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

工具：

1. 蓝图，材质编辑器
2. DCC Digital Content Creation数字内存创造物 的 导入导出工具器

功能：

1. 功能顺序 + tick
2. 多线程

资源：

1. 需要转换成最佳的格式 （图片：DTS GPU纹理）
2. 资源互相的关系的文件 + GUID资源唯一号
3. 资产管理器Handle系统（返回你要的东西）
4. 资源内存生命周期 + 延迟加载

核心：

1. 数学效率
2. 数据结构（防止内存碎片 ， STL 不太行）
3. 内存管理 （pool，数据集中，顺序读取，批量增删）

平台：

1. 文件系统
2. 图形库（虚函数分库）
3. 提供的新硬件设备

初始化和回收：

1. 问题：默认构造析构导致执行次序不可预测
2. 方法1：静态单例化管理器，管理器内嵌到小管理器，并且不使用构造和析构函数，独立启动和终止函数 （影响开闭原则）
3. 方法2：全局优先级队列
4. 方法3：构建依赖图，自动计算优先级，自动按顺序初始化

内存管理：

1. 问题：malloc/free & new/delete 在内存随机分配，管理非常慢（用户到内核的上下文切换）
2. 法则：少堆分配，不在紧凑循环中堆分配
3. 方法1：自制栈分配器（或双端），安排一个指针指栈顶 （游戏关卡）
4. 方法2：池分配器，以元素为单位申请N倍空间，用链表包含所有元素，分配时拆断下一个元素，回收时插入，直接用地址（偏移）作为指针 （链表只有两个元素，后面元素就是一大片内存）
5. 方法3：对齐的分配器，总是2的幂为单位，使用掩码（单位-1），直接和内存AND得到余多少，最后加上差的部分（单位-余多少）。 这个差的值可以用多分配的内存来记录
6. 方法4：单帧分配器，用方法1，但每一帧自动全部清除
7. 方法5：双缓冲分配器，两个单帧交错，安心使用结果

内存碎片：

1. 问题：多次分配&释放后，内存不连续，分配失败
2. 方法1：堆栈或池分配器
3. 方法2：碎片整理，将已分配内存逐一往低位移动，并更新指针（或使用Handle或smart ptr）
   1. Handle：是句柄表的索引，句柄表的每个元素是一个指针 ？
   2. Smart ptr：智能指针是类， ？
   3. 运用某些库时内存不能被定位，要独立出来不整理
   4. 限制每帧可整理次数，分摊成本

自定义容器：

1. 可以完全掌控，优化，定制，无第三方依赖，自制并发结构
2. STL慢，内存大，多动态分配，游戏机编译有差异 （少量内存才用 ）
3. Boost lib太多，许可问题
4. STL port 方便各种编译器
5. 字典，二叉树，用键值排序（二分法搜索）
6. 散列表，键值按mod运算存储在表中（有可能碰撞）
   1. 开放散列表，碰撞时存在同一个位置
   2. 闭合散列表，碰撞时用规则算法探查下一个空位，大小最好为质数
   3. 探查法，线性探查（找下一个，或前后交替找），二次探查（前后交替）
   4. 散列法，把键转换为整数，再mod运算得到索引（越好越少碰撞）
   5. 字符串散列法，LOOKUP3,CRC32,MD5等

字符串处理：

1. 问题：动态分配，本地化
2. 字符串类可能产生多次拷贝等开销，（建议少用，引用传递）
3. 唯一标识符
4. 用散列表储存字符串集合
5. 本地化，字符编码，数据库管理所有字符

引擎配置：

1. 选项的保存，可配置性，每个开发者自己的设置

I/O系统：

1. C的I/O API : fopen()为有缓冲功能，open()则为无缓冲功能
2. 封装I/O API，可以多平台，简化调用，衍生功能
3. C的I/O都是同步的，程序必须等待函数返回
4. asyncOpen() asyncReadFile() asyncWait() 为异步，可以加入回调函数响应
5. 异步处理要注意优先权，先读取优先权高的

