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

·ScriptableRenderContext

渲染管道的API交互对象类，可以直接用SRC成员进行API操作，也可以使用CommandBuffer。最后均需用Submit()提交执行。

·CommandBuffer

创建该对象，将所需操作依次通过成员函数调用进行排程schedule，最后使用ScriptableRenderContext.ExecuteCommandBuffer()将CommandBuffer对象传递给SRC对象。再使用SRC成员Submit()提交执行，和CB成员Release()释放CB对象。也可以使用Graphics.ExecuteCommandBuffer()立即提交执行CB对象，避开SRC传递和提交过程。

1）SRP Batcher

一个可以对使用少数相同shader的大量materials/objects进行批处理以提高渲染速度的技术。

For a Shader to be compatible with the SRP Batcher:

·You must declare all built-in engine properties in a single CBUFFER named “UnityPerDraw”. For example, unity\_ObjectToWorld, or unity\_SHAr.

·You must declare all Material properties in a single CBUFFER named UnityPerMaterial.

## Meshes, Materials, Shaders and Textures

### Writing Shaders

◆Shader Reference - Writing vertex and fragment shaders - Shader semantics

着色器语义是用以告诉编译器变量使用目的的关键字。

·Vertex Input

float3/4 POSITION 顶点物体空间坐标

float3 NORMAL

float2/3/4 TEXCOORDN 高精度输入数据，一般用于纹理坐标或其他信息

float4 TANGENT

float4 COLOR 每顶点颜色

uint SV\_VertexID 标明顶点数的输入数据

·Vertex Output，也称interpolators或varyings

float3 SV\_POSITION 裁剪空间clip space顶点坐标

TEXCOORDN 高精度输出数据

COLORN 低精度输出数据

·Fragment Input

UNITY\_VPOS\_TYPE VPOS 屏幕空间像素坐标 #pragma target 3.0

类型VFACE 当前表面是否为正面，正数为正面，负数为反面

·Fragment Output

类型SV\_TargetN 片段着色器输出目标，N为0时可省略

类型SV\_Depth 片段着色器深度输出，一般不在此阶段更改深度，特殊情况下才使用

◆Shader Reference - Writing vertex and fragment shaders - Accessing shader properties in Cg/HLSL

·MaterialPropertyBlock（ShaderLab）数据类型和HLSL中数据类型对应关系

Color：float4，half4或fixed4

Float：float，half或fixed

2D：sampler2D

CUBE：samplerCUBE

·3种方式为shader传递参数（优先级依次降低）

使用MaterialPropertyBlock传递数据，是为per-renderer或per-instance。可以对相同的材料在不同的renderer上执行不同的override。

使用Material对象传递数据，但对material修改将会改变所有使用该material的物体。

使用全局global shader properties，即对所有shaders均有效的数据，使用Shader类静态函数设置。

·特殊的纹理属性

由Unity自动命名并生成的针对纹理的一些附加信息数据。

Texture tiling & offset：float4 {TextureName}\_ST

x contains X tiling value

y contains Y tiling value

z contains X offset value

w contains Y offset value

Texture size：float4 {TextureName}\_TexelSize

x contains 1.0/width

y contains 1.0/height

z contains width

w contains height

Texture HDR parameters：float4 {TextureName}\_HDR

◆Shader Reference - Writing vertex and fragment shaders - Providing vertex data to vertex programs

详见shader semantics

◆Shader Reference - Writing vertex and fragment shaders - Built-in shader include files

·HLSLSupport.cginc - (automatically included) Helper macros and definitions for cross-platform shader compilation.

·UnityShaderVariables.cginc - (automatically included) Commonly used global variables.

·UnityCG.cginc - commonly used helper functions.

·AutoLight.cginc - lighting & shadowing functionality, e.g. surface shaders use this file internally.

·Lighting.cginc - standard surface shader lighting models; automatically included when you’re writing surface shaders.

·TerrainEngine.cginc - helper functions for Terrain & Vegetation shaders.

