GO：GameObject

## Component基本组件类型

|  |
| --- |
| **GameObject**  PROPERTIES  transform  scene  tag  layer  bool activeSelf //该GO自身是否激活，若parent有任何不激活，场景中便不激活  bool activeHierarchy //检查该GO在场景中是否激活  PUBLIC FUNCTIONS  Component AddComponent<T>() //添加类型T的component并返回该component  Component GetComponent<T>() //返回T类型的component，T也可以是Script  Component GetComponentInChildren<T>() //上面函数的变种  SetActive(bool X) //设置该GO的激活/不激活状态  STATIC FUNCTION  Destroy(Object X, float t = 0.0F) //延迟t秒后删除物体X，如果为0则永不删除  DontDestroyOnLoad(Object X) //在加载新场景时不毁灭物体X，一般用于Awake()  GameObject Find(string name) //返回名称为name的GO  GameObject [] FindGameObjectWithTag(string tag) //返回所有标签为tag的GO  Object Instantiate(Object X, Vector3 Y, Quaternion Z)  Object Instantiate(Object X)  Object Instantiate(Object X, Transform parent)  //实体化X（一般为prefab），位置位向由YZ决定 |
| **Transform**  VARIABLES  Vector3 position //设置物体的世界坐标位置  Vector3 localPosition //设置相对parent的position  Vector3 localEulerAngles  Vector3 forward //物体LCR的z轴（蓝色）在WCR的单位向量  Vector3 localScale //设置相对parent的scale  PUBLIC FUNCTION  Rotate(Vector3 X, Space relativeTo = Space.Self) //X代表欧拉角组成的向量，旋转顺序是zxy，单位是角度，Space.Self指的是相对LCR  Rotate(float x, float y, float z, Space relativeTo = Space.Self)  Rotate(Vector3 axis, float angle, Space relativeTo = Space.Self) //设定绕轴旋转  Translate(Vector3 X, Space relativeTo = Space.Self) //如果是相对世界坐标，用Space.World  Translate(float x, float y, float z, Space relativeTo = Space.Self)  LookAt(Transform X, Vector3 Y = Vector3.up) //控制当前物体朝向X的原点，Y定位大致的up方向，道理类似OpenGL中相关知识  Transform Find(string name) //找寻名称为name的child，返回其transform，注意与GameObject.Find的区别  Transform GetChild(int index) //返回index指定的child的transform  **Space**  *enumeration*  World  Self |

## Properties组件属性数据类型

|  |
| --- |
| **AnimationCurve**  PUBLIC METHODS  Evaluate(float time) //返回time处的值 |
| **Color**  CONSTRUCTOR  Color(float r, float g, float b, float a)  STATIC VARIABLES  black, blue, clear, cyan, gray, green, grey, magenta, red, white, yellow  //Color.X代表常见的几种颜色的color常量  VARIABLES  float r,g,b,a //颜色分量 |

# Graphics

## Render pipelines

Scriptable Render Pipeline

SRP是URP和HDRP的支持基础。

·RenderPipelineAsset

基类，用以继承创建pipeline的配置文件，override CreatePipeline()以创建渲染管道的实例

·RenderPipeline

基类，用以创建pipeline instance，override Render()以执行自定义渲染过程

·GraphicsSettings.renderPipelineAsset 可更改或获取当前的渲染管道配置文件

## Shaders

·Unity中的shader由一种叫ShaderLab语言包装而成，核心的shader使用CG/HLSL语言编写。大致结构如下：

Shader "MyShader" {

Properties

{

\_MyTexture ("My Texture", 2D) = "white" { }

// place other properties here, such as colors or vectors.

}

SubShader

{

// place the shader code here for your:

// - surface shader,

// - vertex and program shader, or

// - fixed function shader

Pass

{

// ... the usual pass state setup ...

CGPROGRAM

// compilation directives for this snippet, e.g.:

#pragma vertex vert

#pragma fragment frag

// the Cg/HLSL code itself

ENDCG

// ... the rest of pass setup ...

}

}

SubShader

{

// a simpler version of the subshader above goes here.

// this version is for supporting older graphics cards.

