pgLib Reference Manual

Version 1.0

Generated by Doxygen 1.3-rc1

Sun Nov 17 20:12:33 2002

Contents

Chapter 1

pgLib Namespace Index

1.1 pgLib Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

pgLib Hierarchical Index

2.1 pgLib Class Hierarchy

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

Chapter 3

pgLib Compound Index

3.1 pgLib Compound List

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

Chapter 4

pgLib Namespace Documentation

4.1 pgIDirectX Namespace Reference

pgIDirectX is the main connection of code outside of pgLib to directx

Enumerations

- enum { CLEAR_Z = 1, CLEAR_COLOR = 2 }

  Flags to be passed to renderBegin().

Functions

- PGLIB_API void init (LPDIRECT3DDEVICE8 nDevice, LPDIRECT3D8 nD3D)

  Initializes the containter class with the given device object.

- PGLIB_API bool renderBegin (int nFlags=CLEAR_Z|CLEAR_COLOR)

  This method has to be called before a new frame is rendered.

- PGLIB_API void renderEnd ()

  This method has to be called after rendering a complete frame.

- PGLIB_API void setClearColor (const pgColor &nColor)

  Sets the color, which is used to clear the color (frame) buffer.

- PGLIB_API void setClearZ (float nZ)

  Sets the z-value which is used to clear the z-buffer.

- PGLIB_API pgColor getClearColor ()

  Returns the color, which is used to clear the color buffer.

- PGLIB_API float getClearZ ()

  Returns the value which is used to clear the z-buffer.

- PGLIB_API void setViewMatrix (const pgMatrix &nMatrix)
  Sets a new view matrix.
- PGLIB_API const pgMatrix & getViewMatrix ()
  returns a pointer to the current view (camera) matrix
- PGLIB_API const pgMatrix & getProjectionMatrix ()
  returns a pointer to the current projection matrix
- PGLIB_API void setProjectionMatrix (const pgMatrix &nMatrix)
  Sets a projection matrix directly.
- PGLIB_API void setProjectionMatrix (float nAspectRatio, float nFOV, float nNear, float nFar)
  Sets the projection matrices with the given aspect ratio, field of view, near plane and far plane.
- PGLIB_API void setAspectRatio (float nAspectRatio)
  Sets a new aspect ratio and updates the projection matrix.
- PGLIB_API float getAspectRatio ()
  Returns the currently set aspect ratio.
- PGLIB_API float getFOVY ()
  Returns the currently set field of view (FOV) for the y-axis.
- PGLIB_API float getFOVX ()
  Returns the currently set field of view (FOV) for the x-axis.
- PGLIB_API float getNearPlane ()
  Returns the distance of the near clipping plane to the camera position.
- PGLIB_API float getFarPlane ()
  Returns the distance of the far clipping plane to the camera position.
- PGLIB_API void setSkyBoxHeightFactor (float nFactor)
  Sets a factor by which the skybox follows the user camera in height.
- PGLIB_API float getSkyBoxHeightFactor ()
  Returns the skybox height factor.
- PGLIB_API void switchWireframe ()
  Switches wireframe rendering on/off.
- PGLIB_API bool getWireframe ()
  Returns true if the wireframe rendering is set to on.
- PGLIB_API void switchUpdateVisibility ()
  Switches updating object visibility on/off.
- PGLIB_API bool getUpdateVisibility ()
  Returns true if the visibility updating is set to on.

- PGLIB_API void setCameraPos (const pgVec3 &nPos)
  Sets the camera's position.
- PGLIB_API pgVec3 & getCameraPos ()
  Returns the camera's position as it was set with setCameraPos().
- PGLIB_API void createClippingPlanes (pgPlane nPlanes[6])
  Creates six clipping planes.
- PGLIB_API void setColor (D3DCOLORVALUE &nDst, const pgVec4 &nSrc)
  Sets a color in pgVec4 (pgColor) format into a D3DCOLORVALUE color value.
- PGLIB_API void setD3DVecFromVec3 (D3DVECTOR &nDst, const pgVec3 &nSrc)
  Sets a 3d vector in pgVec3 format into a D3DVECTOR value.
- PGLIB_API void setVec3FromVecD3D (pgVec3 &nDst, const D3DVECTOR &nSrc)
  Sets a 3d vector in pgVec3 format into a D3DVECTOR value.
- PGLIB_API void resetStats ()
  Resets all triangle counters.
- PGLIB_API void addLevelTris (int nNum)
  Adds triangles to the level triangle counter.
- PGLIB_API void addTerrainTris (int nNum)
  Adds triangles to the terrain triangle counter.
- PGLIB_API void addNonLevelTris (int nNum)
  Adds triangles to the non-level triangle counter.
- PGLIB_API void addParticles (int nNum)
  Adds triangles to the particle triangle counter.
- PGLIB_API int getNumLevelTris ()
  Returns how many level triangles have been counted.
- PGLIB_API int getNumTerrainTris ()
  Returns how many terrain triangles have been counted.
- PGLIB_API int getNumNonLevelTris ()
  Returns how many non-level triangles have been counted.
- PGLIB_API int getNumParticles ()
  Returns how many particle triangles have been counted.
- PGLIB_API LPDIRECT3DDEVICE8 getDevice ()
  Returns the D3D render device.
- PGLIB_API LPDIRECT3D8 getD3D ()

Returns the D3D device.

- PGLIB_API int getAvailableTextureMemory ()

  Returns the availble texture memoy in megabytes.

- PGLIB_API int getNumTextureBlendStages ()

  Returns the number of texture blending stages the device has.

- PGLIB_API bool canUseCompressedTextureFormat (pgImage::FORMAT nFormat)

  Returns true if the device is able to work with compressed textures.

4.1.1 Detailed Description

pgIDirectX is the main connection of code outside of pgLib to directx

pgIDirectX allows you to take advantage of working directly with directx. If you write new classes for pgLib surely have to work with directx. pgIDirectX allows you to access the view and projection matrix, although you should be careful when changing these, since a lot of classes depend on them.

4.1.2 Enumeration Type Documentation

4.1.2.1 anonymous enum

Flags to be passed to renderBegin().

In order to pass more than one flag combine them with the or-operator ('|')

Enumeration values:

CLEAR_Z Tells the render device to clear the z-buffer

CLEAR_COLOR Tells the render device to clear the color-buffer

Definition at line 42 of file pgIDirectX.h.

```
42          {
43             CLEAR_Z = 1,
44             CLEAR_COLOR = 2
45     };
```

4.1.3 Function Documentation

4.1.3.1 PGLIB_API int getAvailableTextureMemory ()

Returns the availble texture memoy in megabytes.

The returned value is only a very coarse value and should not be taken for exact calculations.

4.1.3.2 PGLIB_API LPDIRECT3D8 getD3D ()

Returns the D3D device.

This method returns the d3d device which was passed to init()

4.1.3.3 PGLIB_API LPDIRECT3DDEVICE8 getDevice ()

Returns the D3D render device.

This method returns the render device which was passed to init()

4.1.3.4 PGLIB_API float getFOVX ()

Returns the currently set field of view (FOV) for the x-axis.

Since normally four aspect ration won't be 1.0 the FOV for x-axis and y-axis is different.

4.1.3.5 PGLIB_API float getFOVY ()

Returns the currently set field of view (FOV) for the y-axis.

Since normally four aspect ration won't be 1.0 the FOV for x-axis and y-axis is different.

4.1.3.6 PGLIB_API void init (LPDIRECT3DDEVICE8 *nDevice*, LPDIRECT3D8 *nD3D*)

Initializes the containter class with the given device object.

This method is automatically called by pgAppStub. (If the application framework is used...)

4.1.3.7 PGLIB_API bool renderBegin (int *nFlags* = CLEAR_Z|CLEAR_COLOR)

This method has to be called before a new frame is rendered.

As flags pass: CLEAR_Z if the z-buffer should be cleared CLEAR_COLOR if the color (frame) buffer should be cleared

4.1.3.8 PGLIB_API void renderEnd ()

This method has to be called after rendering a complete frame.

This method calls DirectX to finish the scene and present the rendered image.

4.1.3.9 PGLIB_API void resetStats ()

Resets all triangle counters.

The statistics count how many triangles have been drawn for level, terrain, particles and other objects (non-level).

4.1.3.10 PGLIB_API void setCameraPos (const pgVec3 & *nPos*)

Sets the camera's position.

This method only saves the camera's position but does not change the view matrix accordingly It's up to the calling method to update the view matrix. The camera's position is only saved, because some objects need to know it.

4.1.3.11 PGLIB_API void setClearColor (const pgColor & *nColor*)

Sets the color, which is used to clear the color (frame) buffer.

pgColor takes 4 components as float values [0.0-1.0]: Red, Green, Blue and Alpha

4.1.3.12 PGLIB_API void setClearZ (float *nZ*)

Sets the z-value which is used to clear the z-buffer.

Except in very special cases you'll always want this value to be 1.0

4.1.3.13 PGLIB_API void setColor (D3DCOLORVALUE & *nDst*, const pgVec4 & *nSrc*)

Sets a color in pgVec4 (pgColor) format into a D3DCOLORVALUE color value.

Use this method to convert colors from pgLib format into DirectX format

4.1.3.14 PGLIB_API void setD3DVecFromVec3 (D3DVECTOR & *nDst*, const pgVec3 & *nSrc*)

Sets a 3d vector in pgVec3 format into a D3DVECTOR value.

Use this method to convert vectors from pgLib format into DirectX format

4.1.3.15 PGLIB_API void setSkyBoxHeightFactor (float *nFactor*)

Sets a factor by which the skybox follows the user camera in height.

Normally a skybox is fixed and the camera stays always in the center of the box. For better emmersion it can make sense to let camera move a little bit in the skybox. Be aware that if the camera moves to far away from the center, the box shape will become appearant. Default value is 0.0

4.1.3.16 PGLIB_API void setVec3FromVecD3D (pgVec3 & *nDst*, const D3DVECTOR & *nSrc*)

Sets a 3d vector in pgVec3 format into a D3DVECTOR value.

Use this method to convert vectors from pgLib format into DirectX format

4.1.3.17 PGLIB_API void setViewMatrix (const pgMatrix & *nMatrix*)

Sets a new view matrix.

Be aware, that strange things can (will) happen, if you change the view matrix during rendering a frame and do not restore it, before rendering an other object.

4.1.3.18 PGLIB_API void switchUpdateVisibility ()

Switches updating object visibility on/off.

It's up to every object that can be rendered to determine wether it really stops updating visibility or not by calling getUpdateVisibility()

4.1.3.19 PGLIB_API void switchWireframe ()

Switches wireframe rendering on/off.

It's up to every object that can be rendered to determine wether it should be rendered as wireframe by calling getWireframe()

4.2 pgIFileTool Namespace Reference

This class provides methods for working with files.

Functions

- PGLIB_API void getAllFiles (const char *nPath, const char *nExt, pgList< pgString > &nFileList)

  returns in nFileList all files which are in directory nPath and have the extension nExt

4.2.1 Detailed Description

This class provides methods for working with files.

4.2.2 Function Documentation

4.2.2.1 PGLIB_API void getAllFiles (const char * *nPath*, const char * *nExt*, pgList< pgString > & *nFileList*)

returns in nFileList all files which are in directory nPath and have the extension nExt

nExt has to be passed without dot ('.')

4.3 pgIImageTool Namespace Reference

This interface provides image utitlity methods.

Functions

- PGLIB_API pgImage * createLightmapFromHeightmap (const pgImage *nHeightmap, const pgVec3n &nDir, float nAmbient, float nDiffuse)

  calculates a lightmap image from a heightmap image

4.3.1 Detailed Description

This interface provides image utitlity methods.

4.3.2 Function Documentation

4.3.2.1 PGLIB_API pgImage* createLightmapFromHeightmap (const pgImage * *nHeightmap*, const pgVec3n & *nDir*, float *nAmbient*, float *nDiffuse*)

calculates a lightmap image from a heightmap image

Light is calculated as direction light. The light's direction is set using the nDir parameter. nAmbient and nDiffuse are used to calculate each destination pixel's luminance.

4.4 pgIResourceManager Namespace Reference

pgIResourceManager is the one and only instance for loading graphics stuff from HD.

Enumerations

- enum SOURCE { SOURCE_STD, SOURCE_BSP, SOURCE_MD2, SOURCE_OBJ }

  Source Paths.

- enum

Functions

- PGLIB_API void init ()

  Init has to be called before any resource can be loaded.

- PGLIB_API pgImage * getRawImage (const pgString &nFileName, int nWidth, int nHeight, pgImage::FORMAT nFormat, SOURCE nSource)

  Returns the image loaded from the given filename.

- PGLIB_API pgTexture * getTexture (const pgString &nFileName, SOURCE nSource, pgImage::FORMAT nFormat=pgImage::UNKNOWN)

  Returns the texture with the given name.

- PGLIB_API pgImage * getRawImage (const pgString &nFileName, int nWidth, int nHeight, pgImage::FORMAT nFormat)

  Returns the image loaded from the given filename.

- PGLIB_API pgTexture * getTexture (const pgString &nFileName, pgImage::FORMAT nFormat=pgImage::UNKNOWN)

  Returns the texture with the given name.

- PGLIB_API pgBaseMesh * getBaseMeshMD2 (const pgString &nFileName, bool nLighting=false)

  Returns the MD2 with the given name as a basemesh object.

- PGLIB_API pgMesh * getMeshOBJ (const pgString &nFileName, bool nLighting=false, bool nCreateNew=false)

  Returns the OBJ with the given name as a basemesh object.

- PGLIB_API pgAnimated * getAnimated (const pgString &nFileName)

  Returns the pgAnimated object with the given name.

- PGLIB_API pgParticleSystem * getParticleSystem (const pgString &nFileName, bool nCreateNew=false)

  Returns the ParticleSystem with the given name.

- PGLIB_API pgPathLinear * getLinearPath (const pgString &nFileName)

  Returns the LinearPath with the given name as a general path object.

- PGLIB_API pgTexture * getTexNotFound ()

  Returns the standard texture, which is used, when another texture could not be loaded.

4.4.1 Detailed Description

pgIResourceManager is the one and only instance for loading graphics stuff from HD.

Resource Types are: Raw Images Textures in format: dds, tga, jpg, bmp

4.4.2 Enumeration Type Documentation

4.4.2.1 enum pgIResourceManager::SOURCE

Source Paths.

Enumeration values:

SOURCE_STD Standard Texture Path

SOURCE_BSP BSP Object and Texture Path

SOURCE_MD2 MD2 Object Texture Path

SOURCE_OBJ Wavefront OBJ and Texture Path

Definition at line 45 of file pgIResourceManager.h.

```
45                  {
46                      SOURCE_STD,
47                      SOURCE_BSP,
48                      SOURCE_MD2,
49                      SOURCE_OBJ,
50      };
```

4.4.3 Function Documentation

4.4.3.1 PGLIB_API pgAnimated* getAnimated (const pgString & *nFileName*)

Returns the pgAnimated object with the given name.

If that object is not in memory yet, it is loaded. nFileName must refer to a .ani script in the ani directory.

4.4.3.2 PGLIB_API pgBaseMesh* getBaseMeshMD2 (const pgString & *nFileName*, bool *nLighting* = false)

Returns the MD2 with the given name as a basemesh object.

If that MD2 is not in memory yet, it is loaded.

4.4.3.3 PGLIB_API pgPathLinear* getLinearPath (const pgString & *nFileName*)

Returns the LinearPath with the given name as a general path object.

If that object is not in memory yet, it is loaded. nFileName must refer to a .path file in the path directory.

4.4.3.4 PGLIB_API pgMesh* getMeshOBJ (const pgString & *nFileName*, bool *nLighting* = false, bool *nCreateNew* = false)

Returns the OBJ with the given name as a basemesh object.

If that OBJ is not in memory yet, it is loaded.

4.4.3.5 PGLIB_API pgParticleSystem* getParticleSystem (const pgString & *nFileName*, bool *nCreateNew* = false)

Returns the ParticleSystem with the given name.

If that object is not in memory yet, it is loaded. nFileName must refer to a .ps script in the ani directory.

4.4.3.6 PGLIB_API pgImage* getRawImage (const pgString & *nFileName*, int *nWidth*, int *nHeight*, pgImage::FORMAT *nFormat*)

Returns the image loaded from the given filename.

Since it is a raw image, the size, height and format are needed too. CAUTION: the raw file has to have extension ".raw".

4.4.3.7 PGLIB_API pgImage* getRawImage (const pgString & *nFileName*, int *nWidth*, int *nHeight*, pgImage::FORMAT *nFormat*, SOURCE *nSource*)

Returns the image loaded from the given filename.

Since it is a raw image, the size, height and format are needed too. CAUTION: the raw file has to have extension ".raw".

4.4.3.8 PGLIB_API pgTexture* getTexture (const pgString & *nFileName*, pgImage::FORMAT *nFormat* = pgImage::UNKNOWN)

Returns the texture with the given name.

If that texture is not in memory yet, it is loaded from the given source. If nFormat is pgImage::UNKNOWN the format is taken from the file.

4.4.3.9 PGLIB_API pgTexture* getTexture (const pgString & *nFileName*, SOURCE *nSource*, pgImage::FORMAT *nFormat* = pgImage::UNKNOWN)

Returns the texture with the given name.

