完成整体render架构，device等基本驱动类设计后可以开始vertex post-processing为止之前的部分开发。整体完成后再进行后续设计。类似螺旋或增量模型。

考虑优化：

1. 数据不变时，不再重复渲染。直接显示前一帧
2. 多线程：多个渲染目标同时渲染，多个pass同时处理(到ps为止)
3. 使用性能测试工具测试性能

Next：

开始实际编码

Completed Rendering PileLine

1

Get Vertices Data

Construct the Polygon Mesh

Output Merger

Pixel Shader

Rasterization

Face culling

Vertex Post-Processing

Vertex Shader

Input Assembler

3

2

1. Clipping
2. Perspective divide
3. Viewport transform

4

4

Stream Output

(Run Time Data IO)

Cull the primitives if the triangles face to the viewer or not(Base on Cull Mode:CW or CCW)

每个阶段必须是只要输送数据据就可以独立开始和独立结束的，与其他阶段不相关的，以便于开展流水线

5

6

Rasterize the primitives

(Convert triangles to pixels)

7

Merge PS output and other render resources(Render Targers) together

Stencil Test

Depth Test

Blending

8

8

Render Targers

9

10

To Swap Chain

‘

Create Device

Create Front/Back Buffer

Windows Message Loop

Device Render

Technique(Only support 1 technique now)

Pass 1 (Run a full pileline here)

VertexShader()

PixelShader()

Pass 2 (Run a full pileline here)

VertexShader()

PixelShader()

Swap the front buffer and back buffer

Other Passes …..

Output Render Result To Back Buffer

Present Front Buffer

Data Flow

Vertices、Adjacency(Faces、Edges)、Attributes(Vertices color、Normals、UV、Materials)

1

PolygonMesh Data

2

3

Vertices, *optional*<Normals>,<UV>

4

PolygonMesh Data with transform

5

PolygonMesh Data, with all vertices projected to the viewport

Triangles

6

7

Every Pixels which fill in the triangles

Pixel Colors/Depth/Other user-defined values

8

9

Buffer data

Buffer data waiting for present

10

Classes Define namespace SREngine

除了高层类或抽象类以外，一个类不应直接涉及其他类

SoftRender\_Engine.h

SRE\_GlobalAndUtils.h

SRE\_Math.h

SRE\_Resource.h

SRE\_Rendering.h

SRE\_Mesh.h

SRE\_Device.h

SRE\_PileLine.h

SRE\_Shader.h

SRE\_DebugLog.h

SRE\_Engine.h

Initialize():void createdevice createswapchain

OnStart():void createtechnique createpass setVSPStoPasss

OnRenderStart():void setmeshToPileLine setVarible

OnRender():void technique->Run

OnRenderEnd():void

OnStop():void

OnReStart():void

Destroy():void

Initialize()

OnStart()

OnRenderStart()

OnRender()

OnRenderEnd()

OnStart()

OnStop()

Destroy()

