！下划线命名法！

问题:

内存管理？？

名称类型？？ char [32] / string

FBX读取没算切线

Over函数没写

Const没写

PSO 暴露参数不够

Pass 的输入改变时， SRVtable需要刷新

窗口缩放 需要resize刷新很多东西

贴图必须有默认的

顶点输入布局应该封装在rendersystem里面

只支持三角形

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

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

[](){};

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