves mesh.

The main tasks of the class `SubMeshesPlanesGenerator` are:

- Load the mesh file that contains all the faces of the leaves.
- Generate a single mesh struct with the quad of each cluster plane.
- Store a mesh file that contains all the small quads that will be the impostor leaves mesh.

4.108.2 Constructor & Destructor Documentation

4.108.2.1 SubMeshesPlanesGenerator::SubMeshesPlanesGenerator (void) [inline]

Constructor method of the class SubMeshesPlanesGenerator.

4.108.2.2 SubMeshesPlanesGenerator::~SubMeshesPlanesGenerator (void) [inline]

Destructor method of the class SubMeshesPlanesGenerator.

4.108.2.3 SubMeshesPlanesGenerator::SubMeshesPlanesGenerator (void) [inline]

Constructor method of the class SubMeshesPlanesGenerator.

4.108.2.4 SubMeshesPlanesGenerator::~SubMeshesPlanesGenerator (void) [inline]

Destructor method of the class SubMeshesPlanesGenerator.

4.108.3 Member Function Documentation

4.108.3.1 void SubMeshesPlanesGenerator::generatePlanesSubMeshes (char * *filenames*[ ]) [inline]

This load the information of the smaller quad associated to each plane and generate the impostor leaves mesh.

Parameters:

filenames Strings that contains all the filenames needed

4.108.3.2 void SubMeshesPlanesGenerator::generatePlanesSubMeshes (char * *filenames*[ ]) [inline]

This load the information of the smaller quad associated to each plane and generate the impostor leaves mesh.

Parameters:

filenames Strings that contains all the filenames needed

4.108.3.3 SubMeshesPlanesGenerator& SubMeshesPlanesGenerator::getSingleton (void) [inline]

Return singleton SubMeshesPlanesGenerator object.

Returns:
Singleton SubMeshesPlanesGenerator object

4.108.3.4 SubMeshesPlanesGenerator& SubMeshesPlanesGenerator::getSingleton (void) [inline]

Return singleton SubMeshesPlanesGenerator object.

Returns:
Singleton SubMeshesPlanesGenerator object

4.108.3.5 SubMeshesPlanesGenerator* SubMeshesPlanesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton SubMeshesPlanesGenerator object.

Returns:
Pointer to singleton SubMeshesPlanesGenerator object

4.108.3.6 SubMeshesPlanesGenerator* SubMeshesPlanesGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton SubMeshesPlanesGenerator object.

Returns:
Pointer to singleton SubMeshesPlanesGenerator object

4.108.4 Member Data Documentation

4.108.4.1 char* SubMeshesPlanesGenerator::cFileName01 [protected]

4.108.4.2 char* SubMeshesPlanesGenerator::cFileName01 [protected]

4.108.4.3 TiXmlDocument SubMeshesPlanesGenerator::doc [protected]

4.108.4.4 struct tm* SubMeshesPlanesGenerator::endTime [protected]

4.108.4.5 struct tm* SubMeshesPlanesGenerator::endTime [protected]

4.108.4.6 TiXmlNode * SubMeshesPlanesGenerator::faceNode [protected]

4.108.4.7 TiXmlNode * SubMeshesPlanesGenerator::facesNode [protected]

4.108.4.8 TIXML_STRING* SubMeshesPlanesGenerator::fileName01 [protected]

4.108.4.9 TIXML_STRING* SubMeshesPlanesGenerator::fileName01 [protected]

4.108.4.10 TiXmlNode* SubMeshesPlanesGenerator::geometryNode [protected]

4.108.4.11 TiXmlNode* SubMeshesPlanesGenerator::geometryNode [protected]

4.108.4.12 struct tm* SubMeshesPlanesGenerator::initTime [protected]

4.108.4.13 struct tm* SubMeshesPlanesGenerator::initTime [protected]

4.108.4.14 bool SubMeshesPlanesGenerator::loadOkay [protected]

4.108.4.15 ofstream SubMeshesPlanesGenerator::materialOutputFile [protected]

4.108.4.16 TiXmlNode* SubMeshesPlanesGenerator::node [protected]

4.108.4.17 TiXmlNode* SubMeshesPlanesGenerator::node [protected]

4.108.4.18 TiXmlNode* SubMeshesPlanesGenerator::nodeIt [protected]

4.108.4.19 TiXmlNode* SubMeshesPlanesGenerator::nodeIt [protected]

4.108.4.20 TiXmlNode * SubMeshesPlanesGenerator::normalNode [protected]

4.108.4.21 unsigned int SubMeshesPlanesGenerator::nPlanes [protected]

4.108.4.22 TiXmlDocument SubMeshesPlanesGenerator::outputFile [protected]

4.108.4.23 TIXML_STRING* SubMeshesPlanesGenerator::outputFilename [protected]

4.108.4.24 TIXML_STRING* SubMeshesPlanesGenerator::outputFilename [protected]

4.108.4.25 TiXmlNode * SubMeshesPlanesGenerator::positionNode [protected]

4.108.4.26 TiXmlNode* SubMeshesPlanesGenerator::rootNode [protected]

4.108.4.27 TiXmlNode* SubMeshesPlanesGenerator::rootNode [protected]

4.108.4.28 TiXmlNode * SubMeshesPlanesGenerator::submeshesNode [protected]

4.108.4.29 TiXmlNode * SubMeshesPlanesGenerator::submeshNode [protected]

4.108.4.30 TiXmlNode * SubMeshesPlanesGenerator::texcoordNode [protected]

4.108.4.31 time_t SubMeshesPlanesGenerator::timer [protected]