SRE\_Math.h

Cross(Vector,Vector):Vector

Dot(Vector,Vector):Vector

Class Vector

+x:float

+y:float

Class Vector3:Vector

+z:float

Class Vector4:Vector3

+w:float

Class Matrix33

Class Matrix

Class Matrix44

SRE\_Resource.h

+CreateBuffer(SRE\_BufferDescript,userDefine\_Data,Buffer\*): RESULT

Class Color

Class Sampler

Class Texture: SRE\_IContainer

Class SRE\_BufferDescript

+BufferSize:int

+PerDataSize:int

+BufferType:SRE\_BUFFERTYPE

+BufferFormat:SRE\_FORMAT

Class SRE\_Buffer: IContainer

+GetDescript():SRE\_BufferDescript

-Descript: SRE\_BufferDescript

-Data:Byte array

SRE\_Rendering.h

Class Material

Name

Diffuse

Ambient

Specular

Class Light

SRE\_GlobalAndUtils.h

enum RESULT {SUC, FAIL, INVALIDARG, OUTMEMORY}

SRE\_FORMAT\_VERTEX\_XY

SRE\_FORMAT\_VERTEX\_XYZ

SRE\_FORMAT\_VERTEX\_XYZW

SRE\_FORMAT\_ATTRIBUTE\_TEXCOORD1

SRE\_FORMAR\_ATTRIBUTE\_TEXCOORD2

SRE\_FORMAR\_ATTRIBUTE\_TEXCOORD3

SRE\_FORMAR\_ATTRIBUTE\_TEXCOORD4

SRE\_FORMAT\_ATTRIBUTE\_NORMAL

SRE\_FORMAT\_ATTRIBUTE\_DIFFUSE

SRE\_FORMAT\_ATTRIBUTE\_SPECULAR

SRE\_FORMAT\_ATTRIBUTE\_BINORMAL

SRE\_FORMAT\_ATTRIBUTE\_TANGENT

SRE\_FORMAT\_PIXEL\_R8G8B8A8

SRE\_FORMAT\_PIXEL\_R8G8B8

SRE\_BUFFERUSAGE\_VERTEXBUFFER

VIEWPORTSIZE

WORLDVIEWPROJMATRIX

WORLDMATRIX

VIEWMATRIX

WORLDVIEWMATRIX

PROJECTIONMATRIX

VIEWPROJMATRIX\_INV

WORLDVIEWPROJMATRIX\_INV

WORLDMATRIX\_INV

VIEWMATRIX\_INV

WORLDVIEWMATRIX\_INV

PROJECTIONMATRIX\_INV

GetSystemBits():int

GetProgramBits():int

SRE\_GlobalAndUtils.h

*Virtual Class BaseContainer*

*Virtual Class BaseTask*

*+Run():void*

SRE\_Mesh.h

*Virtual Class IMesh*

*+SetName(string):void*

*+GetName():string*

*-Name:string*

Mesh 访问层

具体Mesh数据

Mesh数据访问层

Abstract class MeshManager

Public:

Face-vertexSize;

Edge-vertexSize;

Edge-faceSize;

Vertex-EdgeSize;

void\* GetFace-Vertex(FaceIndex,index)

void\* GetEdge-Vertex(EdgeIndex,index)

void\* GetEdge-Face(EdgeIndex,index)

void\* GetVertex-Edge(VertexIndex,index)

void\* GetVertex (VertexIndex)

void\* GetAttribute (VertexIndex)

private:

vertexFormat

attributeFormat

Mesh具体数据层

Abstract class IMesh，friend class MeshManager

实现上面的MeshManager接口，并添加一个指向具体mesh类的指针。使用时初始化该类并传入该类的指针，Manager就可以用内部函数遍历该具体mesh类的数据。

取数据部分大致就是这样，还要再参考下directx以及其他图形学书籍来变更一下。

1. 输入数据：用户设置顶点格式，传递struct
2. 读取数据：根据顶点格式和struct大小读取
3. 具体使用以下数据结构来存储数据：

VERTEX4\* -> VertexList

BYTE\* -> AttributesList

INT\*\* -> FaceList & EdgeList

Class Trianglemesh

VertexList

AttributeList

**~~E-VFList~~**

F-VList

V-EList

SRE\_Mesh.h

Class SRE\_TriangleMesh: IMesh

Data Structure:

Face -Vertex List

F0 {vertex1, vertex2, vertex3,

edge1,edge2, edge3}

F1 {vertex1, vertex2, vertex3,

edge1,edge2, edge3}

…….

Edge-FaceVertex List

E1 {face1,face2,vertex1,vertex2}

E2 {face1,face2,vertex1,vertex2}

…….

Vertex-Edge List

V1 { edge1,edge2,edge3…..}

V2 { edge1,edge2,edge3…..}

…….

Attributes {*optional*<Normals>,<Color>,<UV>}

-VertexList

-AttributeList

~~-IndexBuffer~~

-EdgeList

-FaceList

Did a little changes to Wing-Edge structure

private:

