SmartMesh User's Manual

Version 1.0.0127

January 2015

Legal Notice

TABLE OF CONTENTS

ABOUT THIS DOCUMENT

This is a technical document and is intended for use by engineers with experience in various aspects of 3D computer graphics and the 'C' programming language.

PURPOSE

This document provides the user documentation for Cintoo3D SmartMesh SDK Release version 1.0

AUDIENCE

This document is for application developers.

PREREQUISITES

The developer must have knowledge in 3D computing, OpenCL, OpenGL or DirectX and C language.

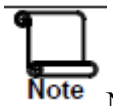
DOCUMENTATION CONVENTIONS

Manual text uses the following conventions	
Convention	***Description***
Italic Dot-Underlined	*Keywords*
Courier Bold	**Commands and keywords that are entered literally as shown.**

Terminal output as shown in examples use the following conventions:	
Convention	***Description***
`Courier`	`Example of information displayed on the terminal`
`Courier Underlined`	`Example of information displayed that depends on the configuration (actual value may be different)`
`Courier Bold`	`Examples of text that must be typed in by the operator.`
`Courier Bold Underlined`	`Examples of text that must be entered but depends on the configuration.`
`password`	`A gray background is applied to text that is not printed to the screen but must be entered by the operator, such as passwords.`

The following conventions are used to attract the attention of the reader:

Means “reader: be careful”. In this situation, the user might do something that could result in equipment damage or loss of data. ! Caution



Note Means “reader: take note”. Notes contain helpful suggestions or references to material not covered in the manual. Note

Some functionality is considered more advanced, for example because it is relatively low-level, or requires special care to be properly used.

Advanced Such functionality is identified like so in the manual.

TERMINOLOGY

SDK: Software Development Kit

LoD: Level of Detail

Patch: a triangle at the coarser LoD

PRODUCT OVERVIEW

Cintoo3D SmartMesh is a SDK that includes different algorithms to process, compress and decompress 3D triangle meshes. The API interface is plain C coding.
The document describes the so-called SmartMesh Release 1.0 API.

SYSTEM CONFIGURATION

The SDK operates on PC and mobile devices equipped with Microsoft Windows Operating System. It is compatible with Windows Vista and higher versions.

DATA FLOWS

The further chapters of the document describe functionalities of each module. But first, make sure to get familiar with the common API concepts used thoroughly in the library.
SmartMesh has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **LOD_generator** : a module allowing the Multi-LOD meshes generation
- **Compression**: a module allowing the compression of the Multi-LOD mesh
- **Decompression**: a module allowing the decompression of the Multi-LOD mesh

USER ACCESS LEVELS

The SmartMesh SDK Trial is limited to a 30 days trial period. It handles only meshes composed by less than **1 Million triangles** and having texture images not exceeding **4096*4096 pixels.**

CONTINGENCIES

In case of power outage data are not saved in internal memory of the operating device.

INSTALLATION

This section describes how to install SmartMesh SDK on Windows systems.

PREREQUISITES

Installing SmartMesh SDK requires:

- a 64bits Windows operating system. The supported operating systems are Windows Vista, Seven and 8/8.1.
- a 12 megabytes free space on the hard disk drive

After the SmartMesh installation on the device, the SDK can be used immediately without any further configuration but requires a few components to be preinstalled.

- a supported compiler (see Section Supported Compilers)
- an OpenCL compatible device with OpenCL 1.1 or higher versions

The viewer sample requires for the visualization purposes:

- OpenGL 2.0 or higher versions

SUPPORTED COMPILERS

In order to build the SmartMesh libraries, you need a C compiler. SmartMesh is supported for the following compilers/operating systems:

Compiler	**Operating System**
MINGW	MS Windows Vista x64, 7 x64, 8/8.1 x64
MS Visual C++ 10.0, 11.0, 12.0 (VISUAL STUDIO 2010, 2012, and 2013)	MS Windows Vista x64, 7 x64, 8/8.1 x64

GRAPHICAL USER INTERFACE INSTALLER

An automated GUI installer wizard called SmartMeshSDK.msi is the preferred method for first time installation. This will guide you through the setup of the SmartMesh SDK installer. You will then use this to manage your installation and to perform further package installations.

To perform your first time SmartMesh SDK installation, you should proceed as follows:

1- download latest available version of the SmartMesh_SDK.msi from the website www.cintoo3d.com

2- Locate the file you have downloaded, and double click on it to start the installer. (Note: depending on your version of Windows, and on your local security policies, you may need to grant administrator permissions for this application to run)

3- When you have set the installation options to suit your preferences, click the Continue button to initiate the installation of SmartMesh SDK

4- When you have completed the foregoing procedure to the point where SmartMesh SDK is installed, you may click the Finish button, to terminate SmartMesh_SDK.msi properly.

