NoiseEffectCompiler

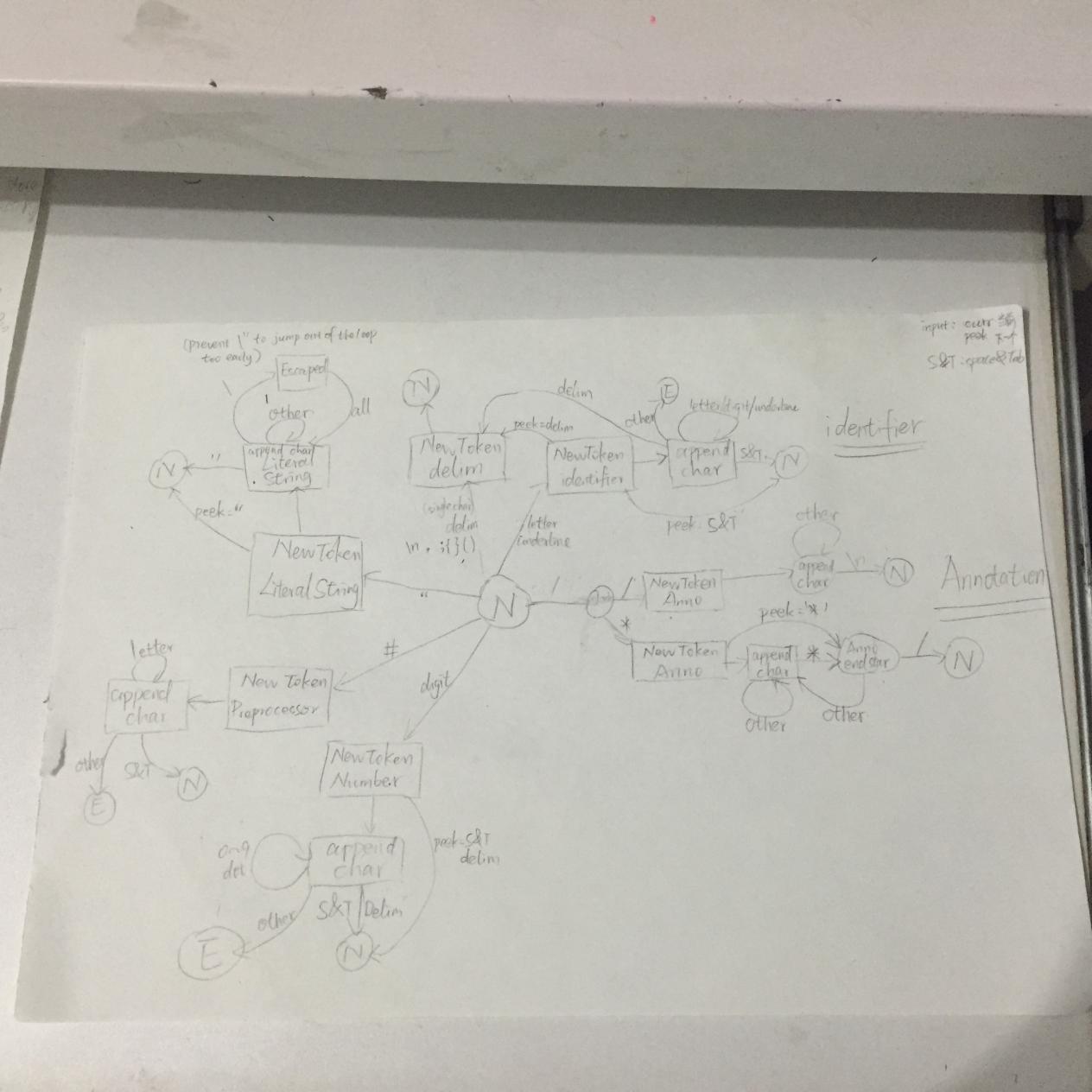
* 简介：反正就是FX11的release版各种错误什么的烦死了，加上这学期在学编译原理，想着要不自己写个简单的Shader组织框架好了，把Effect11给抛弃掉（？）。所以就开始了折磨自己的旅程
* (2016.10.16)一些支持的东西：现在是打算这样的，NoiseEffectCompiler只负责解析.NoiseEffect文件，获取Technique/Passes/不重复的shader 的列表，然后获取到所有要编译的shader以后，hlsl的代码交给D3DCompiler来做，之后再用D3DReflect来得到“资源绑定”的信息（哪个CB，Texture绑定在哪个shader上，在用slot几）（话说d3d的资源绑定貌似也是基于状态机,Slot的编号只是指定当前状态的某个槽而已，槽数应该就是同一状态可绑定的最多资源数目了）。之后解析好NoiseEffect文件以及编译好HLSL之后，就生成一个中间文件，储存了各个shader的二进制数据以及资源绑定的数据，到时在Noise3D里面加载这些“中间代码”就可以像FX11一样，创建Effect，创建Technique，Passes等了。
* (2016.10.16)支持的语法：c风格注释，””字符串，整数，c风格预编译指令，c风格标识符。#include 加上要编译的源文件（到时直接拿这些文件去D3DCompile），Technique和Pass跟FX11差不多，要求有匹配的花括号｛｝，但是要求一定要起名字（？），在Pass里设置shader用以下语法：

VS(函数入口点,版本);

PS(函数入口点,版本);

GS(函数入口点,版本);

