

MILKSHAPE 3D EXPORTER

Every game programmer needs one day to make a choice. He asks a question to himself – „what file format for my models should I use” ?

There are few answers. He can try to understand and use existing file format created by big companies or „big small” programmers. Some formats are very easy to understand but for example we cannot use them in commercial applications. The best choice in that situation is creating own file format for 3D models.

The best we can do is to write an export plugin for one of the well known editors. We can take 3DStudio, Maya, Blender, Milkshape etc. I have chosen the last one because writing plugin for this editor is ridiculously easy and Milkshape is also easy to get.

I will describe making plugin in next sections of this tutorial

Things we need to start:

- Milkshape 3D – download it at <http://www.milkshape3d.com> (it is shareware version but absolutely sufficient for our task)
- Milkshape 3D SDK – download it from the same place as above
- C++ compiler (I will be using MSVC++ 6.0)
- skills in writing DLL's

So, let's start.

Each plugin (it doesn't matter for which editor) is a DLL (dynamic link library). In short, we must create a library which will load with our editor and in proper moment will export to the file everything we want.

We are writing a DLL:

from menu File choose New and then „Win32 Dynamic-Link Library”. Owners of other IDE must do it accordingly to their software. I will focus only on MS Visual C++.

At this point one IMPORTANT advice. We must set the name of a library according to the template: „ms*.dll”. For example „msMyPlugin.dll” or „msExport.dll”. It is required by Milkshape. Ok, let's name our project „msPlugin” and then choose „a simple DLL project”. At the end hit OK. Visual C++ will generate code of simple library. Before we start writing the code, configure directories in compiler for .h and .lib files. You remember what you've downloaded from milkshape web page ? That's it – file with SDK characters in name. We must unpack it somewhere on HDD. You can see there few folders. One named „msLIB”. That's what we need. „msLib.h”, „msPlugIn.h” and in subfolder „lib” the file „msModelLibd.lib”. Set proper options for our compiler.

At the beginning of the code (in msPlugin.cpp) add:

```
#include "msLib.h"  
#include "msPlugIn.h"
```

Press ALT+F7 and tell linker to link „msModelLibd.lib”. Just add the name to Object/Library Modules in Link tab.

Now we have to create a class needed to execute few methods (These functions are executed by Milkshape in proper time)

```
class CPlugin : public cMsPlugIn  
{
```

```

char szTitle[64];

public:
    CPlugin ();
    virtual ~CPlugin ();

public:
    int      GetType ();
    const char *  GetTitle ();
    int      Execute (msModel* pModel);
};

```

Our declared methods are virtual functions from inherited class so we need to write the implementation for them.

```

CPlugin::CPlugin ()
{
    strcpy (szTitle, "Geometry exporter");
}

```

```

CPlugin::~~CPlugin ()
{
}

```

```

int CPlugin::GetType ()
{
    return cMsPlugIn::eTypeExport;
}

```

```

const char* CPlugin::GetTitle ()
{
    return szTitle;
}

```

```

int CPlugin::Execute (msModel *pModel)
{
    return 0;
}

```

In the constructor, szTitle means the name which will be displayed in Exports menu of Milkshape. I called it simply geometry exporter.

Destructor is the thing we don't need to touch. Next method GetType returns us type of the plugin we are writing. It is enum type and possible values are:

```

eTypeImport
eTypeExport
eTypeTool
eTypeEdit
eTypeVertex
eTypeFace
eTypeAnimate

```

Our plugin will be exporting so we choose eTypeExport.

GetTitle() returns the name set in the constructor.

Last and the most important method is Execute. Milkshape calls it when we choose export option. Here all the fun will start :) Here we call functions to get vertices, faces, normals etc. This method should return 0 if succeeded.

One more important thing. We need to declare one more function to make our plugin active:

```
cMsPlugIn *CreatePlugIn ()
{
    return new CPlugin ();
}
```

Its type is described in msPlugIn.h and it is needed by Milkshape to recognize our plugin at runtime. This function must be an export type so tell our compiler to make it. Create file named msPlugIn.def in project directory and add to it two lines:

```
LIBRARY    "msPlugin"
EXPORTS    CreatePlugIn
```

Additionally press ALT+F7 and in the Link tab, field Project Options add:

```
/def:".msPlugin.def"
```

Now try to compile our library. If everything went ok we should get a file named msPlugin.dll in Debug directory of the project. Copy this file to the main directory of Milkshape and run the editor. When you choose File->Export you should see our „Geometry Exporter” on the list. When you click it now – nothing will happen because our Execute method is empty.