ENVIRONMENT VARIABLE

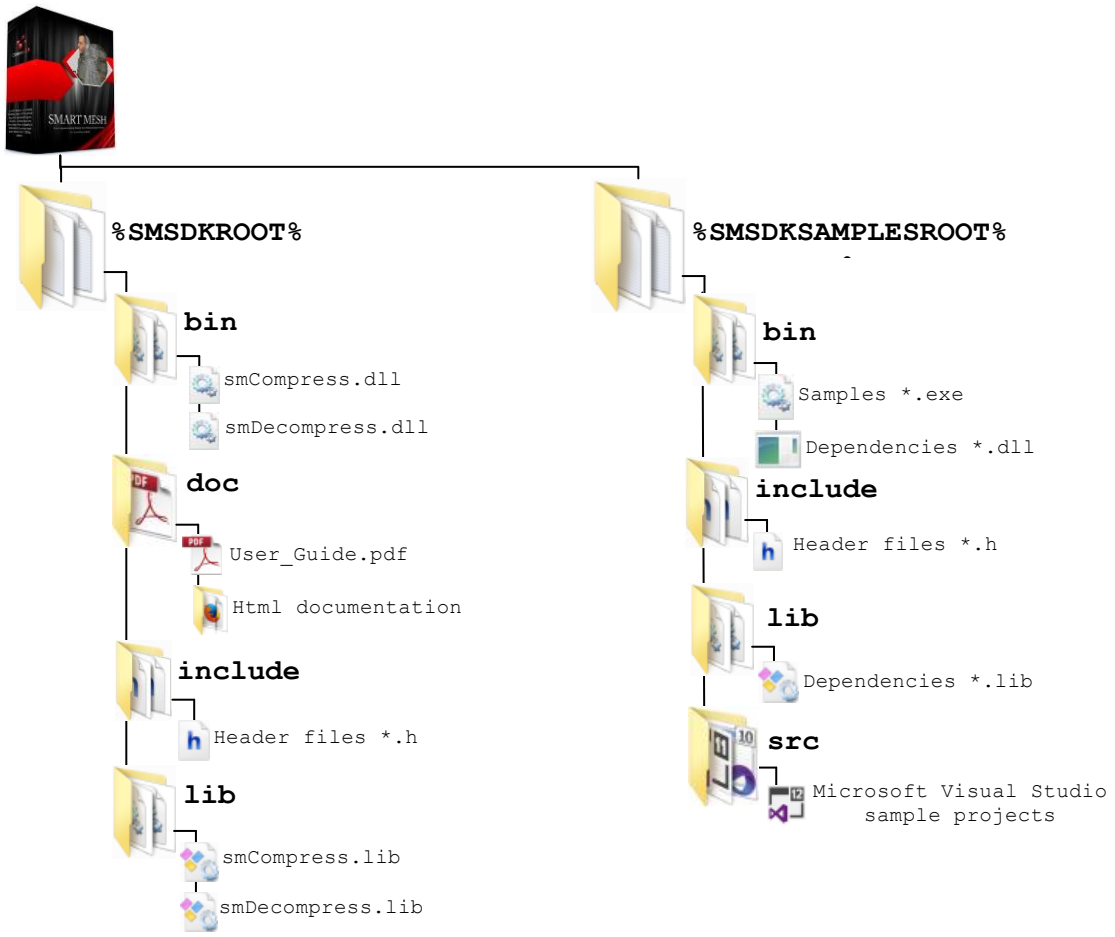
Two new system environment variables %SMSDKROOT% and %SMSDKSAMPLESROOT% are created pointing respectively to the installation root path and the samples path. To get the value of these variables, enter the following at a command prompt :

```
echo %SMSDKROOT%
```

The installation updates also the windows PATH environment variables by adding `%SMSDKROOT%bin`. It allows you to conveniently run any application using the SDK from any directory without having to copy the required dll libraries.

FILES TREE

The SmartMesh SDK installation creates 2 main folders. The first one associated to the environment variable %SMSDKROOT% is created by default on the 64bits programs folder. The second one associated to the environment variable %SMSDKSAMPLESROOT% is created by default on the user personal folder. The following figure shows the main files and folders installed:



UPDATING OR CHANGING YOUR SMARTMESH SDK INSTALLATION

To uninstall the SmartMesh SDK use the Windows Installer Manager located the Control panel.

To update the SmartMesh SDK with a new version please uninstall the old one first.

GETTING STARTED

GENERAL RULES

All the SmartMesh type, data structures and functions are prefixed by the keyword **sm**

```
SM_RUNNER_LOD_GENERATOR     // sm macros
sm_error status;            // sm type
smLodGenerateParamCreate(); // sm function
```

OPENCL LAYER

The Smartmesh SDK offers a smart layer to simplify the interactions and the manipulation of OpenCL. Indeed, it defines the **sm_runner_id** structure which is a high level structure taking as input the OpenCL Command Queue and the set of tasks that could be processed and creates a "runner". The application has to create the OpenCL Command Queue using OpenCL standard API before creating the runner.

The different tasks that a "runner" can do are:

- LOD generation,
- Compression
- Decompression

The application can create the **runner** using the **smRunnerCreate** function by giving three parameters. The first one is the **OpenCL Command queue.** The second argument is a bitfield flag to specify the the task to do which can be:

- SM_GENERATE
- SM_COMPRESS
- SM_DECOMPRESS
- or a mix of any of them.

