

Heightmap module

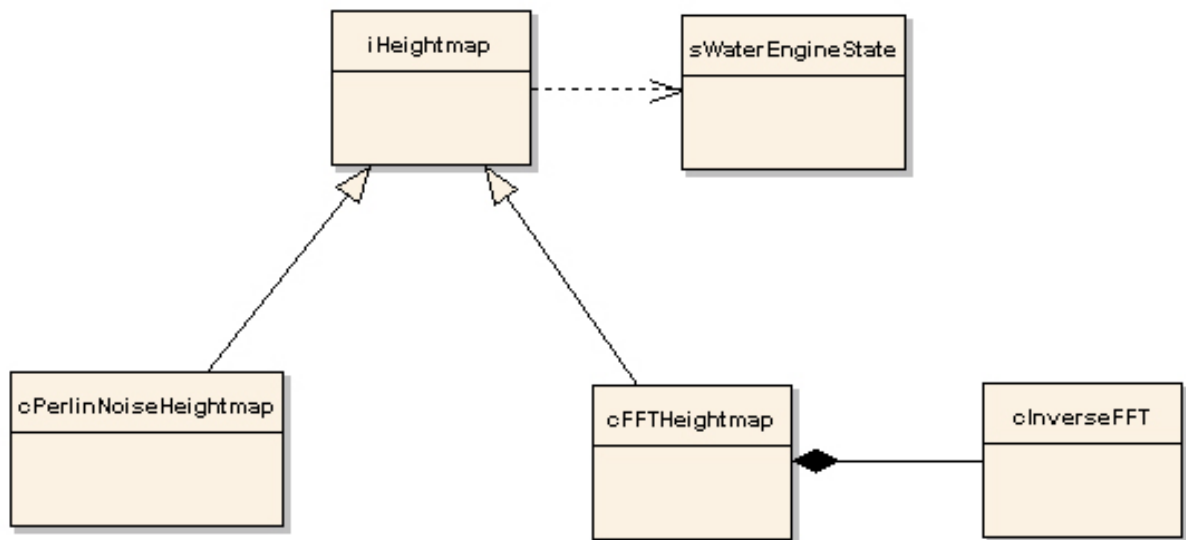


Figure 13 The class diagram (UML) for the heightmap generation module

Remarks: Both heightmap generation classes derive from one abstract base class *iHeightmap*. The base class defines what methods must be implemented in the derived classes. Both classes depend on *sWaterEngineState* class, which is just used to save the state of the engine by implementing the visitor pattern.

Each class (in current case : *cPerlinNoiseHeightmap*, *cFFTHeightmap*), which is related to the engine must confirm its state in order to save the engine state to file. This is done by implementing the function "SaveState(*sWaterEngineState* &)" for each class. This method saves the current state by filling corresponding attributes from the *sWaterEngineState* object.

***IHeightmap* class** (iHeightmap.h, iHeightmap.cpp)

This class is an abstract base class from which are derived two classes - cFFTHeightmap and cPerlinNoiseHeightmap. A list of must-override functions are defined in this class. Current implementation of heightmaps does not support arbitrary size heightmaps; only power of 2 size is accepted, height=width.

The following is a list of public class methods and their description.

```
virtual bool onLostDevice ()=0;
```

Will be implemented in derived classes

```
virtual bool onResetDevice ()=0;
```

Will be implemented in derived classes

```
virtual bool onReleaseDevice ()=0;
```

Will be implemented in derived classes

```
virtual IDirect3DTexture9* GetHeightmapTexture ()=0;
```

Will be implemented in derived classes

```
virtual const float* GetHeightmapData ()=0;
```

Will be implemented in derived classes

```
virtual int GetHeightmapResolution ()=0;
```

Will be implemented in derived classes

```
virtual bool WriteDataToTexture (IDirect3DDevice9 *d3d_device)=0;
```

Will be implemented in derived classes

```
virtual bool Update (float delta)=0;
```

