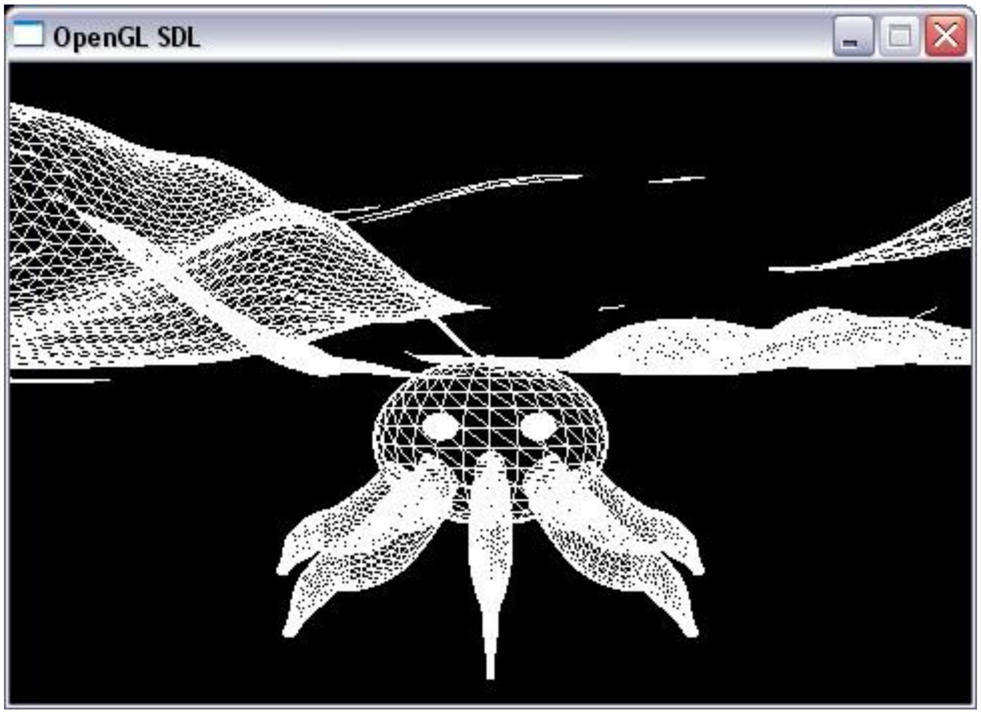
Loading 3DS Max Models

Following on from this week's lecture we will be looking at the code required to load 3 dimensional models into your scene. This tutorial provides the code required to load 3DS Max models and links to online tutorials for an obj loader.

At the end of this tutorial you should see a 3D octopus that was modelled in 3DS Max and reconstructed in your program



The framework we are using for this week's task will be a continuation from last weeks tutorial

You will also require the octopus.3ds file, which is also available on Blackboard.

Task 1 – Object Type

To begin we need to add some more structure types to the Commons file. Open this and add the following code:

```
 //------------------------------------------------------------------------------------------
------------------------------------------
//3DS Max Object type details.
#define MAX_VERTICES 15000
#define MAX_POLYGONS 15000

// The polygon (triangle), 3 numbers that aim 3  vertices
struct Triangle {
	int	a;
	int b;
	int c;
};

// The object type
typedef struct
{
	char name[20];
	int vertices_qty;
	int polygons_qty;
	Vertex3D vertex[MAX_VERTICES];
	Triangle polygon[MAX_POLYGONS];
	TexCoord mapcoord[MAX_VERTICES];
	int	id_texture;
}	obj_type, *obj_type_ptr;
//------------------------------------------------------------------------------------------
------------------------------------------
```

(You should already have implemented the TexCoord and Vertex3D types from previous tutorials)

Task 2 – 3DSLoader class

Next we will write the code for reading in a 3DS Max file. Go ahead and create yourself a header and source file called 3dsLoader. In the header add the following code:

```
#include "Commons.h"

extern char Load3DS(obj_type_ptr ogg, char *filename);
```

Next open the source file and add the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include "Commons.h"
#include "3dsLoader.h"
#include <sys\stat.h>

//------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------

long filelength(int f)
{
	struct stat buf;

	fstat(f, &buf);

	return(buf.st_size);
}

//------------------------------------------------------------------------------------
------------------------------------------

char Load3DS(obj_type_ptr p_object, char *p_filename)
{
	int i; //Index variable
	FILE *inFile; //File pointer

	unsigned short chunkId, s_temp; //Chunk identifier unsigned
	int chunkLength; //Chunk length
	unsigned char name; //Char variable
	unsigned short size; //Number of elements in each chunk

	unsigned short faceFlag; //Flag that stores some face information

	if ((inFile = fopen(p_filename, "rb")) == NULL)
		return 0; //Open the file

	while (ftell(inFile) < filelength(fileno(inFile))) //Loop to scan the whole file
	{
		fread(&chunkId, 2, 1, inFile); //Read the chunk header
		fread(&chunkLength, 4, 1, inFile); //Read the length of the chunk
		//getchar(); //Insert this command for debug (to wait for keypress for each
chuck reading)

		switch (chunkId)
		{
			//--------------------------------------------------- MAIN3DS ------
---------------------------------------
			// Description: Main chunk, contains all the other chunks
			// Chunk ID: 4d4d
			// Chunk Length: 0 + sub chunks
			//-------------------------------------------------------------------
--------------------------------------------
```