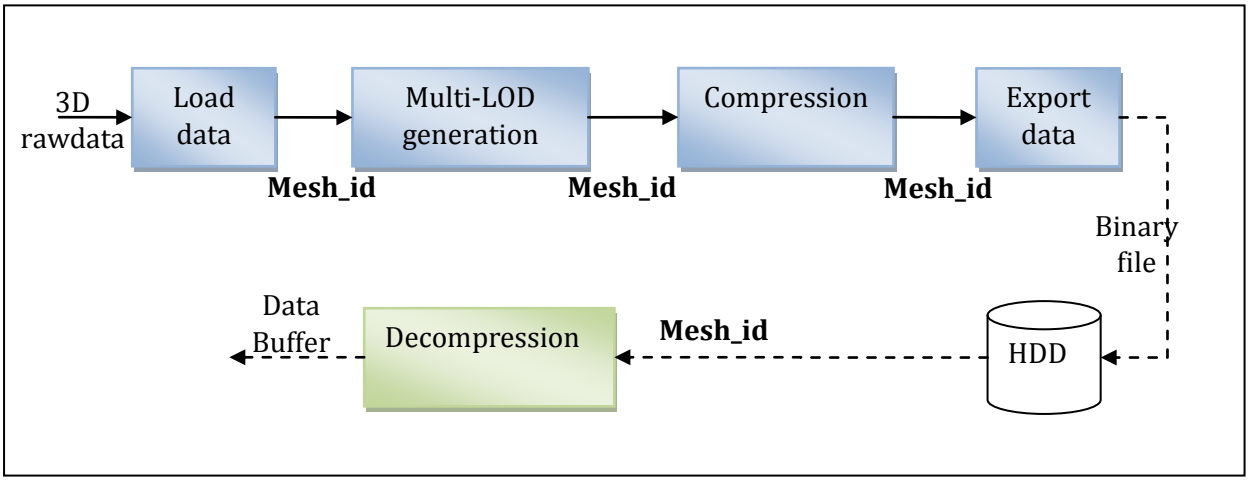
Finally, the last argument permits to verify if an error is occurred.

This code sample shows how to create through the same existing OpenCL Command Queue called **my_command_queue**, two runners: the first one called **comp_runner** for the generation and the compression of the multi-lod mesh and the second one called **decomp_runner** for its decompression.

```
#include "compress.h"

...

sm_error status;
sm_runner_id comp_runner = smRunnerCreate (my_command_queue, SM_GENERATE|SM_COMPRESS,
                                           &status);
if (status != SM_SUCCESS)
       return status;

sm_runner_id decomp_runner = smRunnerCreate (my_command_queue, SM_DECOPRESS, &status);
if (status != SM_SUCCESS)
       return status;
...
```

THE GLOBAL PROCESS

The global process is given by the following flowchart:



It consists of two independent parts. The first one allows generating the compressed bitstream by computing the multi-lod mesh then compressing it. The second part allows the decompression of the compressed mesh.

The SDK provides two libraries:

- **Compression library**: It allows the generation of the multi-LOD mesh and its compression.
- **Decompression library**: It provides functions to decompress the binary file.

We will detail here four steps of the global process:

- Data initialization
- Muti-LOD generation
- Compression
- Decompression

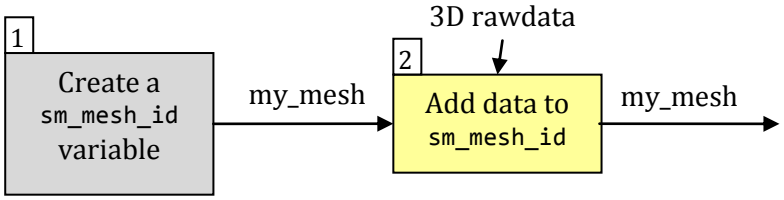
For all these steps the application manipulates the same variable called for example "`my_mesh`" of type `sm_mesh_id`. After each step "`my_mesh`" will get a new status. The application can check the step status in the mesh info structure using the function **`smMeshInfoGet.`**

The following figure shows the different possible steps for a mesh and their descriptions.

- ERR: "init not done or major failure";
- NODATA: "no data owned or empty";
- DATA: "data owned";
- LOD: "smLodGenerate called successfully";
- CMPR: "smMeshCompress called successfully ";
- SAVED: "smMeshsave called successfully";
- DECOMP: "smMeshDecompress called successfully";

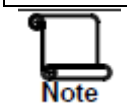
SmartMesh handles its own memory automatically but provides functions and methods having destructors that free the underlying memory buffers when needed.

The initialization of the data begins by instantiating a variable of type **sm_mesh_id** and then initializes it with the return of the function **smMeshCreate.** Then, the 3D data is filled using **smMeshSet_fu32** (see the API Reference) where f is 32 bits floating point type and u32 is 32 bits unsigned integer type.



The following code sample shows how to initialize data:

```
#include "compress.h"
...
sm_error status;
sm_mesh_id my_mesh = smMeshCreate(&status);
if (status != SM_SUCCESS)
	return status;
status = smMeshSet_fu32(	my_mesh,
				nb_vertex,
				nb_texcoords,
				nb_normals,
				vertices_array,
				texcoords_array,
				normals_array,
				nb_triangle,
				index_vertex_array,
				index_tex_coord_array,
				index_normal_array,
				nb_material,
				material_array,
				NULL,
				material_map_kad);
if (status != SM_SUCCESS)
	return status;
...
```



if the mesh has no normal vertex attributes for example, the application has to set `nb_normals to 0` and `normals_array to NULL`.