Will be implemented in derived classes

```
virtual DWORD GetHeightmapType ()=0;
```

Will be implemented in derived classes

```
virtual bool EnableIntegerTextureFormat (IDirect3D9  
*pD3D9Interface, IDirect3DDevice9 *pD3D9Device)=0;
```

Will be implemented in derived classes

```
virtual bool EnableFloatingPointTextureFormat(IDirect3D9
*pD3D9Interface, IDirect3DDevice9 *pD3D9Device)=0;
```

Will be implemented in derived classes

```
virtual D3DFORMAT GetTextureFormatType ()=0;
```

Will be implemented in derived classes

```
virtual void SaveState(sWaterEngineState &stateInOut)=0;
```

Will be implemented in derived classes

***cFFTHeightmap* class**

(*cFFTHeightmap.h*, *cFFTHeightmap.cpp*)

This class implements the heightmap generation using the oceanographic statistical based model. It produces animated, tileable heightmap.

The algorithm is described in “Simulating ocean water”, Jerry Tessendorf , available here:

<http://www.finelightvisualtechnology.com/pages/coursematerials.php>

This class uses the *cInverseFFT* class to perform calculations.

The following is a list of public class methods and their description.

```
virtual bool onLostDevice ();
```

Must be called when the *IDirect3DDevice9* device is about to be restored. Returns true on success, false otherwise.

```
virtual bool onResetDevice ();
```

Must be called immediately when the *IDirect3DDevice9* device was restored, returns true on success, false otherwise.

```
virtual bool onReleaseDevice ();
```

Deprecated.

```
virtual IDirect3DTexture9* GetHeightmapTexture ();
```

Returns the heightmap data in form of texture. The texture format is either *D3DFMT_A8R8G8B8* or floating point (*D3DFMT_A32B32G32R32F* or *D3DFMT_R32F*). The texture format can be changed by calling functions: “*this->EnableIntegerTextureFormat()*” or “*this->EnableFloatingPointTextureFormat()*”. By default, at object creation, the format is *D3DFMT_A8R8G8B8*. The texture will be created when the function *this->WriteDataToTexture()* will be called first time, and the heightmap will be initialized. The texture resolution is equal to heightmap resolution. Returns *NULL* if the texture is not created.

```
virtual const float* GetHeightmapData ();
```

Returns a pointer to an 1D array of float. This array holds the heightmap data and is updated in first place when the heightmap is updated by calling function “*this->Update()*”. The size of array is: resolution * resolution, where resolution can be obtained by calling the method “*this->GetHeightmapResolution()*”. Returns *NULL* if the heightmap is not created.

```
virtual int GetHeightmapResolution ();
```

Returns the resolution (which is the same for height and width) of the heightmap, or -1 if the heightmap was not created.

```
virtual bool WriteDataToTexture(IDirect3DDevice9 *d3d_device);
```

Writes the heightmap data to texture. When first time called this function will create a texture. Return true on success or false in the following cases:

- The heightmap is not created
- The texture cannot be created
- The texture cannot be locked to write into

A pointer to valid IDirect3DDevice9 device must be supplied.

```
virtual bool Update(float delta);
```

Updates the heightmap according to the time spent “delta”. Returns true on success or false if the heightmap was not created by calling function “this->Create()”.

```
virtual DWORD GetHeightmapType();
```

Returns FFT_HEIGHTMAP

```
virtual bool EnableIntegerTextureFormat(IDirect3D9 *pD3D9Interface, IDirect3DDevice9 *pD3D9Device);
```

Enables the D3DFMT_A8R8G8B8 texture format. Calling this function will release the current texture, if the hardware has support for this format. The new texture will be created on the first call to function “this->WriteDataToTexture()”. Returns true if the hardware support the D3DFMT_A8R8G8B8 texture format or false otherwise. If false is returned the previously allocated texture is not released.

The D3DFMT_A8R8G8B8 is the default texture format on object creation.