4.108.4.32 TiXmlNode * SubMeshesPlanesGenerator::vertexbufferNode [protected]

4.108.4.33 TiXmlNode * SubMeshesPlanesGenerator::vertexNode [protected]

4.108.4.34 vector<SPlane> SubMeshesPlanesGenerator::vPlanesInfo [protected]

4.108.4.35 vector<SPlane> SubMeshesPlanesGenerator::vPlanesInfo [protected]

4.109 TextureGenerator Class Reference

This class applies a render to texture for all the leaves associated to each impostor plane. The textures that this class generate will be applied to each plane of the impostor leaves mesh that was created by the SubMeshesPlanesGenerator class.

Public Member Functions

- TextureGenerator (void)

  Constructor method of the class `TextureGenerator`.

- ~TextureGenerator (void)

  Destructor method of the class `TextureGenerator`.

- TextureGenerator * getSingletonPtr (void)

  Return pointer to singleton TextureGenerator object.

- TextureGenerator & getSingleton (void)

  Return singleton TextureGenerator object.

- void init (unsigned int currFrame, bool genTexImp, bool gpuGenTexImp, bool debugGpuGenTexImp, vector< InfoPlane > vIPlane, unsigned int numPlanes)

  This method initialise all the parameters needed to generate the textures.

- unsigned int getDepth (unsigned int nPlanes)
- void generateLeavesTextures (void)

  This method is called each frame for generate the textures associated to each impostor plane and store it as a file.

- TextureGenerator (void)

  Constructor method of the class `TextureGenerator`.

- ~TextureGenerator (void)

  Destructor method of the class `TextureGenerator`.

- TextureGenerator * getSingletonPtr (void)

  Return pointer to singleton TextureGenerator object.

- TextureGenerator & getSingleton (void)

  Return singleton TextureGenerator object.

- void init (unsigned int currFrame, bool genTexImp, bool gpuGenTexImp, bool debugGpuGenTexImp, vector< InfoPlane > vIPlane, unsigned int numPlanes)

  This method initialise all the parameters needed to generate the textures.

- unsigned int getDepth (unsigned int nPlanes)
- void generateLeavesTextures (void)

This method is called each frame for generate the textures associated to each impostor plane and store it as a file.

Protected Attributes

- RenderSystem * rendSys
- SceneManager * mSceneMgr
- Camera * mCamera
- unsigned int mCurrFrame
- bool generatingTextureImpostors
- GpuProgramParametersSharedPtr fragParams01
- GpuProgramParametersSharedPtr fragParams02
- GpuProgramParametersSharedPtr vertParams01
- GpuProgramParametersSharedPtr vertParams02
- bool gpuGenTexImpostors
- bool debugGpuGenTexImpostors
- unsigned int mNumPlanes
- unsigned int mPlaneVisible
- Vector3 vTopLeft
- Vector3 vTopRight
- Vector3 vBottomRight
- Vector3 vBottomLeft
- vector< InfoPlane > vInfoPlane
- Vector3 pNormal
- RenderSystem * rendSys
- SceneManager * mSceneMgr
- Camera * mCamera
- vector< InfoPlane > vInfoPlane

4.109.1 Detailed Description

This class applies a render to texture for all the leaves associated to each impostor plane. The textures that this class generate will be applied to each plane of the impostor leaves mesh that was created by the SubMeshesPlanesGenerator class.

The main tasks of the class `TextureGenerator` are:

- Select for each frame one impostor plane.
- Load the leaves submesh associated to the impostor plane.
- Render the leaves to a texture.
- Store the textures generated for each impostor plane.

4.109.2 Constructor & Destructor Documentation

4.109.2.1 TextureGenerator::TextureGenerator (void) [inline]

Constructor method of the class `TextureGenerator`.

4.109.2.2 TextureGenerator::~TextureGenerator (void) [inline]

Destructor method of the class `TextureGenerator`.

4.109.2.3 TextureGenerator::TextureGenerator (void) [inline]

Constructor method of the class `TextureGenerator`.

4.109.2.4 TextureGenerator::~TextureGenerator (void) [inline]

Destructor method of the class `TextureGenerator`.

4.109.3 Member Function Documentation

4.109.3.1 void TextureGenerator::generateLeavesTextures (void) [inline]

This method is called each frame for generate the textures associated to each impostor plane and store it as a file.

4.109.3.2 void TextureGenerator::generateLeavesTextures (void) [inline]

This method is called each frame for generate the textures associated to each impostor plane and store it as a file.

4.109.3.3 unsigned int TextureGenerator::getDepth (unsigned int *nPlanes*) [inline]

4.109.3.4 unsigned int TextureGenerator::getDepth (unsigned int *nPlanes*) [inline]

4.109.3.5 TextureGenerator& TextureGenerator::getSingleton (void) [inline]

Return singleton TextureGenerator object.