注意前两个include文件仅针对CGPROGRAM，对HLSLPROGRAM的shader并不自动添加

◆Shader Reference - Writing vertex and fragment shaders - Predefined Shader preprocessor macros

预编译宏有的可以存储平台相关信息，有的可以单纯通过其是否定义来判断程序状态或信息，有的则是类似函数的功能。

·Target platform

| Macro: | Target platform: |
| --- | --- |
| SHADER\_API\_D3D11 | Direct3D 11 |
| SHADER\_API\_GLCORE | Desktop OpenGL “core” (GL 3/4) |
| SHADER\_API\_GLES | OpenGL ES 2.0 |
| SHADER\_API\_GLES3 | OpenGL ES 3.0/3.1 |
| SHADER\_API\_METAL | iOS/Mac Metal |
| SHADER\_API\_VULKAN | Vulkan |
| SHADER\_API\_D3D11\_9X | Direct3D 11 “feature level 9.x” target for Universal Windows Platform |
| SHADER\_API\_PS4 | PlayStation 4. SHADER\_API\_PSSL is also defined. |
| SHADER\_API\_XBOXONE | Xbox One |
| SHADER\_API\_MOBILE | All general mobile platforms (GLES, GLES3, METAL). |
| SHADER\_TARGET\_GLSL | Target shading language is GLSL (always true for OpenGL/GLES platforms). |

·Shader target model

SHADER\_TARGET is defined to a numeric value that matches the Shader target compilation model (that is, matching #pragma target directive). For example, SHADER\_TARGET is 30 when compiling into Shader model 3.0.

·Unity version

UNITY\_VERSION contains the numeric value of the Unity version. For example, UNITY\_VERSION is 501 for Unity 5.0.1.

·Shader stage being compiled

SHADER\_STAGE\_VERTEX,

SHADER\_STAGE\_FRAGMENT,

SHADER\_STAGE\_DOMAIN,

SHADER\_STAGE\_HULL,

SHADER\_STAGE\_GEOMETRY,

SHADER\_STAGE\_COMPUTE

are defined when compiling each Shader stage.

·Platform difference helpers

平台差异化辅助宏，慎用

| Macro: | Use: |
| --- | --- |
| UNITY\_BRANCH | Add this before conditional statements to tell the compiler that this should be compiled into an actual branch. Expands to [branch] when on HLSL platforms. |
| UNITY\_FLATTEN | Add this before conditional statements to tell the compiler that this should be flattened to avoid an actual branch instruction. Expands to [flatten] when on HLSL platforms. |
| UNITY\_NO\_SCREENSPACE\_  SHADOWS | Defined on platforms that do not use cascaded screenspace shadowmaps (mobile platforms). |
| UNITY\_NO\_LINEAR\_  COLORSPACE | Defined on platforms that do not support Linear color space (mobile platforms). |
| UNITY\_NO\_RGBM | Defined on platforms where RGBM compression  for lightmaps  is not used (mobile platforms). |
| UNITY\_NO\_DXT5nm | Defined on platforms that do not use DXT5nm normal-map compression (mobile platforms). |
| UNITY\_FRAMEBUFFER\_  FETCH\_AVAILABLE | Defined on platforms where “framebuffer color fetch” functionality can be available (generally iOS platforms - OpenGL ES 2.0, 3.0 and Metal). |
| UNITY\_USE\_RGBA\_FOR\_  POINT\_SHADOWS | Defined on platforms where point light shadowmaps use RGBA Textures with encoded depth (other platforms use single-channel floating point Textures). |
| UNITY\_ATTEN\_CHANNEL | Defines which channel of light attenuation Texture contains the data; used in per-pixel lighting code. Defined to either ‘r’ or ‘a’. |
| UNITY\_HALF\_TEXEL\_  OFFSET | Defined on platforms that need a half-texel offset adjustment in mapping texels to pixels (e.g. Direct3D 9). |
| UNITY\_UV\_STARTS\_AT\_  TOP | Always defined with value of 1 or 0. A value of 1 is on platforms where Texture V coordinate is 0 at the “top” of the Texture. Direct3D-like platforms use value of 1; OpenGL-like platforms use value of 0. |
| UNITY\_MIGHT\_NOT\_HAVE\_  DEPTH\_Texture | Defined if a platform might emulate shadow maps or depth Textures by manually rendering  depth into a Texture. |
| UNITY\_PROJ\_COORD(a) | Given a 4-component vector, this returns a Texture coordinate suitable for projected Texture reads. On most platforms this returns the given value directly. |
| UNITY\_NEAR\_CLIP\_VALUE | Defined to the value of near clipping plane . Direct3D-like platforms use 0.0 while OpenGL-like platforms use –1.0. |
| UNITY\_VPOS\_TYPE | Defines the data type required for pixel position input (VPOS): float2 on D3D9, float4 elsewhere. |
| UNITY\_CAN\_COMPILE\_  TESSELLATION | Defined when the Shader compiler “understands” the tessellation Shader HLSL syntax (currently only D3D11). |
| UNITY\_INITIALIZE\_  OUTPUT(type,name) | Initializes the variable *name* of given *type* to zero. |
| UNITY\_COMPILER\_HLSL,  UNITY\_COMPILER\_HLSL2GLSL,  UNITY\_COMPILER\_CG | Indicates which Shader compiler is being used to compile Shaders - respectively: Microsoft’s HLSL, HLSL to GLSL translator, and NVIDIA’s Cg. See documentation on Shading Languages for more details. Use this if you run into very specific Shader syntax handling differences between the compilers, and want to write different code for each compiler. |
| UNITY\_REVERSED\_Z | defined on plaftorms using reverse Z buffer. Stored Z values are in the range 1..0 instead of 0..1. |