A pointer to valid IDirect3D9 interface and IDirect3DDevice9 device must be supplied.

```
virtual bool EnableFloatingPointTextureFormat(IDirect3D9 *pD3D9Interface, IDirect3DDevice9 *pD3D9Device);
```

Enables one of the following texture formats: D3DFMT_R32F, D3DFMT_A32B32G32R32F. Calling this function will release the current texture if the hardware support floating point textures. The new texture will be created on the first call to function “this->WriteDataToTexture()”. Returns true if hardware has support for floating point textures or false otherwise. If false is returned the current texture is not released.

A pointer to valid IDirect3D9 interface and IDirect3DDevice9 device must be supplied.

Remarks: You will need the floating point texture format for SM3 rendering engine. In order to displace the vertices of the geometry on the Graphics Processing Unit you need to supply a 32 bit per component floating point format texture. You can also supply an integer texture format. In that case the vertices will not be displaced.

```
virtual D3DFORMAT GetTextureFormatType();
```

Returns the current texture format type. This can be D3DFMT_A8R8G8B8 or D3DFMT_R32F or D3DFMT_A32B32G32R32F.

```
virtual void SaveState(sWaterEngineState &stateInOut);
```

Fills the corresponding attributes from the sWaterEngineState structure with current heightmap state parameters.

```
bool Create(int resolutionIN, int gridResolutionIN, D3DXVECTOR2  
&windVectorIN, float amplitudeIN);
```

Creates a new fft based heightmap. Clears previous initialization. May be called multiple times. It takes the following parameters:

- resolutionIN – the resolution of heightmap (width=height=resolution). Only power of 2 values are valid.
- gridResolutionIN – the physical size of surface. Only positive values are accepted
- windVectorIN – the vector specifying the wind direction
- amplitudeIN – the amplitude of waves. This coefficient determines the strength of normals when they are generated using the cNormalmapFFT class.

This function will call “this->Update()” method after the creation of heightmap is complete.

Returns true on success. False otherwise. False can be returned in following cases:

- failure to initialize the inverse fourier transformation (cInverseFFT class)
- failure to update the heightmap with delta=0

```
void SetKWPower(float value);
```

This function will set the smoothness of waves. Valid values are 0,2,4,6,8... . Must be called before the call to “this->Create()” function. The higher the value the smoother the waves are.

```
const std::complex<float>* GetFourierHeightmapData();
```

Returns the wave data in fourier domain. This function should not be ever user. It is used by cNormalmapFFT class to generate the normalmap from the wave data in fourier space.

```
int GetGridResolution();
```

Returns the physical dimension of the heightmap (the gridResolutionIN value on creation)

```
void SetAmplitude(float value);
```

Deprecated.

```
void SetHeightmapNormalizationFactor(float normalizationFactorIN);
```

Set the heightmap scaling factor. If a value of “0” is passed automatic scaling is performed. When the heightmap is generated its height values can be very different depending on what parameters were passed at heightmap creation. For example you may have that the heightmap value range from 300 to 400. Most of the time we need these values to be in

range 0..1. Automatic normalization is not the best way to scale the values to the range 0..1. Because automatic normalization will find the maximum value from the current heightmap values and will scale the rest of values relative to maximum value. One time you may get one maximum value, say 200, other time you may get other maximum value, say 400. The visual results will be different after scaling. The best way to scale heightmap values to range 0..1 is to do it manually. That means, you have to set your scaling factor. You can do that by finding the maximum value from the heightmap, experimenting with it, tweaking it and after that pass the desired scaling factor to this function. To find the maximum value you must call function "GetMaximumValue()" from the cInverseFFT class. To obtain a pointer to the cInverFFT class call "this->GetInverseFFT()". Lets summarize what you need to do, in order to perform manual heightmap scaling.

- Create the heightmap
- Get the inverse fft ("this->GetInverseFFT()")
- Get the maximum value ("inverseFFTObject->GetMaximumValue()")
- Tweak this value
- Call "this->SetHeightmapNormalizationFactor()", and pass to this function your scaling value

```
cInverseFFT *GetInverseFFT ();
```

Returns a pointer to the inverse fast fourier transformation object, which is used in calculations of heightmap.

