用于记录个人在使用C++时不认识的函数

## emplace\_back()

向Vector末尾添加一个元素

与push\_back的区别：

push\_back会先创建这个元素，然后再移动或者复制到容器中

emplace\_back则是直接在容器尾部创建这个元素，省去了移动或复制这一步骤。

## std::weak\_ptr

弱类型指针

### lock

创建新的 std::shared\_ptr 对象，它共享被管理对象的所有权。若无被管理对象，即 \*this 为空，则返回亦为空的 shared\_ptr 。

等效地返回 expired() ? shared\_ptr<T>() : shared\_ptr<T>(\*this) ，原子地执行。

\_transparent\_column.lock()->add\_child(fusion\_object\_node);

以上代码可能存在问题，

1、weak\_ptr.lock()在运行的期间，对象可能被释放。

2、如果无被管理的对象时，则该行代码应该会报错。

## glm::identity

glm::identity<glm::mat4>();

用于生成一个单位矩阵。

即：

position为 0,0,0

Scale为1,1,1

Rotation为0,0,0

## glm::mix

类似unity中的lerp，用于取插值操作。

计算公式a+(b-a)\*t

## Position

glm::vec3 GetPosition(const glm::mat4 & matrix)

{

return glm::vec3(matrix[3]);

}

### matrix[3]

matrix[3] 代表了获取该矩阵的第四列

### glm中的矩阵

glm的矩阵元素如下所示：

m00 m10 m20 m30 ->第一列

m01 m11 m21 m31 ->第二列

m02 m12 m22 m32 ->第三列

m03 m13 m23 m33 ->第四列

### scale

glm::vec3 GetScale(const glm::mat4 & matrix)

{

return glm::vec3(glm::length(glm::vec3(matrix[0])),

glm::length(glm::vec3(matrix[1])),

glm::length(glm::vec3(matrix[2])));

}

#### glm::length(glm::vec3 vector3)

用于获取向量的长度

### rotation

//获取旋转矩阵

glm::mat3 rotationMatrix;

rotationMatrix[0] = glm::vec3(matrix[0]) / scale.x;

rotationMatrix[0] = glm::vec3(matrix[1]) / scale.y;

rotationMatrix[0] = glm::vec3(matrix[2]) / scale.z;

//将旋转矩阵转换为四元数或者欧拉角

glm::quat quaternion = glm::quat\_cast(rotationMatrix);//四元数

glm::vec3 eulerAngle = glm::eulerAngles(quaternion);//欧拉角

旋转矩阵为正交矩阵。

## opengl中模型矩阵的计算顺序 ：

1. 缩放
2. 旋转（不确定hmi项目中的旋转顺序是什么，在unity中是zxy）
3. 位移

## 类型转换

### static\_cast

实行安全的向上转换，即子类转父类；基本数据类型的转换

在编译时执行类型检查，不进行运行时类型安全检查，因此效率较高。如果进行下转型，必须确保转换是安全的，否则行为未定义。

### dynamic\_cast

实行安全的向下转换，即父类转子类

在运行时检查类型的安全性，如果转换失败（比如转换的类型不正确），返回nullptr（针对指针）或抛出std::bad\_cast异常（针对引用）。

### 强制转换

不安全，若使用，需确保转换的正确性