If that texture is not in memory yet, it is loaded from the given source. If nFormat is pgImage::UNKNOWN the format is taken from the file.

4.5 pgIStringTool Namespace Reference

This interface provides utility methods for string processing.

Functions

- PGLIB_API const char * readLine (FILE *nFp)

  Reads a line and returns a temporary copy for it.

- PGLIB_API const char * skipSpaces (const char *nLine)

  Skips all spaces and tabs and returns the new pointer to the same data buffer.

- PGLIB_API const char * skipNonSpaces (const char *nLine)

  Skips all non spaces and non tabs and returns the new pointer to the same data buffer.

- PGLIB_API const char * getPosAfter (const char *nLine, const char *nSearch)

  Searches for the given string and returns the position after it in nLine.

- PGLIB_API bool startsWith (const char *nStr, const char *nStart)

  Returns true if nStr starts with nStart (ignores heading white spaces).

- PGLIB_API bool startsWithIgnoreCase (const char *nStr, const char *nStart)

  Returns true if nStr starts with nStart ignoring case sensitivity (ignores heading white spaces).

- PGLIB_API bool isEmpty (const char *nStr)

  Returns true if the line is empty (white space do not count).

- PGLIB_API void removePathAndExtension (const char *nFullPath, char *nFileName)

  Takes a (probably) full path of a file and tries to remove the path and the extenions from it.

- PGLIB_API void removeExtension (const char *nFullPath, char *nFileName)

  Takes a (probably) full path of a file and tries to remove the extenions from it.

- PGLIB_API bool readVec2 (const char *nString, pgVec2 &nVec2)

  Reads a Vec2 from a String.

- PGLIB_API bool readVec3 (const char *nString, pgVec3 &nVec3)

  Reads a Vec3 from a String.

- PGLIB_API bool readVec4 (const char *nString, pgVec4 &nVec4)

  Reads a Vec4 from a String.

4.5.1 Detailed Description

This interface provides utility methods for string processing.

4.5.2 Function Documentation

4.5.2.1 PGLIB_API const char* readLine (FILE * *nFp*)

Reads a line and returns a temporary copy for it.

Be aware not to call this method from outside of the pgLib DLL, since FILE pointers may not be passed over DLL boundaries.

4.5.2.2 PGLIB_API bool readVec2 (const char * *nString*, pgVec2 & *nVec2*)

Reads a Vec2 from a String.

If reading fails, false is returned.

4.5.2.3 PGLIB_API bool readVec3 (const char * *nString*, pgVec3 & *nVec3*)

Reads a Vec3 from a String.

If reading fails, false is returned.

4.5.2.4 PGLIB_API bool readVec4 (const char * *nString*, pgVec4 & *nVec4*)

Reads a Vec4 from a String.

If reading fails, false is returned.

4.5.2.5 PGLIB_API void removeExtension (const char * *nFullPath*, char * *nFileName*)

Takes a (probably) full path of a file and tries to remove the extenions from it.

nFileName must be large enough to hold the resulting string.

4.5.2.6 PGLIB_API void removePathAndExtension (const char * *nFullPath*, char * *nFileName*)

Takes a (probably) full path of a file and tries to remove the path and the extenions from it.

nFileName must be large enough to hold the resulting string.

4.6 pgITime Namespace Reference

pgTime provides precise timing methods

Functions

- PGLIB_API void update ()

  Update must only be called once per frame (before any other pgTime method is used).

- PGLIB_API float getLastFrameTime ()

  Returns in seconds, how long the last frame took.

- PGLIB_API float getFPS ()

  returns the current frames per second based on the last frame time

- PGLIB_API pgTimeInstance getCurrentTime ()

  returns the current time in very high precision as pgTimeInstance object

- PGLIB_API float getTimeSince (const pgTimeInstance &nInst)

  returns the time if seconds since which have passed since nInst

4.6.1 Detailed Description

pgTime provides precise timing methods

pgTime uses the processors VERY HIGH PRECISSION timer functions, which are precise enough to bench even single assembler commands...

4.6.2 Function Documentation

4.6.2.1 PGLIB_API float getLastFrameTime ()

Returns in seconds, how long the last frame took.

The time difference between the last two calls to update() is returned.

Chapter 5

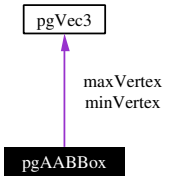
pgLib Class Documentation

5.1 pgAABBox Class Reference

Axis aligned bounding box.

`#include <pgAABBox.h>`

Collaboration diagram for pgAABBox:



Public Methods

- void setMin (const pgVec3 &nVertex)

  Sets one corner of the bounding box.

- void setMax (const pgVec3 &nVertex)

  Sets another corner of the bounding box.

- void addVertex (const pgVec3 &nVertex)

  Extends the box to contain the given vertex.

- void addBox (const pgAABBox &nOther)

  Extends the box to contain an other box.

- pgVec3 getCenter () const

  Calculates and returns the center of the box.

- void move (const pgVec3 &nMove)

  Moves the box by nMove.

- void reset ()

  Resets the box to contain everything.

- void setMid (const pgVec3 &nMid, float d)

  Resets the box to be centered at nMid and to have each side of length d.

- void setMid (const pgVec3 &nMid, float dx, float dy, float dz)

  Resets the box to be centered at nMid and to have size dx,dy,dz.

- const pgVec3 & getMin () const

  returns one point of the box

- const pgVec3 & getMax () const

  returns the other point of the box

- pgVec3 getMid () const

  returns the center of the box

5.1.1 Detailed Description

Axis aligned bounding box.

An axis aligned bounding box can be calculated very effectivly. A AABBox is defined by only two vertices (corners).

Definition at line 30 of file pgAABBox.h.

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

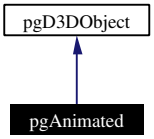
- pgAABBox.h

5.2 pgAnimated Class Reference

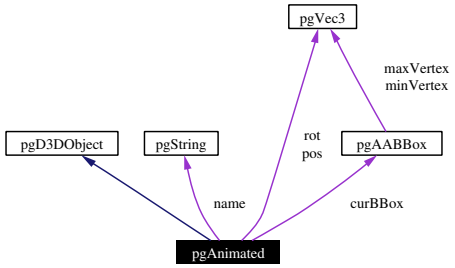
A pgAnimated object resembles an animated object such as a player character.

`#include <pgAnimated.h>`

Inheritance diagram for pgAnimated:



Collaboration diagram for pgAnimated:



Public Types

- enum FRAMEOP { BLEND, SWITCH }

  Type of operation done when changing between animation frames.

Public Methods

- bool load (const char *nFileName)

  Loads the given .char file.

- void startAnimation (int nAnim)

  Starts the animation with the given index.

- void startAnimation (const char *nName)

  starts the animation with the given name

- int getAnimationIndex (const char *nName)

  Returns the index of the given animaion name.

- pgMesh * getActiveMesh ()

  Returns the mesh that is currently in use.

- const pgAABBox & getCurrentBBox () const
  Returns the bounding box which is formed by the current mesh.
- const pgString & getName ()
  Returns the name of the object.
- bool getDidMeshLoop () const
  Returns true if the mesh animation did loop during the last update.
- void resetDidMeshLoop ()
  Resets the didloop flag.
- void setFreeze (bool nFreeze)
  If nFreeze is true, the animation is stopped (frozen).
- void setHidden (bool nHidden)
  If nHidden is true, the object is not drawn until it is unhidden.
- virtual void render ()
  Renders the current underlying mesh.
- virtual void deleteDeviceObjects ()
  The implementing object has to destroy all device dependent objects.
- virtual bool restoreDeviceObjects ()
  The implementing object has to recreate all device dependent objects.

5.2.1 Detailed Description

A pgAnimated object resembles an animated object such as a player character.

A pgAnimated object is constructed by loading an .ani file. Several animations can be stored in one object, which can each be started by calling startAnimation()

Definition at line 38 of file pgAnimated.h.

5.2.2 Member Enumeration Documentation

5.2.2.1 enum pgAnimated::FRAMEOP

Type of operation done when changing between animation frames.

Enumeration values:

BLEND Do vertex blending for smooth animation

SWITCH Just switch between frames (faster but looks awfull

Definition at line 42 of file pgAnimated.h.

```
42                  {
43                     BLEND,
44                     SWITCH
45     };
```

5.2.3 Member Function Documentation

5.2.3.1 virtual void pgAnimated::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.2.3.2 int pgAnimated::getAnimationIndex (const char * *nName*)

Returns the index of the given animaion name.

If the name is invalid -1 is returned

5.2.3.3 bool pgAnimated::getDidMeshLoop () const [inline]

Returns true if the mesh animation did loop during the last update.

For several reasons it can be useful for the owner of a pgAnimated object to know when the animation was finished and restarted because looping is active.

Definition at line 93 of file pgAnimated.h.

```
93 {  return didMeshLoop;  }
```

5.2.3.4 bool pgAnimated::load (const char * *nFileName*)

Loads the given .char file.

See the specification in the manual

5.2.3.5 virtual bool pgAnimated::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

5.2.3.6 void pgAnimated::setHidden (bool *nHidden*) [inline]

If nHidden is true, the object is not drawn until it is unhidden.

The animation goes on (is updated), but the object is not drawn. Call setFreeze(true) if the animation should stop too.

Definition at line 108 of file pgAnimated.h.

```
108 {  hidden = nHidden;  }
```

5.2.3.7 void pgAnimated::startAnimation (int *nAnim*)

Starts the animation with the given index.

If the index is invalid the animation nr 0 is started

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

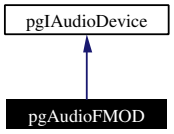
- pgAnimated.h

5.3 pgAudioFMOD Class Reference

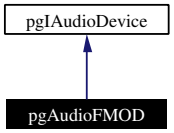
internal class for audio usage. use pgAudio for basic audio functionality

`#include <pgAudioFMOD.h>`

Inheritance diagram for pgAudioFMOD:



Collaboration diagram for pgAudioFMOD:



Protected Methods

- virtual void init ()
  must be called before any sound can be loaded or played
- virtual void cleanup ()
  should be called before quiting the programm
- virtual void update ()
  must be called every frame to update the audio engine
- virtual int getNumChannels ()
  returns the total number of channels available for playing sounds
- virtual pgISample * loadSample (const char *nName)
  loads a sound sample (.wav, .mp3, etc...)

Friends

- class pgIAudio

5.3.1 Detailed Description

internal class for audio usage. use pgAudio for basic audio functionality

Definition at line 26 of file pgAudioFMOD.h.

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

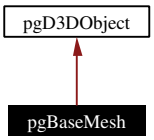
- pgAudioFMOD.h

5.4 pgBaseMesh Class Reference

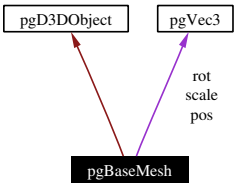
pgBaseMesh represents a renderable generic mesh without texturing.

`#include <pgBaseMesh.h>`

Inheritance diagram for pgBaseMesh:



Collaboration diagram for pgBaseMesh:



Public Types

- enum TYPE { TYPE_UNDEFINED, TYPE_INDEXED }

  Indexed or nonindexec type of mesh.

Public Methods

- void render (int nFrame)

  Renders frame number nFrame.

- void renderTweened (int nFrame0, int nFrame1, float nBlendWeight)

  Renders the mesh by tweening between the frames nFrame0 and nFrame1 by the weight nBlendWeight.

- int getNumFrames () const

  This method returns the number of keyframes in the basemesh.

- virtual void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- virtual bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

Friends

- class pgMeshUtil

5.4.1 Detailed Description

pgBaseMesh represents a renderable generic mesh without texturing.

Therefore the basemesh can be shared by many meshes each using different textures. A pgBaseMesh can consist of several frames in order to animate an object. pgBaseMesh does NOT setup any render or blending stages.

Definition at line 69 of file pgBaseMesh.h.

5.4.2 Member Enumeration Documentation

5.4.2.1 enum pgBaseMesh::TYPE

Indexed or nonindexec type of mesh.

Enumeration values:

TYPE_UNDEFINED Undefined: indicates, that the mesh has not been created yet

TYPE_INDEXED The mesh uses indexed vertices

Definition at line 75 of file pgBaseMesh.h.

```
75                {  TYPE_UNDEFINED,
76                   TYPE_INDEXED,
77                   TYPE_NONINDEXED  };
```

5.4.3 Member Function Documentation

5.4.3.1 virtual void pgBaseMesh::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.4.3.2 virtual bool pgBaseMesh::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

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

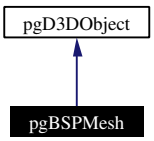
- pgBaseMesh.h

5.5 pgBSPMesh Class Reference

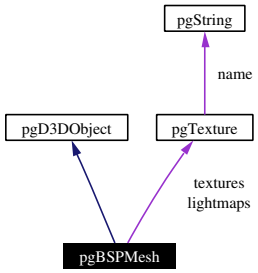
pgBSPMesh can load and render Q3 mesh objects

`#include <pgBSPMesh.h>`

Inheritance diagram for pgBSPMesh:



Collaboration diagram for pgBSPMesh:



Public Types

- enum { LIGHTMAP_SIZE = 128, MAX_VERTICES = 65536, MAX_INDICES = 65536 }

  Limits of the pgBSPMesh class.

Public Methods

- bool load (const char *nBSPFileName)

  Loads the ibsp and the corresponding shader file.

- void setSubFact (int nSubFact)

  Sets the subdivision factor for bezier patches.

- void render ()

  Renders the bsp with the current camera position.

- const char * getReportString ()

  Fills a string with statistics about the level.

- bool slideSphere (const pgVec3 &nPos0, float nRadius, const pgVec3 &nPos1, pgVec3 &nFinalPos)

  Slides a sphere with radius nRadius with planned movement from nPos0 to nPos1 through the bsp level.

- bool collideSphere (const pgVec3 &nSpherePos, float nRadius, const pgVec3 &nVelocity, pgVec3 &nNearestColSpherePos, pgVec3 &nNearestColTriPos, float &nNearestDistance, bool &nStuck)

  Returns true if the sphere collides with the bsp level.

- pgVec3 findSavePos (const pgVec3 &nSavePos, float nRadius)

  Returns a position near nSavePos which does not collide with the level.

- void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

5.5.1 Detailed Description

pgBSPMesh can load and render Q3 mesh objects

Definition at line 39 of file pgBSPMesh.h.

5.5.2 Member Enumeration Documentation

5.5.2.1 anonymous enum

Limits of the pgBSPMesh class.

Enumeration values:

LIGHTMAP_SIZE Size of one lightmap

MAX_VERTICES Maximum number of vertices to be drawn at once

MAX_INDICES Maximum number of indices to be drawn at once

Definition at line 45 of file pgBSPMesh.h.

```
45          {
46             LIGHTMAP_SIZE = 128,
47             MAX_VERTICES = 65536,
48             MAX_INDICES =  65536
49     };
```

5.5.3 Member Function Documentation

5.5.3.1 bool pgBSPMesh::collideSphere (const pgVec3 & *nSpherePos*, float *nRadius*, const pgVec3 & *nVelocity*, pgVec3 & *nNearestColSpherePos*, pgVec3 & *nNearestColTriPos*, float & *nNearestDistance*, bool & *nStuck*)

Returns true if the sphere collides with the bsp level.

During movement from nSpherePos with nVelocity Returns the point of the sphere which will collide with a triangle (nNearestColSpherePos) the point on a triangle with which it will collide (nNearestColTriPos) the distance to go until the collision (nNearestDistance) stuck as true if the sphere was already colliding on the initial position true if there actually will be a collision

5.5.3.2 void pgBSPMesh::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.5.3.3 bool pgBSPMesh::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

5.5.3.4 void pgBSPMesh::setSubFact (int *nSubFact*) [inline]

Sets the subdivision factor for bezier patches.

The number of faces created for a bezier patch is (2^n * 2^n). A value of 6 is therefore most times more than enough.

Definition at line 61 of file pgBSPMesh.h.

```
61 {  patchSubFact = nSubFact;  }
```

5.5.3.5 bool pgBSPMesh::slideSphere (const pgVec3 & *nPos0*, float *nRadius*, const pgVec3 & *nPos1*, pgVec3 & *nFinalPos*)

Slides a sphere with radius nRadius with planned movement from nPos0 to nPos1 through the bsp level.