The created mesh has different accessible information that will be filled in as the different steps are achieved. By specifying the information to query using the type **sm_mesh_info**, the function **smMeshInfoGet** returns the actual value of the queried mesh. The supported information are given by the following table:

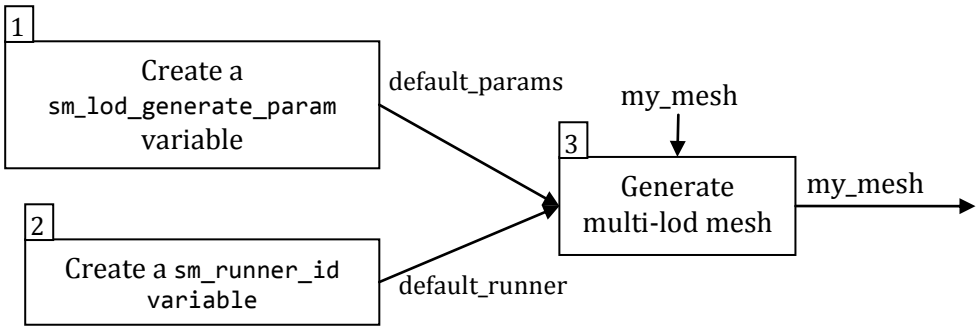
sm_mesh_info	Return type	Info. returned
SM_MESH_INFO_NB_VERTEX	size_t	Number of vertices
SM_MESH_INFO_NB_FACE	size_t	Number of faces
SM_MESH_INFO_HAS_TEXTURE	int	1 if mesh is textured and 0 if not
SM_MESH_INFO_HAS_NORMAL	int	1 if mesh has normals and 0 if not
SM_MESH_INFO_NB_VERTEX_ATTRIB	size_t	Number of vertex attributes
SM_MESH_INFO_NB_FACE_ATTRIB	size_t	Number of face attributes
SM_MESH_INFO_NB_LOD	size_t	Number of LODs
SM_MESH_INFO_NB_PATCH	size_t	Number of patchs (base level triangles)
SM_MESH_INFO_STATUS	sm_mesh_status	Most advanced step validated
SM_MESH_INFO_BOUNDING_BOX	float*	Bounding box : 6*float (XYZminXYZmax)
SM_MESH_INFO_CENTER	float*	Center point of the mesh : 3*float
SM_MESH_INFO_DIAG_BOUNDING_BOX	float	The distance of the bounding box diagonal

To get the value of a given `sm_mesh_info`, the application has to create a variable of the corresponding return type (see last table). Then allocate it with the right size if necessary. The size can be retrieved using the same function **`smMeshInfoGet.`** The following code sample shows how to get the value of the Center point of the mesh which is an array of three floats.

```
float *mesh_center = NULL;
size_t mesh_center_size = 0;
sm_error status;
 /* get the size in bytes of the array to store the center of the mesh */
status = smMeshInfoGet(mesh, SM_MESH_INFO_CENTER, 0, NULL,
                       &mesh_center_size);
if (status != SM_SUCCESS)
       return status;

/* allocate the needed ressources */
mesh_center = (float*)malloc(g_mesh_center_size);
/* get the coordinates of the center of mesh */
status = smMeshInfoGet(    mesh,
                           SM_MESH_INFO_CENTER,
                           mesh_center_size,
                           mesh_center, NULL);
if (status != SM_SUCCESS)
       return status;

printf("Mesh center: %f %f %f\n", mesh_center[0], mesh_center[1], mesh_center[2]);
```

MULTI-LOD GENERATION

The next step after the data initialization consists of generating the multi-lod mesh. To reduce the computation time, this process is implemented using OpenCL. The application creates a "runner" and specifies the device that will run the task through the given OpenCL Command Queue. For more details about runner creation please see section OpenCL layer.



This code sample shows how to generate a multi-lod mesh using the default remeshing parameters. In this case the SmartMesh SDK computes the optimal parameters depending on the input mesh.

```
#include "compress.h"
...
sm_error status;
sm_lod_generate_param default_ param = smLodGenerateParamCreate();
sm_runner_id my_runner = smRunnerCreate (my_cl_command_queue, SM_GENERATE, &status);
if (status != SM_SUCCESS)
	return status;
status = smLodGenerate(my_mesh, default_ params, my_runner, my_callback);
if (status != SM_SUCCESS)
	return status;
...
```

my_mesh status has to be at least **DATA** (SM_MESH_STATUS_DATA)
The runner must contains the task SM_GENERATE