}

}

Surface Shaders

◆Surface shaders实际上是unity为了便于书写带有光照计算的shader而创造的概念，书写一个surface shader会被编译器自动生成对应的各种底层vertex/pixel shader。Surface shader只能放在Pass块之外，因为它会被编译成多个Pass。且使用如下语句声明身份（该语句须至于CGPROGRAM块内）：

#pragma surface surfaceFunction lightModel [optionalparams]

·surfaceFunction - which Cg function has surface shader code. The function should have the form of   
void surf (Input IN, inout SurfaceOutput o),   
where Input is a structure you have defined. Input should contain any texture coordinates and extra automatic variables needed by surface function.

·lightModel - lighting model to use.   
- “Standard” lighting model uses SurfaceOutputStandard output struct, and matches the Standard (metallic workflow) shader in Unity.  
- “StandardSpecular” lighting model uses SurfaceOutputStandardSpecular output struct, and matches the Standard (specular setup) shader in Unity.  
- “Lambert” and “BlinnPhong” lighting models are not physically based (coming from Unity 4.x), but the shaders using them can be faster to render on low-end hardware.

◆Surface shader的输出结构：

struct SurfaceOutput

{

fixed3 Albedo; // diffuse color

fixed3 Normal; // tangent space normal, if written

fixed3 Emission;

half Specular; // specular power in 0..1 range

fixed Gloss; // specular intensity

fixed Alpha; // alpha for transparencies

};

struct SurfaceOutputStandard

{

fixed3 Albedo; // base (diffuse or specular) color

fixed3 Normal; // tangent space normal, if written

half3 Emission;

half Metallic; // 0=non-metal, 1=metal

half Smoothness; // 0=rough, 1=smooth

half Occlusion; // occlusion (default 1)

fixed Alpha; // alpha for transparencies

};

struct SurfaceOutputStandardSpecular

{

fixed3 Albedo; // diffuse color

fixed3 Specular; // specular color

fixed3 Normal; // tangent space normal, if written

half3 Emission;

half Smoothness; // 0=rough, 1=smooth

half Occlusion; // occlusion (default 1)

fixed Alpha; // alpha for transparencies

};

◆Surface shader的输入结构：

纹理坐标必须以uv开头，或uv2以使用第二组坐标，其余可用输入：

·float3 viewDir - contains view direction, for computing Parallax effects, rim lighting etc.

·float4 with COLOR semantic - contains interpolated per-vertex color.

·float4 screenPos - contains screen space position for reflection or screenspace effects. Note that this is not suitable for GrabPass; you need to compute custom UV yourself using ComputeGrabScreenPos function.

·float3 worldPos - contains world space position.

·float3 worldRefl - contains world reflection vector if surface shader does not write to o.Normal. See Reflect-Diffuse shader for example.

·float3 worldNormal - contains world normal vector if surface shader does not write to o.Normal.

·float3 worldRefl; INTERNAL\_DATA - contains world reflection vector if surface shader writes to o.Normal. To get the reflection vector based on per-pixel normal map, use WorldReflectionVector (IN, o.Normal). See Reflect-Bumped shader for example.

·float3 worldNormal; INTERNAL\_DATA - contains world normal vector if surface shader writes to o.Normal. To get the normal vector based on per-pixel normal map, use WorldNormalVector (IN, o.Normal).

SURFACE SHADER AND RENDERING PATH

Vertex and Fragment Shaders

◆顶点和片段shader写在CGPROGRAM…ENDCG或HLSLPROGRAM…ENDHLSL之间。一般来说二者等效，但前者更推荐。

◆compilation directives常用编译指令如下（指定顶点和片段着色器的指令是必须的，还有更多可选的），一般写在最前面：

| **Statement** | **Function** |
| --- | --- |
| #pragma vertex name | Compile function name as the **vertex shader**. |
| #pragma fragment name | Compile function name as the **fragment shader** . |
| #pragma geometry name | Compile function name as DX10 geometry shader. This option automatically turns on #pragma target 4.0 as described in the table below. |
| #pragma hull name | Compile function name as DX11 hull shader. This option automatically turns on #pragma target 5.0, as described in the table below. |
| #pragma domain name | Compile function name as DX11 domain shader. This option automatically turns on #pragma target 5.0, as described in the table below. |

# Animation

·最新的Unity动画系统也称Mecanim。主要基于Animator（组件）-Animator Controller（状态机资产）-动画片段clips结构。

·Animation概念不仅仅限于动画，对于任何随时间变化的数值属性均可应用此技术。

Avatar：Unity中对于人形动画进行的广泛抽象，可用于移植人形动画至不同模型

## Animation Window

Preview Mode：预览动画

Record Mode（auto-key mode）：记录动画

Dopesheet/Curves：两种显示动画数据的方式

## Animation Event

Unity允许在clip中添加event，即播放到该位置调用某个方法。该方法的脚本位置应在与Animator组件相同的GO中，否则Unity Inspector中无法设定。

