！下划线命名法！

工具层 （编辑）

功能层 （逻辑 输入 相机 角色控制 动画 物理 渲染 网络 IO 内存管理 其他）

资源层

核心层 （创建线程 内存分配 容器创建 数学模块）

平台层

只能上调下，不能下跨上

问题:

内存管理？？内存泄漏问题？

名称类型？？ char [32] / string

帧资源切换

FBX读取没算切线

PSO 暴露参数不够

窗口缩放 需要resize刷新很多东西

贴图必须有默认的

只支持三角形列表 （需要提供多种输入方式 比如三角带）

需要提供运算符重载 比如[]

Shader反射提供pass的输入输出

9.22.2022

Idea: pass的构建按照 pass\_layout,具体构建需要遍历object的type 及内部mesh的类型，输入输出texture的名称

Done: cpu基础类构建，

Todo:在render\_system 里构建pass，具体到 render的子类里面构建 ok

9.23.2022

Idea: stre\_render 类全部封装为抽象类， 具体变量保存在子类中。

base\_type 也保存在stre\_render中，使外部用相同的基础类型

所有数据都保存在resource的data\_ptr中，只能在cg\_resource\_factory初始化和读取

每个资源类型必须有自己的结构体保存自己的数据（包括父类的数据）。

data\_ptr头储存为内容的结构体，后面为数组缓存区域，需要调整内容的指针指向这些区域

一个子物体一个材质一张贴图

Done: cpu基础类构建,内存的管理结构，fbx静态读取

Todo: 动画部分

9.24.2022

Idea: resource 具有布局引导render构建GPU资源

外部没有必要控制GPU的指令和内容

不同的render\_system可以用不同的render\_api,

gpu\_resource有多个gpu\_resource\_element以应对一个物体强耦合的多个资源

render内只具体化构建gpu\_resource\_element

Done：gpu资源构建的结构

Todo ：render内具体化构建 ok

9.25.2022

Idea: 资源从 CPU到GPU

1.利用reource\_factory构建默认的resource  
 2.再利用resource 构建对应物体类型  
 3.物体类型的构造函数中构建 GPU layout

Done：gpu\_resource 的具体构建

Todo ：pass的具体构建 ok

9.26.2022

Idea: Pass的堆 需要独立构建， texture的堆也是独立的（跟随mesh吗？？）

光照也是数组 SRV

GPU\_layout必须构建好，尤其是index 和vertex，（数量即元素数量）

必须按照顺序写shader

Done：利用pass 和 场景资源的DX API draw

Todo：各个物体类型的CB 和GPU\_layout的构建

9.28.2022

Idea: 摄像头和灯集合在场景里

模型的材质需要重新保存为GPU格式才能输入进GPU

Done：GPU导入结构

Todo：数据刷新和参数配置

10.09.2022

Idea: gpu\_layout 和gpu\_resource强力绑定

每次希望更新gpu\_resource需要标记layout为need\_update

Done：render元素级别的更新函数

Todo：render\_system提供回调给外部，收集所有的更新请求再使用render更新

10.10.2022

Done:render\_system 具备回调队列，回调类，添加函数

Todo：具体化resource\_manager 调用render\_system的添加函数控制资源的GPU内容更新

///////////////////////////////////////////////////////////////////////////////

/////////////////////大改架构///////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