Examples of using *cFFTHeightmap* class

Example 1: Simple

```
////////////////////////////////////  
//The following code fragment is execute once at program initialization//  
////////////////////////////////////  
  
//declare an object  
cFFTHeightmap fft;  
//set smoothness of waves  
fft.SetKWPower(2.0)  
//create the object  
if(!fft.Create(128,4,D3DXVECTOR2(1,1),1.0f))  
{  
    return false;  
}
```

Example 2: Using manual heightmap scaling and changing the texture format

```
////////////////////////////////////  
//The following code fragment is execute once at program initialization//  
////////////////////////////////////  
  
//declare an object  
cFFTHeightmap fft;  
  
//create the object  
if(!fft.Create(128,4,D3DXVECTOR2(1,1),1.0f))  
{  
    return false;  
}  
//update the heightmap, with a time value of 100  
if(!fft.Update(100))  
{  
    return false;  
}  
//get the maximum value from current heightmap  
float maxValue=fft.GetInverseFFT()->GetMaximalValue();  
//set the manual scale factor  
fft.SetHeightmapNormalizationFactor(maxValue);  
//enable floating point texture format, if false is returned the hardware  
does not have support for floating point textures  
if(!fft.EnableFloatingPointTextureFormat(d3d_interface,d3d_device))  
{  
    return false;  
}
```

cPerlinNoiseHeightmap class

(cPerlinNoiseHeightmap.h, cPerlinNoiseHeightmap.cpp)

This class will produce a tileable, animated heightmap using 3D perlin noise concept.

A good description of perlin noise can be found at:

http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

The following is a list of public class methods and their description.

```
virtual bool onLostDevice();
```

Must be called when a device is about to be restored. Returns true on success, false otherwise.

```
virtual bool onResetDevice();
```

Must be called immediately when the Direct3D9 device was restored, returns true on success, false otherwise.

```
virtual bool onReleaseDevice();
```

Deprecated.

```
virtual IDirect3DTexture9* GetHeightmapTexture();
```

Returns the heightmap data in form of texture. The texture format is either D3DFMT_A8R8G8B8 or floating point (D3DFMT_A32B32G32R32F or D3DFMT_R32F). The texture format can be changed by calling functions: "this->EnableIntegerTextureFormat()" or "this->EnableFloatingPointTextureFormat()". By default, at object creation, the format is D3DFMT_A8R8G8B8. The texture will be created when the function "this->WriteDataToTexture()" will be called first time, and the heightmap will be initialized. The texture resolution is equal to heightmap resolution. Returns NULL if the texture is not created.

```
virtual const float* GetHeightmapData();
```

Returns a pointer to an 1D array of float. This array holds the heightmap data and is updated in first place when the heightmap is updated by calling function "this->Update()". The size of array is: resolution * resolution, where resolution can be obtained by calling the method "this->GetHeightmapResolution()". Returns NULL if the heightmap is not created.

```
virtual int GetHeightmapResolution();
```

Returns the resolution (which is the same for height and width) of the heightmap, or -1 if the heightmap was not created.

```
virtual bool WriteDataToTexture(IDirect3DDevice9 *d3d_device);
```

Writes the heightmap data to texture. When first time called this function will create a

texture. Return true on success or false in the following cases:

- The heightmap is not created
- The texture cannot be created
- The texture cannot be locked to write into

A pointer to a valid IDirect3DDevice9 device must be supplied.

```
virtual bool Update(float delta);
```

Updates the heightmap according to the time spent “delta”. Returns true on success or false if the heightmap was not created by calling function “this->Create()”.

```
virtual DWORD GetHeightmapType();
```

Returns PERLIN_HEIGHTMAP

```
virtual bool EnableIntegerTextureFormat(IDirect3D9 *pD3D9Interface, IDirect3DDevice9 *pD3D9Device);
```

Enables the D3DFMT_A8R8G8B8 texture format. Calling this function will release the current texture, if the hardware has support for this format.. The new texture will be created on the first call to function "this->WriteDataToTexture()". Returns true if the hardware support the D3DFMT_A8R8G8B8 texture format or false otherwise. If false is returned the texture is not released.