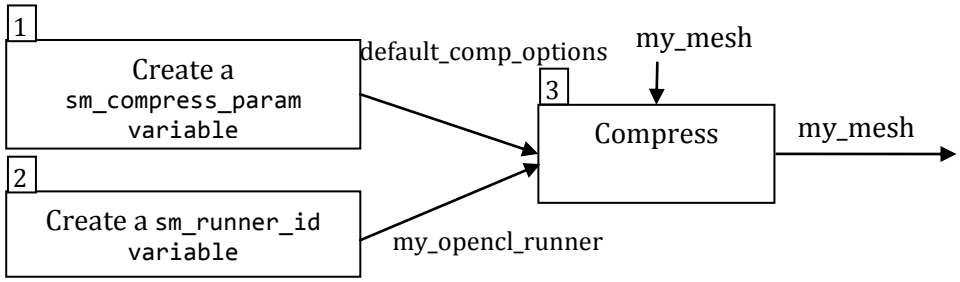
The application can specify its own parameters by modifying the variable **default_params** using the parameters setter **smLodGeneratorParamSet**. The application can set for example: the numbers of LOD, the number of patches, the total number of triangles…
This code sample shows how the application can set its own remeshing parameters and then call the function **smLodGenerate**. The parameters to modify are SM_LOD_GENERATE_INFO_NB_PATCH_MIN and SM_LOD_GENERATE_INFO_NB_LOD_MAX

```
#include "compress.h"
...
sm_error status;
uint8_t  lodMax = 5;
size_t   nb_patches = 1000;
sm_lod_generate_param my_lod_param = smLodGenerateParamCreate();
/* setting the max number of lod param */
status = smLodGenerateParamSet(my_lod_param,
                               SM_LOD_GENERATE_INFO_NB_LOD_MAX,
                               &lodMax,
                               sizeof(uint8_t));
```

```
if (SM_SUCCESS != status)
	return status;
/* setting the minimum patch number param */
status = smLodGenerateParamSet(my_lod_param,
                               SM_LOD_GENERATE_INFO_NB_PATCH_MIN,
                               &nb_patches,
                               sizeof(size_t));
if (SM_SUCCESS != status)
	return status;
...
status = smLodGenerate(my_mesh, my_lod_param, my_runner, my_callback);
if (SM_SUCCESS != status)
	return status;
...
```

COMPRESSION

The compression step allows generating the final bitstream to save. It can be applied only when the multi-lod generation step is called successfully. It requires a mesh with a status at least equal to SM_MESH_STATUS_LOD. This task requires also a valid runner.



This code sample shows how to compress a multi-lod mesh using the default compression parameters and a custom OpenCL runner. The runner is initialized with a valid OpenCL Command Queue provided by the application.

```
#include "compress.h"
...
sm_error status;
sm_mesh_compress_param default_comp_param = smCompressParamCreate();
sm_runner_id my_opencl_runner= smRunnerCreate(my_command_queue, SM_RUNNER_COMPRESS,
&status);
if (SM_SUCCESS != status)
	return status;
status = smMeshCompress(my_mesh, default_comp_param, my_opencl_runner);
```

The application can specify its own compression parameters by modifying the variable **default_comp_param** using the function **smCompressParamSet**.

my_mesh status has to be at least **LOD** (SM_MESH_STATUS_LOD)
The runner must contains the task SM_COMPRESS

SAVING

At this step you can save the compressed data by using smMeshSave. This function generates a binary file with .**i3sc** extension. If the input is textured a binary **i3st** file is generated containing the new compressed texture images.

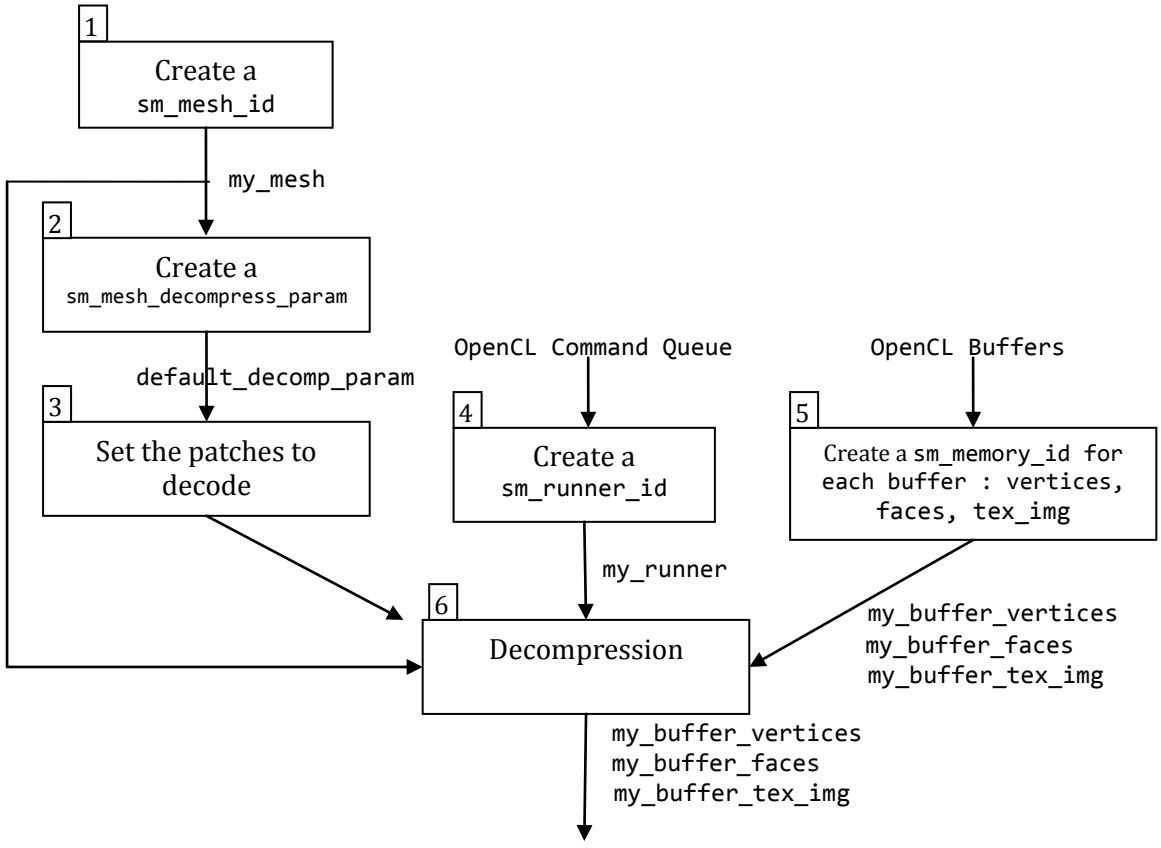
This code sample shows how to save the compressed multi-lod mesh.

```
#include "compress.h"
...
sm_error status;
status = smMeshSave(my_mesh, filename);
if (SM_SUCCESS != status)
       return status;

...
```

my_mesh status has to be at least at **COMPR** step (SM_MESH_STATUS_COMPR)

