//游戏流程:

// GameManager,UIManager,PlayerInput,CharactorController/EnemyController(Animator,Rigidbody), EnemyBehavior(Navmesh,NavmeshAgent) ,CameraController,HotFix ,Network

// Physics, Animator / Animation, Mesh(Meshfilter), Meshrender(shader) (IO模块), ParticleSystem, AI, Event, Audio, UI, Input

// 数学相关

// Matrix4x4 Mathf Vector3 Quaternion Transform / GameObject Camera

// 物理引擎相关

// Ray Plane Bounds BoundingSphere

// 基础碰撞检测图形

// line, ray, segment - plane, circle, triangle, rectangle,

// polygon, box(Oriented bounding box(OOB)) / aab(Axis - aligned bounding box(AABB)), sphere

// aab - aab

// box - box, capsule

// plane - plane, sphere, triangle, aab / box

// sphere - sphere, aab / box

// triangle - triangle

// 凸包 : Fixed directions hulls, k - DOP, convex hull 没有任何一个内角是优角

// 凹包 : Concave hull 内角至少有一个优角

// 优角(reflex angle)亦称凹角，180° < θ < 360°。

// 劣角(inferior angle)亦称凸角，0° < θ < 180°

// bullet物理引擎

// Convex Primitives :

// btBoxShape btSphereShape btCapsuleShape btCylinderShape btConeShape btMultiSphereShape

// Compound Shapes :

// btCompoundShape

// Convex Hull Shapes :

// btConvexHullShape

// Concave Triangle Meshes :

// btBvhTriangleMeshShape

// Forward Dynamic : Apply Gravity Predict Transforms

// BroadPhase : Compute AABBs Detect Pairs(区域分割 四叉树 / 八叉树)

进程:不共享内存区

线程:共享内存区域

// NarrowPhase : Compute Contacts(碰撞检测 GJK / EPA算法)

// Forward Dynamic : Solve constrains Integrate Position

// 1--------------------------cpu(和内存(RAM),硬盘(HDD,SSD)交互):

堆区(heap)

向上增长

栈区(stack)

向下增长

代码区:

定义

函数堆栈(function)

常量区(const)

local const, static const

heap const

成员变量(member variable)

(不需要初始化)

局部变量(local variable)

(需要初始化)

//文件名/其他:FileTest,file\_test

//包名：com.deamerstudio.xxxtest

//类名/结构体,接口名,枚举：CSharpTest/StructTest,IInterfaceTest,EnumTest(ENUM\_VALUE)

//函数名: DataTest,dataTest

//变量名：memberTest (属性:MemberTest)(字段:memberTest,m\_MemberTest,\_memberTest)

//常量名：CONST\_TEST,k\_CONST\_TEST

//静态名: Instance, s\_Instance

//// (c++ 值/指针传递(pass by value/pointer),引用传递(pass by reference))编译型语言(compiled language);

// (c#值/指针传递,引用传递)解释型语言(interpreted languages)

// js,lua,phthon 脚本语言(script language)

静态区static extern

2--------------------------cpu和显存交互, gpu(和显存交互) 猜测:dx在此阶段发送drawcall?

Input Assembler(IA) 输入装配阶段 --√

Unity自定义流水数据,写入常量缓冲区: 灯光颜色,灯光位置,环境光颜色,自定义颜色,相机位置, MVP矩阵

Unity固定流水数据,写入顶点,索引缓冲区: 顶点,法线,切线,纹理坐标,

3--------------------------//cpu和显存交互, gpu(和显存交互): // 未知点: 图元(primitives)?

1------------------ Vertex Shader(VS) 顶点着色阶段 --√

已知可以:计算裁剪空间坐标(MVP矩阵),计算模型世界坐标法线,传递uv坐标

2-------------------中间其他阶段----------------------------------

3------------------Rasterizer(RS) 光栅化阶段 --√

把顶点数据转换为片元(fragments) 已知:采样(sampling),反走样 (anti-aliasing)(抗锯齿(anti-jaggies)处理)

4------------------Pixel Shader(PS) 像素着色器阶段 --√

已知可以: 通过 灯光颜色,灯光位置,环境光颜色,自定义颜色,相机位置计算灯光颜色, 通过法线, 视线计算颜色

=> Pixel OwnerShip Test(像素所有者测试)=> Scissor Test (裁剪测试)=>Alpha Test(clip函数) => Stencil Test => Depth Test =>Blending => Dithering(抖动) =>Logic Op

=>GBuffer=>front buffer=>frame buffer //不知道是不是这个阶段进行的

5------------------Output Merger(OM) 输出合并阶段 --√ //不确定干啥的?