The final position is returned in nFinalPos This method returns true if there was a collision

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

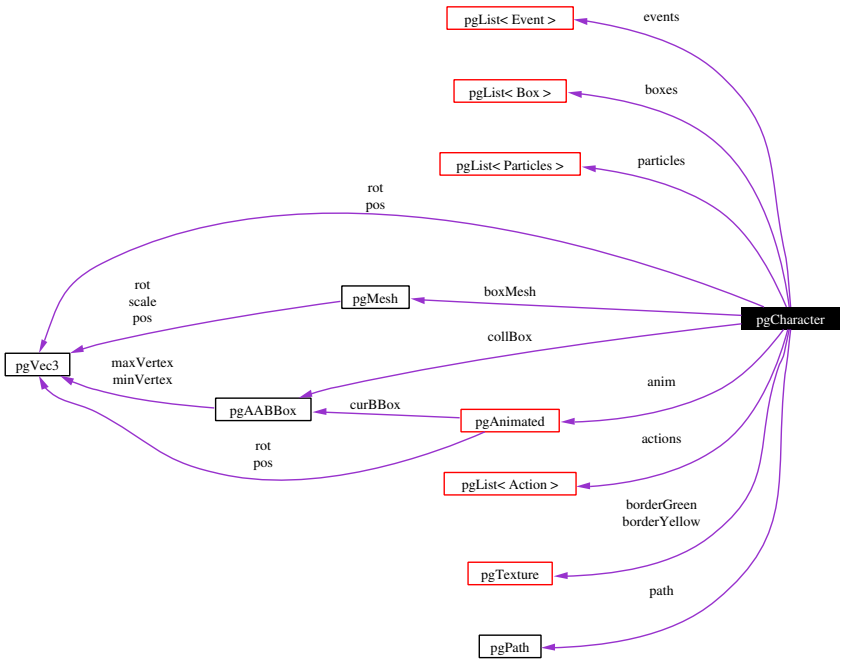
- pgBSPMesh.h

5.6 pgCharacter Class Reference

Animated Character Class.

`#include <pgCharacter.h>`

Collaboration diagram for pgCharacter:



Public Types

- enum WAY { WAY_LINE, WAY_PATH }

  Type of way a character move along.

- enum RELATION { REL_NONE, REL_ABS, REL_REL, REL_LOCAL, REL_GLOBAL }

  Type of relation between the character and a (trigger or collision) box or a particle system.

- enum MOVEMENT { MOV_FOLLOW, MOV_FIX }

  Type of relation between the character and a (trigger or collision) box.

- enum EVENT { EVENT_UNDEFINED, EVENT_STARTUP, EVENT_GUIDE, EVENT_PLAYER }

  Type of events.

Public Methods

- bool load (const char *nFileName, bool nFullPath=false)

  Loads a character from a .char file.

- void setEvent (EVENT nEvent, const pgAABBox *nBox)

  Tells the character, that an event has happened.

- EVENT getEventTypeFromString (pgString nEventStr)

  Converts a string into an event.

- void update ()

  Updates the character before it is rendered.

- bool setGameProgress (float nVal)

  Sets how far the game progress currently is.

- virtual void render ()

  Renders the character.

5.6.1 Detailed Description

Animated Character Class.

This class provides functionality to created scripted objects which can react onto events during the game.

Definition at line 41 of file pgCharacter.h.

5.6.2 Member Enumeration Documentation

5.6.2.1 enum pgCharacter::EVENT

Type of events.

Enumeration values:

EVENT_UNDEFINED No event

EVENT_STARTUP Startup event; sent after programm start

EVENT_GUIDE Event by the Guide

EVENT_PLAYER Event by the Player

Definition at line 69 of file pgCharacter.h.

```
69                  {
70                      EVENT_UNDEFINED,
71                      EVENT_STARTUP,
72                      EVENT_GUIDE,
73                      EVENT_PLAYER
74      };
```

5.6.2.2 enum pgCharacter::MOVEMENT

Type of relation between the character and a (trigger or collision) box.

Enumeration values:

MOV_FOLLOW The box moves together with the character

MOV_FIX The box does not moves with the character (stays at its initial position

Definition at line 62 of file pgCharacter.h.

```
62                      {
63                           MOV_FOLLOW,
64                           MOV_FIX
65      };
```

5.6.2.3 enum pgCharacter::RELATION

Type of relation between the character and a (trigger or collision) box or a particle system.

Enumeration values:

REL_NONE No relation

REL_ABS The box is initially positioned in absolute coordinates

REL_REL The box is initially positioned relativ to the character

REL_LOCAL Emitted particles stay in the coordinate system of the character (follow each move)

REL_GLOBAL Emitted particles move independent of the character (do not follow)

Definition at line 52 of file pgCharacter.h.

```
52                      {
53                           REL_NONE,
54                           REL_ABS,
55                           REL_REL,
56                           REL_LOCAL,
57                           REL_GLOBAL
58      };
```

5.6.2.4 enum pgCharacter::WAY

Type of way a character move along.

Enumeration values:

WAY_LINE Moves along a line (defined by points in the .CHAR file)

WAY_PATH Moves along a path (defined by many points in a .PATH file)

Definition at line 45 of file pgCharacter.h.

```
45                {
46                     WAY_LINE,
47                     WAY_PATH
48      };
```

5.6.3 Member Function Documentation

5.6.3.1 bool pgCharacter::load (const char * *nFileName*, bool *nFullPath* = false)

Loads a character from a .char file.

If nFullPath is true the filename is used directly. Otherwise it is extended by the default path and extension

5.6.3.2 void pgCharacter::setEvent (EVENT *nEvent*, const pgAABBox * *nBox*)

Tells the character, that an event has happened.

Isually every frame there will be events such as player, guide, etc...

5.6.3.3 bool pgCharacter::setGameProgress (float *nVal*)

Sets how far the game progress currently is.

Values must be between 0.0 and 1.0 this value is used to

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

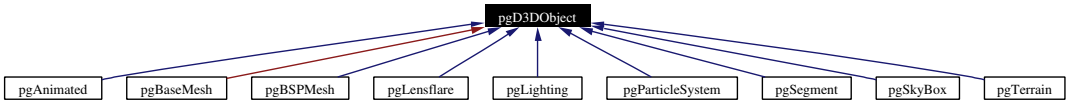
- pgCharacter.h

5.7 pgD3DObject Class Reference

This class is the base class of all pgLib objects which directly use the render device.

`#include <pgD3DObject.h>`

Inheritance diagram for pgD3DObject:



Public Methods

- bool checkDevice (const char *nMsg)

  Checks if the render device is set or can be retrieved by pgDirectX.

- virtual void deleteDeviceObjects ()=0

  The implementing object has to destroy all device dependent objects.

- virtual bool restoreDeviceObjects ()=0

  The implementing object has to recreate all device dependent objects.

5.7.1 Detailed Description

This class is the base class of all pgLib objects which directly use the render device.

Definition at line 23 of file pgD3DObject.h.

5.7.2 Member Function Documentation

5.7.2.1 bool pgD3DObject::checkDevice (const char * *nMsg*)

Checks if the render device is set or can be retrieved by pgDirectX.

If the the render device can not be retrieved, the message nMsg is logged as an error and false is returned.

5.7.2.2 virtual void pgD3DObject::deleteDeviceObjects () `[pure virtual]`

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implemented in pgBSPMesh, pgAnimated, pgBaseMesh, pgLensflare, pgLighting, pgParticleSystem, pgSegment, pgSkyBox, and pgTerrain.

5.7.2.3 virtual bool pgD3DObject::restoreDeviceObjects () [pure virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implemented in pgBSPMesh, pgAnimated, pgBaseMesh, pgLensflare, pgLighting, pgParticleSystem, pgSegment, pgSkyBox, and pgTerrain.

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

- pgD3DObject.h

5.8 pgIAudio Class Reference

Defines the basic audio interface.

`#include <pgIAudio.h>`

Collaboration diagram for pgIAudio:

Public Types

- enum TYPE { TYPE_NONE, TYPE_FMOD }

  Type of audio devices to be used.

Static Public Methods

- void init (TYPE nType)

  Must be called before any sound can be loaded or played.

- void cleanup ()

  Should be called before quiting the programm.

- void update ()

  Must be called every frame to update the audio engine.

- int getNumChannels ()

  Returns the total number of channels available for playing sounds.

- pgISample * loadSample (const char *nName)

  Loads a sound sample (.wav, .mp3, etc...).

5.8.1 Detailed Description

Defines the basic audio interface.

This class is the entry point for audio support in pgLib, such as loading audio samples.

Definition at line 29 of file pgIAudio.h.

5.8.2 Member Enumeration Documentation

5.8.2.1 enum pgIAudio::TYPE

Type of audio devices to be used.

Enumeration values:

TYPE_NONE No Audio Device

TYPE_FMOD FMOD Audio Engine

Definition at line 33 of file pgIAudio.h.

```
33                {
34                    TYPE_NONE,
35                    TYPE_FMOD
36    };
```

5.8.3 Member Function Documentation

5.8.3.1 void pgIAudio::init (TYPE *nType*) [static]

Must be called before any sound can be loaded or played.

Currently only FMOD is supported as underlying audio engine.

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

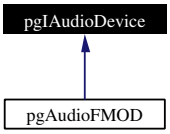
- pgIAudio.h

5.9 pgIAudioDevice Class Reference

This interface is used internally by pgIAudio.

`#include <pgIAudioDevice.h>`

Inheritance diagram for pgIAudioDevice:



Protected Methods

- virtual void init ()=0
  must be called before any sound can be loaded or played
- virtual void cleanup ()=0
  should be called before quiting the programm
- virtual void update ()=0
  must be called every frame to update the audio engine
- virtual int getNumChannels ()=0
  returns the total number of channels available for playing sounds
- virtual pgISample * loadSample (const char *nName)=0
  loads a sound sample (.wav, .mp3, etc...)

Friends

- class pgIAudio

5.9.1 Detailed Description

This interface is used internally by pgIAudio.

All audiodevices which should be created by pgIAudio have to implement this interface

Definition at line 27 of file pgIAudioDevice.h.

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

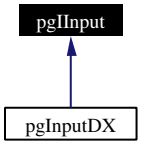
- pgIAudioDevice.h

5.10 pgIInput Class Reference

This class defines a basic input device.

`#include <pgIInput.h>`

Inheritance diagram for pgIInput:



Public Types

- enum TYPE { KEYBOARD = 1, MOUSE = 2 }
  Input type.
- enum BUTTON { BUTTON_LEFT = 1, BUTTON_MIDDLE = 2, BUTTON_RIGHT = 4 }
  Mouse Buttons.
- enum KEY
  Key definitions (identical to dxinput).

Public Methods

- virtual bool init (int nTypes, HINSTANCE hInst, HWND hWnd)=0
  Has to be called before the first update and get methods can be called.
- virtual void cleanup ()=0
  Should be called at the end of the program.
- virtual void update ()=0
  Update() has to be called every frame before any get methods are called.
- virtual bool isKeyDown (KEY nKey) const=0
  Returns true if the given key is currently pressed down.
- virtual bool isKeyNewDown (KEY nKey) const=0
  Returns true if the given key is currently down and was not down last frame.
- virtual int getMouseX () const=0
  Returns the x-position of the mouse in screen coordinates.
- virtual int getMouseY () const=0
  Returns the y-position of the mouse in screen coordinates.

- virtual bool isButtonDown (BUTTON nButton) const=0

  Returns true if the given button is currently pressed down.

- virtual void processWindowMsg (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)=0

  Processes windows messages.

5.10.1 Detailed Description

This class defines a basic input device.

Definition at line 21 of file pgIInput.h.

5.10.2 Member Enumeration Documentation

5.10.2.1 enum pgIInput::BUTTON

Mouse Buttons.

Enumeration values:

BUTTON_LEFT Left mouse button

BUTTON_MIDDLE Middle mouse button

BUTTON_RIGHT Right mouse button

Definition at line 32 of file pgIInput.h.

```
32                  {
33                  BUTTON_LEFT = 1,
34                  BUTTON_MIDDLE = 2,
35                  BUTTON_RIGHT = 4
36      };
```

5.10.2.2 enum pgIInput::TYPE

Input type.

Enumeration values:

KEYBOARD Creates a keyboard input device

MOUSE Creates a mouse input device

Definition at line 25 of file pgIInput.h.

```
25                {
26                  KEYBOARD = 1,
27                  MOUSE = 2
28      };
```

5.10.3 Member Function Documentation

5.10.3.1 virtual void pgIInput::processWindowMsg (HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*) [pure virtual]

Processes windows messages.

For some device type - such as mouse input - it is necessary that the input device has access to the window procedure. This method is automatically called by the framework for every message that is received by the application.

Implemented in pgInputDX.

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

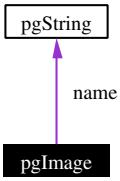
- pgIInput.h

5.11 pgImage Class Reference

pgImage objects store images

`#include <pgImage.h>`

Collaboration diagram for pgImage:



Public Types

- enum FORMAT { UNKNOWN = D3DFMT_UNKNOWN, RGB888 = D3DFMT_R8G8B8, ARGB8888 = D3DFMT_A8R8G8B8, XRGB8888 = D3DFMT_X8R8G8B8, RGB565 = D3DFMT_R5G6B5, XRGB1555 = D3DFMT_X1R5G5B5, ARGB1555 = D3DFMT_A1R5G5B5, ARGB4444 = D3DFMT_A4R4G4B4, A8 = D3DFMT_A8, P8 = D3DFMT_P8, L8 = D3DFMT_L8, DXT1 = D3DFMT_DXT1, DXT2 = D3DFMT_DXT2, DXT3 = D3DFMT_DXT3, DXT4 = D3DFMT_DXT4, DXT5 = D3DFMT_DXT5 }

  Image format specification.

Public Methods

- pgImage ()

  Creates an undefinded image (size 0∗0).

- pgImage (int nWidth, int nHeight, FORMAT nFormat, const pgString &nName, unsigned char ∗nData=NULL, bool nCopy=true)

  Creates an image with size nWidth ∗ nHeight and format nFormat.

- int getWidth () const

  Returns the width of the image in pixels.

- int getHeight () const

  Returns the height of the image in pixels.

- const unsigned char ∗ getData () const

  Returns a pointer to the image's pixels.

- unsigned char ∗ getData ()

  Returns a pointer to the image's pixels.

- int getPixelSize () const

  Returns the number of bytes a single pixel takes in the image.

- bool isCompressed () const

  Returns true if the current image format is a compressed format.

- const pgString & getName () const

  Returns the name of the image which was set during construction.

Static Public Methods

- int getPixelSize (FORMAT nFormat)

  Toolmethod to calculate the pixelsize from a format.

- pgString getFormatString (D3DFORMAT nFormat)

  Returns the name of the format as a string.

- pgString getFormatString (FORMAT nFormat)

  Returns the name of the format as a string.

- D3DFORMAT getD3DFormat (FORMAT nFormat)

  Converts a D3D pixel format into a pgLib pixel format.

- bool isCompressed (FORMAT nFormat)

  Returns true if the passed image format is a compressed format.

5.11.1 Detailed Description

pgImage objects store images

Each image is defined by its resolution and format

Definition at line 28 of file pgImage.h.

5.11.2 Member Enumeration Documentation

5.11.2.1 enum pgImage::FORMAT

Image format specification.

Enumeration values:

UNKNOWN Undefined Format

RGB888 24-bit RGB

ARGB8888 32-bit ARGB

XRGB8888 32-bit RGB

RGB565 16-bit RGB

XRGB1555 16-bit RGB

ARGB1555 16-bit ARGB

ARGB4444 16-bit ARGB

A8 8-bit Alpha

P8 8-bit Indexed

L8 8-bit Gray

DXT1 Opaque / one-bit alpha

DXT2 Explicit alpha (premultiplied)

DXT3 Explicit alpha

DXT4 Interpolated alpha (premultiplied)

DXT5 Interpolated alpha

Definition at line 32 of file pgImage.h.

```
32                  {
33                      UNKNOWN = D3DFMT_UNKNOWN,
34                      RGB888 = D3DFMT_R8G8B8,
35                      ARGB8888 = D3DFMT_A8R8G8B8,
36                      XRGB8888 = D3DFMT_X8R8G8B8,
37                      RGB565 = D3DFMT_R5G6B5,
38                      XRGB1555 = D3DFMT_X1R5G5B5,
39                      ARGB1555 = D3DFMT_A1R5G5B5,
40                      ARGB4444 = D3DFMT_A4R4G4B4,
41                      A8 = D3DFMT_A8,
42                      P8 = D3DFMT_P8,
43                      L8 = D3DFMT_L8,
44                      DXT1 = D3DFMT_DXT1,
45                      DXT2 = D3DFMT_DXT2,
46                      DXT3 = D3DFMT_DXT3,
47                      DXT4 = D3DFMT_DXT4,
48                      DXT5 = D3DFMT_DXT5
49      };
```

5.11.3 Constructor & Destructor Documentation

5.11.3.1 pgImage::pgImage (int *nWidth*, int *nHeight*, FORMAT *nFormat*, const pgString & *nName*, unsigned char * *nData* = NULL, bool *nCopy* = true)

Creates an image with size nWidth * nHeight and format nFormat.

If nData is unequal to NULL the data is taken to initialize the image object. If nCopy is true the data is copied and not just referenced.

5.11.4 Member Function Documentation

5.11.4.1 int pgImage::getPixelSize (FORMAT *nFormat*) [static]

Toolmethod to calculate the pixelsize from a format.

The pixelsize is returned as number of bytes each pixel takes using the passed format

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

- pgImage.h

5.12 pgIMathTool Class Reference

pgIMathTool provides mathematical routines for 3d calculations

`#include <pgIMathTool.h>`

Static Public Methods

- void initSinCos ()

  Initializes the tables used by sin() and cos().

- float sin (float nVal)

  Uses a sine table for fast sinus calculation.

- float cos (float nVal)

  Uses a cosine table for fast calculation.

- float random (float nMin, float nMax)

  Calculates a random number that lies between nMin and nMax.