DECOMPRESSION

The decompression process is detailed by the following flowchart:



The decompression step begins with the creation of a valid variable `sm_mesh_id` by calling **`smMeshCreateFromFile`**. Then the application has to create and set the decompression parameters. It specifies for examples the different patchs to decode and the desired lod for each one. These values can be, for example, the result of the culling process. After that, the application has to create a valid runner and a valid **`sm_memory_id`** for each vertex attribute to decode and store the result). The size of the **`sm_memory_id`** is determined by the function **`smMeshDecompressParamGet`** and depends on the params currently set using **`smMeshDecompressParamSet`**.

This code sample shows how to decompress all triangles of a non textured mesh at the second lod following the last flowchart. A complete decoding sample can be found at this location: %SMSDKROOT%sample\viewer.c (c.f. Environment variable)

```
#include "decompress.h"
...
sm_error status;
size_t vertex_size;
size_t face_size;
size_t nbr_patchs;
uint8_t *my_patchs_lod;
sm_mesh_id my_mesh = smMeshCreateFromFile(file_name, &status); // i3sc file
if (SM_SUCCESS != status)
       return status;
/* Get the number of patches of the mesh */
status = smMeshInfoGet(g_mesh,
                     SM_MESH_INFO_NB_PATCH,
                     sizeof(size_t),
                     &nbr_patchs, NULL);
if (SM_SUCCESS != status)
       return status;

/* initialize the patch lod array to set in decompress param */
my_patchs_lod = (uint8_t*)malloc(nbr_patchs);
/* decode the entire mesh at the second level*/
memset(my_patchs_lod, 2, nbr_patchs * sizeof(uint8_t));
sm_mesh_decompress_param decomp_param = smMeshDecompressParamCreate(my_mesh, &status);
if (SM_SUCCESS != status)
       return status;
status = smMeshDecompressParamSet(decomp_param,
                                  SM_DECOMPRESS_PARAM_PATCH,
                                  nbr_patch* sizeof(uint8_t),
                                  (void*) my_patchs_lod);
if (SM_SUCCESS != status)
       return status;

// once the patchs have been set, use smMeshDecompressParamGet to get the
// corresponding sizes of buffers
/* get the size of vertex and face buffers (related to the last param set) */
status = smMeshDecompressParamGet(g_mesh_param,
                                  SM_DECOMPRESS_PARAM_VERTEX_SIZE,
                                  sizeof(size_t),
                                  &vertex_size,
                                  NULL);
if (status!= SM_SUCCESS)
       return status;
status = smMeshDecompressParamGet(decomp_param,
                                  SM_DECOMPRESS_PARAM_FACE_SIZE,
                                  sizeof(size_t),
                                  &face_size,
                                  NULL);
```

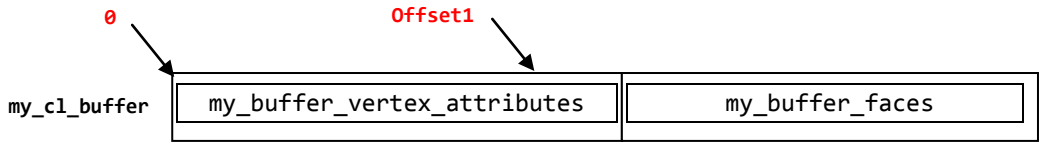
```
if (status!= SM_SUCCESS)
	return status;
// after creating the opencl buffers (my_cl_buffer_vertices and my_cl_buffer_faces)
using face_size and vertex_size we create the corresponding sm_memory_id
sm_memory_id my_buffer_vertices = smMemoryCreate(my_cl_buffer_vertices, 0, &status);
if (status!= SM_SUCCESS)
	return status;
sm_memory_id my_buffer_faces = smMemoryCreate(my_cl_buffer_faces, 0, &status);
if (status!= SM_SUCCESS)
	return status;
sm_runner my_runner= smRunnerCreate(my_command_queue, SM_DECOMPRESS, &status);
if (status!= SM_SUCCESS)
	return status;

/* run async decompression */
status = smMeshDecompress( my_mesh,
                           decomp_param,
                           my_runner,
                           my_buffer_vertices,
                           my_buffer_faces,
                           NULL);
if (status!= SM_SUCCESS)
	return status;
...
```

The following table details each step of the decompression process.