10.26.2022

Done:大改架构

gpu\_resource 自己更新 （使得随意添加资源可能）

gpu\_pass 自己更新,factory添加gpu\_resource指针 （使得随意添加资源可能）

render直接读pass的res指针装填数据 （使得随意添加资源可能）

cpu\_manager为模板处理各种类型的cpu数据以及控制gpu\_resource 更新和装载 （方便添加新类型）

Render\_system读取factory存储的更新函数functor队列执行GPU更新。（使得统一更新和需求化更新）

Todo:完善各种manager，

10.27.2022

Done: texture 和table的package

gpu texture的构建

新加followmesh的资源类型，跟随子物体索引替换CBV

Todo：验证贴图导入和SRVdesc的关系

完善各种manager

Texture清理

11.02.2022

Done： 清理了目前的报错，（内存分配器停用，改回new）

Todo：设计库导出接口

11.04.2022

Done：文档简单的类介绍

基本类型的接口设计 （取消了资源类型的策略模式，改回工厂）

Todo：简单的场景实现

11.05.2022

Done：自定义资源为帧资源，随当前帧刷新，更新时将全部更新，被使用时会刷新fence值

贴图和rt资源都只有一张

内藏backbuffer，pass设为is\_output直接输出到屏幕

帧指针，交换链指针，栅栏的实现

Todo：简单的场景实现

11.09.2022

Done: 输出默认画面

修改了交换链backbuffer的表示方法

RT不用const

清理RT和切换RT状态

Todo：简单的场景实现

11.16.2022

Idea：命令模式，UI不做任何操作（包括UI的生成），

所有操作由命令完成，

数据传输方法目前为全局变量

Done: 初步完成命令模式的QT调用，

管线窗口的初步代码

Todo：使用调试和完善前端代码和命令

11.20.2022

Done ：简单的界面和debug跑通

Todo：内存释放，析构函数

共享指针

界面功能完善（texture package ，pass allocate time）

11.21.2022

Idea:由于GPU资源没有互相指向的结构，可以全数交给共享指针管理内存

Done: gpu\_shader\_resource全面使用共享指针

Todo: 前端的内存控制

11.21.2022

Done:前端的内存控制， QT组件析构释放资源，跑通画面

Todo:前端功能实现，shader反射

11.23.2022

Idea:暂时要求shader和Pass连接时直接更新队列，执行编译后，马上反射到pass控件上

Done:shader反射，并反射到前端接口重建

Todo:前端功能实现，gpu\_resource和寄存器号解耦

11.27.2022

Done：删除功能但未调试

Todo：调试，多pass实现

11.28.2022

Done:删除功能调试, 暴露属性类型选择未调试

Todo:调试属性类型

11.29.2022

Done: 用户控制pass优先级的多pass实现，初步debug\_output 的构建

Todo: 调试， 让debug\_output兼容大多数错误并引导用户操作

11.30.2022

Idea：重建gpu前必须要执行一遍gpu命令

Done:重构命令模式，让参数作为命令类的成员传递。

debug\_output的实现。

维修各种连线逻辑的bug。

Shaderd的根签名输入贴图（根参数要注意保留生命到根签名创建结束）

在创建设备前启动debug检查DX错误

Todo：release工程调试，多pass成功执行

12.2.2022

Done:多pass成功执行，选中边框加深，连线位置修改

Todo：摄像机 灯 模型导入 材质和贴图的关系

12.06.2022

Done：模型导入，摄像机灯光组件，摄像机移动，常量缓存和着色器资源缓存的输入

Todo：Debug

12.09.2022

Done：Debug相机视角移动，维修了模板资源控制器中的内存大小获取问题

将帧资源提高到5个缓存，保证有充足时间执行命令队列

暴露pass和维修mesh的属性

Todo：Debug，mesh的属性

Mesh的法线有损坏？

光照流程

12.11.2022

Done：改bug，改变object矩阵计算顺序，view的刷新,控件拖拽连线刷新，light pass

检查pass所有的输入是否正常才拿去渲染

Todo：Debug light pass

断连debug，

暴露光参数，

检查为什么不能两个一样的srv输入

12.12.2022

Done: 暴露光参数,Debug light pass

Todo: 检查为什么不能两个一样的srv输入 ，输入同一个相机

12.13.2022

Done：GPU资源带表记录pass的根签名索引Debug light pass，录制和制作视频

Future：FBX 贴图导入，保存管线，动画运行，材质系统

02.20.2023

Done: 改善了下UI ，和一些致命bug

Todo：PCSS实现

02.27.2023

Done: PCSS

Todo:GI

02.28.2023

Done: GI-SSR

Todo: 自定义参数反射到界面 ， 摄像机移动问题，摄像机分辨率，可视范围调节，阴影图大小和深度精度问题

多pass实现 ：

用户控制pass优先级

按照优先级顺序绘制过程

属性窗体设计：

Mesh ：选择类型

FBX路径

OBJ位置

贴图

Texture: 选择类型

贴图路径

Shader:类型 + 路径

[]()->return\_type{};