Returns:
Singleton TextureGenerator object

4.109.3.6 TextureGenerator& TextureGenerator::getSingleton (void) [inline]

Return singleton TextureGenerator object.

Returns:
Singleton TextureGenerator object

4.109.3.7 TextureGenerator* TextureGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton TextureGenerator object.

Returns:
Pointer to singleton TextureGenerator object

4.109.3.8 TextureGenerator* TextureGenerator::getSingletonPtr (void) [inline]

Return pointer to singleton TextureGenerator object.

Returns:
Pointer to singleton TextureGenerator object

4.109.3.9 void TextureGenerator::init (unsigned int *currFrame*, bool *genTexImp*, bool *gpuGenTexImp*, bool *debugGpuGenTexImp*, vector< InfoPlane > *vIPlane*, unsigned int *numPlanes*) [inline]

This method initialise all the parameters needed to generate the textures.

Parameters:

currFrame The number when we want to start to generate the textures.

genTexImp True if we want to store the textures.

gpuGenTexImp True if we want to generate the textures using a shader.

debugGpuGenTexImp True if we want to show a debug image to check if the generated image is correct.

vIPlane The list of impostor planes of which the textures should be generated.

numPlanes The number of impostor planes.

4.109.3.10 void TextureGenerator::init (unsigned int *currFrame*, bool *genTexImp*, bool *gpuGenTexImp*, bool *debugGpuGenTexImp*, vector< InfoPlane > *vIPlane*, unsigned int *numPlanes*) [inline]

This method initialise all the parameters needed to generate the textures.

Parameters:

currFrame The number when we want to start to generate the textures.

genTexImp True if we want to store the textures.

gpuGenTexImp True if we want to generate the textures using a shader.

debugGpuGenTexImp True if we want to show a debug image to check if the generated image is correct.

vIPlane The list of impostor planes of which the textures should be generated.

numPlanes The number of impostor planes.

4.109.4 Member Data Documentation

4.109.4.1 bool TextureGenerator::debugGpuGenTexImpostors [protected]

4.109.4.2 GpuProgramParametersSharedPtr TextureGenerator::fragParams01 [protected]

4.109.4.3 GpuProgramParametersSharedPtr TextureGenerator::fragParams02 [protected]

4.109.4.4 bool TextureGenerator::generatingTextureImpostors [protected]

4.109.4.5 bool TextureGenerator::gpuGenTexImpostors [protected]

4.109.4.6 Camera* TextureGenerator::mCamera [protected]

4.109.4.7 Camera* TextureGenerator::mCamera [protected]

4.109.4.8 unsigned int TextureGenerator::mCurrFrame [protected]

4.109.4.9 unsigned int TextureGenerator::mNumPlanes [protected]

4.109.4.10 unsigned int TextureGenerator::mPlaneVisible [protected]

4.109.4.11 SceneManager* TextureGenerator::mSceneMgr [protected]

4.109.4.12 SceneManager* TextureGenerator::mSceneMgr [protected]

4.109.4.13 Vector3 TextureGenerator::pNormal [protected]

4.109.4.14 RenderSystem* TextureGenerator::rendSys [protected]

4.109.4.15 RenderSystem* TextureGenerator::rendSys [protected]

4.109.4.16 Vector3 TextureGenerator::vBottomLeft [protected]

4.109.4.17 Vector3 TextureGenerator::vBottomRight [protected]

4.109.4.18 GpuProgramParametersSharedPtr TextureGenerator::vertParams01 [protected]

4.109.4.19 GpuProgramParametersSharedPtr TextureGenerator::vertParams02 [protected]

4.109.4.20 vector<InfoPlane> TextureGenerator::vInfoPlane [protected]

4.109.4.21 vector<InfoPlane> TextureGenerator::vInfoPlane [protected]

4.109.4.22 Vector3 TextureGenerator::vTopLeft [protected]

4.109.4.23 Vector3 TextureGenerator::vTopRight [protected]

4.110 Vertex Class Reference

Public Member Functions

- Vertex ()
- Vertex (double xx, double yy, double zz)
- ~Vertex ()
- Vertex & operator= (const Ogre::Vector3 &rkVector)

4.110.1 Detailed Description

this class represents a Ogre::Vector3 encapsulation

4.110.2 Constructor & Destructor Documentation

4.110.2.1 Vertex::Vertex ()

4.110.2.2 Vertex::Vertex (double *xx*, double *yy*, double *zz*)

4.110.2.3 Vertex::~Vertex ()

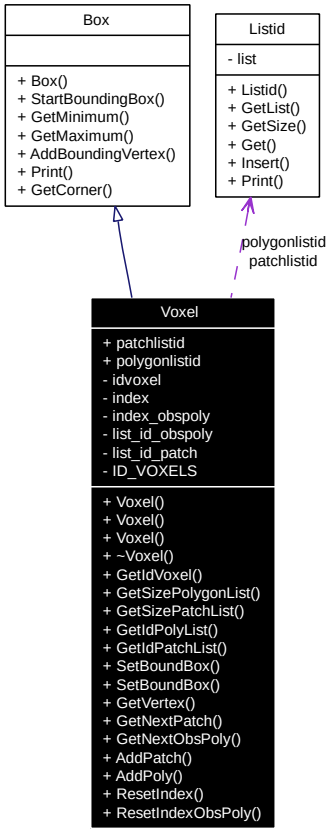
4.110.3 Member Function Documentation

4.110.3.1 Vertex& Vertex::operator= (const Ogre::Vector3 & *rkVector*)

4.111 Voxel Class Reference

Inheritance diagram for Voxel:

Collaboration diagram for Voxel:



Public Member Functions

- Voxel (Box &box)
- Voxel ()
- Voxel (const Voxel &v)
- ~Voxel ()
- unsigned int GetIdVoxel () const
- int GetSizePolygonList () const
- int GetSizePatchList () const
- std::vector< unsigned int > * GetIdPolyList ()
- std::vector< unsigned int > * GetIdPatchList ()
- void SetBoundBox (Box *b)
- void SetBoundBox (float x1, float y1, float z1, float x2, float y2, float z2) void Print()
- Vertex * GetVertex (int i)
- unsigned int GetNextPatch ()
- unsigned int GetNextObsPoly ()
- bool AddPatch (unsigned int id)
- bool AddPoly (unsigned int id)
- void ResetIndex ()
- void ResetIndexObsPoly ()

Public Attributes

- Listid patchlistid
- Listid polygonlistid

4.111.1 Detailed Description

Voxel

4.111.2 Constructor & Destructor Documentation

4.111.2.1 Voxel::Voxel (Box & *box*)

4.111.2.2 Voxel::Voxel ()

4.111.2.3 Voxel::Voxel (const Voxel & *v*)

4.111.2.4 Voxel::~Voxel () [inline]

4.111.3 Member Function Documentation

4.111.3.1 bool Voxel::AddPatch (unsigned int *id*)

Adds one patch (only the id) to list_id_patch

4.111.3.2 bool Voxel::AddPoly (unsigned int *id*)

Adds one patch (only the id) to list_id_obspoly

4.111.3.3 std::vector<unsigned int>* Voxel::GetIdPatchList () [inline]

4.111.3.4 std::vector<unsigned int>* Voxel::GetIdPolyList () [inline]

return pointer to list_id_obspoly;

4.111.3.5 unsigned int Voxel::GetIdVoxel () const [inline]

The real voxel's id from scene, for indexing purpose

4.111.3.6 unsigned int Voxel::GetNextObsPoly ()

4.111.3.7 unsigned int Voxel::GetNextPatch ()

4.111.3.8 int Voxel::GetSizePatchList () const [inline]

4.111.3.9 int Voxel::GetSizePolygonList () const [inline]

Returns the number of polygons in the voxel

4.111.3.10 Vertex* Voxel::GetVertex (int *i*)

4.111.3.11 void Voxel::ResetIndex () [inline]

Resets the index of the patch list

4.111.3.12 void Voxel::ResetIndexObsPoly () [inline]

Resets the index of the obspolylist

4.111.3.13 void Voxel::SetBoundBox (float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*)

print voxel info

4.111.3.14 void Voxel::SetBoundBox (Box * *b*)

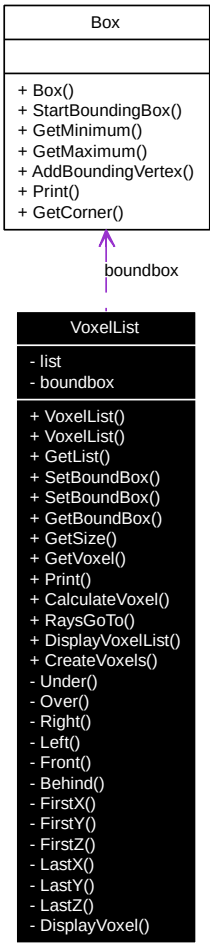
4.111.4 Member Data Documentation

4.111.4.1 Listid Voxel::patchlistid

4.111.4.2 Listid Voxel::polygonlistid

4.112 VoxelList Class Reference

Collaboration diagram for VoxelList:



Public Member Functions

- VoxelList ()
- VoxelList (Box ∗)
- std::vector< Voxel > ∗ GetList ()
- bool SetBoundBox (Box ∗)
- bool SetBoundBox (float, float, float, float, float, float)
- Box ∗ GetBoundBox ()
- int GetSize () const
- Voxel ∗ GetVoxel (unsigned int)
- void Print ()
- int CalculateVoxel (Vertex ∗)
- int RaysGoTo (Ray ∗)

- void DisplayVoxelList (Ogre::SceneManager *scenemanager=NULL)
- bool CreateVoxels ()

4.112.1 Detailed Description

this class represents a scene voxels list

4.112.2 Constructor & Destructor Documentation

4.112.2.1 VoxelList::VoxelList ()

4.112.2.2 VoxelList::VoxelList (Box *)

4.112.3 Member Function Documentation

4.112.3.1 int VoxelList::CalculateVoxel (Vertex *)

calculate the voxel that vertex go into, called by AddPatch2Voxel

4.112.3.2 bool VoxelList::CreateVoxels ()

return the next voxel that rays go to

4.112.3.3 void VoxelList::DisplayVoxelList (Ogre::SceneManager * *scenemanager* = NULL)

return the next voxel that rays go to

4.112.3.4 Box* VoxelList::GetBoundBox () [inline]

return pointer to boundbox of scene

4.112.3.5 std::vector<Voxel>* VoxelList::GetList () [inline]

return the pointer to the list of voxels

4.112.3.6 int VoxelList::GetSize () const [inline]

return pointer to boundbox of scene

4.112.3.7 Voxel* VoxelList::GetVoxel (unsigned *int*)

return pointer to boundbox of scene

4.112.3.8 void VoxelList::Print ()

print stats

4.112.3.9 int VoxelList::RaysGoTo (Ray *)

return the next voxel that rays go to

4.112.3.10 bool VoxelList::SetBoundBox (float, float, float, float, float, float)