## State Machine

Solo：用于测试状态机运行，只允许当前transition可运行

Mute：同上，关闭当前transition

Write Defaults：控制状态是否将被操作数据恢复原default值

·Unity的animation状态机不仅仅可用于装载动画，还可以当做通用脚本（行为）状态机来使用。在某个状态的inspector中Add Behaviour即可添加脚本。注意脚本需要继承StateMachineBehaviour。详见后面Animation Scripting。

## Sub-State Machine和Animation Layers

Sub-State Machine：其实是多个相关的子状态组成一组，成为上一级的一个状态，是一种hierarchical state machine。

Animation Layers：是将不同骨骼部分的动画状态机独立设置并融合在一起的工具

Animation Layer Syncing：使某个layer重用另一个layer的状态结构，但允许使用不同的动画，比如正常状态和负伤状态。

## Animation Clip

Root Motion：root指代动画骨骼相对位置的基本原点，有些移动型动画的root会表达角色的移动信息，这就是root motion。

·在导入和设置动画片段时可以设置如何建立root motion和角色模型的关系。

## Animation Scripting

|  |
| --- |
| **Animator**  STATIC METHODS  int StringToHash(string name)  //将一个字符串转化为hash值，主要用于将名称转化为id  PUBLIC METHODS  void SetBool(string X, bool value) //将名为X的变量设为value，也可用int类型的id  void SetFloat(…)  void SetInteger(…)  void SetTrigger(…)  void ResetTrigger(…)  AnimatorStateInfo GetCurrentAnimatorStateInfo(int layerIndex)  //返回当前的状态信息类型变量  AnimatorStateInfo GetNextAnimatorStateInfo(int layerIndex)  bool IsInTransition(int layerIndex) //该layer下的状态机是否处于过渡状态  **AnimatorStateInfo**  PROPERTIES  int tagHash //该状态的tag，是字符串hash值  PUBLIC METHODS  bool IsName(string X) //返回是否当前状态名为X |
| **StateMachineBehaviour**  PUBLIC METHODS  OnStateMachineEnter(Animator animator, int stateMachinePathHash)  //涉及多线程，暂放  OnStateMachineExit(…)  MESSAGES  OnStateEnter(Animator animator, AnimatorStateInfo animatorStateInfo, int layerIndex)  OnStateExit(…)  OnStateUpdate(…) //在除了首尾的每一帧Update时调用  OnStateMove(…) //在MonoBehaviour.OnAnimatorMove()之后调用  OnStateIK(…)  **SharedBetweenAnimatorsAttribute**  用于指明本statemachinebehaviour适用于所有animator |

数学工具

|  |
| --- |
| **Mathf** //unity中的数学函数类  STATIC VARIABLES  float Infinity //只读，正无穷  STATIC FUNCTIONS  float Sin(float radian) //返回弧度角正弦 |
| **Vector3**  STATIC VARIABLES  back, down, forward, left, right, up //即常量(0,0,-1),(0,-1,0)…  one, zero //(1,1,1),(0,0,0)  VARIABLES  x,y,z  float magnitude //向量长度  Vector3 normalized //正交化/单位化后的向量  STATIC METHODS  float Distance(Vector3 a, Vector3 b) //返回ab距离  float Dot(Vector3 a, Vector3 b) //返回点积 |
| **Quaternion**  STATIC VARIABLES  Quaternionidentity //identity rotation  STATIC METHODS  Quaternion LookRotation(Vector3 forward, Vector3 upwards = Vector3.up)  Quaternion Lerp(  //创建一个以forward为Z轴的旋转 |

其他辅助工具类型

|  |
| --- |
| **Debug**  STATIC FUNCTIONS  DrawRay(Vector3 start, Vector3 dir, Color color = Color.white, float duration = 0.0f,  bool depthTest = true) //从start沿dir绘制一条线（WCR），并且设置颜色持续时间  和深度检测，持续时间为0表示只渲染1帧  Log(string X) //向console输出字符串X |
| **Time**  STATIC VARIABLES  deltaTime //储存当前帧与上一帧之间的时间差  fixedDeltaTime  //储存当前帧与上一FixedUpdate帧之间的时间差，但document中建议无论在Update还是FixedUpdate中都用deltaTime获取时间差 |