OPENMDL USER MANUAL

Author:	Eric H. den Boer
Contact:	info@openmdl.org
Website:	www.openmdl.org
Version:	0.95.2 beta
Date:	December 13, 2010

MANUAL IS WORK IN PROGRESS

CONTENTS

WHAT IS OPENMDL AND WHY SHOULD YOU USE IT?

OpenMDL is a fast and compact game-oriented model format. It is designed specifically for use in games. As a result, all of the features that are not required in games have been stripped. OpenMDL supports: Geometry, Textures (multi-layered), Materials, Shaders, Cameras, Lights, Joints, Metadata and various forms of Animation (Vertex Animation, Bone Animation and Morph Targets).

One of the richest features OpenMDL incorporates is its material system. The OpenMDL materials support a wide variety colors, coefficients, texture maps and custom hardware shaders like HLSL, GLSL, Cg and CgFX - basically any shader type that your 3D content creation software can output. Next to this, OpenMDL's materials are ready for the future and feature a wide variety of properties often used in ray tracers and path tracers, such as refraction, reflection, surface thickness and light absorbance properties.

Although usually not part of the model file itself, cameras and lights have still been implemented, with the independent developer community in mind. These developers usually do not have the resources for extensive world editors or engines that come with world editors. By using the content creation software's light and camera objects, the software can be used to preview the world before exporting it to the game engine.

The data that is contained in OpenMDL model files is stored as binary data, resulting in fast loading of the models in your game. Unlike model formats that store their data in ASCII, OpenMDL's data can be streamed into memory straight out of the Importer - no ASCII to binary conversion needed. This also creates compact data: a 3D point in ASCII can easily take up to 62% more storage.

The importer that comes with OpenMDL features a built-in scene manager; data is read out of a file once and is then read from the cache from that point on: no multiple reads from the same file. This data is saved to an internal scene graph; all relationships between the nodes and groups that the artist made are reflected in the model file and thus to you: the programmer. Next to the scene manager and scene graph, OpenMDL also features the ability to share instanced material, joint and animation data in-between models (scenes). This saves loading time as well as cuts down the memory footprint.

The vanilla OpenMDL importer and exporters that OpenMDL incorporates do not require external libraries to run: all that is needed is a C++-compliant compiler.

INSTALLATION: AUTODESK MAYA

1. Go to the installation directory of Autodesk Maya. Usually this is: **C:/Program Files/Autodesk/Maya****/** where the **** are replaced by your version.

2. Inside the installation directory go to **bin/plug-ins/**.

3. Copy or move the appropriate OpenMDL version into this folder.

4. Inside the installation directory go to **scripts/others/**.

5. Copy or move the **OpenMDLMaya.mel** file into this folder.

6. Startup Autodesk Maya and go to **Window > Settings/Preferences > Plug-in Manager**. Once you're at the Plug-in Manager, tick the Load and Auto Load checkboxes beside the **OpenMDLMaya****.mll** plug-in.

7. You can now start using OpenMDL to export your Maya models to a quick and compact game-oriented format!

INSTALLATION: AUTODESK 3D STUDIO MAX

Currently OpenMDL does not yet support Autodesk 3D Studio Max. This exporter is expected to hit the market around January 2011.

INSTALLATION: OPENMDL C++ IMPORTER

1. Go to your project directory and copy the contents of the **C++ Importer** folder into your project directory or one of your choosing.

2. Include OpenMDL's header file (**./importer/OpenMDL.h**) and link to its library file (**./importer/OpenMDL.lib**)

3. All of OpenMDL's classes and functionality is within the **OpenMDL namespace**.

TECHNICAL DETAILS

For its file storage, OpenMDL uses little endian for integer types (8-bit, 16-bit, 32-bit and 64-bit) and IEEE 754 for floating point types. All integer and floating point data is 32-bit. For all integer data types whose value cannot be less than zero, unsigned integers are used. Booleans are represented as unsigned 8-bit integers, where 0x00 equals false and 0xFF equals true.

USAGE EXAMPLE 1: LOADING GEOMETRY

Note: This code is not meant to be optimal. This is meant to demonstrate where one can find the data instead.

The following is a straight-forward usage example of OpenMDL's C++ Importer. As you'll see, loading up a model including all its data (geometry, materials, bones, animation) is a walk in the park and will speed-up your production pipeline. The objects' transform is a row-major 4x4 matrix, relative to its parent.

```
// create an importer
OpenMDL::Importer importer;

// load a scene file
bool newInstance = false;
OpenMDL::Scene* scene = importer.LoadSceneFile("my_model_file.omd", &newInstance);
// NOTE: newInstance will be true if the scene file was loaded for the first time.

// Get a mesh count
unsigned int meshCount = scene->GetMeshes().size();
for(unsigned int m = 0; m < meshCount; ++m)
{
        // retrieve mesh data
        OpenMDL::Mesh* mesh = scene->GetMeshes()[m];
        OpenMDL::String& meshName = mesh->GetName();

        unsigned int materialCount = mesh->GetMaterials().Size();
        OpenMDL::Material* meshMaterial = mesh->GetMaterial(0);

        // retrieve the mesh's attributes / metadata
        OpenMDL::Attribute* meshAttribute = mesh->FindAttribute("myAttributeName");

        // process the vertex data
        unsigned int vertexCount = mesh->GetVertexIndexCount();
        for(unsigned int v = 0; v < vertexCount; ++v)
        {
                // vertex indices are always exported when geometry is exported
                unsigned int vertexIndex = mesh->GetVertexIndices()[v];

                // normal indices are exported when normals are
                unsigned int normalIndex = mesh->GetNormalIndices()[v];

                // same for tangent indices
                unsigned int tangentIndex = mesh->GetTangentIndices()[v];

                // texcoord indices go per UV layer, in this case, layer 0 is selected
                // NOTE: You're going to have to do some checking
                unsigned int uvIndex = mesh->GetTexCoordIndices(0)[v];

                OpenMDL::float3& vertex = mesh->GetVertices()[vertexIndex];
                OpenMDL::float3& normal = mesh->GetNormals()[normalIndex];
                OpenMDL::float3& binormal = mesh->GetBinormals()[tangentIndex];
                OpenMDL::float3& tangent = mesh->GetTangents()[tangentIndex];
                OpenMDL::float2& uv = mesh->GetTexCoordSets(0)[uvIndex];

                // (...)
                // Insert some vertex processing code here
        }
}
```

This is all that is required to access the model's geometry data.