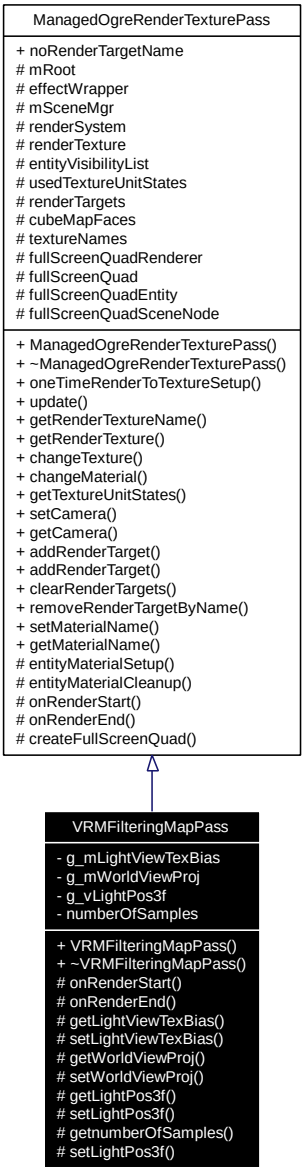
4.112.3.11 bool VoxelList::SetBoundBox (Box *)

set the boundbox

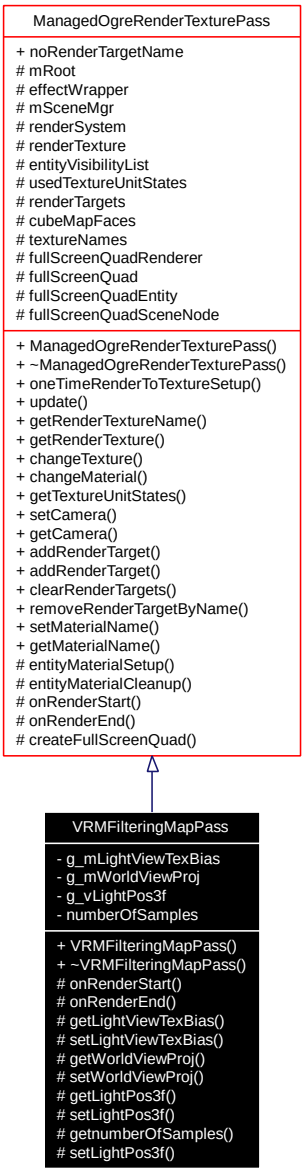
4.113 VRMFilteringMapPass Class Reference

Filters the shadow.

Inheritance diagram for VRMFilteringMapPass:



Collaboration diagram for VRMFilteringMapPass:



Public Member Functions

- VRMFilteringMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~VRMFilteringMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

- void onRenderEnd (NameValuePairList *namedParams=0)
- Matrix4 getLightViewTexBias ()

  Returns the value of g_mLightViewTexBias matrix.

- void setLightViewTexBias (Matrix4 matrix4)

  Sets the value of g_mLightViewTexBias matrix.

- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

- float * getLightPos3f ()

  Returns the value of g_vLightPos3f.

- void setLightPos3f (float *vector3)

  Sets the value of g_vLightPos3f.

- int getnumberOfSamples ()

  Returns the value of numberOfSamples.

- void setLightPos3f (int nos)

  Sets the value of numberOfSamples.

4.113.1 Detailed Description

Filters the shadow.

SuperClass: ManagedOgreRenderTexturePass Class: VRMFilteringMapPass The instances of this class are to filter the hard shadow based on previously generated information. The resulting texture is a PF_-FLOAT32_RGBA type texture

4.113.2 Constructor & Destructor Documentation

4.113.2.1 VRMFilteringMapPass::VRMFilteringMapPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.
height int, The height of the texture.
texType TextureType, Texure type.
internalFormat PixelFormat, Format of the pixel.
miscParams NameValuePairList, Pairs for names and values.
fullScreenQuadRenderer bool, Do we render a full screen quad.

4.113.2.2 VRMFilteringMapPass::~VRMFilteringMapPass () [inline]

Destructor.

4.113.3 Member Function Documentation

4.113.3.1 float* VRMFilteringMapPass::getLightPos3f () [protected]

Returns the value of g_vLightPos3f.

4.113.3.2 Matrix4 VRMFilteringMapPass::getLightViewTexBias () [protected]

Returns the value of g_mLightViewTexBias matrix.

4.113.3.3 int VRMFilteringMapPass::getnumberOfSamples () [protected]

Returns the value of numberOfSamples.

4.113.3.4 Matrix4 VRMFilteringMapPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.113.3.5 void VRMFilteringMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.113.3.6 void VRMFilteringMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.113.3.7 void VRMFilteringMapPass::setLightPos3f (int *nos*) [protected]

Sets the value of numberOfSamples.