//mvp矩阵

//(模型/物体/本地坐标系)local/model/space ->(model matrix(unity\_ObjectToWorld模型变换)), transform的trs矩阵 == >

//未知:css还未确定?

// (世界坐标系)world space(dx左手, opengl右手, unity左手,css左手) ->(view matrix(unity\_MatrixV视点变换)), camera的transform的trs矩阵 == >

// (眼坐标系)view space->(perspective / orthographic projection matrix(glstate\_matrix\_projection投影变换投影变换)), camera的project矩阵 == >

// (归一化坐标系 normalized device coordinates)ndc space -> (齐次裁剪坐标系)homogeneous clip space ->(viewport transform视口变换) == >

// (屏幕坐标系)screen space(dx左上角原点(0,1), opengl左下角原点(-1,1), unity左下角原点(0,1),css(左上角原点(0,1)))

//

// uv坐标: opengl( opengl左下角原点(0,1),dx左上角原点(0,1),unity左下角原点(0,1),)

// 惯性坐标系:不旋转的本地坐标系

// aspect ratio : 宽高比

//

//Point:需要构造(x,y,z,1)(只要w分量!=0就可以表示点)

//Vector:需要构造(x,y,z,0)

//旋转矩阵(矩阵的乘法)(右手坐标系): // 逆矩阵: 旋转矩阵是一个正交矩阵,所以求转置矩阵就行

// 绕x轴矩阵(yz平面): //pitch,gradient

//x′=x | 1,0, 0, 0 | |x|

//y′=ycosθ−zsinθ | 0,cosθ,-sinθ, 0 | \*|y|

//z′=ysinθ+zcosθ | 0,sinθ, cosθ, 0 | |z|

// | 0,0, 0, 1 | |1|

// 绕z轴矩阵(xy平面): //roll,banking

//x′=ysinθ+zcosθ | cosθ,-sinθ, 0,0 | |x|

//y′=ycosθ−zsinθ | sinθ, cosθ,0,0 | \* |y| //未知: 旋转矩阵还需要更多推导?

//z′=z | 0,0, 1, 0| |z|

// | 0,0, 0, 1 | |1|

// 绕y轴矩阵(xz平面): // yaw,turn, heading

//x′=zsinθ+xcosθ | cosθ,0, sinθ, 0 | |x|

//y′=y | 0, 1,0 ,0 | \*|y|

//z′=zcosθ−xsinθ | -sinθ, 0,cosθ, 0 | |z|

// | 0,0, 0, 1 | |1|

//最终的顶点\*旋转矩阵: E(α,π/2,β) = Rz(β) \* Ry(π/2) \* Rx(α) = Ryz(α) \*Rzx(π/2)\* Rxy(β) = Ry(π/2) \* Rx(α-β)

//// quaternion顶点\*轴角: 3d空间中任意一个v沿着单位向量u旋转θ角度之后的v'为: v' = cos(θ)\*v + (1-cos(θ))\*(u.v).u + sin(θ)\*(uxv)

///

/// 赋值给transform的rotation(Quat=Quat),

/// 对已有Vec3变换(Quat\*Vec3),

/// 运算Quat\*Quat

///

//平移矩阵(矩阵的加法): // 逆矩阵(矩阵的减法):

//x' | 1,0,0,dx| |x| |x+dx| | 0,0,0,-dx| |x| |x-dx|

//y' | 0,1,0,dy|\*|y| = |y+dy| | 0,0,0,-dy|\*|y| = |y-dy|

//z' | 0,0,1,dz| |z| |z+dz| | 0,0,0,-dz| |z| |z-dz|

// | 0,0,0,1 | |1| |1| | 0,0,0,1 | |1| |1|

//缩放矩阵(矩阵的乘法) //逆矩阵:

//x' | sx,0,0,0| |x| | 1/sx,0,0,0| |x|

//y' | 0,sy,0,0|\*|y| | 0,1/sy,0,0|\*|y|

//z' | 0,0,sz,0| |z| | 0,0,1/sz,0| |z|

// | 0,0,0,1 | |1| | 0,0,0, 1 | |1|

//正交投影矩阵:

//透视投影矩阵: