Path Map Module Reference Manual

Generated by Doxygen 1.4.6-NO

Thu Apr 27 17:17:41 2006

Contents

Chapter 1

Path Map Module Hierarchical Index

1.1 Path Map Module Class Hierarchy

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

Chapter 2

Path Map Module Class Index

2.1 Path Map Module Class List

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

Chapter 3

Path Map Module Class Documentation

3.1 HitRec Class Reference

Result record for a ray-casting operation.

Public Member Functions

- **HitRec** ()
  constructor

Public Attributes

- **Intersectable** ∗ **object**
  Intersectable(p. 7) *hit by the ray.*
- **Vector point**
  point of the intersection
- **Vector normal**
  normal at the intersection
- **Vector texUV**
  texture coordinates at the intersection
- const **Material** ∗ **material**
  Material(p. 12) *of the* **Intersectable**(p. 7) *that was hit.*
- float **depth**
  distance from ray origin
- bool **isIntersect**
  valid flag

3.1.1 Detailed Description

Result record for a ray-casting operation.

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

- HitRec.hpp

3.2 Intersectable Class Reference

Base interface for all object types that can be ray-traced. The most important implementing classes are **TriangleMesh**(p. 26) and Transformed (usually referring to a **TriangleMesh**(p. 26)). A **KDTree**(p. 8) will build a hierarchy of Intersectables. TraingleMesh::Patch is also an implementing class, and a **TriangleMesh**(p. 26) contains a **KDTree**(p. 8) of patches.

Inherited by Cylinder, Occluder, PlaneXAligned, PlaneYAligned, PlaneZAligned, Sphere, Transformed, Triangle, **TriangleMesh**, and **TriangleMesh::Patch**.

Public Member Functions

- virtual bool **intersect** (const **Ray** &ray, float &depth, float rayMin, float rayMax)=0

  pure virtual function, must be implemented to carry out the intersection test

- virtual float **getSurfaceArea** ()

  return the surface area

- virtual void **sampleSurface** (**Radion** &radion)

  return a random surface point

Public Attributes

- BoundingBox **bbox**

  bounding box for the kd-tree

- unsigned int **lastTestedRayId**

  result of the last intersection test

3.2.1 Detailed Description

Base interface for all object types that can be ray-traced. The most important implementing classes are **TriangleMesh**(p. 26) and Transformed (usually referring to a **TriangleMesh**(p. 26)). A **KDTree**(p. 8) will build a hierarchy of Intersectables. TraingleMesh::Patch is also an implementing class, and a **TriangleMesh**(p. 26) contains a **KDTree**(p. 8) of patches.

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

- Intersectable.hpp
- Intersectable.cpp

3.3 KDTree Class Reference

Private Member Functions

- void **deleteLeaves** (unsigned int nodeID)
- void **build** (unsigned int nodeID, unsigned int *boundsArray, unsigned int **nObjects**, BoundingBox &limits, unsigned char axisNmask)

  recursive algorithm to build the tree bits 1,0 identify cut axis to be used

- void **quickSort** (unsigned int *lo0, unsigned int *hi0, unsigned char axis)

  qSort an array segment of object extremes

- bool **compare** (unsigned int aIndex, unsigned int bIndex, unsigned char axis)

  compare two object extrema according to the coordinate specified

- float **getBoundValue** (unsigned int *extremeIndex, unsigned char axis)

  return the value of the object extreme referenced

- unsigned int * **findBound** (unsigned int *extremeArrayStart, unsigned int nBounds, float loc, unsigned char axis)

  binary search: finds the position of value 'loc' in a sorted array partition

- bool **isLeaf** (unsigned int xnode)

  tell if a node is a leaf

- bool **followChildren** (unsigned int &parent, unsigned int &leftChild, unsigned int &rightChild)
- bool **getFreeNodes** (unsigned int parent, unsigned int &leftFree, unsigned int &rightFree)
- void **createBoundingBox** (**Intersectable** **iObjects, int **nObjects**)

  accumulate object extrema

- unsigned int **makePointer** (unsigned int originalNode)

Private Attributes

- unsigned int **nCacheLines**

  cache line sized memory segments used

- unsigned int **maxNCacheLines**

  nodes per cache line

- unsigned int **nCacheLineNodes**

  nodes per cache line

- unsigned int **nCacheLineBytes**

  bytes per cache line

- BoundingBox **boundingBox**

  bounding box containing all objects

- float * **poolNodeTable**

memory allocated for nodes

- float * **nodeTable**

  array aligned on cache lines

- unsigned int **nObjects**

  objects(p. 9)

- **Intersectable** ** **objects**

  array objects

- Heap< unsigned int > * **freeNodes**

  minimum heap to store indices of free nodes during the build

- **KDTree::TraverseStack** * **traverseStack**

  stack to implement the recursive traversal algorithm

Classes

- struct **TraverseStack**

  stack to implement the recursive traversal algorithm

3.3.1 Detailed Description

This class implements a kd-tree. Splitting, termination and cutting of empty space is based on cost estimation. Memory representation uses cache-line sized sub-trees and mapping of an unbalanced tree into an array.

3.3.2 Member Function Documentation

3.3.2.1 void KDTree::build (unsigned int *nodeID*, unsigned int * *boundsArray*, unsigned int *nObjects*, BoundingBox & *limits*, unsigned char *axisNmask*) [private]

recursive algorithm to build the tree bits 1,0 identify cut axis to be used

bits 5,4,3 indicate z,y,x failed cut attempts

Parameters:

nodeID node index of kd-tree node to be created

boundsArray array of duplicated object indices, first bit tells mimima and maxima apart

nObjects **objects**(p. 9) (boundsArray's size is 2*nObjects)

limits kd-tree node volume

3.3.2.2 bool KDTree::compare (unsigned int *aIndex*, unsigned int *bIndex*, unsigned char *axis*) [inline, private]

compare two object extrema according to the coordinate specified

this is more complicated then one would predict. rules are:

- if values are significantly different, no problem
- the minimum of a patch is smaller than its maximum
- if a maximum and a minimum are near, the other extrema of the patches decide
- if all four extrema are near, the 2 ends of a patch have to be simultaneusly smaller or bigger than the other two ends

3.3.2.3 void KDTree::deleteLeaves (unsigned int *nodeID*) [private]

traces the tree defined by root node index 'nodeID', and frees the memory occupied by the object lists

3.3.2.4 bool KDTree::followChildren (unsigned int & *parent*, unsigned int & *leftChild*, unsigned int & *rightChild*) [inline, private]

retrieve the node indices corresponding to the children of the specified node return false if the node is a leaf

3.3.2.5 bool KDTree::getFreeNodes (unsigned int *parent*, unsigned int & *leftFree*, unsigned int & *rightFree*) [inline, private]

retrieve the node indices corresponding to the children of the specified node if they are not suitable to be the root of a free sub-tree, return false

3.3.2.6 unsigned int KDTree::makePointer (unsigned int *originalNode*) [inline, private]

If the node referenced by oNode is on the last level of a cache-line sized sub-tree, a pointer to a free node is placed, and the index of this free node is returned. Else, originalNode is returned.

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

- KDTree.hpp
- KDTree.cpp

3.4 KDTree::TraverseStack Struct Reference

stack to implement the recursive traversal algorithm

Public Attributes

- float **rayMin**
  minimum distance from origin
- float **rayMax**
  maximum distance from origin
- unsigned int **tNode**
  node index of kd-tree node to be traced

3.4.1 Detailed Description

stack to implement the recursive traversal algorithm

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

- KDTree.hpp

3.5 Material Class Reference

Material(p. 12) class for ray tracing. A material that can be assigned to Intersectables. Implements [lambert + phong + ideal reflection + refraction + emission] model. Descendants may override this by texturing (**TexturedMaterial**(p. 24)) or procedural texturing.

Inherited by **TexturedMaterial**, and **WoodMaterial**.

Protected Attributes

- **Vector kd**
  diffuse coefficient
- **Vector ks**
  specular energy reflected for perpendicular light
- float **ns**
  specularity
- **Vector ki**
  ideal
- **Vector kr**
  refractive
- float **rf**
  refraction ratio
- float **ad**
  diffuse albedo
- float **as**
  specular albedo
- float **ai**
  ideal albedo
- float **ar**
  refraction albedo
- **Vector ed**
  diffuse emission
- **Vector es**
  directional (near surface normal) emission
- float **ens**
  directional (near surface normal) emission exponent
- float **aed**

diffuse emission albedo

- float **aes**

  directional emission albedo

3.5.1 Detailed Description

Material(p. 12) class for ray tracing. A material that can be assigned to Intersectables. Implements [lambert + phong + ideal reflection + refraction + emission] model. Descendants may override this by texturing (**TexturedMaterial**(p. 24)) or procedural texturing.

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

- Material.hpp
- Constants.cpp

3.6 PathMapEffect Class Reference

Main class for the PRM computation and usage application. This class encapsulates all resources needed for computing PRMs and using them in the final rendering. PRM resources may be generated, saved to files, or restored.

Public Member Functions

- **PathMapEffect** (LPDIRECT3DDEVICE9 **device**)

  constructor: allocates all resources

- ∼**PathMapEffect** (void)

  destructor: releases all resources

- void **render** ()

  renders the scene using the currently selected method

- void **renderWithPRM** ()

  renders the scene, usings the PRMs for indirect illumination

- void **showPRMTexture** ()

  displays a part of a PRM texture

- void **move** (float fElapsedTime)

  moves the virtual world objects (camera, light)

- void **savePathMaps** ()

  store all precomputed data for the scene in folder prm

- void **loadPathMaps** ()

  restore all precomputed data for the scene from folder prm

Static Public Member Functions

- static void **addUiParameters** (Parameters ∗**parameters**)

  adds controls for the user-adjustable application parameters

Public Attributes

- CFirstPersonCamera ∗ **camera**

  camera

- CFirstPersonCamera ∗ **lightCamera**

  primary light source (can be moved just like the real camera, and can be used as the camera)

Private Member Functions

- void **loadMesh** (LPCWSTR fileName)

  private method that loads a mesh and its textures

- LPDIRECT3DTEXTURE9 **loadTexture** (LPCWSTR fileName)

  private method to load a texture

- void **releaseTextures** ()

  release material texture resources

- void **releaseMeshes** ()

  release mesh resources

- void **releaseEntities** ()

  release entities

- void **renderFullScreen** (float depth=0.0f)
- void **createPRMTextures** ()

  allocates PRM resources for every entity

- void **sampleSurfaceRadion** (**Radion** &starter)

  finds a random entry point (all entities considered with equal probability)

- void **sampleSurfaceRadionUniform** (**Radion** &starter)

  finds a random entry point (all entities considered with probability proporstional to surface area)

- void **shootRadionBush** (**Radion** &starter, std::vector< **Radion** > &bushRadions)

  shoot virtual light sources from original entry point and add them to the vector

- int **clusterRadions** (**Radion** *partition, int psize, char axis)

  sort radions into initial, uniform length clusters

- void **clusterRadionsKMeans** ()

  use K-means clustering to cluster radions

- void **precompute** ()

  perform computaions: generate entry points, render PRMs

- void **uploadRadions** ()

  fill 'radionsTexture' from 'bushStarters'

- void **fillRadionPosArray** (void *pData)

  *fill *pData (will point to locked 'starterVertexBuffer') from 'bushStarters'*

Static Private Member Functions

- static void **sampleShootingDiffuseDirection** (int depth, const **Vector** &normal, **Vector** &outDir)

  generates a random direction (cosine distribution) near a normal vector

- static void **sampleSphereDirection** (**Vector** &outDir)

  generates a random direction an the unit sphere

- static void **drawFullScreenQuad** (LPDIRECT3DDEVICE9 **device**, float depth=0.0f, float fLeftU=0.0f, float fTopV=0.0f, float fRightU=1.0f, float fBottomV=1.0f)

  render a full-screen quad

Private Attributes

- LPDIRECT3DDEVICE9 **device**

  pointer to main DX device

- LPD3DXEFFECT **effect**

  pointer to main DX effect

- LPDIRECT3DSURFACE9 **frameColorBuffer**
- LPDIRECT3DSURFACE9 **frameDepthStencilBuffer**
- LPDIRECT3DTEXTURE9 **depthMapTexture**

  depth map texture

- LPDIRECT3DSURFACE9 **depthMapDepthStencilBuffer**

  depth map texture's surface

- LPDIRECT3DTEXTURE9 **fakeTexture**

  depth map dummy render texture

- LPDIRECT3DSURFACE9 **fakeSurface**

  depth map dummy render surface

- LPDIRECT3DSURFACE9 **prmBlendingDepthStencilBuffer**

  stencil buffer for rendering a virtual light source to the PRM

- LPDIRECT3DVERTEXBUFFER9 **starterVertexBuffer**

  vertex buffer with entry point positions, for entry point visualization

- LPDIRECT3DTEXTURE9 **weightsTexture**

  render target texture to which current entry point weights are computed

- LPDIRECT3DSURFACE9 **weightsSurface**

  weights texture's surface

- LPDIRECT3DTEXTURE9 **sysMemWeightsTexture**

  weights texture copy in system mem

- LPDIRECT3DSURFACE9 **sysMemWeightsSurface**

  surface of weights texture copy in system mem

- LPDIRECT3DTEXTURE9 **radionTexture**

  texture containing entry point data, input for weight computation

- LPDIRECT3DSURFACE9 **radionSurface**

  entry point texture's surface

- **KDTree** $*$ **kdtree**

  the kd-tree that contains the scene geometry in raytraceable format

- **PathMapEffect::Method method**

  clever enum for supported final rendering methods

- std::vector< **RenderMesh** $*$ > **renderMeshes**

  vector of loaded meshes

- std::vector< LPDIRECT3DTEXTURE9 > **materialTextures**

  vector of loaded textures (textures specified in mesh's material will be loaded here)

- LPDIRECT3DTEXTURE9 **emptyTexture**

  for materials with no texture we will render with this texture

- std::vector< Entity > **entities**

  vector of virtual world objects

- float **weights** [NCLUSTERS]

  the array of averaged cluster weights

- unsigned int **clusterLenghts** [NCLUSTERS]

  array thet contasins the number of entry points in every cluster (entry points are stored continously [bushStarters, starterVertexBuffer])

- float **sumSurfaceArea**

  summed surface area of all entities

- std::vector< **Radion** > **bushStarters**

  entry points

Static Private Attributes

- static Parameters $*$ **parameters** = NULL

  pointer to global user-adjustable application parameters object

Friends

- struct **Entity**

  struct that represents a virtual world object: a mesh and a model-world transformation matrix

Classes

- class **Method**

  clever enum for supported final rendering methods

- struct **RenderMesh**

  struct containing all mesh related data. Will be filled from X files in **PathMapEffect**(p. 14) *constructor*

3.6.1 Detailed Description

Main class for the PRM computation and usage application. This class encapsulates all resources needed for computing PRMs and using them in the final rendering. PRM resources may be generated, saved to files, or restored.

3.6.2 Member Function Documentation

3.6.2.1 void PathMapEffect::renderFullScreen (float *depth* = `0.0f`) `[private]`

renders a full screen quad, invoking the pixel shader for all pixels used for rendering the environment map to the background

3.6.3 Member Data Documentation

3.6.3.1 LPDIRECT3DSURFACE9 PathMapEffect::frameColorBuffer `[private]`

frame color buffer surface saved before render-to-texture, and restored as the render target for the final rendering to the screen

3.6.3.2 LPDIRECT3DSURFACE9 PathMapEffect::frameDepthStencilBuffer `[private]`

frame depth buffer surface saved before render-to-texture, and restored as the render target for the final rendering to the screen

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

- PathMapEffect.h
- PathMapEffect.cpp

3.7 PathMapEffect::Method Class Reference

clever enum for supported final rendering methods

Static Public Attributes

- static const **Method PRM**
  method constants

Static Private Attributes

- static const wchar_t * **methodNames** [10]
  method name strings to be displayed on screen

3.7.1 Detailed Description

clever enum for supported final rendering methods

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

- PathMapEffect.h
- PathMapEffect.cpp

3.8 PathMapEffect::RenderMesh Struct Reference

struct containing all mesh related data. Will be filled from X files in **PathMapEffect**(p. 14) constructor

Public Member Functions

- HRESULT **setVertexFormat** (DWORD fvf, LPDIRECT3DDEVICE9 **device**)
  rebuild D3D to have differnet vertex format
- void **buildEdgeVertexBuffer** (LPDIRECT3DDEVICE9 **device**)
  compute line primitives to edgeVertexBuffer

Public Attributes

- unsigned int **nSubsets**
  number of submeshes (with possible different material)
- LPD3DXMESH **mesh**
  D3D mesh.
- LPD3DXBUFFER **materialBuffer**
  D3D material buffer.
- D3DXMATERIAL * **materials**
  D3D material array.
- LPDIRECT3DTEXTURE9 * **textures**
  D3D textures for the materials.
- **Material** * **rayTraceMaterial**
  material used in for ray tracing
- **TriangleMesh** * **rayTraceMesh**
  the raytracable representation of the mesh
- LPDIRECT3DVERTEXBUFFER9 **edgeVertexBuffer**
  a set of line primitives for atlas (PRM) rendering
- int **nEdges**
  number of line primitives in edgeVertexBuffer

3.8.1 Detailed Description

struct containing all mesh related data. Will be filled from X files in **PathMapEffect**(p. 14) constructor

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

- PathMapEffect.h
- PathMapEffect.cpp

3.9 Radion Class Reference

Class that describes a virtual light source on a diffuse surface.

Public Attributes

- **Vector normal**
  surface normal at the virtual light source
- **Vector radiance**
  power of the virtual light source
- **Vector position**
  position of the virtual light source
- float **probability**
  probability of generation (useful for clipping the form factor)

3.9.1 Detailed Description

Class that describes a virtual light source on a diffuse surface.

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

- Radion.hpp
- Radion.cpp

3.10 Ray Class Reference

Class that describes a ray to be traced.

Public Attributes

- **Vector dir**
  must be normalised
- int **id**
  must be initialised and unique

3.10.1 Detailed Description

Class that describes a ray to be traced.

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

- Ray.hpp

3.11 TexturedMaterial Class Reference

A **Material**(p. 12) that can be assigned to an **Intersectable**(p. 7) (typically a **TriangleMesh**(p. 26)), and contains a texture. The texture must be square (equal height and width), 32bpp, BGRA.

Inherits **Material**.

Public Member Functions

- **TexturedMaterial** (void *pdata, int pitch, int textureSize)

  Constructor.

3.11.1 Detailed Description

A **Material**(p. 12) that can be assigned to an **Intersectable**(p. 7) (typically a **TriangleMesh**(p. 26)), and contains a texture. The texture must be square (equal height and width), 32bpp, BGRA.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 TexturedMaterial::TexturedMaterial (void * *pdata*, int *pitch*, int *textureSize*)

Constructor.

Parameters:

pdata pointer to the memory containing the image

pitch Number of bytes in a row. (Containg padding zeros, if any.)

textureSize Width and height in pixels. (Width and height must be equal.)

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

- TexturedMaterial.h
- TexturedMaterial.cpp

3.12 Transformation Class Reference

3D linear transformation + translation class. Used by the ray-tracing system to store entity modelling transformations. Class Transformed is an Intersecable that refers to an **Intersectable**(p. 7) and contains a **Transformation**(p. 25).

Public Member Functions

- void **transformPoint** (const **Vector** &vIn, **Vector** &vOut) const
  vOut = Matrix ∗ vIn, matrix is on the left side
- void **transformDirection** (const **Vector** &vIn, **Vector** &vOut) const
  vOut = Matrix ∗ vIn, matrix is on the left side
- void **transformPointTransposed** (const **Vector** &vIn, **Vector** &vOut) const
  vOut = vIn ∗ Matrix, matrix is on the right side
- void **transformDirectionTransposed** (const **Vector** &vIn, **Vector** &vOut) const
  vOut = vIn ∗ Matrix, matrix is on the right side

3.12.1 Detailed Description

3D linear transformation + translation class. Used by the ray-tracing system to store entity modelling transformations. Class Transformed is an Intersecable that refers to an **Intersectable**(p. 7) and contains a **Transformation**(p. 25).

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

- Transformation.hpp

3.13 TriangleMesh Class Reference

ray-tracable representaion of a mesh **TriangleMesh**(p. 26) encapsulates a kd-tree containing triangles. It can be constructed using the vertex and index buffers of a mesh. Ray-intersection and random surface sampling are supported.

Inherits **Intersectable**.

Public Member Functions

- float **getSurfaceArea** ()

  return total surface area

- void **sampleSurface** (**Radion** &radion)

  return random surface radion

- bool **intersect** (const **Ray** &ray, float &depth, float rayMin, float rayMax)

  pure virtual function, must be implemented to carry out the intersection test

- **TriangleMesh** (**Material** *material, unsigned short *indexBuffer, unsigned int nFaces, D3DVERTEX *vertexBuffer, unsigned int nVertices)

  constructor to build a **TriangleMesh**(p. 26) *from mesh buffers. Vertex format has to be per struct D3DVERTEX.*

Private Member Functions

- void **buildAreaTree** ()

  build area tree from root

- double **buildAreaTree** (unsigned int u)

  build area tree from given node (recursively)

- void **sampleSurface** (unsigned int u, double rnd, **Radion** &radion)

  return random surface radion from area subtree under node u (recursive)

Private Attributes

- **TriangleMesh::Patch** * **meshPatches**

  A triangle with ray-intersection. mesh triangles array.

- unsigned int **nMeshPatches**

  number of mesh triangles

- unsigned int **nAreaTreeNodes**

  number of nodes in area tree (for selection $\sim$ area)

- double **surfaceArea**

total surface

- double * **areaTree**

  area tree nodes array

Classes

- class **Patch**

  A triangle with ray-intersection.

3.13.1 Detailed Description

ray-tracable representaion of a mesh **TriangleMesh**(p. 26) encapsulates a kd-tree containing triangles. It can be constructed using the vertex and index buffers of a mesh. Ray-intersection and random surface sampling are supported.

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

- TriangleMesh.h
- TriangleMesh.cpp

3.14 TriangleMesh::Patch Class Reference

A triangle with ray-intersection.

Inherits **Intersectable**.

Public Member Functions

- bool **intersect** (const **Ray** &ray, float &depth, float rayMin, float rayMax)
  pure virtual function, must be implemented to carry out the intersection test
- void **sampleSurface** (**Radion** &radion)
  return a random surface point

Public Attributes

- **Vector flatNormal**
  triangle normal
- float **hyperPlaneShiftOffset**
  triangle plane offset
- **Vector inverseVertexMatrix** [3]
  Descartes to barycentric matrix.
- unsigned short **vertexIndices** [3]
  triangle vertex indices

Static Public Attributes

- static **Vector** ∗ **meshVertices** = 0x0
  a temporary reference to the owner TriangleMesh's vertex array
- static **Vector** ∗ **meshNormals** = 0x0
  a temporary reference to the owner TriangleMesh's normal array
- static **Vector** ∗ **meshTexCoords** = 0x0
  a temporary reference to the owner TriangleMesh's texcoord array

3.14.1 Detailed Description

A triangle with ray-intersection.

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

- TriangleMesh.h
- TriangleMesh.cpp

3.15 Vector Class Reference

3D vector class with overloaded operators. Used for positions, directions, colors, etc. It has the same memory layout as D3DXVECTOR3.

3.15.1 Detailed Description

3D vector class with overloaded operators. Used for positions, directions, colors, etc. It has the same memory layout as D3DXVECTOR3.

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

- Vector.hpp
- Constants.cpp

3.16 WoodMaterial Class Reference

A procedural material.

Inherits **Material**.

3.16.1 Detailed Description

A procedural material.

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

- WoodMaterial.hpp