Parameters:
nos Contains the new value of numberOfSamples.

4.113.3.8 void VRMFilteringMapPass::setLightPos3f (float * *vector3*) [protected]

Sets the value of g_vLightPos3f.

Parameters:
vector3 Contains the new value of g_vLightPos3f.

4.113.3.9 void VRMFilteringMapPass::setLightViewTexBias (Matrix4 *matrix4*) [protected]

Sets the value of g_mLightViewTexBias matrix.

Parameters:
matrix4 Contains the new value of g_mLightViewTexBias matrix.

4.113.3.10 void VRMFilteringMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

Sets the value of g_mWorldViewProj matrix.

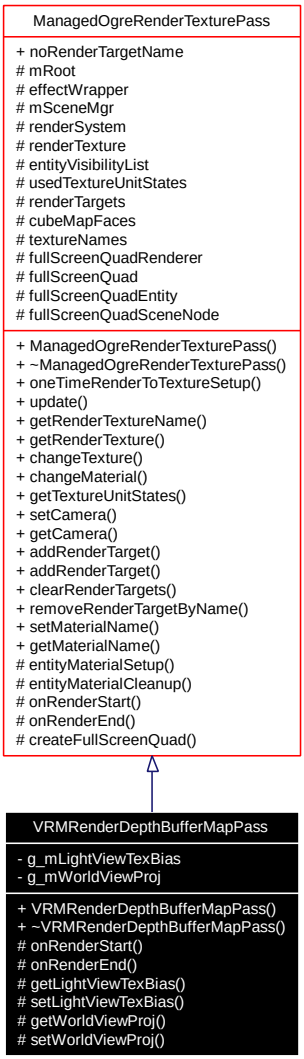
Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

4.114 VRMRenderDepthBufferMapPass Class Reference

Renders the depthbuffer of a light.

Inheritance diagram for VRMRenderDepthBufferMapPass:

Collaboration diagram for VRMRenderDepthBufferMapPass:



Public Member Functions

- VRMRenderDepthBufferMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~VRMRenderDepthBufferMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)
- Matrix4 getLightViewTexBias ()

Returns the value of g_mLightViewTexBias matrix. The light view matrix with bias.

- void setLightViewTexBias (Matrix4 matrix4)

  Sets the value of g_mLightViewTexBias matrix.

- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

4.114.1 Detailed Description

Renders the depthbuffer of a light.

SuperClass: ManagedOgreRenderTexturePass Class: VRMRenderDepthBufferMapPass The resulting texture is a PF_FLOAT32_RGBA type texture

4.114.2 Constructor & Destructor Documentation

4.114.2.1 VRMRenderDepthBufferMapPass::VRMRenderDepthBufferMapPass (Root * *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList * *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.114.2.2 VRMRenderDepthBufferMapPass::~VRMRenderDepthBufferMapPass () `[inline]`

Destructor.

4.114.3 Member Function Documentation

4.114.3.1 Matrix4 VRMRenderDepthBufferMapPass::getLightViewTexBias () [protected]

Returns the value of g_mLightViewTexBias matrix. The light view matrix with bias.

4.114.3.2 Matrix4 VRMRenderDepthBufferMapPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.114.3.3 void VRMRenderDepthBufferMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.114.3.4 void VRMRenderDepthBufferMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.114.3.5 void VRMRenderDepthBufferMapPass::setLightViewTexBias (Matrix4 *matrix4*) [protected]

Sets the value of g_mLightViewTexBias matrix.

Parameters:
matrix4 Contains the new value of g_mLightViewTexBias matrix.

4.114.3.6 void VRMRenderDepthBufferMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

Sets the value of g_mWorldViewProj matrix.

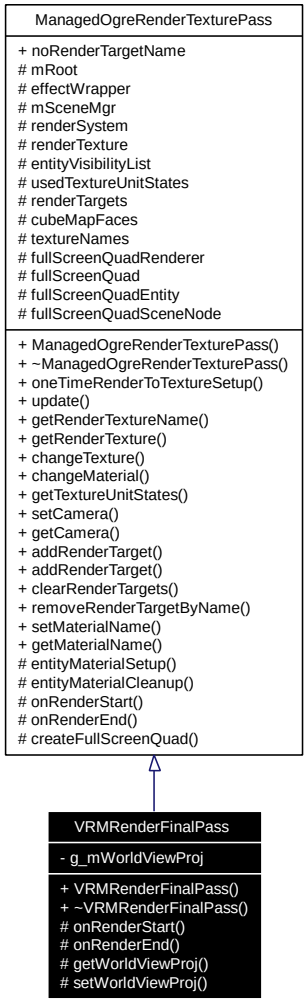
Parameters:

matrix4 Contains the new value of g_mWorldViewProj matrix.