·Shadow mapping marcos

Declaring and sampling shadow maps can be very different depending on the platform. Unity has several macros to help with this:

| Macro: | Use: |
| --- | --- |
| UNITY\_DECLARE\_SHADOWMAP(tex) | Declares a shadowmap Texture variable with name “tex”. |
| UNITY\_SAMPLE\_SHADOW(tex,uv) | Samples shadowmap Texture “tex” at given “uv” coordinate (XY components are Texture location, Z component is depth to compare with). Returns single float value with the shadow term in 0..1 range. |
| UNITY\_SAMPLE\_SHADOW\_PROJ(tex,uv) | Similar to above, but does a projective shadowmap read. “uv” is a float4, all other components are divided by .w for doing the lookup. |

NOTE: Not all graphics cards support shadowmaps. Use SystemInfo.SupportsRenderTextureFormat to check for support.

·Constant buffer macros

Direct3D 11 groups all Shader variables into “constant buffers”. Most of Unity’s built-in variables are already grouped, but for variables in your own Shaders it might be more optimal to put them into separate constant buffers depending on expected frequency of updates.Use CBUFFER\_START(name) and CBUFFER\_END macros for that.

·Texture/Sampler declaration macros

Usually you would use texture2D in Shader code to declare a Texture and Sampler pair. However on some platforms (such as DX11), Textures and Samplers are separate objects, and maximum possible Sampler count is quite limited. Unity has some macros to declare Textures without Samplers, and to sample a Texture using a Sampler from another Texture.

| Macro: | Use: |
| --- | --- |
| UNITY\_DECLARE\_TEX2D(name) | Declares a Texture and Sampler pair. |
| UNITY\_DECLARE\_TEX2D\_NOSAMPLER(name) | Declares a Texture without a Sampler. |
| UNITY\_DECLARE\_TEX2DARRAY(name) | Declares a Texture array Sampler variable. |
| UNITY\_SAMPLE\_TEX2D(name, uv) | Sample from a Texture and Sampler pair, using given Texture coordinate. |
| UNITY\_SAMPLE\_TEX2D\_SAMPLER(name, samplername, uv) | Sample from Texture (name), using a Sampler from another Texture (samplername). |
| UNITY\_SAMPLE\_TEX2DARRAY(name, uv) | Sample from a Texture array with a float3 UV; the z component of the coordinate is array element index. |
| UNITY\_SAMPLE\_TEX2DARRAY\_LOD(name, uv, lod) | Sample from a Texture array with an explicit mipmap level. |

·Surface shader pass indicators

在编译surface shader至多个pass的过程中定义

| Macro: | Use: |
| --- | --- |
| UNITY\_PASS\_FORWARDBASE | Forward rendering  base pass (main directional light, lightmaps, SH). |
| UNITY\_PASS\_FORWARDADD | Forward rendering additive pass (one light per pass). |
| UNITY\_PASS\_DEFERRED | Deferred shading  pass (renders g buffer). |
| UNITY\_PASS\_SHADOWCASTER | Shadow caster and depth Texture rendering pass. |
| UNITY\_PASS\_PREPASSBASE | Legacy deferred lighting base pass (renders normals and specular exponent). |
| UNITY\_PASS\_PREPASSFINAL | Legacy deferred lighting final pass (applies lighting and Textures). |

·Disable Auto-Upgrade

UNITY\_SHADER\_NO\_UPGRADE allows you to disable Unity from automatically upgrading or modifying your shader file.