step	Note	Caution
1	The application call **smMeshCreateFromFile to get a valid** `sm_mesh_id` from a **i3sc** file	call **smMeshClear** if a previous mesh exist on this variable
3	To set the patches to decode, the application gives an array of type **`uint8_t`** having the same number of element as number of patches. The value of each element corresponds to the LoD to be decoded at. The 0 value means that the patch will not be decoded.	If the value corresponding to a patch is superior to the number of levels, the patch will be decoded at the higher level. The version 1 of SmartMesh can only decode patches at the same level
4	**my_runner** defines the device and the set of tasks to be performed.	the application has to initialize the runner with the **`SM_DECOMPRESS`** flag
5	The application creates its own **`sm_memory_id`** buffers using the OpenCL memory buffers and offsets. All the vertex attributes are stored on the first **`sm_memory_id`** buffer, the second buffer is dedicated to face indices and the third one for the texture image. See MEMORY BUFFERS INITIALISATION AND USAGE below for details on how to initialize and use the **`sm_memory_id`** buffers.	The application cannot call the decompression using the same **`sm_memory_id`** for the different buffers
6	The application may decode only the vertex attributes by setting to null the other **`sm_memory_id`** buffers. The choice of the vertex attributes to be decoded is done using a flag. The available flags are `SM_DECOMPRESS_POSITION`, `SM_DECOMPRESS_TEXCOORD` and `SM_DECOMPRESS_NORMAL`	The decompression requires at least one **`sm_memory_id`** buffer. Otherwise it returns `SM_ERROR_INVALID_PARAMETER`

MEMORY BUFFERS INITIALIZATION AND USAGE

The decompression function takes 3 buffers of type **sm_memory_id**. The first one to store the vertex attributes, the second one to store face indices and the third one for the texture image. The application can use the same OpenCL buffer to store the vertex attributes and the face indices using different offsets. The following code sample shows how to use the same OpenCL buffer to create these 2 **sm_memory_id** buffers.

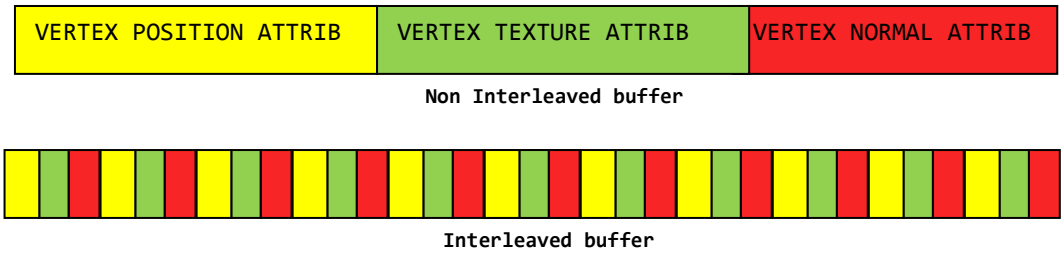



```
#include "decompress.h"
...
sm_error status;
sm_memory_id my_buffer_vertex_attributes = smMemoryCreate(my_cl_buffer, 0, &error);
if (status!= SM_SUCCESS)
	return error;
sm_memory_id my_buffer_faces = smMemoryCreate(my_cl_buffer, Offset1, &error);
if (status!= SM_SUCCESS)
	return error;
...
```

The first buffer is reserved to store the vertex attributes which can be:

- SM_VERTEX_ATTRIB_POSITION
- SM_VERTEX_ATTRIB_TEXTURE
- SM_VERTEX_ATTRIB_NORMAL

These attributes can be interleaved or not by specifying for each attribute the parameters: stride and offset. The following figure shows the two ways of formatting attributes



The following table gives for the two cases the values of the different parameters. By default the parameters are initialized to 0:

Parameter	Non-interleaved	interleaved
SM_MEM_ATTRIB_POSITION_OFFSET	0	0
SM_MEM_ATTRIB_POSITION_STRIDE	0	0
SM_MEM_ATTRIB_TEXCOORD_OFFSET	Offset1	0
SM_MEM_ATTRIB_TEXCOORD_STRIDE	0	3*sizeof(float) (position xyz)
SM_MEM_ATTRIB_NORMAL_OFFSET	Offset2	0
SM_MEM_ATTRIB_NORMAL_STRIDE	0	3*sizeof(float) + 2*sizeof(float) (position xyz + texture coord uv)

The following code sample shows how to interleave the vertex attributes on the vertex buffer

```
#include "decompress.h"
...
sm_error status;
sm_memory_id my_buffer_vertices = smMemoryCreate(my_cl_buffer, 0, & status);
if (status!= SM_SUCCESS)
	return status;
unsigned int stride = 0;
status = smMemoryAttribSet(my_buffer_vertices, SM_MEM_ATTRIB_POSITION_STRIDE,
sizeof(unsigned int), &stride);
if (status!= SM_SUCCESS)
	return status;
stride = 3*sizeof(float); /* position(xyz) */
status = smMemoryAttribSet(my_buffer_vertices, SM_MEM_ATTRIB_TEXCOORD_STRIDE,
sizeof(unsigned int), &stride);
if (status!= SM_SUCCESS)
	return status;
stride = 3*sizeof(float) + 2*sizeof(float);/* position(xyz)+texture coordinates (uv)*/
status = smMemoryAttribSet(my_buffer_vertices, SM_MEM_ATTRIB_NORMAL_STRIDE,
sizeof(unsigned int), &stride);
if (status!= SM_SUCCESS)
	return status;
...
```

FREQUENTLY ASKED QUESTIONS

1- *What is SmartMesh SDK?*

SmartMesh is a SDK that provides different algorithms to process, compress and decompress 3D triangle meshes.

2- How can I compile and execute the demos and examples on Debian / Ubuntu?

SmartMesh SDK is now only available for Microsoft windows platforms but we welcome enquiries from companies working on other Operating systems and interested in adding or integrating our technologies with their products and solutions.

3- What is a runner?

The Smartmesh SDK offers a smart layer to simplify the interactions and the manipulation of OpenCL. Indeed, it defines the **`sm_runner_id`** structure which is a high level structure taking as input the OpenCL Command Queue and the set of tasks that could be processed and creates a "runner". For more details please see section OpenCL layer.

4- What is a sm_memory_id?

The `sm_memory_id` buffer is created from an OpenCL Buffer. Like the runner, it allows easy manipulation of OpenCL buffers (c.f. section OpenCL layer).

5- How can I choose the device to do the decompression of the mesh?

The **`smMeshDecompress`** function requires a valid runner to decompress the mesh. The application will create a "runner" and choose through the given OpenCL Command Queue, the device that will run the process. For more detail about runner creation please see section OpenCL layer.

6- How can I compress a quadrangular mesh ?

The SmartMesh SDK process only triangular meshes. You have to convert the mesh into triangula mesh before processing it.

7- Can I use the SmartMesh SDK to simplify a mesh ?

Yes, to simplify a mesh you have to call the function **`smLodGenerate`** with the values 1 for the parameter `SM_LOD_GENERATE_INFO_NB_LOD_MAX` and the desired number of final triangles for the parameter `SM_LOD_GENERATE_INFO_NB_PATCH_MIN.`

8- Can I decompress only the position attributes of a mesh ?

The choice of the vertex attributes to be decoded is done using the function `smMeshDecompressParamSet.` The application set the parameter `SM_DECOMPRESS_PARAM_VERTEX_ATTRIB` using the SM_DECOMPRESS_POSITION as value.

9- Why the size of the i3st file is bigger than the jpeg ?

The trial version of the SmartMesh SDK compresses texture images using a lossless algorithm. With the full version, the application can choose between the lossy and the lossless algorithm.

10- Why the decompression and visualization is slow?

The visualization should be smooth. For optimal performance:

- Use the gpu as device for the decompression;
- The application should choose the optimal level of detail depending on available resources and viewpoint;
- A culling techniques can be used to reduce the number of patchs to decode.

11- Why the given obj file loader does not load my file?

The given obj file loader is just a sample code and it's given for didactic purposes. The application shall handle mesh data loading. We welcome enquiries from companies interested by specific developments.

12- Why the number of levels of the output mesh is inferior to the asked one?

The parameter `SM_LOD_GENERATE_INFO_NB_LOD_MAX` sets the maximum number of LoD of the output mesh. When the number of triangles of the input mesh is not sufficient to get the asked number of levels, the SmartMesh SDK uses an inferior value.

13- Why some connected components are not present on the output file?

Some connected components will be ignored when an error occurred during the Lod generation step. Mostly when there is not enough memory to fulfill the process.

14- Why the number of triangles of the output mesh is different from the input mesh

The number of triangles of the output mesh depends essentially on the number of LoD and the number of triangles of the coarser level (number of patchs). The LoD generation tolerates up to 5 times more triangles on the last LoD than the input mesh.

15- Why the number of patchs of the output mesh is different from the asked one?

The number of patchs (coarser level triangles) depends on the topological complexity of the input mesh. The coarser level preserves the holes and the genus of the input mesh which may requires additional patchs.

16- Can the input mesh be non-manifold?

The SmartMesh SDK cleans the input meshes by removing topological artifacts such as:

- Duplicated vertices;
- Degenerated triangles;
- Non-manifoldenss (vertices and edges);
- …

KNOWN ISSUES

Environment variable

If the SmartMesh SDK is installed from a standard user having no administrator rights you may have some issues with the environment variable. In this case a simple logoff/logon on the session of the current user should fix it.

Range of the texture coordinates

The SmartMesh SDK handles texture coordinates within the range [0, 1] where `(0,0)` is conventionally the bottom-left corner and `(1,1)` is the top-right corner of the texture image.

If coordinates outside the range limits are detected, a `SM_ERROR_INVALID_INPUT` error is returned by **`smLodGenerate`**.

Texture images

The list below gives the different supported texture image format by the SmartMesh SDK:

- Windows bitmaps - *.bmp, *.dib ,
- JPEG files - *.jpeg, *.jpg, *.jpe ,
- JPEG 2000 files - *.jp2,
- Portable Network Graphics - *.png
- Portable image format - *.pbm, *.pgm, *.ppm
- TIFF files - *.tiff, *.tif

The other formats like tga are not supported and a `SM_ERROR_FILE_CORRUPTED` is returned.

Compressed output Files

The trial version of the SmartMesh SDK compresses texture images using a lossless algorithm. With the full version, the application can be choose between the lossy and the lossless algorithm

Out of memory error during LoD generation

If you get an out of memory error during the LoD generation step, the application should use a device with more memory. For example switching from a gpu based device having 1GB of memory to a CPU device having 8 GB may fix the problem. The application can also reduce the number of levels requested.

SUPPORT

Feel free to contact us at support@cintoo3d.com