The D3DFMT_A8R8G8B8 is the default texture format on object creation;

A pointer to valid IDirect3D9 interface and IDirect3DDevice9 device must be supplied.

```
virtual bool EnableFloatingPointTextureFormat(IDirect3D9 *pD3D9Interface, IDirect3DDevice9 *pD3D9Device);
```

Enables one of the following texture formats: D3DFMT_R32F, D3DFMT_A32B32G32R32F. Calling this function will release the current texture if the hardware support floating point textures. The new texture will be created on the first call to function “this->WriteDataToTexture()”. Returns true if hardware has support for floating point textures or false otherwise. If false is returned the current texture is not released.

A pointer to valid IDirect3D9 interface and IDirect3DDevice9 device must be supplied.

Remarks: You will need the floating point texture format for SM3 rendering engine. In order to displace the vertices of the geometry on the Graphics Processing Unit you need to supply a 32 bit per component floating point format texture. You can also supply an integer texture format. In that case the vertices will not be displaced.

```
virtual D3DFORMAT GetTextureFormatType();
```

Returns the current texture format type. This can be D3DFMT_A8R8G8B8 or D3DFMT_R32F or D3DFMT_A32B32G32R32F.

```
virtual void SaveState(sWaterEngineState &stateInOut);
```

Fills the corresponding attributes from the sWaterEngineState structure with current heightmap state parameters.

```
bool Create(int resolutionIN, float persistenceIN, int octaveCountIN, int  
gridSizeIN, bool squaredHeightIN);
```

Creates the perlin noise heightmap. May be called multiple times. The function will clear previously allocated data and allocate new memory. The function takes the following parameters:

- resolutionIN – resolution of the heightmap (width=height=resolution). Must be a power of 2 value.
- octaveCountIN – the number of octaves (layers) that are added successively to form the heightmap. Each higher layer is twice the size (physical) of previous one. Valid values are greater than 0. This class has special optimization rules and will usually restrict the octave count, internally to a specific value, depending on the values of other parameters.
- persistenceIN – a parameter specifying how much each higher octave contribute to the final heightmap. The value must be in range 0..1. The contribution is calculated in the following way: $c = \text{pow}(\text{persistenceIN}, i)$, where “i” is the index of octave.
- gridSizeIN – the physical size of the surface. Valid values are greater than 0.
- squaredHeightIN – a parameters specifying to square or not the heightmap values after heightmap generation. May produce nice results.

After the heightmap is ready, this function will call the “this->Update()” function.

Return true on success, false otherwise. False can be returned in the following cases:

- Invalid parameters (ex. octaveCountIN == 0)
- Failure to update the heightmap with a delta value of “0”

Examples of using *cPerlinNoiseHeightmap* class

Example 1: Simple

```
////////////////////////////////////  
//The following code fragment is execute once at program initialization//  
////////////////////////////////////  
  
//declare object  
cPerlinNoiseHeightmap perlin;  
  
//create the heightmap  
if(!perlin.Create(128,0.9,4,8,true))  
{  
    return false;  
}  
}
```

Example 2: Changing the texture format

```
////////////////////////////////////  
//The following code fragment is execute once at program initialization//  
////////////////////////////////////  
  
//declare object  
cPerlinNoiseHeightmap perlin;  
  
//create the heightmap  
if(!perlin.Create(128,0.9,4,8,true))  
{  
    return false;  
}  
//enable floating point format for textures, if false is returned the  
hardware does not have support for floating point textures  
if(!perlin.EnableFloatingPointTextureFormat(d3d_interface,d3d_device))  
{  
    return false;  
}  
}
```