***Direct X Homework 3:***

***Textured Cube***

**Objective**

This homework assignment will build off of the last two labs, and introduce things from this lecture. You’ll be doing texturing, multi-threading, and writing your own pixel shader today.

**Homework Requirements**

Features you’re expected to implement:

* The scene is properly displaying on screen.
* The textured cube is drawing properly, with a constantly changing texture.
* The cube for the skybox is drawing and displaying the skybox texture properly.
* There are at least two threads, one used for streaming in textures, the other used for drawing the geometry.

**Homework Instructions**

**Drawing the Cubes**

* While the skybox and cube are technically different sizes, but this can be achieved with the same vertex and index data, you’ll just need to scale up the skybox and reverse the culling in the rasterizer state.
* You’ll need to have the cube switch between a series of textures. The best way to approach this would be to load one texture from the folder at a time, and display it.
* A skybox uses a cube texture and uses the normalized vertex positions as its texture coordinates. This MUST be a TextureCube shader side if you want it to sample properly.

**Texture Loading**

* D3D11 doesn’t have any built in texture loading functionality, so you’ll be using the one supplied for you in the utilities folder. This loader only loads DDS files though, so you’ll want to take the textures folder from the server, or convert the files to DDS yourself.
* We can pull the description of a texture using its ShaderResourceView. This will allow us to check what KIND of texture, amongst other things, that the texture we loaded in is.

**Threading**

* You’ll need to use a DIFFERENT deferred context within each of your threads you use. It works very similar to an immediate context so it shouldn’t be too hard to transition to.
* Your load thread should load in all possible textures, then randomly choose one to present to the cube. This means if you add textures to the folder, it will choose from more textures.

**Thread Naming**

* If you want to be able to name a thread to make debugging easier,  you can use this handy function and structure to do so:

// HANDY THREAD NAMING FUNCTION FOR DEBUGGING (see a threads name in the debugger)

typedef struct tagTHREADNAME\_INFO  
{  
DWORD dwType; // must be 0x1000  
LPCSTR szName; // pointer to name (in user address space)  
DWORD dwThreadID; // thread ID (-1 = caller thread)  
DWORD dwFlags; // reserved for future use, must be zero  
} THREADNAME\_INFO;

void SetThreadName( DWORD dwThreadID, LPCSTR szThreadName )  
{  
THREADNAME\_INFO info;  
info.dwType = 0x1000;  
info.szName = szThreadName;  
info.dwThreadID = dwThreadID;  
info.dwFlags = 0;

\_\_try  
{  
RaiseException( 0x406D1388, 0,  
sizeof(info) / sizeof(DWORD),  
(const ULONG\_PTR\*)&info );  
}  
\_\_except( EXCEPTION\_CONTINUE\_EXECUTION ) {  
}  
}

// Example usage:  
///SetThreadName(-1, "Main thread");

**Waiting for threads**

* to make your life even easier, here's a simple wait macro you can also use:

// Macro which lets us safely wait for a thread if it is still alive  
#define WAIT\_FOR\_THREAD(\_thread) if((\_thread).joinable()) (\_thread).join();

**Homework Bonus**

**Instancing**

* If you want to attempt instancing this is a great lab to try it with. You can define in your Input Layout data that is INPUT\_PER\_INSTANCE instead of the standard INPUT\_PER\_VERTEX data.
* With this information, you can have a series of different world matrices to represent different cubes.
* If you have a branching pixel shader that uses an Index for textures, you can give each of the instances a different index as well, so the cubes are textured differently!