- bool isPointInSphere (const pgVec3 &nPoint, const pgVec3 &nSpherePos, float nRadius)

  Returns true if nPoint lies in the sphere.

- bool isPointInTriangle (const pgVec3 &nPoint, const pgVec3 &nA, const pgVec3 &nB, const pgVec3 &nC)

  Returns true if the point lies inside the triangle.

- pgVec3 getClosestPointOnLine (const pgVec3 &nPoint, const pgVec3 &nA, const pgVec3 &nB)

  Returns the closest point from nPoint on the given line.

- pgVec3 getClosestPointOnTriangle (const pgVec3 &nPoint, const pgVec3 &nA, const pgVec3 &nB, const pgVec3 &nC)

  Returns the closest point from nPoint which lies on the triangles edges.

- float getDistanceToPlane (const pgVec3 &nPos, const pgPlane &nPlane)

  Returns the signed distance of the given position to the plane.

- bool isPointBehindPlane (const pgVec3 &nPos, const pgPlane &nPlane)

  Returns true if a point lies behind a plane (planes are always directed).

- bool isPointBehindPlanes (const pgVec3 &nPos, int nNumPlanes, const pgPlane *nPlanes)

  Returns true if a point lies behind all planes.

- bool isPointOnPlane (const pgVec3 &nPos, const pgPlane &nPlane)

  Returns true if a point lies on the plane.

- bool arePointsBehindPlane (int numPos, const pgVec3 *nPos, const pgPlane &nPlane)

  Returns true if at least one of the points lies behind the plane.

- bool findIntersectionRaySphere (const pgVec3 &nPos0, const pgVec3n &nDir0, const pgVec3 &nSpherePos, float nRadius, float &nT)

  Finds the intersection of a ray and a sphere.

- bool findIntersectionRayPlane (const pgPlane &nPlane, const pgVec3 &nPos0, const pgVec3n &nDir, pgVec3 &nPos, float &nT)

  Finds the intersection of a ray and a plane.

- bool findIntersectionPlanes (const pgPlane &nPlane1, const pgPlane &nPlane2, const pgPlane &nPlane3, pgVec3 &nPoint)

  Finds the intersection of three planes.

- bool findIntersectionSphereTriangle (const pgVec3 &nSpherePos, float nRadius, const pgVec3 &nDir, const pgVec3 &nCorner1, const pgVec3 &nCorner2, const pgVec3 &nCorner3, const pgPlane *nPlane, float &nDist, pgVec3 &nColSpherePos, pgVec3 &nColTrianglePos, bool &nStuck)

  Finds the intersection of the given sphere moving into direction nDir with the given triangle.

5.12.1 Detailed Description

pgIMathTool provides mathematical routines for 3d calculations

Definition at line 40 of file pgIMathTool.h.

5.12.2 Member Function Documentation

5.12.2.1 float pgIMathTool::cos (float *nVal*) [static]

Uses a cosine table for fast calculation.

Returns the cosine of nVal CAUTION: nVal has to be passed in degrees

5.12.2.2 bool pgIMathTool::findIntersectionPlanes (const pgPlane & *nPlane1*, const pgPlane & *nPlane2*, const pgPlane & *nPlane3*, pgVec3 & *nPoint*) [static]

Finds the intersection of three planes.

Returns the point in nPoint Returns true if nPoint is valid

5.12.2.3 bool pgIMathTool::findIntersectionRayPlane (const pgPlane & *nPlane*, const pgVec3 & *nPos0*, const pgVec3n & *nDir*, pgVec3 & *nPos*, float & *nT*) [static]

Finds the intersection of a ray and a plane.

Returns the position in parameter pos and signed distance of nPos0 and pos in nT Returns true as return value if there was an intersection

5.12.2.4 bool pgIMathTool::findIntersectionRaySphere (const pgVec3 & *nPos0*, const pgVec3n & *nDir0*, const pgVec3 & *nSpherePos*, float *nRadius*, float & *nT*) [static]

Finds the intersection of a ray and a sphere.

Returns the length of the ray in nT Returns true if the ray really hit the sphere

5.12.2.5 bool pgIMathTool::findIntersectionSphereTriangle (const pgVec3 & *nSpherePos*, float *nRadius*, const pgVec3 & *nDir*, const pgVec3 & *nCorner1*, const pgVec3 & *nCorner2*, const pgVec3 & *nCorner3*, const pgPlane * *nPlane*, float & *nDist*, pgVec3 & *nColSpherePos*, pgVec3 & *nColTrianglePos*, bool & *nStuck*) [static]

Finds the intersection of the given sphere moving into direction nDir with the given triangle.

If the sphere did already collide with the triangle nStuck is set to true If nPlane is not NULL it must lie inside the triangle Returns: the distance the sphere moves until it collides the position on the sphere, that will collide with the triangle the position on the triangle that will collide with the sphere sets nStuck to true if there was already a collision on the inital position

5.12.2.6 pgVec3 pgIMathTool::getClosestPointOnTriangle (const pgVec3 & *nPoint*, const pgVec3 & *nA*, const pgVec3 & *nB*, const pgVec3 & *nC*) [static]

Returns the closest point from nPoint which lies on the triangles edges.

It's assumed that nPoint lies NOT inside the triangle

5.12.2.7 void pgIMathTool::initSinCos () [static]

Initializes the tables used by sin() and cos().

This method must be called before sin() or cos() can be called.

5.12.2.8 bool pgIMathTool::isPointInTriangle (const pgVec3 & *nPoint*, const pgVec3 & *nA*, const pgVec3 & *nB*, const pgVec3 & *nC*) [static]

Returns true if the point lies inside the triangle.

It's assumed that the point always lies in the triangles plane

5.12.2.9 float pgIMathTool::sin (float *nVal*) [static]

Uses a sine table for fast sinus calculation.

Returns the sine of nVal CAUTION: nVal has to be passed in degrees

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

- pgIMathTool.h

5.13 pgInFile Class Reference

Base class for file reading classes.

`#include <pgInFile.h>`

5.13.1 Detailed Description

Base class for file reading classes.

Definition at line 20 of file pgInFile.h.

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

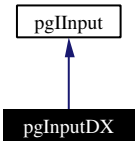
- pgInFile.h

5.14 pgInputDX Class Reference

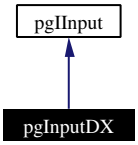
This class implements the pgInput interface using directinput.

`#include <pgInputDX.h>`

Inheritance diagram for pgInputDX:



Collaboration diagram for pgInputDX:



Public Methods

- virtual bool init (int nTypes, HINSTANCE hInst, HWND hWnd)

  Has to be called before the first update and get methods can be called.

- virtual void cleanup ()

  Should be called at the end of the program.

- virtual void update ()

  Update() has to be called every frame before any get methods are called.

- virtual bool isKeyDown (KEY nKey) const

  Returns true if the given key is currently pressed down.

- virtual bool isKeyNewDown (KEY nKey) const

  Returns true if the given key is currently down and was not down last frame.

- virtual void processWindowMsg (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM l-Param)

  Processes windows messages.

- virtual int getMouseX () const

  Returns the x-position of the mouse in screen coordinates.

- virtual int getMouseY () const

  Returns the y-position of the mouse in screen coordinates.

- virtual bool isButtonDown (BUTTON nButton) const

  Returns true if the given button is currently pressed down.

5.14.1 Detailed Description

This class implements the pgInput interface using directinput.

See pgInput for more documentation

Definition at line 27 of file pgInputDX.h.

5.14.2 Member Function Documentation

5.14.2.1 virtual void pgInputDX::processWindowMsg (HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*) [virtual]

Processes windows messages.

For some device type - such as mouse input - it is necessary that the input device has access to the window procedure. This method is automatically called by the framework for every message that is received by the application.

Implements pgIInput.

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

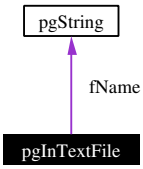
- pgInputDX.h

5.15 pgInTextFile Class Reference

This class provides basic methods for reading text files.

`#include <pgInTextFile.h>`

Collaboration diagram for pgInTextFile:



Public Methods

- pgInTextFile ()

  Creates a pgInTextFile object which is not yet attached to a file.

- virtual bool open (const pgString &nFullPath)

  Opens a text file in text mode. If the file could not be opened false is returned.

- virtual void close ()

  Closes an open text file.

- virtual int read (unsigned char *nBuffer, int nMaxRead)

  Reads up to nMaxRead bytes into nBuffer.

- virtual int getFileSize ()

  Returns the filesize of a file.

- const char * readLine ()

  Reads one line of a file and returns the data as a pointer.

- virtual bool eof ()

  Returns true if the end of the file was reached.

5.15.1 Detailed Description

This class provides basic methods for reading text files.

You should always use this class if you want to pass a text file object to a pgLib class, since standard c FILE pointers can not be passed over DLL boundaries (parts of the FILE objects data is stored as static members in the C lib and unfortunatelly not ”DLL safe”)

Definition at line 31 of file pgInTextFile.h.

5.15.2 Member Function Documentation

5.15.2.1 virtual void pgInTextFile::close () [virtual]

Closes an open text file.

If the file is not open, this call is ignored.

5.15.2.2 virtual int pgInTextFile::getFileSize () [virtual]

Returns the filesize of a file.

Before you can call this method a file must have been opened.

5.15.2.3 virtual int pgInTextFile::read (unsigned char * *nBuffer*, int *nMaxRead*) [virtual]

Reads up to nMaxRead bytes into nBuffer.

The return value resembles the number of actually read bytes.

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

- pgInTextFile.h

5.16 pgISample Class Reference

Basic sound sample interface.

`#include <pgISample.h>`

Public Methods

- virtual const char * getSampleName () const=0

  returns the name of the sound sample

- virtual void destroy ()=0

  Sets the sample to be destroyed as soon as possible.

- virtual void play (bool nLoop=false)=0

  Starts playing a sound.

- virtual void stop ()=0

  Stops playing the sound immediately.

- virtual void stopLooping ()=0

  Keeps on playing the sound is finished, but the sound will not loop anymore.

- virtual bool isPlaying ()=0

  Returns true if the sound is currently playing.

5.16.1 Detailed Description

Basic sound sample interface.

Use this interface to play samples. Samples can be played once or looped.

Definition at line 24 of file pgISample.h.

5.16.2 Member Function Documentation

5.16.2.1 virtual void pgISample::destroy () [pure virtual]

Sets the sample to be destroyed as soon as possible.

It is not garantied when the sample will be destroyed. The concrete time of destroying and freeing up of resources is up to the audio engine.

5.16.2.2 virtual void pgISample::play (bool *nLoop* = false) [pure virtual]

Starts playing a sound.

If nLoop is passed as true the sound will play looped until stop() is called.

5.16.2.3 virtual void pgISample::stop () [pure virtual]

Stops playing the sound immediately.

How ”immediately” the sound stops playing is up to the underlying audio engine. Durations around 50ms can be seen as ”very immediately”

5.16.2.4 virtual void pgISample::stopLooping () [pure virtual]

Keeps on playing the sound is finished, but the sound will not loop anymore.

After this method is called the sample keeps on playing until it is finished but will not loop again.

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

- pgISample.h

5.17 pgISettings Class Reference

Stores general Settings.

`#include <pgISettings.h>`

Static Public Methods

- void init ()

  init() has to be called before any path can be retrieved

- const char * getAppPath ()

  Returns the path where the exe file is located.

- char * tmpFullPath (const char *nFileName, const char *nExt=0)

  Returns a temporary copy of an ani-extended path. nExt is added.

- const char * getAniPath ()

  Returns the path where the ani files are stored.

- void extendToAniPath (const char *nFileName, char *nFullName)

  Extends a path by prefixing the bsp path.

- char * tmpFullAniPath (const char *nFileName, const char *nExt=0)

  Returns a temporary copy of an ani-extended path. nExt is added.

- const char * getBSPPath ()

  Returns the path where the bsp files are stored.

- void extendToBSPPath (const char *nFileName, char *nFullName)

  Extends a path by prefixing the bsp path.

- char * tmpFullBSPPath (const char *nFileName, const char *nExt=0)

  Returns a temporary copy of an bsp-extended path. nExt is added.

5.17.1 Detailed Description

Stores general Settings.

The class pgISettings has only static members, since it is not expected and can not be instantiated. Its primary work is to store global settings and do path calculations.

Definition at line 26 of file pgISettings.h.

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

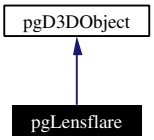
- pgISettings.h

5.18 pgLensflare Class Reference

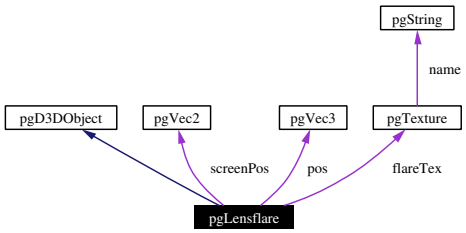
Renders a lensflare.

`#include <pgLensflare.h>`

Inheritance diagram for pgLensflare:



Collaboration diagram for pgLensflare:



Public Methods

- void setPosition (const pgVec3 &nPos)

  Sets the position of the lensflare.

- void loadStdFlareImages ()

  Loads a set of standard lensflare images.

- void render ()

  Renders the lensflare. The viewing direction is retrieved from pgIDirectX.

- void setBaseAlpha (float nAlpha)

  Sets the basic alpha value of the lensflare.

- virtual void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- virtual bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

5.18.1 Detailed Description

Renders a lensflare.

This class draws a nice lensflare onto the screen. Currently only a set of standard images can be loaded. The lensflare is drawn on a line between the set position and the center of the screen.

Definition at line 33 of file pgLensflare.h.

5.18.2 Member Function Documentation

5.18.2.1 virtual void pgLensflare::deleteDeviceObjects () [inline, virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

Definition at line 66 of file pgLensflare.h.

```
66 {}
```

5.18.2.2 virtual bool pgLensflare::restoreDeviceObjects () [inline, virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

Definition at line 67 of file pgLensflare.h.

```
67 {  return true;  }
```

5.18.2.3 void pgLensflare::setBaseAlpha (float *nAlpha*) [inline]

Sets the basic alpha value of the lensflare.

The lower the alpha value is, the less the lensflare will be visible. nAlpha must be in range [0.0-1.0]

Definition at line 63 of file pgLensflare.h.

```
63 {  baseAlpha = nAlpha;  }
```

5.18.2.4 void pgLensflare::setPosition (const pgVec3 & *nPos*) [inline]

Sets the position of the lensflare.

If the lensflare shall follow a specific object (such as a skybox) it is important, that this position lies on the object and not just in that direction.

Definition at line 47 of file pgLensflare.h.

```
47 {  pos = nPos;  }
```

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

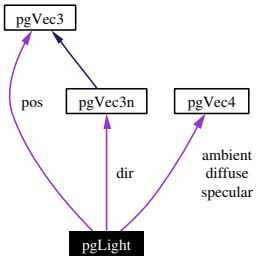
- pgLensflare.h

5.19 pgLight Class Reference

pgLight stores all information about a light

`#include <pgLight.h>`

Collaboration diagram for pgLight:



Public Types

- enum TYPE { TYPE_POINT = D3DLIGHT_POINT, TYPE_SPOT = D3DLIGHT_SPOT, TYPE_DIRECTIONAL = D3DLIGHT_DIRECTIONAL }

  Type of light source.

Public Methods

- pgLight ()

  Creates a default white light.

- pgLight (TYPE nType, const pgVec3n &nDir, const pgVec3 &nPos, const pgVec4 &nAmbient, const pgVec4 &nDiffuse, const pgVec4 &nSpecular, float nRange)

  Creates a light where the most important values are set.

- void setColors (const pgVec4 &nAmbient, const pgVec4 &nDiffuse, const pgVec4 &nSpecular)

  Sets all color values at once.

- void setType (TYPE nType)

  Sets the type of light (point, spot, directional).

- void setDirection (const pgVec3n &nDir)

  Sets the direction of the light.

- void setPosition (const pgVec3 &nPos)

  Sets the position of the light.

- void setAmbient (const pgVec4 &nAmbient)

  Sets the ambient color property.

- void setDiffuse (const pgVec4 &nDiffuse)

Sets the diffuse color property.

- void setSpecular (const pgVec4 &nSpecular)

  Sets the specular color property.

- void setRange (float nRange)

  Sets the range of the light.

Friends

- class pgLighting

5.19.1 Detailed Description

pgLight stores all information about a light

pgLight object can be a point, a spot or a directional light. Some of the properties only make sense for specific light types.

Definition at line 30 of file pgLight.h.

5.19.2 Member Enumeration Documentation

5.19.2.1 enum pgLight::TYPE

Type of light source.

Enumeration values:

TYPE_POINT Point light: direction has to be calculated for each vertex

TYPE_SPOT Spot light: the slowest type of light

TYPE_DIRECTIONAL Directional light: fasted type of light. direction is same for each vertex

Definition at line 35 of file pgLight.h.

```
35              {
36                TYPE_POINT = D3DLIGHT_POINT,
37                TYPE_SPOT = D3DLIGHT_SPOT,
38                TYPE_DIRECTIONAL = D3DLIGHT_DIRECTIONAL
39    };
```

5.19.3 Member Function Documentation

5.19.3.1 void pgLight::setDirection (const pgVec3n & *nDir*)

Sets the direction of the light.