没有[=,&]捕获的lambda函数才能转换成函数指针！！！

启发：

1. 类里面有该变量或指针（强关联，需要在意include结构），也可以在别的地方用数组保存用id识别
2. 父类包含许多获取的虚函数但不拥有变量，子类继承该虚函数并拥有私有变量，只允许用户使用父类，保证变量安全性
3. 工厂模式构建子类但返回父类指针，保证dll按父类的结构执行
4. 通用的遍历或者统一的格式太庞大，使用回调将通用代码分解开，各自交由外部实现（回调传递具体的函数，例如各种资源的创建和更新）
5. 回调架构：
   1. 回调类C包含 **输入参数** 和 **函数指针**
   2. 函数指针**形参**包含 **输入参数** 和 **能被利用的内部私有参数**
   3. A 函数构建时在内部使用 **私有参数写逻辑**
   4. B 收集函数构建回调类C入队
   5. B 执行函数遍历队列执行出队
6. DLL导出： 需要导出隐藏函数类 .cpp函数声明为 \_declspec(dllexport) .h声明为\_declspec(dllimport)
7. Git, commit为做记录保存到本地，push为将本地的commit提交到云端，pull为将云端的commit拖到本地，megre时检查两个分支有没有修改相同的地方。

模板编译和多cpp多类型的问题？？？？

模板类不允许分支特化？（需要用技巧？）

在库中编译模板必须要显示实例化所有情况，（模板只是提供给外面扩展用的，并不会自动编译库内容，（外面编译时并不会再编译一遍库））

模板被使用时，要确保模板的方法已被定义（所以最基本的模板会直接在头文件定义）

控制台程序窗口程序等类型改变

进入project->setting->Link, 在Project options中的/subsystem:windows.

用时间+地址确立uid唯一性

Triangle strip 三角带 不需要索引，ABCDEF 顶点集合 直接翻译成—》 ABC、BCD、CDE 和 DEF三角形

QT会自动释放孩子组件 不需要自己在析构中释放

QT点击事件函数return父组件事件触发函数会将事件传播给父组件

QT的库用环境变量导入

每个函数应该保证对输入有检查，并规定输入要求和预期输出结果

List删除时要it = list.erase() 获得下一个位置

头文件声明的static变量只会导致每个源文件都包含一个新的本地该变量。应该头文件extern，在随便一个cpp中声明

命令模式中，命令所需求的变量可以以命令的类变量形式输入

TEXTURE2D 必须以描述符表Table形式作为根签名和绑定输入

SRV UAV只用于RAW 和structbuffer

创建根签名时最好把根参数的生命周期延长到创建结束

每帧自带分配器？防止前一帧的命令没执行完？

DS的SR有对应输入格式要求。

SetGraphicsRootShaderResourceView 第一个参数是根签名的顺序！不是寄存器号

Cbuffer CBV常量缓冲区的意思是为256的倍数的大小的缓冲区（大小不能改变？），不是不能改变数值的缓冲区

Structedbuffer SRV意思是可变大小缓冲区

DXGI\_FORMAT\_R8G8B8A8\_UNORM 代表归一化 范围最多为0~1

&会将整个引用交出去改变，返回时重新获取引用，而GetAddressOf()只会交出指针，引用不变（比如建PSO时，引用也会改变，所以只交出指针会导致啥都没建）

矩阵乘法不可逆，因为旋转原点一直在世界原点，所以object的世界矩阵顺序应该先缩放再旋转最终再平移，

一定要保证pass执行时获取得了资源才指向地址，不然会爆炸

注意ComPtr<ID3D12Resource> 不能被同时使用

设计结构和功能时，需要考虑

1. 输入参数的范围 和处理能力（比如为nullptr该怎么办）
2. 输出的结果是怎样 （错误怎么处理，修改了什么 或者说结果是什么）
3. 尽量让功能为独立功能，放哪都能用的功能，不局限于输入的类型，减少使用的要求

均值采样容易锯齿，要用各种分布保证采样均匀 ，比如泊松分布

.sample()输入的是（0~1）uv坐标 TEXCOORD

Texture[][]输入的是像素索引值（1980\*1080）

POSITION VS输出的是NDC坐标(-1,-1,-1,-1)~(1,1,1,1)

经过rasterizate输入到PS变SV\_POSITION 屏幕坐标，xy是像素索引值（1980\*1080）

//poisson分布

float2 poissonDisk[NUM\_SAMPLES\_SSR];

float ANGLE\_STEP = PI2 \* float(NUM\_RINGS) / float(NUM\_SAMPLES);

float INV\_NUM\_SAMPLES = 1.0 / float(NUM\_SAMPLES);

float angle = rand\_2to1(pIn\_uv) \* PI2;

float radius = INV\_NUM\_SAMPLES;

float radiusStep = radius;

for (int k = 0; k < NUM\_SAMPLES\_SSR; k++)

{

poissonDisk[k] = float2(cos(angle), sin(angle)) \* pow(radius, 0.75);

radius += radiusStep;

angle += ANGLE\_STEP;

}