```
		case 0x4d4d:
			break;

			//-----------------------------------------------------     EDIT3DS
------------------------------
			// Description: 3D Editor chunk, objects layout info
			// Chunk ID: 3d3d (hex)
			// Chunk Length: 0 + sub chunks
			//-------------------------------------------------------------------
-------------------------------------------
		case 0x3d3d:
			break;

			//--------------------------------------------- EDIT_OBJECT -------
---------------------------------------
			// Description: Object block, info for each object
			// Chunk ID: 4000 (hex)
			// Chunk Length: len(object name) + sub chunks
			//-------------------------------------------------------------------
--------------------------------------------
		case 0x4000:
			i = 0;
			do
			{
				fread(&name, 1, 1, inFile);
				p_object->name[i] = name;
				i++;
			} while (name != '\0'		&&	i<20);
			break;

			//--------------------------------------------- OBJ_TRIMESH -------
---------------------------------------
			// Description: Triangular mesh, contains chunks for 3d mesh info
			// Chunk ID: 4100 (hex)
			// Chunk Length: 0 + sub chunks
			//-------------------------------------------------------------------
-------------------------------------------
		case 0x4100:
			break;

			//--------------------------------------------- TRI_VERTEXL -------
---------------------------------------
			// Description: Vertices list
			// Chunk ID: 4110 (hex)
			// Chunk Length:	1 x unsigned short (number of vertices)
			//				+ 3 x float (vertex coordinates) x
(number of vertices)
			//				+ sub chunks
			//-------------------------------------------------------------------
---------------------------------------------
		case 0x4110:
			fread(&size, sizeof(unsigned short), 1, inFile);
			p_object->vertices_qty = size;

			for (i = 0; i<size; i++)
			{
				fread(&p_object->vertex[i].x, sizeof(float), 1, inFile);
				fread(&p_object->vertex[i].y, sizeof(float), 1, inFile);
```

```
                                fread(&p_object->vertex[i].z, sizeof(float), 1, inFile);
                        }
                        break;
                        //------------------------------------------------ TRI_FACEL1 ---------
---------------------------------------
                        // Description: triangles (faces) list
                        // Chunk ID: 4120 (hex)
                        // Chunk LengtH:     1 x unsigned short (number of triangles)
                        //                                  + 3 x unsigned short (triangle
points) x (number of triangles)
                        //                                  + sub chunks
                        //-------------------------------------------------------------------
---------------------------------------------
                case 0x4120:
                        fread(&size, sizeof(unsigned short), 1, inFile);
                        p_object->triangles_qty = size;

                        for (i = 0; i<size; i++)
                        {
                                fread(&s_temp, sizeof(unsigned short), 1, inFile);
                                p_object->triangle[i].a = s_temp;

                                fread(&s_temp, sizeof(unsigned short), 1, inFile);
                                p_object->triangle[i].b = s_temp;

                                fread(&s_temp, sizeof(unsigned short), 1, inFile);
                                p_object->triangle[i].c = s_temp;

                                fread(&faceFlag, sizeof(unsigned short), 1, inFile);
                        }
                        break;

                        //-------------------------------------- TRI_MAPPINGCOORS ---------
---------------------------
                        // Description: Vertices list
                        // Chunk ID: 4140 (hex)
                        // Chunk Length: 1 x unsigned short (number of mapping points)
                        // + 2 x float (mapping  coordinates) x (number of mapping points)
                        // + sub chunks
                        //-------------------------------------------------------------------
---------------------------------------------
                case 0x4140:
                        fread(&size, sizeof(unsigned short), 1, inFile);
                        for (i = 0; i < size; i++)
                        {
                                fread(&p_object->texcoord[i].u, sizeof(float), 1, inFile);

                                fread(&p_object->texcoord[i].v, sizeof(float), 1, inFile);
                        }
                        break;

                        //-------------------------------- Skip unknown chunks ------------
------------------------
                        //We need to skip all the chunks that currently we don't use
                        //We use the chunk lenght information to set the file pointer
                        //to the same level next chunk
                        //-------------------------------------------------------------------
---------------------------------------------
```