This property is only valid for spot or directional lights.

5.19.3.2 void pgLight::setPosition (const pgVec3 & *nPos*)

Sets the position of the light.

This property is only valid for point and spot lights.

5.19.3.3 void pgLight::setRange (float *nRange*)

Sets the range of the light.

This property is only valid for point and spot lights.

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

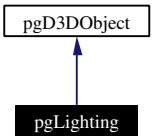
- pgLight.h

5.20 pgLighting Class Reference

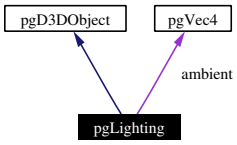
A pgLighting is a fix set of lights which can be applied to an object up to be rendered.

`#include <pgLighting.h>`

Inheritance diagram for pgLighting:



Collaboration diagram for pgLighting:



Public Methods

- void addLight (pgLight *nLight)
  Adds a light (only up to 8 lights are allowed).
- int getNumLights () const
  Returns how many lights are set.
- pgLight * getLight (int nIndex)
  Retrieves a specific light from the lighting.
- void removeLight (int nIndex)
  Removes a light.
- void removeAllLights ()
  Removes all lights at once.
- void setBaseAmbient (const pgVec4 &nColor)
  Sets the base ambient color value.
- void applyLighting ()
  Applies the lighting to the render device.
- virtual void deleteDeviceObjects ()
  The implementing object has to destroy all device dependent objects.
- virtual bool restoreDeviceObjects ()

The implementing object has to recreate all device dependent objects.

- void turnOffLighting ()

  Turns of lighting.

5.20.1 Detailed Description

A pgLighting is a fix set of lights which can be applied to an object up to be rendered.

A pgLighting can conist of up to 8 lights. Additionally a common base ambient value can be set.

Definition at line 31 of file pgLighting.h.

5.20.2 Member Function Documentation

5.20.2.1 void pgLighting::applyLighting ()

Applies the lighting to the render device.

This method activates lighting. After this method was called lighting is enabled until turnOffLighting() is called

5.20.2.2 virtual void pgLighting::deleteDeviceObjects () [inline, virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

Definition at line 75 of file pgLighting.h.

```
75 {}
```

5.20.2.3 virtual bool pgLighting::restoreDeviceObjects () [inline, virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

Definition at line 76 of file pgLighting.h.

```
76 {  return true;  }
```

5.20.2.4 void pgLighting::setBaseAmbient (const pgVec4 & *nColor*)

Sets the base ambient color value.

This ambient color value is added to all lights

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

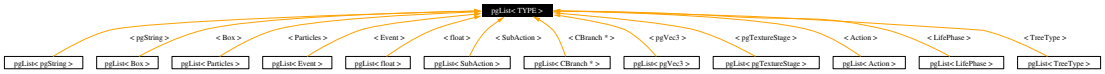
- pgLighting.h

5.21 pgList< TYPE > Class Template Reference

pgList is a template class for storing simple objects directly in the list (no pointers)

`#include <pgList.h>`

Inheritance diagram for pgList< TYPE >:



Public Methods

- int getSize () const

  Returns the number of objects currently stored in the list.

- bool isEmpty () const

  Returns true if the list is empty.

- bool checkIndex (int index) const

  Checks if the index is valid.

- void setGrowingSize (int size)

  Sets how many elements the list grows when growing is done.

- bool enlargeList (int size=-1)

  Lets the list grow 'size' elemens.

- bool enlargeListSet (int size=-1)

  Enlarge the lists by a given size and set the lists size.

- int setSize (int nSize)

  Enlarges/Reduces the list to a given size.

- virtual int addHead (const TYPE &item)

  Adds one element to the head of the list (SLOW).

- virtual int addTail (const TYPE &item)

  Adds one element to the tail of the list.

- virtual int addTailAgain ()

  Adds the last element of the list to the list again.

- virtual int addTail (const TYPE *item, int nSize)

  Adds 'nSize' elements to the tail of the list.

- virtual int insertAfter (int index, const TYPE &item)

  Inserts one element after the given index (SLOW).

- virtual int insertBefore (int index, const TYPE &item)
  Inserts one element before the given index (SLOW).
- virtual bool setList (const TYPE *item, int nSize)
  Takes 'nSize' elements starting at the given address and copies them into the list.
- virtual bool setAt (int index, const TYPE &item)
  Overwrites one element.
- virtual bool replaceIndices (int nIndexA, int nIndexB)
  Replaces the indices of the 2 items.
- virtual bool replace (const TYPE &old, const TYPE &item)
  Replaces the first found element which is equal to 'old' with 'item'.
- virtual bool replaceAll (const TYPE &old, const TYPE &item)
  Replaces all elements which are equal to 'old' with 'item'.
- bool getHead (TYPE &item) const
  Returns the first element of the list.
- TYPE & getHead () const
  Returns the first element of the list.
- bool getTail (TYPE &item) const
  Returns the last element of the list.
- TYPE & getTail () const
  Returns the last element of the list.
- bool getAt (int index, TYPE &item) const
  Returns the element at the given position.
- TYPE & getAt (int index) const
  Returns the element at the given position.
- bool removeHead ()
  Removing the first element of the list (SLOW).
- bool removeTail ()
  Removing the last element of the list.
- bool removeItem (const TYPE &item)
  Removes the first element in the list which is equal to 'item'.
- virtual bool removeIndex (int index)
  Removes the element a position 'index'.
- virtual bool removeAll ()
  Empties the list.

- int find (const TYPE &item)
  Finds the first element which is equal to 'item'.
- int findAfter (const TYPE &item, int after)
  Finds the first element which is equal to 'item' and positioned after 'after'.
- int findBefore (const TYPE &item, int before)
  Finds the first element which is equal to 'item' and positioned before 'before'.
- virtual pgList * getSubList (int from, int to)
  Returns a part of the list.
- TYPE & operator[ ] (unsigned index) const
  Gives direct access to the list members.
- pgList< TYPE > & operator= (const pgList< TYPE > &other)
  Copies a list from another list.

Friends

- bool operator== (const pgList &, const pgList &)
  Compares two lists.

5.21.1 Detailed Description

template<class TYPE> class pgList< TYPE >

pgList is a template class for storing simple objects directly in the list (no pointers)

See also: plPtrList

Definition at line 30 of file pgList.h.

5.21.2 Member Function Documentation

5.21.2.1 template<class TYPE> bool pgList< TYPE >::getAt (int *index*, TYPE & *item*) const [inline]

Returns the element at the given position.

Returns false if the list is empty

Definition at line 620 of file pgList.h.

References pgList< TYPE >::checkIndex().

```
622 {
623     if (!checkIndex(index))
624     {
625         raiseError("IndexError");
```

```
626         assert(1);
627         return false;
628     }
629     try
630     {
631         item = array[index];
632     }
633     catch(...)
634     {
635         raiseError("in Get At");
636         return false;
637     }
638     return true;
```

5.21.2.2 template<class TYPE> bool pgList< TYPE >::getHead (TYPE & *item*) const [inline]

Returns the first element of the list.

Returns false if the list is empty

Definition at line 548 of file pgList.h.

```
550 {
551     if (actualItemCount>0)
552     {
553         raiseError("Item not found");
554         return false;
555     }
556     try
557     {
558         item = array[0];
559     }
560     catch(...)
561     {
562         raiseError("in GetHead");
563         return false;
564     }
565     return true;
```

5.21.2.3 template<class TYPE> bool pgList< TYPE >::getTail (TYPE & *item*) const [inline]

Returns the last element of the list.

Returns false if the list is empty

Definition at line 584 of file pgList.h.

```
586 {
587     if (actualItemCount<1)
588     {
589         raiseError("Item not found");
590         return false;
591     }
592     try
593     {
594         item = array[actualItemCount-1];
595     }
596     catch(...)
```

```
597     {
598         raiseError("in GetTail");
599         return false;
600     }
601     return true;
```

5.21.3 Friends And Related Function Documentation

5.21.3.1 template<class TYPE> bool operator== (const pgList< TYPE > &, const pgList< TYPE > &) [friend]

Compares two lists.

Each item is compared. (therefore TYPE must have an operator==). If all items are ident, true is returned.

Definition at line 228 of file pgList.h.

```
229 { return false; }
```

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

- pgList.h

5.22 pgLog Class Reference

Use this class from anywhere to do logging into a file.

`#include <pgLog.h>`

Static Public Methods

- void init (const char *nLogFileName)

  init() has to be called before the first log can be saved.

- void info (const char *szFormat,...)

  Logs an info message.

- void error (const char *szFormat,...)

  Logs an error message.

- void trace (const char *szFormat,...)

  Writes a message to debug output in visual studio.

5.22.1 Detailed Description

Use this class from anywhere to do logging into a file.

Before the first logging method is invoked, init() must be called.

Definition at line 23 of file pgLog.h.

5.22.2 Member Function Documentation

5.22.2.1 void pgLog::error (const char * *szFormat*, ...) [static]

Logs an error message.

Parameters work same as for printf

5.22.2.2 void pgLog::info (const char * *szFormat*, ...) [static]

Logs an info message.

Parameters work same as for printf

5.22.2.3 void pgLog::init (const char * *nLogFileName*) [static]

init() has to be called before the first log can be saved.

Since most of pgLibs classes do logging during their initialization, this method has to be called, before any other initialization is done. Therefore calling pgLog::init() should be the first call of any pgLib method.

5.22.2.4 void pgLog::trace (const char * *szFormat, ...*) [static]

Writes a message to debug output in visual studio.

Parameters work same as for printf

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

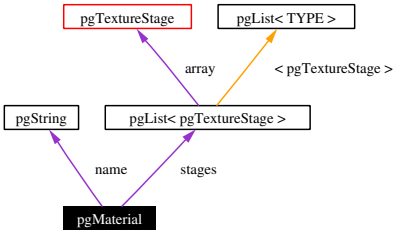
- pgLog.h

5.23 pgMaterial Class Reference

pgMaterial defines the properties of a surface

`#include <pgMaterial.h>`

Collaboration diagram for pgMaterial:



Public Types

- enum CULLING { CULL_NONE = D3DCULL_NONE, CULL_CW = D3DCULL_CW, CULL_CCW = D3DCULL_CCW }

  Type of culling performed for each triangle.

- enum BLEND { BLEND_NONE = 0, BLEND_ZERO = D3DBLEND_ZERO, BLEND_ONE = D3DBLEND_ONE, BLEND_SRCCOLOR = D3DBLEND_SRCCOLOR, BLEND_INVSRCCOLOR = D3DBLEND_INVSRCCOLOR, BLEND_SRCALPHA = D3DBLEND_SRCALPHA, BLEND_INVSRCALPHA = D3DBLEND_INVSRCALPHA, BLEND_DESTALPHA = D3DBLEND_DESTALPHA, BLEND_INVDESTALPHA = D3DBLEND_INVDESTALPHA, BLEND_DESTCOLOR = D3DBLEND_DESTCOLOR, BLEND_INVDESTCOLOR = D3DBLEND_INVDESTCOLOR }

  Type of calculation how stage sources are combined.

Public Methods

- void addStage (const pgTextureStage &nStage)

  Adds a texture stage to the material.

- void setCulling (CULLING nCull)

  Sets a culling method (clockwise, counterclockwise or none).

- D3DMATERIAL8 * getDDMaterial ()

  Returns the underlying Direct3D material which is applied to the render device.

Friends

- class pgSegment

5.23.1 Detailed Description

pgMaterial defines the properties of a surface

A pgMaterial object defines lighting properties (how the surface reacts to lights) and texture stages. Each pgSegment object must have a pgMaterial in order to be rendered.

Definition at line 34 of file pgMaterial.h.

5.23.2 Member Enumeration Documentation

5.23.2.1 enum pgMaterial::BLEND

Type of calculation how stage sources are combined.

Enumeration values:

BLEND_NONE Undefined

BLEND_ZERO Value is replaced (multiplied) by zero

BLEND_ONE Value is multiplied by one (unchanged)

BLEND_SRCCOLOR Value is multiplied by source color

BLEND_INVSRCCOLOR Value is multiplied by one minus source color

BLEND_SRCALPHA Value is multiplied by source alpha

BLEND_INVSRCALPHA Value is multiplied by one minus source alpha

BLEND_DESTALPHA Value is multiplied by destination alpha

BLEND_INVDESTALPHA Value is multiplied by one minus destination alpha

BLEND_DESTCOLOR Value is multiplied by destination color

BLEND_INVDESTCOLOR Value is multiplied by one minus destination color

Definition at line 47 of file pgMaterial.h.

```
47                  {   BLEND_NONE              = 0,
48                      BLEND_ZERO              = D3DBLEND_ZERO,
49                      BLEND_ONE               = D3DBLEND_ONE,
50                      BLEND_SRCCOLOR          = D3DBLEND_SRCCOLOR,
51                      BLEND_INVSRCCOLOR       = D3DBLEND_INVSRCCOLOR,
52                      BLEND_SRCALPHA          = D3DBLEND_SRCALPHA,
53                      BLEND_INVSRCALPHA       = D3DBLEND_INVSRCALPHA,
54                      BLEND_DESTALPHA         = D3DBLEND_DESTALPHA,
55                      BLEND_INVDESTALPHA      = D3DBLEND_INVDESTALPHA,
56                      BLEND_DESTCOLOR         = D3DBLEND_DESTCOLOR,
57                      BLEND_INVDESTCOLOR      = D3DBLEND_INVDESTCOLOR,
58                      BLEND_SRCALPHASAT       = D3DBLEND_SRCALPHASAT,
59                      BLEND_BOTHSRCALPHA      = D3DBLEND_BOTHSRCALPHA,
60                      BLEND_BOTHINVSRCALPHA   = D3DBLEND_BOTHINVSRCALPHA
61      };
```

5.23.2.2 enum pgMaterial::CULLING

Type of culling performed for each triangle.

Enumeration values:

CULL_NONE No culling performed

CULL_CW Clockwise culling

CULL_CCW Counterclockwise culling

Definition at line 40 of file pgMaterial.h.

```
40                  {  CULL_NONE = D3DCULL_NONE,
41                     CULL_CW = D3DCULL_CW,
42                     CULL_CCW = D3DCULL_CCW
43     };
```

5.23.3 Member Function Documentation

5.23.3.1 void pgMaterial::addStage (const pgTextureStage & *nStage*) [inline]

Adds a texture stage to the material.

Up to 8 texture stages are allowed Keep care that the render device can not render more stages than pgIDirectX::getNumTextureBlendStages() returns

Definition at line 88 of file pgMaterial.h.

```
88 {  stages.addTail(nStage);  }
```

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

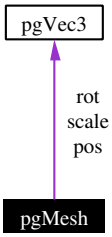
- pgMaterial.h

5.24 pgMesh Class Reference

A pgMesh is a complex 3D object which can consist of several segments.

`#include <pgMesh.h>`

Collaboration diagram for pgMesh:



Public Methods

- void addSegment (pgSegment *nSegment)

  Adds a segment to the mesh.

- int getNumFrames () const

  Returns the number of frames the segments have.

- int renderTweenedWithBlendTextures (float nFrame, int nMinFrame, int nMaxFrame, pgTexturePtr nTexture0, pgTexturePtr nTexture1, float nTexBlend)

  Renders the mesh by blending each segement between two frames.

- int renderTweened (float nFrame, int nMinFrame, int nMaxFrame)

  Renders the mesh by blending each segement between two frames.

- void render (int nFrame=0, int nSegment=-1)

  Renders the set frame. if nSegment is -1 all segments are rendered.

- pgAABBox getBBox (int nIndex)

  Returns a bounding box which contains the frame number nIndex in all segments.

Friends

- class pgMeshUtil

5.24.1 Detailed Description

A pgMesh is a complex 3D object which can consist of several segments.

A pgMesh object contains one or more segment in order to support more than one texture per mesh. A pgMesh object has to be created/loaded by pgMeshUtil.

Definition at line 31 of file pgMesh.h.

5.24.2 Member Function Documentation

5.24.2.1 void pgMesh::addSegment (pgSegment * *nSegment*) [inline]

Adds a segment to the mesh.

The mesh can constist of any number of segments

Definition at line 42 of file pgMesh.h.

```
42 {  segments.addTail(nSegment);  }
```

5.24.2.2 int pgMesh::getNumFrames () const [inline]

Returns the number of frames the segments have.

It's considered that each segment has the same number of frames.

Definition at line 55 of file pgMesh.h.

```
55 {  return segments.getSize() ? segments[0]->getNumFrames() : 0;  }
```

5.24.2.3 int pgMesh::renderTweened (float *nFrame*, int *nMinFrame*, int *nMaxFrame*)

Renders the mesh by blending each segement between two frames.

Works exactly the as renderTweenedWithBlendTextures, except that the texture settings are used from the segments' stage settings

5.24.2.4 int pgMesh::renderTweenedWithBlendTextures (float *nFrame*, int *nMinFrame*, int *nMaxFrame*, pgTexturePtr *nTexture0*, pgTexturePtr *nTexture1*, float *nTexBlend*)

Renders the mesh by blending each segement between two frames.

this methods expects the mesh to have at least nMaxFrame frames nFrame must be between 0.0 and 1.0. the method automatically determines between which two frames it has to tween nTexture0 must always be !=NULL. if nTexture1 is !=NULL and nTexBlend != 0.0 then a blending between the two textures is performed

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

- pgMesh.h

5.25 pgMeshUtil Class Reference

Creates and updates pgBaseMesh and pgMesh objects.