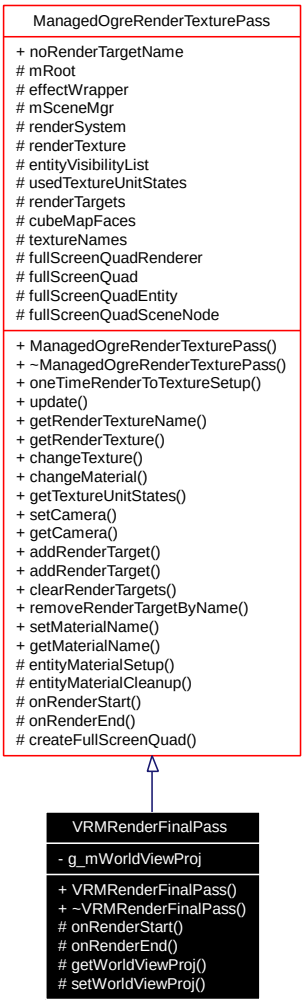
4.115 VRMRenderFinalPass Class Reference

Renders the entity into the backbuffer.

Inheritance diagram for VRMRenderFinalPass:



Collaboration diagram for VRMRenderFinalPass:



Public Member Functions

- VRMRenderFinalPass (Root *mRoot)

  Constructor.

- ~VRMRenderFinalPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)
- void onRenderEnd (NameValuePairList *namedParams=0)
- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

4.115.1 Detailed Description

Renders the entity into the backbuffer.

SuperClass: ManagedOgreRenderTexturePass Class: VRMRenderFinalPass

4.115.2 Constructor & Destructor Documentation

4.115.2.1 VRMRenderFinalPass::VRMRenderFinalPass (Root * *mRoot*)

Constructor.

Constructor.

Parameters:
mRoot Root, The root class of the Ogre system.

4.115.2.2 VRMRenderFinalPass::~VRMRenderFinalPass () [inline]

Destructor.

4.115.3 Member Function Documentation

4.115.3.1 Matrix4 VRMRenderFinalPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.115.3.2 void VRMRenderFinalPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.115.3.3 void VRMRenderFinalPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.115.3.4 void VRMRenderFinalPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

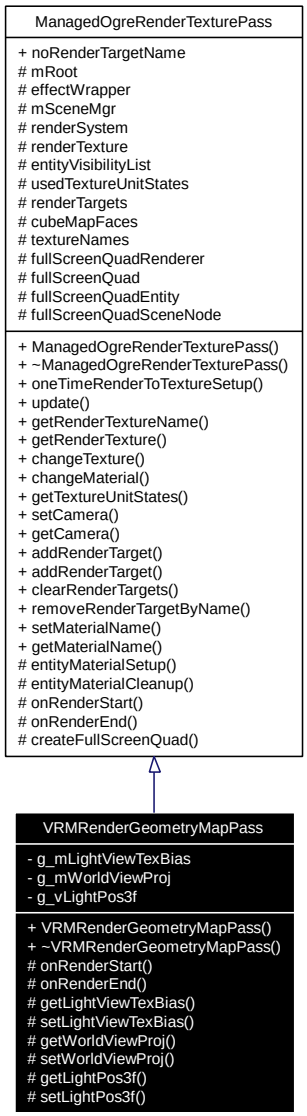
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

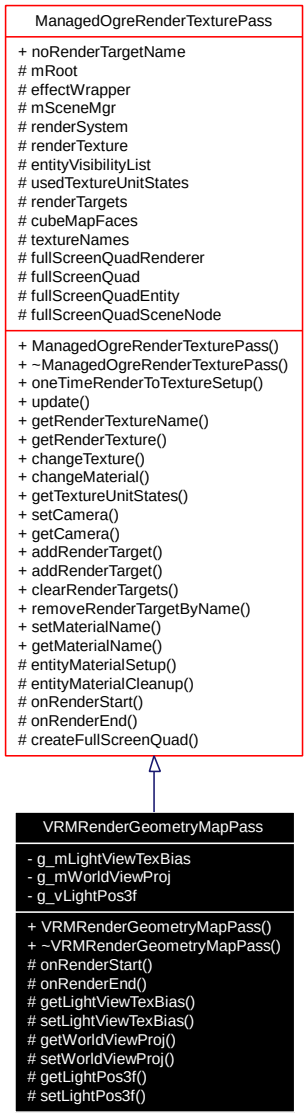
4.116 VRMRenderGeometryMapPass Class Reference

Creates a texture to store geometry information from the entity.

Inheritance diagram for VRMRenderGeometryMapPass:



Collaboration diagram for VRMRenderGeometryMapPass:



Public Member Functions

- VRMRenderGeometryMapPass (Root *mRoot, const String &renderTextureName, unsigned int width, unsigned int height, TextureType texType=TEX_TYPE_2D, PixelFormat internalFormat=PF_X8R8G8B8, const NameValuePairList *miscParams=0, bool fullScreenQuadRenderer=false)

  Constructor.

- ~VRMRenderGeometryMapPass ()

  Destructor.

Protected Member Functions

- void onRenderStart (NameValuePairList *namedParams=0)

- void onRenderEnd (NameValuePairList ∗namedParams=0)
- Matrix4 getLightViewTexBias ()

  Returns the value of g_mLightViewTexBias matrix.

- void setLightViewTexBias (Matrix4 matrix4)

  Sets the value of g_mLightViewTexBias matrix.

- Matrix4 getWorldViewProj ()

  Returns the value of g_mWorldViewProj matrix.

- void setWorldViewProj (Matrix4 matrix4)

  Sets the value of g_mWorldViewProj matrix.