Try to open a file „msLib.h” from Milkshape SDK. What do you see ? Every possible structures and functions we can use in our Execute() method. Everyone should orient himself what they do because every name is adequate to what it really does. For example:

int msModel_GetMeshCount (msModel *pModel) – returns number of the meshes created in the workspace

msMesh* msModel_GetMeshAt (msModel *pModel, int nIndex) – returns pointer on a mesh with number nIndex

int msMesh_GetVertexCount (msMesh *pMesh) – returns number of vertices for a given Mesh

There are many functions but I will not describe all of them. You need to work by yourself :)

For our Execute method, we need something which will export all vertices from all meshes to given file.

So add one more directive at the beginning of DLL main file:

```
#include <fstream.h>
```

And finally the code:

```
ofstream file;
file.open("c:\\export.txt"); //name of the file
int meshcount, vertexcount;
meshcount=msModel_GetMeshCount(pModel);
file<<meshcount<<endl; //number of meshes
for (int x=0; x<meshcount; x++) //go through all meshes
```

```

{
    msMesh *mesh;
    mesh=msModel_GetMeshAt(pModel,x);
    vertexcount=msMesh_GetVertexCount(mesh);
    file<<"Mesh"<<endl;
    file<<vertexcount<<endl; //number of vertices for given mesh
    for (int t=0; t<vertexcount ; t++) //go through all vertices
    {
        msVertex *vertex;
        msVec3 vx;
        vertex=msMesh_GetVertexAt(mesh,t);
        msVertex_GetVertex(vertex,vx);
        file<<vx[0]<<" "<<vx[1]<<" "<<vx[2]<<endl;
    }
}
MessageBox(NULL,"Export OK",NULL,MB_OK);
file.close();

```

And here is a description of the code:

First line is a declaration of file, second opens the file for writing. Variables meshcount and vertexcount hold numbers representing how much meshes and vertices in a mesh we have.

Next we call first function from SDK - msModel_GetMeshCount(pModel);

Its parameter is a pointer on model which we are editing now in Milkshape. We have this parameter from Execute() method.

We know how many meshes we have so let's write it to the file. (the code from next line)

A loop comes, which repeats as many times as the number of meshes.

We declare the pointer for msMesh structure and we will write to it data about our first mesh.

Struct looks like below:

```

typedef struct msMesh
{
    byte    nFlags;
    char    szName[MS_MAX_NAME];
    char    nMaterialIndex;

    word    nNumVertices;
    word    nNumAllocedVertices;
    msVertex* pVertices;

    word    nNumNormals;
    word    nNumAllocedNormals;
    msVec3* pNormals;

    word    nNumTriangles;
    word    nNumAllocedTriangles;
    msTriangle* pTriangles;
} msMesh;

```

names of a structure variables are very intuitive and there is no sense to describe them.

We call msModel_GetMeshAt(pModel,x). It returns a pointer on mesh with x number.

When we have first mesh as the structure we can take everything we want from there.

For each structure there is a number of functions which can take anything we need in a simple way.

We need vertices coordinates so try to find out how many vertices is in our mesh.

(msMesh_GetVertexCount(mesh)) and make another loop which will read every single vertex and save it to the file.

In this loop we encounter unknown structure msVertex. So look to the msLib.h and check what it is:

```
typedef struct msVertex
{
    byte    nFlags;
    msVec3  Vertex;
    float   u, v;
    char    nBoneIndex;
} msVertex;
```

we see that there is something like msVec3 type. It is not much more than array of three variables representing vertex coordinates. There are also texture coordinates and variable describing bones. We don't need them for this tutorial.

We declare variable of msVec3 type and will write vertex coordinates to it.

Function msMesh_GetVertexAt(mesh,t) returns the pointer to the vertex of t number from given mesh. Next function msVertex_GetVertex(vertex,vx) will fill the array described earlier. It's parameters are: pointer to the vertex and destination array of vertex coordinates. Thank to this, we get only what we need from the structure – x,y and z coordinates.

After succesful pass of the Execute() method we get the message that everything went OK and our file appears on disk C.

Possibilities of export are huge. I will not describe all of them because you can try them by yourself, studying „msLib.h”.

You can export everything, from simple data about vertices through textures to the bones.

That's all.

Have a nice coding,

SirMike

sirmike@poczta.onet.pl