`#include <pgMeshUtil.h>`

Static Public Methods

- pgBaseMesh * createBox (const pgAABBox &nBox, DWORD nColor)

  Creates a box mesh with texture coordinates. All vertices get the given color.

- bool updateBox (pgBaseMesh *nMesh, const pgAABBox &nBox, DWORD nColor)

  Updates the mesh to be a box with the given coordinates.

- pgBaseMesh * loadMD2 (const char *nName, bool nLighting=false)

  Creates an pgBaseMesh from an MD2 file.

- pgMesh * loadOBJ (const char *nName, bool nLighting=true)

  Creates an pgMesh from an OBJ file.

5.25.1 Detailed Description

Creates and updates pgBaseMesh and pgMesh objects.

Definition at line 36 of file pgMeshUtil.h.

5.25.2 Member Function Documentation

5.25.2.1 pgBaseMesh* pgMeshUtil::loadMD2 (const char * *nName*, bool *nLighting* = false) [static]

Creates an pgBaseMesh from an MD2 file.

If nLighting is true a mesh with lighting support (normals, no fix vertex colors) is created

5.25.2.2 pgMesh* pgMeshUtil::loadOBJ (const char * *nName*, bool *nLighting* = true) [static]

Creates an pgMesh from an OBJ file.

If nLighting is true a mesh with lighting support (normals) is created

5.25.2.3 bool pgMeshUtil::updateBox (pgBaseMesh * *nMesh*, const pgAABBox & *nBox*, DWORD *nColor*) [static]

Updates the mesh to be a box with the given coordinates.

Number of vertices and indices must already be correct (usually the mesh will already be a box, which only gets new coordinates)

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

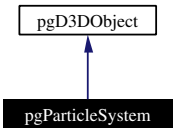
- pgMeshUtil.h

5.26 pgParticleSystem Class Reference

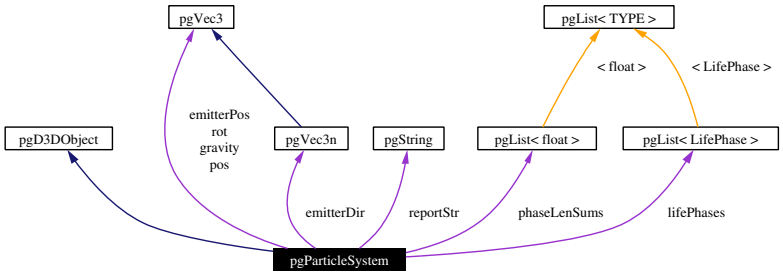
Creates and animates a particle system from a .ps file.

`#include <pgParticleSystem.h>`

Inheritance diagram for pgParticleSystem:



Collaboration diagram for pgParticleSystem:



Public Types

- enum
- enum EFFECT { EFFECT_NORMAL, EFFECT_SPHERE_NORMAL, EFFECT_SPHERE_FLEE, EFFECT_CYLINDER_NORMAL, EFFECT_CYLINDER_FLEE }

  Predefined effect type.

- enum BLEND { BLEND_UNKNOWN = 0, BLEND_ZERO = D3DBLEND_ZERO, BLEND_ONE = D3DBLEND_ONE, BLEND_SRCCOLOR = D3DBLEND_SRCCOLOR, BLEND_INVSRCCOLOR = D3DBLEND_INVSRCCOLOR, BLEND_SRCALPHA = D3DBLEND_SRCALPHA, BLEND_INVSRCALPHA = D3DBLEND_INVSRCALPHA, BLEND_DESTALPHA = D3DBLEND_DESTALPHA, BLEND_INVDESTALPHA = D3DBLEND_INVDESTALPHA, BLEND_DESTCOLOR = D3DBLEND_DESTCOLOR, BLEND_INVDESTCOLOR = D3DBLEND_INVDESTCOLOR }

  Type of calculation how stage sources are combined.

Public Methods

- bool load (const char *nFile)

  Loads settings from a .ps file.

- void setEffect (EFFECT nEffect)

This methods are only needed if the particle system is created manually.

- bool create (int nMaxParticles)

  Creates the particle system after all properties have been set.

- void cleanup ()

  Frees all resource from the particle system.

- void emitParticles (int nNum)

  Tells the particle system to emit (additional) nNum new particles during the next update.

- void update ()

  Updates the particle system. All particles are moved and "grow older".

- void render ()

  Renders the particle system.

- void startEmitting ()

  Tells the particle system to automatically emit as many particles per second as set with setEmitPerSecond().

- void stopEmitting ()

  Stops automatic particle emitting.

- bool isEmitting ()

  Returns true if the automatic particle emitting is currently active.

- virtual void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- virtual bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

5.26.1 Detailed Description

Creates and animates a particle system from a .ps file.

For more information about .ps files see the manual

Definition at line 31 of file pgParticleSystem.h.

5.26.2 Member Enumeration Documentation

5.26.2.1 enum pgParticleSystem::BLEND

Type of calculation how stage sources are combined.

Enumeration values:

BLEND_UNKNOWN Undefined

BLEND_ZERO Value is replaced (multiplied) by zero

BLEND_ONE Value is multiplied by one (unchanged)

BLEND_SRCCOLOR Value is multiplied by source color

BLEND_INVSRCCOLOR Value is multiplied by one minus source color

BLEND_SRCALPHA Value is multiplied by source alpha

BLEND_INVSRCALPHA Value is multiplied by one minus source alpha

BLEND_DESTALPHA Value is multiplied by destination alpha

BLEND_INVDESTALPHA Value is multiplied by one minus destination alpha

BLEND_DESTCOLOR Value is multiplied by destination color

BLEND_INVDESTCOLOR Value is multiplied by one minus destination color

Definition at line 65 of file pgParticleSystem.h.

```
65                 {
66                     BLEND_UNKNOWN           = 0,
67                     BLEND_ZERO              = D3DBLEND_ZERO,
68                     BLEND_ONE               = D3DBLEND_ONE,
69                     BLEND_SRCCOLOR          = D3DBLEND_SRCCOLOR,
70                     BLEND_INVSRCCOLOR       = D3DBLEND_INVSRCCOLOR,
71                     BLEND_SRCALPHA          = D3DBLEND_SRCALPHA,
72                     BLEND_INVSRCALPHA       = D3DBLEND_INVSRCALPHA,
73                     BLEND_DESTALPHA         = D3DBLEND_DESTALPHA,
74                     BLEND_INVDESTALPHA      = D3DBLEND_INVDESTALPHA,
75                     BLEND_DESTCOLOR         = D3DBLEND_DESTCOLOR,
76                     BLEND_INVDESTCOLOR      = D3DBLEND_INVDESTCOLOR,
77                     BLEND_SRCALPHASAT       = D3DBLEND_SRCALPHASAT,
78                     BLEND_BOTHSRCALPHA      = D3DBLEND_BOTHSRCALPHA,
79                     BLEND_BOTHINVSRCALPHA   = D3DBLEND_BOTHINVSRCALPHA
80     };
```

5.26.2.2 enum pgParticleSystem::EFFECT

Predefined effect type.

Enumeration values:

EFFECT_NORMAL Particles emit at the emitter and work only with gravity : DEFAULT

EFFECT_SPHERE_NORMAL Particles emit on a sphere and work only with gravity

EFFECT_SPHERE_FLEE Particles emit on a sphere and move away from the center

EFFECT_CYLINDER_NORMAL Particles emit on a cylinder and work only with gravity

EFFECT_CYLINDER_FLEE Particles emit on a cylinder and move away from the center axis

Definition at line 55 of file pgParticleSystem.h.

```
55                 {
56                    EFFECT_NORMAL,
57                    EFFECT_SPHERE_NORMAL,
58                    EFFECT_SPHERE_FLEE,
59                    EFFECT_CYLINDER_NORMAL,
60                    EFFECT_CYLINDER_FLEE,
61                    EFFECT_CYLINDER_ROTATE };
```

5.26.3 Member Function Documentation

5.26.3.1 bool pgParticleSystem::create (int *nMaxParticles*)

Creates the particle system after all properties have been set.

If the particle system was created manually (without loading from file) the creation process has to be finished by calling create().

5.26.3.2 virtual void pgParticleSystem::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.26.3.3 bool pgParticleSystem::load (const char * *nFile*)

Loads settings from a .ps file.

create() is called internally

5.26.3.4 virtual bool pgParticleSystem::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

5.26.3.5 void pgParticleSystem::setEffect (EFFECT *nEffect*) [inline]

This methods are only needed if the particle system is created manually.

See the technical manual for a description of these settings.

Definition at line 116 of file pgParticleSystem.h.

```
116 {  effect = nEffect;  }
```

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

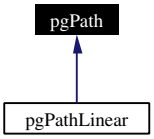
- pgParticleSystem.h

5.27 pgPath Class Reference

This class defines the basic path interface which all path classes have to implement.

`#include <pgPath.h>`

Inheritance diagram for pgPath:



Public Types

- enum TYPE { TYPE_NONE, TYPE_LINEAR }

  Types of paths.

Public Methods

- virtual pgVec3 getPosition (float nPos)=0

  Returns the interpolated position on the path.

- virtual pgVec3n getDirection (float nPos)=0

  Returns the interpolated rotation on the path.

- virtual int getPosDir (float nPos, pgVec3 &nPosition, pgVec3n &nDirection)=0

  Returns the interpolated position and rotation.

- virtual TYPE getType () const

  Returns the type of the path.

- virtual float getLength () const=0

  Returns the length of the path.

5.27.1 Detailed Description

This class defines the basic path interface which all path classes have to implement.

Some of the get-methods are not constant, since a path might want to cache request data (section of last request, etc...)

Definition at line 26 of file pgPath.h.

5.27.2 Member Enumeration Documentation

5.27.2.1 enum pgPath::TYPE

Types of paths.

Enumeration values:

TYPE_NONE Undefined Path Type

TYPE_LINEAR Linear Path

Definition at line 30 of file pgPath.h.

```
31                {
32                     TYPE_NONE,
33                     TYPE_LINEAR
```

5.27.3 Member Function Documentation

5.27.3.1 virtual pgVec3n pgPath::getDirection (float *nPos*) [pure virtual]

Returns the interpolated rotation on the path.

nPos must be in range [0.0-1.0]

Implemented in pgPathLinear.

5.27.3.2 virtual int pgPath::getPosDir (float *nPos*, pgVec3 & *nPosition*, pgVec3n & *nDirection*) [pure virtual]

Returns the interpolated position and rotation.

This method is far more efficient than calling getPosition() and getRotation() seperately nPos must be in range [0.0-1.0] Returns the index of the base vertex used

Implemented in pgPathLinear.

5.27.3.3 virtual pgVec3 pgPath::getPosition (float *nPos*) [pure virtual]

Returns the interpolated position on the path.

nPos must be in range [0.0-1.0]

Implemented in pgPathLinear.

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

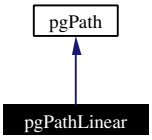
- pgPath.h

5.28 pgPathLinear Class Reference

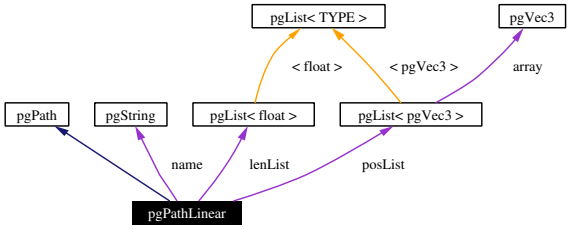
This class implements the basic pgPath interface.

`#include <pgPathLinear.h>`

Inheritance diagram for pgPathLinear:



Collaboration diagram for pgPathLinear:



Public Methods

- bool load (const char *nFileName)
  Loads a path from the file.
- void setInterpolateRotation (bool nEnable)
  Enables interpolation of the rotation.
- int getNumCorners () const
  Returns the number of corners, which build this linear path.
- const pgVec3 & getCorner (int nIndex) const
  Returns a specific corner.
- virtual pgVec3 getPosition (float nPos)
  Returns the interpolated position on the path.
- virtual pgVec3n getDirection (float nPos)
  Returns the interpolated rotation on the path.
- virtual int getPosDir (float nPos, pgVec3 &nPosition, pgVec3n &nDirection)
  Returns the interpolated position and rotation.
- virtual float getLength () const
  Returns the length of the path.

5.28.1 Detailed Description

This class implements the basic pgPath interface.

It provides a path contructed of linear subsection (polyline). the rotation can be interpolated if requested.

Definition at line 27 of file pgPathLinear.h.

5.28.2 Member Function Documentation

5.28.2.1 virtual pgVec3n pgPathLinear::getDirection (float *nPos*) [virtual]

Returns the interpolated rotation on the path.

nPos must be in range [0.0-1.0]

Implements pgPath.

5.28.2.2 virtual int pgPathLinear::getPosDir (float *nPos*, pgVec3 & *nPosition*, pgVec3n & *nDirection*) [virtual]

Returns the interpolated position and rotation.

This method is far more efficient than calling getPosition() and getRotation() seperately nPos must be in range [0.0-1.0] Returns the index of the base vertex used

Implements pgPath.

5.28.2.3 virtual pgVec3 pgPathLinear::getPosition (float *nPos*) [virtual]

Returns the interpolated position on the path.

nPos must be in range [0.0-1.0]

Implements pgPath.

5.28.2.4 bool pgPathLinear::load (const char * *nFileName*)

Loads a path from the file.

This method load a path from a file Each line must contain three float values. In order to support 3dsMAX the y & z values are exchanged.

5.28.2.5 void pgPathLinear::setInterpolateRotation (bool *nEnable*) [inline]

Enables interpolation of the rotation.

Not implemented yet.

Definition at line 46 of file pgPathLinear.h.

```
46 {  interpolRot = nEnable;  }
```

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

- pgPathLinear.h

5.29 pgPlane Class Reference

This class implements a plane.

`#include <pgPlane.h>`

5.29.1 Detailed Description

This class implements a plane.

Definition at line 24 of file pgPlane.h.

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

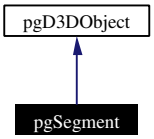
- pgPlane.h

5.30 pgSegment Class Reference

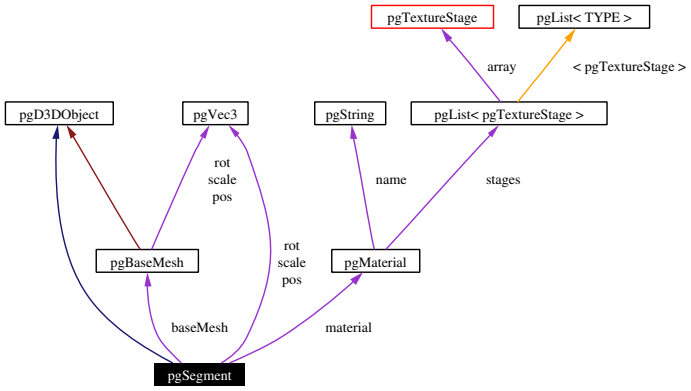
A pgSegment object can render a 3d object having exactly one material.

`#include <pgSegment.h>`

Inheritance diagram for pgSegment:



Collaboration diagram for pgSegment:



Public Types

- enum SETTINGS { SET_LIGHT = 1, SET_ZTEST = 2, SET_ZWRITE = 4, SET_ALPHATEST = 8, SET_FILL = 16, SET_TEXTURED = 32 }

  Properties of a pgSegment.

Public Methods

- void setBaseMesh (pgBaseMesh *nBaseMesh)

  Sets the base mesh, which is rendered during render().

- void setRenderSettings (int nSettings)

  Sets the current render settings. all previous render settings are lost.

- void enableRenderSettings (int nSettings)

  Enables specific render settings.

- void disableRenderSettings (int nSettings)

Disables specific render settings.

- void setMaterial (pgMaterial *nMaterial)

  Sets the material to use during rendering.

- void updateRenderSettings ()

  Updates the render settings().

- int getNumFrames () const

  Returns the number of frames which are stored in the base mesh.

- int renderTweenedWithBlendTextures (float nFrame, int nMinFrame, int nMaxFrame, pgTexturePtr nTexture0, pgTexturePtr nTexture1, float nTexBlend)

  Renders the mesh by blending each segement between two frames.

- int renderTweened (float nFrame, int nMinFrame, int nMaxFrame)

  Renders the mesh by blending each segement between two frames.

- void render (int nFrame)

  Renders the set frame.

- pgAABBox * getBBox (int nIndex)

  Returns the basemesh's bounding box.

- void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

Friends

- class pgMeshUtil

5.30.1 Detailed Description

A pgSegment object can render a 3d object having exactly one material.

For optimal rendering speed it is necessary to store triangles sorted by material. A pgSegment object has exactly one material and can therefore be rendered by one single DirectX call. In order to create/use objects consisting of more than one material use pgMesh which stores an arbitrary number of pgSegments.

Definition at line 37 of file pgSegment.h.

5.30.2 Member Enumeration Documentation

5.30.2.1 enum pgSegment::SETTINGS

Properties of a pgSegment.

Enumeration values:

SET_LIGHT Mesh will render with light

SET_ZTEST Mesh will do z-tests

SET_ZWRITE Mesh will do z-testing

SET_ALPHATEST Mesh will do alpha testing

SET_FILL Mesh will fill its triangles

SET_TEXTURED Mesh will draw textures

Definition at line 43 of file pgSegment.h.

```
43                      {
44                          SET_LIGHT       = 1,
45                          SET_ZTEST       = 2,
46                          SET_ZWRITE      = 4,
47                          SET_ALPHATEST   = 8,
48                          SET_FILL        = 16,
49                          SET_TEXTURED    = 32
50      };
```

5.30.3 Member Function Documentation

5.30.3.1 void pgSegment::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.30.3.2 int pgSegment::renderTweened (float *nFrame*, int *nMinFrame*, int *nMaxFrame*)

Renders the mesh by blending each segement between two frames.

Works exactly the as renderTweenedWithBlendTextures, except that the texture settings are used from the segments' stage settings

5.30.3.3 int pgSegment::renderTweenedWithBlendTextures (float *nFrame*, int *nMinFrame*, int *nMaxFrame*, pgTexturePtr *nTexture0*, pgTexturePtr *nTexture1*, float *nTexBlend*)

Renders the mesh by blending each segement between two frames.

this methods expects the mesh to have at least nMaxFrame frames nFrame must be between 0.0 and 1.0. the method automatically determines between which two frames it has to tween nTexture0 must always be !=NULL. if nTexture1 is !=NULL and nTexBlend != 0.0 then a blending between the two textures is performed

5.30.3.4 bool pgSegment::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

5.30.3.5 void pgSegment::setRenderSettings (int *nSettings*) [inline]

Sets the current render settings. all previous render settings are lost.

See pgSegment::SETTING for a list of all possible settings. (all settings can be combined freely)

Definition at line 68 of file pgSegment.h.

```
68 {  settings = nSettings;  }
```

5.30.3.6 void pgSegment::updateRenderSettings ()

Updates the render settings().

If any render setting of property of the set material is changed, this method has to be called in order to update the render settings

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

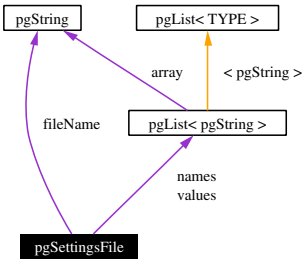
- pgSegment.h

5.31 pgSettingsFile Class Reference

This class loads settings (ini) files.

`#include <pgSettingsFile.h>`

Collaboration diagram for pgSettingsFile:



Public Methods

- void reset ()

  Removes all names and values which where retrieved by a load() call.

- bool load (const pgString &nFileName)

  Opens a setting file and reads all names and values from it.

- void setLogging (bool nSet)

  Activates or deactivates logging.

- bool getValueYes (const pgString &nName, bool &nYes) const

  Returns true if the value of nName is 'yes'.

- bool getValueString (const pgString &nName, pgString &nString) const

  Returns the value of nName as a string.

- bool getValueInt (const pgString &nName, int &nInt) const

  Returns the value of nName as an integer.

- bool getValueFloat (const pgString &nName, float &nFloat) const

  Returns the value of nName as a float.

- bool getValueVec2 (const pgString &nName, pgVec2 &nVec2) const

  Returns the value of nName as a vec2.

- bool getValueVec3 (const pgString &nName, pgVec3 &nVec3) const

  Returns the value of nName as a vec3.

- bool getValueVec4 (const pgString &nName, pgVec4 &nVec4) const

  Returns the value of nName as a vec4.

5.31.1 Detailed Description

This class loads settings (ini) files.

Each setting consists of the parts: a name and a value. Values can consists of more than one item. E.g.: a vec3 consists of three floats. In values consisting of more than one element all elements must be seperated by a space (no ”,” oder ”;”) All lines which are empty or start with # are ignored

Definition at line 36 of file pgSettingsFile.h.

5.31.2 Member Function Documentation

5.31.2.1 bool pgSettingsFile::getValueFloat (const pgString & *nName*, float & *nFloat*) const

Returns the value of nName as a float.

If the nName is not found false is returned.

5.31.2.2 bool pgSettingsFile::getValueInt (const pgString & *nName*, int & *nInt*) const

Returns the value of nName as an integer.

If the nName is not found false is returned.

5.31.2.3 bool pgSettingsFile::getValueString (const pgString & *nName*, pgString & *nString*) const

Returns the value of nName as a string.

If the nName is not found false is returned.

5.31.2.4 bool pgSettingsFile::getValueVec2 (const pgString & *nName*, pgVec2 & *nVec2*) const

Returns the value of nName as a vec2.

If the nName is not found or the value can not be converted into a vec2, false is returned.

5.31.2.5 bool pgSettingsFile::getValueVec3 (const pgString & *nName*, pgVec3 & *nVec3*) const

Returns the value of nName as a vec3.

If the nName is not found or the value can not be converted into a vec3, false is returned.

5.31.2.6 bool pgSettingsFile::getValueVec4 (const pgString & *nName*, pgVec4 & *nVec4*) const

Returns the value of nName as a vec4.

If the nName is not found or the value can not be converted into a vec4, false is returned.

5.31.2.7 bool pgSettingsFile::getValueYes (const pgString & *nName*, bool & *nYes*) const

Returns true if the value of nName is ’yes’.

If the nName is not found false is returned.

5.31.2.8 bool pgSettingsFile::load (const pgString & *nFileName*)

Opens a setting file and reads all names and values from it.

The list of names and values is not cleared, which allows to gather information from several setting files and merge them in one pgSettingsFile object.

5.31.2.9 void pgSettingsFile::setLogging (bool *nSet*) [inline]

Activates or deactivates logging.

If logging is enabled all names which are not found in the name list are logged as errors.

Definition at line 61 of file pgSettingsFile.h.

```
61 {  doLogging = nSet;  }
```

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

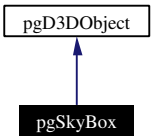
- pgSettingsFile.h

5.32 pgSkyBox Class Reference

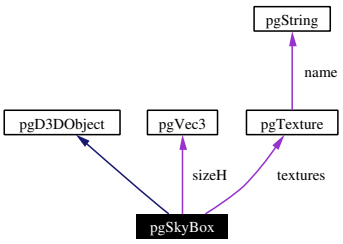
Creates and renders a skybox.

`#include <pgSkyBox.h>`

Inheritance diagram for pgSkyBox:



Collaboration diagram for pgSkyBox:



Public Methods

- pgSkyBox (bool nDownSide=true)

  If nDownSide is false the bottom texture of the skybox is not drawn.

- void setTextures (pgTexture *nFront, pgTexture *nBack, pgTexture *nLeft, pgTexture *nRight, pgTexture *nUp, pgTexture *nDown)

  Sets the textures for the six sides of the skybox cube.

- void create ()

  Creates the skybox mesh.

- void render ()

  Renders the skybox.

- void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

5.32.1 Detailed Description

Creates and renders a skybox.

The skybox is automatically rotated to the direction retrieved from pgIDirectX

Definition at line 27 of file pgSkyBox.h.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 pgSkyBox::pgSkyBox (bool *nDownSide* = true)

If nDownSide is false the bottom texture of the skybox is not drawn.

In order to save texture memory nDown should be passed as NULL if nDownSide is true.

5.32.3 Member Function Documentation

5.32.3.1 void pgSkyBox::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.32.3.2 bool pgSkyBox::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

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

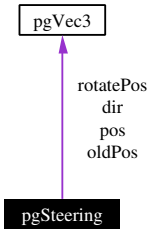
- pgSkyBox.h

5.33 pgSteering Class Reference

pgSteering provides basic camera movement.

`#include <pgSteering.h>`

Collaboration diagram for pgSteering:



Public Types

- enum MODE { MODE_INSPECT, MODE_FLY, MODE_ROTATE }

  Steering Mode.

Public Methods

- void setMode (MODE nMode)

  Sets a new movement mode.

- void switchMode ()

  Switches to the next movement mode.

- void setNewPosition (const pgVec3 &nPos)

  Sets the current position.

- void setOldPosition (const pgVec3 &nPos)

  Sets the position before update was called.

- void setDirection (const pgEuler &nDir)

  Sets a new direction.

- void setInspectModeSpeed (float nForwardSpeed, float nPitchSpeed, float nUpSpeed, float nRotateSpeed)

  Sets the inspect-mode moving and rotation speed.

- void setInspectModeFactors (float nSpeedupFactor, float nMouseXFactor, float nMouseYFactor)

  Sets the inspect-mode speedup and mouse factors.

- void setFlyModeHead (float nDHeadSpeed, float nMaxHeadSpeed)

  Sets heading properties for fly-mode.

- void setFlyModePitch (float nDPitchSpeed, float nMaxPitchSpeed, float nMaxPitch)

  Sets pitching properties for fly-mode.

- void setFlyModeForward (float nDForwardSpeed, float nMaxForwardSpeed)

  Sets forward moving properties for fly-mode.

- void setRotateModeSpeed (float nForwardSpeed, float nPitchSpeed, float nUpSpeed, float nRotateSpeed)

  Sets the rotate-mode moving and rotation speed.

- void setRotateModeFactors (float nSpeedupFactor, float nMouseXFactor, float nMouseYFactor)

  Sets the rotate-mode speedup and mouse factors.

- void setFlySpeedupFactor (float nSpeedupFactor)

  Sets the fly speedup factor.

- MODE getMode () const

  Returns the currently set mode.

- const pgVec3 & getOldPosition () const

  Returns the position before update() was called.

- const pgVec3 & getNewPosition () const

  Returns the new position.

- const pgEuler & getDirection () const

  Returns the direction.

- void update (const pgIInput &nInput)

  Calculates a new position and direction.

- void createViewMatrix (bool nApply=true)

  Creates a matrix from current position and rotation.

- const pgMatrix & getViewMatrix () const

  Returns the current view matrix.

- void formatPositionString (pgString &nString)

  Formats a string containing the current position.

- void formatDirectionString (pgString &nString)

  Formats a string containing the current direction.

- void formatModeString (pgString &nString)

  Formats a string containing the mode.

5.33.1 Detailed Description

pgSteering provides basic camera movement.

pgSteering has two moving modes: MODE_INSPECT and MODE_FLY MODE_INSPECT is designed to provide as pricise and free movement as possible. MODE_FLY moves the camera like a flying aeroplane.

Definition at line 39 of file pgSteering.h.

5.33.2 Member Enumeration Documentation

5.33.2.1 enum pgSteering::MODE

Steering Mode.

Enumeration values:

MODE_INSPECT Designed to provide as pricise and free movement as possible

MODE_FLY Moves the camera like a flying aeroplane

MODE_ROTATE Rotates the camera around a point in front of it instead of strafing

Definition at line 45 of file pgSteering.h.

```
45              {
46                MODE_INSPECT,
47                MODE_FLY,
48                MODE_ROTATE,
49    };
```

5.33.3 Member Function Documentation

5.33.3.1 void pgSteering::createViewMatrix (bool *nApply* = true)

Creates a matrix from current position and rotation.

If nApply is true pgSterring calls pgIDirectX::setViewMatrix() to set the matrix as current rendering view matrix.

5.33.3.2 void pgSteering::setFlySpeedupFactor (float *nSpeedupFactor*) [inline]

Sets the fly speedup factor.

Speedup is activated if the speedup key is pressed. (Default: left shift)

Definition at line 114 of file pgSteering.h.

```
114 {  flySpeedupFactor = nSpeedupFactor;  }
```

5.33.3.3 void pgSteering::setInspectModeFactors (float *nSpeedupFactor*, float *nMouseXFactor*, float *nMouseYFactor*) [inline]

Sets the inspect-mode speedup and mouse factors.

Speedup is activated if the speedup key is pressed. (Default: left shift)

Definition at line 86 of file pgSteering.h.

```
86 {  inspectSpeedupFactor = nSpeedupFactor; inspectMouseXFactor = nMouseXFactor;  inspectMouseYFactor = n
```

5.33.3.4 void pgSteering::setOldPosition (const pgVec3 & *nPos*) [inline]

Sets the position before update was called.

pgSteering stores the previous (old) position when a new one is calculated, which is needed for collision handling. This method lets you specify this old position explicitly.

Definition at line 70 of file pgSteering.h.

```
70 {  oldPos = nPos;  }
```

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

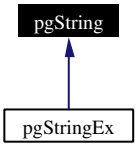
- pgSteering.h

5.34 pgString Class Reference

Class for storing and formating single byte character strings.

`#include <pgString.h>`

Inheritance diagram for pgString:



Public Methods

- pgString ()
  Creates an empty string.
- pgString (const pgString &other)
  Creates a string as a copy of another pgString object.
- pgString (const char *nString)
  Creates a string as a copy of a char pointer.
- pgString & set (const char *nString,...)
  Formats the given string and stores it.
- pgString & cat (const char *nString)
  Formats the given string and adds it.
- void setBuffer (char *nBuffer)
  Sets a new buffer directly.
- int length () const
  Returns the length of the string.
- int getLength () const
  Returns the length of the string.
- bool consistsJustOf (const pgString &cSet) const
  Does the string consists just the chars of the cSet??
- int getIndex (const char c, bool forward=true) const
  Returns the first/last occurence of c;.
- int getIndex (int start, const char c, bool forward=true) const
  Returns the first/last occurence of c; after start.
- int getIndexNot (const char c, bool forward=true) const

Returns the first/last char that is not c.

- int getIndexNot (int start, const char c, bool forward=true) const
- int getIndex (const pgString &other, bool forward=true) const

  Returns the first/last occurence of str;.

- int getIndex (int start, const pgString &other, bool forward=true) const
- int getCNum (const char c) const

  Returns the amount of occurence of c.

- const char * get () const

  Gives direct access to the string's data.

- pgString getSubString (int from, int count) const

  Returns a substring.

- pgString & toLower ()

  Converts all charater to lower characters.

- pgString & toUpper ()

  Converts all charater to upper characters.

- pgString & cutC (const char c, bool forward=true)

  Cuts all leading/ending c-s.

- pgString & cut (char c)

  Cuts all chars c of.

- int find (const pgString &nSubString)

  Returns the index of the substring. Returns -1 of not found.

Static Public Methods

- char toLower (char nC)

  Returns the lower of the passed character.

- char toUpper (char nC)

  Returns the upper of the passed character.

Friends

- pgString operator+ (const pgString &left, const pgString &right)

  concanates two strings

5.34.1 Detailed Description

Class for storing and formating single byte character strings.

Definition at line 22 of file pgString.h.

5.34.2 Member Function Documentation

5.34.2.1 bool pgString::consistsJustOf (const pgString & *cSet*) const

Does the string consists just the chars of the cSet??

If the string only contains the characters in cSet true is returned.

5.34.2.2 int pgString::getIndex (int *start*, const pgString & *other*, bool *forward* = true) const

Returns the first/last occurence of str If start==-1, then the search begins either at index 0 or Last()

5.34.2.3 int pgString::getIndex (int *start*, const char *c*, bool *forward* = true) const

Returns the first/last occurence of c; after start.

If start is -1, then the search begins either at index 0 or Last()

5.34.2.4 int pgString::getIndexNot (int *start*, const char *c*, bool *forward* = true) const

Returns the first/last char that is not c. If start==-1, then the search begins either at index 0 or Last().

5.34.2.5 void pgString::setBuffer (char * *nBuffer*)

Sets a new buffer directly.

CAUTION: the pgString object takes the ownership of the buffer; so do not delete it.

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

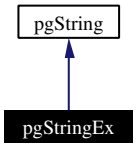
- pgString.h

5.35 pgStringEx Class Reference

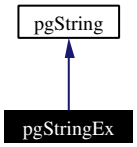
pgStringEx has a formating contructor

`#include <pgString.h>`

Inheritance diagram for pgStringEx:



Collaboration diagram for pgStringEx:



Public Methods

- pgStringEx (const char *nString,...)

  Lets you create a pgString object with a formated string.

5.35.1 Detailed Description

pgStringEx has a formating contructor

Definition at line 184 of file pgString.h.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 pgStringEx::pgStringEx (const char * *nString*, ...)

Lets you create a pgString object with a formated string.

Formating works same as sprintf()

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

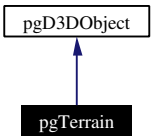
- pgString.h

5.36 pgTerrain Class Reference

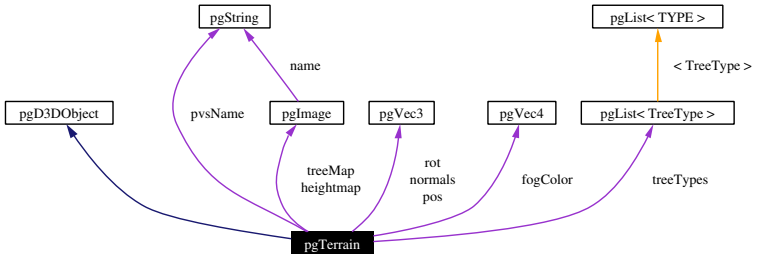
pgTerrain can render terrain by using geo-mipmapping

`#include <pgTerrain.h>`

Inheritance diagram for pgTerrain:



Collaboration diagram for pgTerrain:



Public Types

- enum RENDERMODE { UNIFIED, SPLIT, MORPH_SW, MORPH_HW }

  Settings how the terrain is rendered.

Public Methods

- void setRenderMode (RENDERMODE nMode)

  Sets a new render mode.

- void setMipmapFilter (pgTextureStage::FILTER nFilter)

  Sets a mipmap filter.

- void setBasePass (pgTexture *nColorTexture, float nRepeatX, float nRepeatY, pgTexture *nLightmap)

  Sets the base passes texture repeat rate and lightmap.

- void addPass (pgTexture *nColorTexture, float nRepeatX, float nRepeatY, pgTexture *nTransTexture)

  Adds another pass to the terrain.

- void setHeightMap (pgImage *nImage)

Sets the height map.

- void setPVSName (const pgString &nName)

  Defines the name of the pvs map file.

- void setTreeMap (pgImage *nImage)

  Sets a map which defines where trees are inserted.

- void setSkyMap (pgTexture *nSkyMap, float nRepeatX, float nRepeatY, float nSpeedX, float nSpeedY)

  Sets a texture which will be painted on the terrain to show cloud shadows.

- void setNumPatches (int nX, int nY)

  Sets the size of the landscape in number of patches.

- void setPatchSize (float nSizeX, float nSizeY, float nSizeZ)

  Sets how large one patch is.

- void setMaxError (float nErr)

  Sets the maximum error which is allowed for patch rendering.

- float getMaxError () const

  Returns the maximum error, which was set by setMaxError().

- void setFogRange (float nNear, float nFar)

  Sets the fog's range.

- void setFogColor (const pgVec4 &nCol)

  Sets the fog's color (default: (1.0, 1.0, 1.0, 1.0).

- void setDesiredFPS (int nFPS)

  Sets the fps which shall be achieved.

- int getDesiredFPS () const

  Returns the desired fps set with setDesiredFPS().

- void build ()

  Build the landscape after all properties have been set.

- void render ()

  Renders the landscape.

- void update ()

  Updates the landscapes tesselation.

- bool intersectLine (const pgVec3 &nPos0, const pgVec3 &nPos1, float &nScalarPos) const

  Does not yet work correctly.

- bool projectDown (const pgVec3 &nPos, pgVec3 &nProjected) const

  Calculates the position on landscape's surface below nPos.

- bool moveAboveSurface (pgVec3 &nPos, float nHeight, float nGlideFactor) const

  Moves nPos to stay above the surface.

- int getNumPatchesRendered ()

  Returns the number of patches which were rendered in the last frame.

- void fillInfoString (pgString &nStr)

  Fills a string with information about the last rendered terrain.

- virtual void deleteDeviceObjects ()

  The implementing object has to destroy all device dependent objects.

- virtual bool restoreDeviceObjects ()

  The implementing object has to recreate all device dependent objects.

5.36.1 Detailed Description

pgTerrain can render terrain by using geo-mipmapping

pgTerrain uses geo-mipmapping in order to display large landscapes at a high framerate. An arbitrary number of texture passes can be applied. An alpha map defines, where each texture is visible on the landscape and where not.

Definition at line 42 of file pgTerrain.h.

5.36.2 Member Enumeration Documentation

5.36.2.1 enum pgTerrain::RENDERMODE

Settings how the terrain is rendered.

Enumeration values:

UNIFIED Renders the terrain with one DrawIndexedPrimitive() call

SPLIT Renders the each patch seperated

MORPH_SW Does Vertex Morphing between two tesselation stages to reduce pops

MORPH_HW Does Vertex Morphing using Hardware Tweeming between two tesselation stages to reduce pops

Definition at line 49 of file pgTerrain.h.

```
49                      {
50             UNIFIED,
51             SPLIT,
52             MORPH_SW,
53             MORPH_HW
54     };
```

5.36.3 Member Function Documentation

5.36.3.1 void pgTerrain::addPass (pgTexture * *nColorTexture*, float *nRepeatX*, float *nRepeatY*, pgTexture * *nTransTexture*)

Adds another pass to the terrain.

This method works same as setBasePass, except that the lightmap has to have an alpha channel, which defines, where the colortexture is visible and where not.

5.36.3.2 virtual void pgTerrain::deleteDeviceObjects () [virtual]

The implementing object has to destroy all device dependent objects.

When switching to fullscreen or changing window size the render device has to be destroyed and afterwards recreated. This enforces, that all device depended objects are destroyed and recreated too. Its the duty of the implementating object destroy all device dependent objects and call all sub-objects' deleteDeviceObjects() methods.

Implements pgD3DObject.

5.36.3.3 int pgTerrain::getDesiredFPS () const [inline]

Returns the desired fps set with setDesiredFPS().

By default the automatic fps feature is disabled and getDesiredFPS() will return 0.

Definition at line 174 of file pgTerrain.h.

```
174 {  return desiredFPS;  }
```

5.36.3.4 bool pgTerrain::moveAboveSurface (pgVec3 & *nPos*, float *nHeight*, float *nGlideFactor*) const

Moves nPos to stay above the surface.

(nGlideFactor is not used yet)

5.36.3.5 bool pgTerrain::projectDown (const pgVec3 & *nPos*, pgVec3 & *nProjected*) const

Calculates the position on landscape's surface below nPos.

If nPos is not above the landscape false is returned.

5.36.3.6 virtual bool pgTerrain::restoreDeviceObjects () [virtual]

The implementing object has to recreate all device dependent objects.

Its the duty of the implementating object recreate all device dependent objects and call all sub-objects' restoreDeviceObjects() methods.

Implements pgD3DObject.

5.36.3.7 void pgTerrain::setBasePass (pgTexture * *nColorTexture*, float *nRepeatX*, float *nRepeatY*, pgTexture * *nLightmap*)

Sets the base passes texture repeat rate and lightmap.

Sets a texture for the base pass. nRepeatX and nRepeatY define how often the texture is repeated (wrapped) throughout the whole terrain. (1.0 means to stretch the texture to cover the whole terrain without repeating). The lightmap is not repeated.

5.36.3.8 void pgTerrain::setDesiredFPS (int *nFPS*) [inline]

Sets the fps which shall be achieved.

If the value is not zero, the terrain will automatically been drawn coarser or finer in order to achieve the desired fps. Pass 0 in order to turn of fps achievment.

Definition at line 166 of file pgTerrain.h.

```
166 {  desiredFPS = nFPS;  }
```

5.36.3.9 void pgTerrain::setFogRange (float *nNear*, float *nFar*) [inline]

Sets the fog's range.

If this method is not called no fog is used

Definition at line 153 of file pgTerrain.h.

```
153 {  fogNear = nNear;  fogFar = nFar;  }
```

5.36.3.10 void pgTerrain::setHeightMap (pgImage * *nImage*)

Sets the height map.

A height value of 0 will create a vertex at pos.y. A height value of 255 will create a vertex at pos.y + sizeY

5.36.3.11 void pgTerrain::setMaxError (float *nErr*) [inline]

Sets the maximum error which is allowed for patch rendering.

If the projected error of a patch on the render window exceeds maxError, the next more detailed tesselation is used.

Definition at line 142 of file pgTerrain.h.

```
142 {  maxError = nErr;  }
```

5.36.3.12 void pgTerrain::setMipmapFilter (pgTextureStage::FILTER *nFilter*) [inline]

Sets a mipmap filter.

Default filter is LINEAR.

Definition at line 71 of file pgTerrain.h.

References pgTextureStage::FILTER.

```
71 {  mipFilter = nFilter;  }
```

5.36.3.13 void pgTerrain::setNumPatches (int *nX*, int *nY*) [inline]

Sets the size of the landscape in number of patches.

nX and nY must correlate to the image which was set as heightmap. The heightmaps size must be (nX∗16)+1 x (nY∗16)+1.

Definition at line 126 of file pgTerrain.h.

```
126 {  patchesX = nX;  patchesY = nY;  }
```

5.36.3.14 void pgTerrain::setPatchSize (float *nSizeX*, float *nSizeY*, float *nSizeZ*)

Sets how large one patch is.

The size of the complete landscape can be calculated by: patchesX∗nSizeX x nSizeY x patchesY∗nSizeZ

5.36.3.15 void pgTerrain::setRenderMode (RENDERMODE *nMode*) [inline]

Sets a new render mode.

Default mode is SPLIT_MORPH. The render mode can not be changed after build() has been called.

Definition at line 64 of file pgTerrain.h.

```
64 {  renderMode = nMode;  }
```

5.36.3.16 void pgTerrain::setSkyMap (pgTexture ∗ *nSkyMap*, float *nRepeatX*, float *nRepeatY*, float *nSpeedX*, float *nSpeedY*)

Sets a texture which will be painted on the terrain to show cloud shadows.

nRepeatX and nRepeatY define of often the texture is repeated (wrapped). nSpeedX and nSpeedY define how much (in texture coordinates) the texture moves in one second.

5.36.3.17 void pgTerrain::update ()

Updates the landscapes tesselation.

It's important that the camera's final position for the current frame has already been set, since it is used to calculate the tesselation depth for each patch.

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

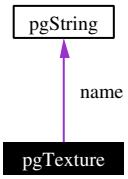
- pgTerrain.h

5.37 pgTexture Class Reference

pgTextur class

`#include <pgTexture.h>`

Collaboration diagram for pgTexture:



Public Methods

- LPDIRECT3DTEXTURE8 getD3DTexture ()

  Returns a pointer to the underlying Direct3D texture.

- operator LPDIRECT3DTEXTURE8 ()

  Cast operator to cast the texture to a Direct3D texture.

- bool isCompressed () const

  Returns true if the texture in a compressed format.

- int getWidth () const

  Returns the width of the texture.

- int getHeight () const

  Returns the height of the texture.

- const pgString & getName () const

  Returns the name of the texture.

- bool getData (unsigned char *&nData, int &nPitch, int nLevel=0, bool nReadOnly=true)

  Retrieves a pointer to the texture data.

- void releaseData ()

  Releases a texture which was by getData().

5.37.1 Detailed Description

pgTextur class

Definition at line 26 of file pgTexture.h.

5.37.2 Member Function Documentation

5.37.2.1 LPDIRECT3DTEXTURE8 pgTexture::getD3DTexture () [inline]

Returns a pointer to the underlying Direct3D texture.

The special case the texture object is invalid (NULL) is caught and NULL is returned

Definition at line 39 of file pgTexture.h.

```
39 {  return (this==NULL) ? NULL : d3dTexture;  }
```

5.37.2.2 bool pgTexture::getData (unsigned char *& *nData*, int & *nPitch*, int *nLevel* = 0, bool *nReadOnly* = true)

Retrieves a pointer to the texture data.

In order to get a pointer to the texture's data, the texture must be locked. Before the texture can be used for further rendering it must be unlocked by calling releaseData()

5.37.2.3 pgTexture::operator LPDIRECT3DTEXTURE8 () [inline]

Cast operator to cast the texture to a Direct3D texture.

The special case the texture object is invalid (NULL) is caught and NULL is returned

Definition at line 47 of file pgTexture.h.

```
47 {  return (this==NULL) ? NULL : d3dTexture;  }
```

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

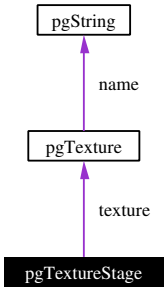
- pgTexture.h

5.38 pgTextureStage Class Reference

pgTextureStage is part of a pgMaterial

`#include <pgTextureStage.h>`

Collaboration diagram for pgTextureStage:



Public Types

- enum FILTER { FILTER_NONE = D3DTEXF_NONE, FILTER_POINT = D3DTEXF_POINT, FILTER_LINEAR = D3DTEXF_LINEAR, FILTER_ANISOTROPIC = D3DTEXF_ANISOTROPIC }

  Filder method.

Friends

- class pgSegment

5.38.1 Detailed Description

pgTextureStage is part of a pgMaterial

pgTextureStage defines the setup of a specific texture stage for a segment.

Definition at line 28 of file pgTextureStage.h.

5.38.2 Member Enumeration Documentation

5.38.2.1 enum pgTextureStage::FILTER

Filder method.

Enumeration values:

FILTER_NONE No filtering (only valid for mipmap, means: no mipmapping)

FILTER_POINT Nearest neighbour

FILTER_LINEAR Linear interpolation

FILTER_ANISOTROPIC Anisotropic filtering

Definition at line 33 of file pgTextureStage.h.

Referenced by pgTerrain::setMipmapFilter().

```
33                  {
34                     FILTER_NONE = D3DTEXF_NONE,
35                     FILTER_POINT = D3DTEXF_POINT,
36                     FILTER_LINEAR = D3DTEXF_LINEAR,
37                     FILTER_ANISOTROPIC = D3DTEXF_ANISOTROPIC
38     };
```

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

- pgTextureStage.h

5.39 pgTimeInstance Class Reference

this class represents a specific instance in time (a distinct point on the timeline)

`#include <pgITime.h>`

5.39.1 Detailed Description

this class represents a specific instance in time (a distinct point on the timeline)

Definition at line 22 of file pgITime.h.

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

- pgITime.h

5.40 pgVec2 Class Reference

2D vector class

`#include <pgVec2.h>`

Public Methods

- pgVec2 ()
  Creates a default vec2 (0,0).
- pgVec2 (float x, float y)
  Creates a vec2 from to floats.
- pgVec2 (float *nValues)
  Creates a vec2 from a pointer to two floats.
- pgVec2 (const pgVec2 &other)
  Creates a vec2 as a copy from an other one.
- float & operator[ ] (unsigned index)
  access to members (x,y) by indices
- const float & operator[ ] (unsigned index) const
  access to members (x,y) by indices
- pgVec2 & operator= (const pgVec2 &)
  copy a vec2 from an other
- pgVec2 operator+ () const
  addition same as for normal value type
- pgVec2 & operator+= (const pgVec2 &)
  add an other vec2 to this one
- pgVec2 operator- () const
  substraction same as for normal value type
- pgVec2 & operator-= (const pgVec2 &)
  substract an other vec2 to this one
- pgVec2 & operator *= (float mult)
  multiply this vec2 with a float
- pgVec2 & operator/= (float div)
  divide this vec2 by a float
- pgVec2 & add (const pgVec2 &v1, const pgVec2 &v2)
  Adds two vec2 and saves the result in this vec2.

- pgVec2 & addScaled (const pgVec2 &v1, float s, const pgVec2 &v2)
  Adds two vec2 scaling the second one and saves the result in this vec2.
- bool almostEqual (const pgVec2 &nV, float nTol) const
  Returns true if this vec2 and nV are at maximum nTol different.
- pgVec2 & combine (float s1, const pgVec2 &v1, float s2, const pgVec2 &v2)
  Adds two vec2 scaling both and saves the result in this vec2.
- pgVec2 & copy (const pgVec2 &v)
  Same a operator=().
- float distance (const pgVec2 &nV) const
  Returns the arithmetic distance between this vec2 and nV.
- float sqrDistance (const pgVec2 &nV) const
  Returns the squared arithmetic distance between this vec2 and nV.
- float dot (const pgVec2 &nV) const
  Returns the dot product between this vec2 and nV.
- bool equal (const pgVec2 &v) const
  Same as operator==().
- float length () const
  Returns the length of this vec2.
- void negate ()
  Negates this vec2.
- float normalize ()
  Normalizes this vec2.
- pgVec2 & scale (float nS)
  Multiplies this vec2 by nS.
- pgVec2 & scale (float nS, const pgVec2 &nV)
  Multiplies nV by nS and stores the result in this vec2.
- pgVec2 & scaleBy (const pgVec2 &tensor)
  Multiplies this vector component-wise by tensor.
- void set (float x, float y)
  Sets new x and y values.
- void set (const float xy[2])
  Sets new x and y values.
- pgVec2 & sub (const pgVec2 &v1, const pgVec2 &v2)
  Subtracts two vec2 and stores the result in this vec2.

Static Public Methods

- float cosine (const pgVec2 &left, Axis nAxis=X_AXIS)

  Returns the angle between the vector and one axis.

- float cosine (const pgVec2 &left, const pgVec2 &right)

  Return the angle between the two vectors.

5.40.1 Detailed Description

2D vector class

Definition at line 21 of file pgVec2.h.

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

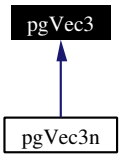
- pgVec2.h

5.41 pgVec3 Class Reference

3D vector class

`#include <pgVec3.h>`

Inheritance diagram for pgVec3:



Public Methods

- float dot (const pgVec3 &nV) const

  Returns the dot product between this vec3 and nV.

5.41.1 Detailed Description

3D vector class

Definition at line 25 of file pgVec3.h.

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

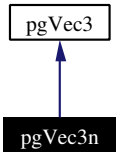
- pgVec3.h

5.42 pgVec3n Class Reference

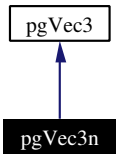
3D normalized vector class

`#include <pgVec3n.h>`

Inheritance diagram for pgVec3n:



Collaboration diagram for pgVec3n:



5.42.1 Detailed Description

3D normalized vector class

Definition at line 21 of file pgVec3n.h.

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

- pgVec3n.h

5.43 pgVec4 Class Reference

4D vector class

`#include <pgVec4.h>`

5.43.1 Detailed Description

4D vector class

Definition at line 20 of file pgVec4.h.

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

- pgVec4.h

Index