- float ∗ getLightPos3f ()

  Returns the value of g_vLightPos3f.

- void setLightPos3f (float ∗vector3)

  Sets the value of g_vLightPos3f.

4.116.1 Detailed Description

Creates a texture to store geometry information from the entity.

SuperClass: ManagedOgreRenderTexturePass Class: VRMRenderGeometryMapPass The resulting texture is a PF_FLOAT32_RGBA type texture

4.116.2 Constructor & Destructor Documentation

4.116.2.1 VRMRenderGeometryMapPass::VRMRenderGeometryMapPass (Root ∗ *mRoot*, const String & *renderTextureName*, unsigned int *width*, unsigned int *height*, TextureType *texType* = `TEX_TYPE_2D`, PixelFormat *internalFormat* = `PF_X8R8G8B8`, const NameValuePairList ∗ *miscParams* = `0`, bool *fullScreenQuadRenderer* = `false`)

Constructor.

Constructor.

Parameters:

mRoot Root, The root class of the Ogre system.

renderTextureName String, The target of the rendering.

width int, The width of the texture.

height int, The height of the texture.

texType TextureType, Texure type.

internalFormat PixelFormat, Format of the pixel.

miscParams NameValuePairList, Pairs for names and values.

fullScreenQuadRenderer bool, Do we render a full screen quad.

4.116.2.2 VRMRenderGeometryMapPass::~VRMRenderGeometryMapPass () [inline]

Destructor.

4.116.3 Member Function Documentation

4.116.3.1 float* VRMRenderGeometryMapPass::getLightPos3f () [protected]

Returns the value of g_vLightPos3f.

4.116.3.2 Matrix4 VRMRenderGeometryMapPass::getLightViewTexBias () [protected]

Returns the value of g_mLightViewTexBias matrix.

4.116.3.3 Matrix4 VRMRenderGeometryMapPass::getWorldViewProj () [protected]

Returns the value of g_mWorldViewProj matrix.

4.116.3.4 void VRMRenderGeometryMapPass::onRenderEnd (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs after the render-texture object is updated place all cleanup code here.

Parameters:
namedParams Contains named parameters for rendering end.

Reimplemented from ManagedOgreRenderTexturePass.

4.116.3.5 void VRMRenderGeometryMapPass::onRenderStart (NameValuePairList * *namedParams* = 0) [protected, virtual]

Runs before the render-texture object is updated place all shader setup here.

Parameters:
namedParams A list of named parameters for rendering setup.

Reimplemented from ManagedOgreRenderTexturePass.

4.116.3.6 void VRMRenderGeometryMapPass::setLightPos3f (float * *vector3*) [protected]

Sets the value of g_vLightPos3f.

Parameters:
vector3 Contains the new value of g_vLightPos3f.

4.116.3.7 void VRMRenderGeometryMapPass::setLightViewTexBias (Matrix4 *matrix4*) [protected]

Sets the value of g_mLightViewTexBias matrix.

Parameters:
matrix4 Contains the new value of g_mLightViewTexBias matrix.

4.116.3.8 void VRMRenderGeometryMapPass::setWorldViewProj (Matrix4 *matrix4*) [protected]

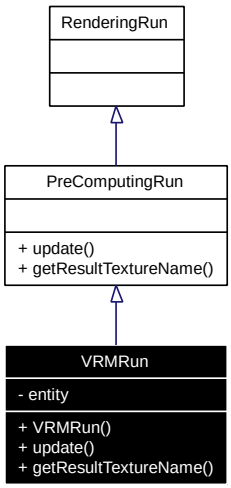
Sets the value of g_mWorldViewProj matrix.

Parameters:
matrix4 Contains the new value of g_mWorldViewProj matrix.

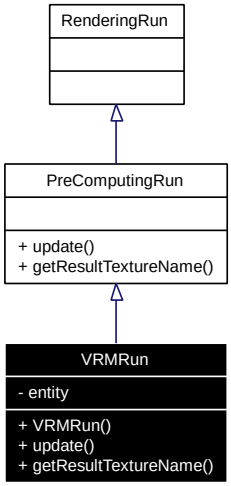
4.117 VRMRun Class Reference

Computes the complete direct irradiance caused by 'area' light sources for an entity..

Inheritance diagram for VRMRun:



Collaboration diagram for VRMRun:



Public Member Functions

- VRMRun (Entity *entity, unsigned int width, unsigned int height)
- void update ()
- const String & getResultTextureName ()

4.117.1 Detailed Description

Computes the complete direct irradiance caused by 'area' light sources for an entity..

SuperClass: PreComputingRun Class: VRMRun The resulting texture is a PF_FLOAT32_RGBA type texture

4.117.2 Constructor & Destructor Documentation

4.117.2.1 VRMRun::VRMRun (Entity * *entity*, unsigned int *width*, unsigned int *height*)

Constructor.

Parameters:

entity Entity, The owner entity of an entity-bound precomputing run.

width int, The width of the texture.

height int, The height of the texture.

4.117.3 Member Function Documentation

4.117.3.1 const String& VRMRun::getResultTextureName () [virtual]

Returns with the created texture.

Reimplemented from PreComputingRun.

4.117.3.2 void VRMRun::update () [virtual]

Recalculates the passes.

Implements PreComputingRun.

Index