其中版本是shader版本，例如VertexShader的vs\_5\_0,PixelShader的ps\_5\_0等

* 词法分析器：编译原理讲了一堆正则表达式转NFA,DFA之类的东西，感觉有点似懂非懂，最后考虑到词法也没多复杂，就用状态机(State Machine)手撕简单的c风格词法，手写词法分析器(lexical analyzer)。基本上只要最多看当前和下一个字符就可以确定状态机要转移的下一个状态（复杂注释的结尾 \*/ 就要读入两个字符来判断。）
* 上图的Lexer状态转移图可能还写的不够完整，就是根据读进不同的字符决定状态机转移到哪个状态。其中除了Literal string和注释以外，其他状态只要读入了空格/Tab/分隔符/斜杠就要强行进入对应的状态，而不是返回至LS\_NORMAL状态。在NoiseEffectCompiler的Tokenizer实现里，状态转移和某个状态的Output动作是分离的，减少耦合。例如转移到LS\_XXX\_NEWTOKEN状态的时候，就给XXX创建一个新的Token（词法单元？），然后再给这个词法单元读取完整。有点懒了...具体看看ITokenizer的代码就好了，一目了然，注释还不少。
* (2016.10.16)现在准备写Parser，就是对词法分析器得到的一大堆Token进行语法分析得到更高层次的信息。语法可能更加简单，而且Effect这种配置shader的小脚本，也不会需要复杂语法和递归等，所以这可能就是LL(1)文法？（从左扫描，只需读取一个Token就可以选择出对应的生成式Production），所以的话，可能采用的方案就递归下降(recursive descent)解析，毕竟Technique，Pass和shader就是树状关系，递归下降应该很符合这种结构。
* (2016.10.30)HLSL Compiler：hlsl的代码还是交给api去做，那就是Win8SDK里的D3DCompile，据说很高端（？），但是这个就不要include 2010June的DXSDK了。然后就是要自己实现一个”ShaderInclude”的结构体并重写它的两个函数Open和Close好像是，不过其实亦可以用一个默认的ShaderInclude，这个看MSDN吧，传入某个MACRO就是使用默认的shaderInclude接口。
* (2016.10.30)ShaderRlection:这个很tm关键，现在是用<D3DCompiler.h>里的D3DReflect()函数对compiled shader bytecode 进行资源绑定分析(Resource Binding)，也就是当前的shader使用到了什么cbuffer,textures,samplers,UAV等东西。这个就是在用ID3D11ShaderReflection接口get到Shader Input Description 得到这个shader用到的各种resource和buffer,。ID3D11ShaderReflectionConstantBuffer接口还能得到cbuffer的byteSize和它附属的各个变量的offset/Size的信息，总之还是挺好用的。不过暂时不打算getVariable，直接一刷刷一个cbuffer先吧。
* (2016.12.1)文件结构：非常像.3ds文件，有一个个block组成，每个block有他的sub-block。文件以一个MAGIC NUMBER开头，可以辨别.nxo的版本的validity，每个block从一个ID开始，就是一个1 byte的常量，之后每个block的具体格式已经规定好。然后整体结构是：文件头+全局资源池+Effect树状结构，其中每个shader都储存了要用到的资源(cb,tb,texture,sampler)等的名字，得到名字可以在全局资源池里面找到对应的具体资源。

//Desc : 4 bytes header

//Father : Global

//Begin of : magic number

//-----used to recognized noiseEffect file

const UINT NOISE\_EBI\_MAGIC\_NUMBER = 0xabcd1234;

//Desc: EBI(1 byte) + NAME(string) + binaryBlockSize(4 bytes) + binaryData (variable)

//Father : Global

//Begin of : new GLOBAL BLOCK

//----- create resources/buffers in unique Global Resource Pool

const uchar NOISE\_EBI\_NEW\_CB = 10;

const uchar NOISE\_EBI\_NEW\_TB = 11;

const uchar NOISE\_EBI\_NEW\_TEXTURE = 12;

const uchar NOISE\_EBI\_NEW\_SAMPLER = 13;

const uchar NOISE\_EBI\_NEW\_SHADER\_BINARY\_BLOCK = 14;

//----- Create new Effect/Technique/Pass belong to current father node

//----- please carefully manager current states

//Desc: EBI(1 byte) + Technique Count(2 byte) + sub-block

//Father:Global

//Begin of: new effect block

const uchar NOISE\_EBI\_NEW\_EFFECT = 0;

//Desc: EBI(1 byte) + Passes Count(2 bytes) + Technique name(string)+ sub-block

//Father: Effect Block

//Begin of : new technique block

const uchar NOISE\_EBI\_NEW\_TECHNIQUE = 1;

//Desc: EBI(1 byte) + Pass Name(string)+ sub-block

//Father: Technique Block

//Begin of : new pass block

const uchar NOISE\_EBI\_NEW\_PASS = 2;

//Desc: EBI(1 byte) + sub-block

//Father: Pass Block

//Begin of : all kinds of new shader blocks

//----- Create new shaders, and identify a start of sub-blocks

const uchar NOISE\_EBI\_NEW\_VS = 200;

const uchar NOISE\_EBI\_NEW\_PS = 201;

const uchar NOISE\_EBI\_NEW\_GS = 202;

//Desc: EBI (1 byte) + name(string)

//Father : Shader Blocks

//Begin of: names of bound resources

//----- resource/buffers can be REFERENCED BY NAME in global resource pool

//----- and these resource (names) belong to current shader

const uchar NOISE\_EBI\_NAME\_CB = 60;

const uchar NOISE\_EBI\_NAME\_TB = 61;

const uchar NOISE\_EBI\_NAME\_TEXTURE = 62;

const uchar NOISE\_EBI\_NAME\_SAMPLER = 63;

const uchar NOISE\_EBI\_NAME\_SHADER\_BINARY\_BLOCK = 64;