```
            default:
                   fseek(inFile, chunkLength-6, SEEK_CUR);
            }
     }

     fclose(inFile);      //Close the file.
     return(1);
}

//---------------------------------------------------------------------------------------
```

Task 3 – A file to hold our Object

Next we need a class which can be used to load the model, position it in the world and render it.

This is just the bare bones of the class. You are encouraged to add functionality to move the object around your scene.

First we need a header and source file for this class. Create both and name them object3DS. In the header add the following code:

```
#ifndef _OBJECT3DS_H_
#define _OBJECT3DS_H_

#include "Commons.h"
#include <string>
using std::string;

class Object3DS
{
public:
       Object3DS(Vector3D startPosition, string modelFileName);
       ~Object3DS(){}

       void update(float deltaTime);
       void render();

       //Load 3ds file
       void loadModel();

       //Load texture for this model.
       void loadTexture();

private:
       Vector3D mPosition;

       char fileName[20];
       char textureName[20];

       obj_type object;
};

#endif //_OBJECT3DS_H_
```

Next open the source file and add the following code:

```
#include "object3DS.h"
#include "../gl/glut.h"
#include "3dsLoader.h"

//----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------

Object3DS::Object3DS(Vector3D startPosition, string modelFileName)
{
	//start position.
	mPosition = startPosition;

	//3ds file to load.
	std::strcpy(fileName, modelFileName.c_str());
	loadModel();
}

//----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------

void Object3DS::loadModel()
{
	if (fileName[0] != '---')
		Load3DS(&object, fileName);
}

//----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------

void Object3DS::loadTexture()
{
	//TODO: Load a texture to map to the object.
}

//----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------

void Object3DS::update(float deltaTime)
{
	//TODO: Move object here.
}

//----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------
```

```
void Object3DS::render()
{
	glPushMatrix();
	glTranslatef(mPosition.x, mPosition.y, mPosition.z);

	//glBindTexture(GL_TEXTURE_2D, object.id_texture); // We set the active texture

	glBegin(GL_TRIANGLES); // glBegin and glEnd delimit the vertices that define a
primitive (in our case triangles)
	for (int l_index = 0; l_index < object.triangles_qty; l_index++)
	{
		//----------------------------------------------------- FIRST VERTEX -------
-----------------------------------------//
		// Texture coordinates of the first vertex
		//glTexCoord2f( object.mapcoord[ object.triangle[l_index].a ].u,
		//		       object.mapcoord[ object.triangle[l_index].a ].v);
		// Coordinates of the first vertex
		glVertex3f(object.vertex[object.triangle[l_index].a].x,
			object.vertex[object.triangle[l_index].a].y,
			object.vertex[object.triangle[l_index].a].z); //Vertex definition
		//----------------------------------------------------- SECOND VERTEX -------
--------------------------------------------
		// Texture coordinates of the second vertex
		//glTexCoord2f( object.mapcoord[ object.triangle[l_index].b ].u,
		//		       object.mapcoord[ object.triangle[l_index].b ].v);
		// Coordinates of the second vertex
		glVertex3f(object.vertex[object.triangle[l_index].b].x,
			object.vertex[object.triangle[l_index].b].y,
			object.vertex[object.triangle[l_index].b].z);

		//----------------------------------------------------- THIRD VERTEX -------
--------------------------------------------
		// Texture coordinates of the third vertex
		//glTexCoord2f( object.mapcoord[ object.triangle[l_index].c ].u,
		//		       object.mapcoord[ object.triangle[l_index].c ].v);
		// Coordinates of the Third vertex
		glVertex3f(object.vertex[object.triangle[l_index].c].x,
			object.vertex[object.triangle[l_index].c].y,
			object.vertex[object.triangle[l_index].c].z);
	}
	glEnd();

	glPopMatrix();
}

//-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
------------------------------
```

You will notice that there is texture code commented out within this file. This can be uncommented and is for you to experiment with. You will need to write the code to load in the texture file for this to work.

Task 4 – Hooking everything up to our GameScene

We now have the code required to load in our 3DS Max model, but before anything will appear in our scene we need to create an object of object3DS type. Open GameScreenLevel1.h and add the following private declaration:

```
Object3DS*  m_p3DSModel;
```

You will obviously need to forward declare this type or include the header.

Next, open the GameScreenLevel1.cpp file and in the constructor set up the object as follows:

```
m_p3DSModel = new Object3DS(Vector3D(0.0f, 0.0f, 0.0f), "Octopus.3ds");
```

The final task is to call the render and update functions of our m_p3DSModel from within the GameScreenLevel1 render and update functions.

Further work

1. Add variables to the object3DS file that will allow the object to move around. Try to get the octopus moving back and forth within the scene.

2. Attempt to load in a model of your own creation.

3. Once you have this model loaded write the function for loading in a texture in the 3DSLoader and then map a texture to the model.

4. Create yourself an OBJ loader. Here is a link explaining the process: https://www.youtube.com/watch?v=XIVUxywOyjE