vertexFormat

vertexNumber

faceNumber

edgeNumber

perAttriSize

SRE\_Mesh.h

Virtual Class IMeshManager

Class X\_ Manager: IMeshManager

Class Pmx\_Manager: IMeshManager

Class Fbx\_Manager: IMeshManager

Class Obj\_Manager: IMeshManager

SRE\_Device.h

+CreateDevice(SRE\_DeviceDescript, SRE\_Device\*):RESULT

Class SRE\_DeviceDescript

+Windowed:bool

+BufferWidth:int

+BufferHeight:int

+BufferFormat: SRE\_PIXEL\_FORMAT

+BufferCount:int

+OutputWindow:HWND

+RefreshRate:int

Class SRE\_Device

+Present():void

+CreateSwapChain():RESULT

+GetDescript():SRE\_DeviceDescript

+GetSwapChain():SRE\_SwapChain

+BondPileLine():RESULT

+SetTechnique():RESULT

-pSwapchain: SRE\_SwapChain\*

-pPileLine:SRE\_PileLine\*

-devicedecript: SRE\_DeviceDescript

Class SRE\_SwapChain

+GetPresentBuffer():RESULT

+GetOutputBuffer():RESULT

+swap():void

-BufferList:array

-CurrentPresent:pointer

-CurrentOutput:pointer

SRE\_PileLine.h

CreateRenderTarget():RESULT

CreateTechnique():RESULT

CreateRenderPass():RESULT

Class BasicInput: BaseContainer

Class BasicOutput:BaseContainer

Class SRE\_Technique: Task

+AddRenderPass():RESULT

+GetRenderPassByIndex():SRE\_RenderPass

+GetRenderPassNumber():int

+GetName():string

+SetName():string

-RenderPassNumber:int

-PassList:array

-Name:string

Class SRE\_RenderPass: Task

+SetVextexShader():RESULT

+SetPixelShader():RESULT

+SetOutputTarget():RESULT

+SetMesh():RESULT

+GetName():string

+SetName():string

-StartPileLine():void

-OutputTarget:pointer

-MeshList:array

-Name:string

每条流水线对应一个RunTimeData，只供流水线内部使用

Class SRE\_RunTimeData: IContainer

template<typename T>

+GetData(int pos,int listIndex,const T \*):RESULT

+SetData(int pos, int listIndex, const T \*):RESULT

+CreateDataList():RESULT

-DataListArray:array

每条流水线对应一个ConstantData Buffer, 该buffer应可供用户任意获取其中的数据，初始化由InputAssembler的passVarible()实现 \*需要用户预定义，比如需预定义WorldViewProjMatrix才能获取正确的数据

Class SRE\_ConstantData: IContainer

-ViewportSize:Vector2

-WorldViewProjMatrix:Matrix

-WorldMatrix:Matrix

-ViewMatrix:Matrix

-WorldViewMatrix:Matrix

-ProjectionMatrix:Matrix

-ViewProjMatrix\_inv:Matrix

-WorldViewProjMatrix\_inv:Matrix

-WorldMatrix\_inv:Matrix

-ViewMatrix\_inv:Matrix

-WorldViewMatrix\_inv:Matrix

-ProjectionMatrix\_inv:Matrix

SRE\_PileLine.h

SRE\_PileLine.h

Class PileLine: Task

Class SRE\_PileLine: PileLine

+SetInputAssembler():RESULT

+SetVertexPostProcesser():RESULT

+SetRasterizer():RESULT

+SetOutputMerger():RESULT

*Virtual Class IPileLineBuilder*

*+BuildProcessPart(ProcessPartBuilder\*)*

*+BuildShaderPart(ShaderBuilder\*)*

*+BuildRasterizerPart()*

*+BuildOutputPart()*

Class SRE\_PileLineBuilder : IPileLineBuilder

+BuildProcessPart(ProcessPartBuilder\*)

+BuildShaderPart(ShaderBuilder\*)

+BuildRasterizerPart()

+BuildOutputPart()

Class SRE\_PileLineCreator

+BuildPileLine(PileLineBuilder\*, PileLine \*):RESULT

*Virtual Class ProcessPartBuilder*

*+BuildProcessPart ():RESULT*

Class VertexPostProccesserBuilder：ProcessPartBuilder

-CreateVertexPostProccesser ():RESULT

Class InputAssemblerBuilder：ProcessPartBuilder

-CreateInputAssembler ():RESULT

SRE\_PileLine.h

Class SRE\_InputAssembler: Task

+CreateRunTimeDataBuffer():RESULT

+CreateConstantBuffer():RESULT

+PassVaribles():void

+SetMesh():RESULT

+GetMesh():RESULT

Class SRE\_VertexPostProccesser: Task

-Clipping():void

-PerspectiveDivide():void

-ViewportTransform():void

-FaceCulling():void

应设计成便于开展多线程的结构，比如一个函数运行2x2像素，那么便可多线程下同时调用这个函数。使用三角形重心坐标进行插值。

主要任务：对每一个屏幕上的三角形，根据顶点位置进行插值获得一个新的像素位置，并传入ps中，同时对normals，pos等值进行插值并作为参数传入ps中。每计算一个像素位置调用一次ps。

Class SRE\_Rasterizer: Task

+SetPixelShader():void

在现阶段先实现wireframe或顶点渲染。后续再完整实现

根据FORMAT进行计算，比如FORMAT指定NORMAL,就在normal()函数中获取或计算normal然后给出，否则返回空

SRE\_Shader.h

Class VSInput: BasicInput

为这2个类添加顶层抽象类

Class VSOutput:BasicOutput

Class VertexShader :Task

+SetCallBackShader(BasicVSOutput (\*func)( const BasicVSInput &)):void

+SetInputArgumentsFormat(SRE\_FORMAT):void

~~-PassArguments():void~~

-SaveOutput():void

-CallBackShader: BasicVSOutput (\*func)( const BasicVSInput &)

-InputArgumentsFormat: SRE\_FORMAT

SRE\_Shader.h

Class PixelShader :Task

+SetCallBackShader(float4 (\*func)( const BasicVSOutput &)):void

+SetInputArgumentsFormat(SRE\_FORMAT):void

+SetOutputTarget():void

-PassArguments():void

-SaveOutput():void

-CallBackShader: float4 (\*func)( const BasicVSOutput &)

-OutputTarget:Texture

-InputArgumentsFormat: SRE\_FORMAT