资源管理：

1. 资源调节管道，转换为引擎格式，调节参数的记录
2. 资源数据库，增删改查，磁盘移动，交叉引用，保存历史版本，多样化搜索联想
3. 三步： exporter原始数据读入引擎，compiler压缩改造，linker依赖资源的互相链接
4. 运行时：生命周期，处理复合资源，交叉引用，内存安排，
5. 目录组织，多路径导致磁盘需要寻道，合并成单一文件（免费的ZIP）更高效
6. 唯一标识符GUID
7. 注册表（字典），保证内存中无重复资源
8. 生命周期，永久，关卡期间，用完就丢（小短片），串流（背景音乐）
   1. 引用计数，先下一个关卡的资源引用加1，再上一个关卡资源引用减1，再导入卸载资源
9. 内存管理，堆分配造成碎片，栈分配限制，池分配需要分割成组块并冗余，
   1. 自由大小组块分配器：链表管理未用组块，组块内带栈分配器（一般用于同生命周期的资源 关卡，使得组块内也可以继续分配空间），！释放还是组块整体释放！
10. 复合资源，保证交叉引用
    1. 方法1，指针只能用于运行时
    2. 方法2，内部引用时，GUID表，保存引用，查找获取指针
    3. 方法3，内部引用时，指针修正表，序列化时内存不连续但相互引用的对象保存到文件时变得连续，文件内保存一个表，储存每个对象的起始偏移值，文件内对象也保存引用对象的该偏移值。P302
    4. 方法4，外部引用时，用GUID表或指针修正表同时，需保存对方的路径，文件内建立交叉引用表，导入时扫描该表。 内存也有主查表记录和获取已有的引用资源。
11. 导入后初始化，C需要一个表对应到初始函数，C++则（类 多态 虚函数 工厂等）

动态数组：

1. C的静态数组很友好
2. 动态数组，需要更多时分配更大的空间再复制过去 （效能慢，旧位置的内存碎片）
3. 外露式链表，节点含指针指向元素 每次都是新分配的节点（适合使用池分配器分配节点）
   1. Template<typename t>

Struct Link

{

Link <t>\* m\_pPrev;

Link <t>\* m\_pNext;

t\* m\_pT;

}

1. 侵入式链表，节点嵌在元素中，无需动态分配节点，或者从节点派生出元素类（但不能像指针一样被共享在多个链表中）
2. 循环链表，简化插入删除逻辑

Moderb C++ Design

游戏循环：

1. 消息泵，先处理来自windows的消息，再处理游戏循环
2. 回调驱动框架，lib要求完善回调函数或衍生类的虚函数
3. 事件更新机制，一个队列保存收到的事件，再在周期中按位置处理事件

时间线：

1. 使用上一帧的时间差来作为下一帧的执行参数有可能导致物理系统步进多次花更多时间
2. 方法1：使用连续几帧的平均时间差
3. 方法2：锁帧率，可以使物理系统使用最佳时间差
4. 多线程获取的时间会不同
5. 当暂停等，帧时间过长时，设置最大帧时间值1/60s等

并行：

1. 方法1：SIMD指令，多个数据同时运算
2. 方法2：分叉/汇合fork/join，
3. 方法3：引擎子系统，主系统控制和同步，子系统执行有隔离性的功能（渲染，物理，动画，音频）
4. 方法4：任务模型，任务（数据和代码组）队列，多个处理器取出处理。
   1. 异步，提交后过段时间等结果继续执行（可提前一帧提交任务）

网络架构：

1. client-server model 客户端输入/渲染，服务器逻辑
2. Peer to peer 每个机器只有特定对象的管辖权

世界编辑器：

1. 组块的构建和管理 定义世界组块，填入动态和静态元素
2. 可视化编辑 （自制渲染引擎，作为DCC工具插件，直接在游戏内调）
3. 导航功能（编辑时摄像机的快捷移动）
4. 选取功能
5. 图层功能
6. 对象的属性操作
7. 辅助（透视，安放，对齐辅助）
8. 特殊对象类型（粒子，灯，线条，声源，体积，导航网格）
9. 加载和保存功能
10. 资产管理工具

游戏性基础系统：

1. 在世界安放游戏对象
2. 关卡管理和串流（大世界加载和预加载）
3. 更新实时对象 / 动态产生和销毁
4. 消息和事件处理
5. 脚本
6. 目标和游戏流程管理
7. 对象能够按需获得底层引擎服务（粒子，物理，等等）
8. 随时间变化
9. 新增游戏对象
10. 唯一标识符
11. 对象的查询和引用
12. 状态机
13. 网络同步
14. 存档功能 （对象状态的存储，运行时类型识别RTTI/反射：获得类及其属性和方法，/抽象构造：直接先构建实例再从硬盘将数据序列化到内存）

1.工具方 2.运行时

以对象为中心：UE4

1. 所有物体都有具体对应的类型。对象中含有各种属性
2. 深且宽的继承关系 （虚函数的定义容易混淆 ，类型分类的标准写死 ）

多重继承，致命钻石：一个类型最多一个有祖父的类，其他都是独立的类（mix-in类），

改善1：冒泡效应，将新功能往基类送，导致基类越来越大

改善2：is-a合成为一个，新类型作为类成员，必须同时构造和销毁

改善3：has-a有一个，将类的功能全部拆分为独立类型（组件），再拥有他们

改善4：纯组件模型，没有根类型，组件上拥有实例化的id，用id将组件联系在一起（通信困难）

以属性为中心：（关系数据库的一列，类似纯组件模型）

1. 属性类含有表，记录各个对象的该属性的值
2. 属性间互相通信实现行为 / 专属脚本控制对象
3. 优点:1更好的内存管理，2轻松定义游戏对象（对象都是在属性表加一个id的事）
4. 缺点:1 属性关系和行为难维持实现 ，2 debug难

收集下第三方工具库和各种引擎（DCC digital content creation数字内容创作）的特点

Blender：建模

Houdini：特效

Maya：离线动画

SSAO:

在摄像机坐标系下利用z轴判断随机点和原点的前后遮挡关系

随机点需要转换到NDC投影中找对应视图的坐标数据

坐标系转换要注意齐次坐标缩放归一

1. 将坐标转换到摄像机局部坐标系，并需要缩放归一？

float4 viewpostion = mul(float4(gbufferd.xyz, 1.0), gView);

viewpostion.xyz /= viewpostion.w;

1. 利用噪声图构建随机方向，和该法线一起构建随机切线空间的转换矩阵

float3 normal = normalize(mul(gbufferb.xyz, gView));

float3 randomVector = NoiseMap.Sample(gsamLinearWrap, pIn.texcoord \* noiseScale).xyz;

float3 tangent = normalize(randomVector - normal \* dot(randomVector, normal));

float3 bitangent = cross(normal, tangent);

float3x3 transformMat = float3x3(tangent, bitangent, normal);

1. 利用随机方向数组获取切线空间的随机点

float3 samplePos = mul(kernelOffsets[i], transformMat);

samplePos = samplePos \* radius + centerDepthPos;

1. 转换到屏幕空间获取对应坐标，并转换到摄像机局部坐标系

float4 offset = mul(float4(samplePos, 1.0), gProj);

offset.xy /= offset.w;

float4 offsetposition = GBuffer[3].Sample(gsamPointClamp, float2(offset.x \* 0.5 + 0.5, -offset.y \* 0.5 + 0.5));

offsetposition = mul(float4(offsetposition.xyz, 1.0), gView);

offsetposition.xyz /= offsetposition.w;

float sampleDepth = offsetposition.z;

1. 判断遮挡和叠加

//距离 大于 半径 则趋近于0 表示未受遮挡

float rangeCheck = smoothstep(0.0, 1.0, radius / abs(centerDepthPos.z - sampleDepth));

//如果采样点在自己前面则不加，否则按照角度叠加遮挡

occlusion += rangeCheck \* step(sampleDepth, samplePos.z) \* nDotS;

}

//反向遮挡值

occlusion = 1.0 - (occlusion / kernelSize);

float finalOcclusion = pow(occlusion, power);