◆Shader Reference - Writing vertex and fragment shaders - Built-in shader helper functions

Vertex transformation functions in UnityCG.cginc

| Function: | Description: |
| --- | --- |
| float4 UnityObjectToClipPos(float3 pos) | Transforms a point from object space to the camera ’s clip space in homogeneous coordinates. This is the equivalent of mul(UNITY\_MATRIX\_MVP, float4(pos, 1.0)), and should be used in its place. |
| float3 UnityObjectToViewPos(float3 pos) | Transforms a point from object space to view space. This is the equivalent of mul(UNITY\_MATRIX\_MV, float4(pos, 1.0)).xyz, and should be used in its place. |

Generic helper functions in UnityCG.cginc

| Function: | Description: |
| --- | --- |
| float3 WorldSpaceViewDir (float4 v) | Returns world space direction (not normalized) from given object space vertex position towards the camera. |
| float3 ObjSpaceViewDir (float4 v) | Returns object space direction (not normalized) from given object space vertex position towards the camera. |
| float2 ParallaxOffset (half h, half height, half3 viewDir) | calculates UV offset for parallax normal mapping. |
| fixed Luminance (fixed3 c) | Converts color to luminance (grayscale). |
| fixed3 DecodeLightmap (fixed4 color) | Decodes color from Unity lightmap  (RGBM or dLDR depending on platform). |
| float4 EncodeFloatRGBA (float v) | Encodes [0..1) range float into RGBA color, for storage in low precision render target. |
| float DecodeFloatRGBA (float4 enc) | Decodes RGBA color into a float. |
| float2 EncodeFloatRG (float v) | Encodes [0..1) range float into a float2. |
| float DecodeFloatRG (float2 enc) | Decodes a previously-encoded RG float. |
| float2 EncodeViewNormalStereo (float3 n) | Encodes view space normal into two numbers in 0..1 range. |
| float3 DecodeViewNormalStereo (float4 enc4) | Decodes view space normal from enc4.xy. |

Forward rendering helper functions in UnityCG.cginc

These functions are only useful when using forward rendering (ForwardBase or ForwardAdd pass types).

| Function: | Description: |
| --- | --- |
| float3 WorldSpaceLightDir (float4 v) | Computes world space direction (not normalized) to light, given object space vertex position. |
| float3 ObjSpaceLightDir (float4 v) | Computes object space direction (not normalized) to light, given object space vertex position. |
| float3 Shade4PointLights (...) | Computes illumination from four point lights, with light data tightly packed into vectors. Forward rendering  uses this to compute per-vertex lighting. |

Screen-space helper functions in UnityCG.cginc

The following functions are helpers to compute coordinates used for sampling screen-space textures. They return float4 where the final coordinate to sample texture with can be computed via perspective division (for example xy/w).

The functions also take care of platform differences in render texture coordinates.

| Function: | Description: |
| --- | --- |
| float4 ComputeScreenPos (float4 clipPos) | Computes texture coordinate for doing a screenspace-mapped texture sample. Input is clip space position. |
| float4 ComputeGrabScreenPos (float4 clipPos) | Computes texture coordinate for sampling a GrabPass texure. Input is clip space position. |

Vertex-lit helper functions in UnityCG.cginc

These functions are only useful when using per-vertex lit shaders (“Vertex” pass type).

| Function: | Description: |
| --- | --- |
| float3 ShadeVertexLights (float4 vertex, float3 normal) | Computes illumination from four per-vertex lights and ambient, given object space position & normal. |

◆Shader Reference - Writing vertex and fragment shaders - Built-in shader variables

·Transformations

All these matrices are float4x4 type, and are column major.

|  |  |
| --- | --- |
| Name | Value |
| UNITY\_MATRIX\_MVP | Current model \* view \* projection matrix. |
| UNITY\_MATRIX\_MV | Current model \* view matrix. |
| UNITY\_MATRIX\_V | Current view matrix. |
| UNITY\_MATRIX\_P | Current projection matrix. |
| UNITY\_MATRIX\_VP | Current view \* projection matrix. |
| UNITY\_MATRIX\_T\_MV | Transpose of model \* view matrix. |
| UNITY\_MATRIX\_IT\_MV | Inverse transpose of model \* view matrix. |
| unity\_ObjectToWorld | Current model matrix. |
| unity\_WorldToObject | Inverse of current world matrix. |

·Camera and screen

These variables will correspond to the Camera that is rendering. For example during shadowmap rendering, they will still refer to the Camera component values, and not the “virtual camera” that is used for the shadowmap projection.

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| \_WorldSpaceCameraPos | float3 | World space position of the camera. |
| \_ProjectionParams | float4 | x is 1.0 (or –1.0 if currently rendering with a flipped projection matrix), y is the camera’s near plane, z is the camera’s far plane and w is 1/FarPlane. |
| \_ScreenParams | float4 | x is the width of the camera’s target texture in pixels , y is the height of the camera’s target texture in pixels, z is 1.0 + 1.0/width and w is 1.0 + 1.0/height. |
| \_ZBufferParams | float4 | Used to linearize Z buffer values. x is (1-far/near), y is (far/near), z is (x/far) and w is (y/far). |
| unity\_OrthoParams | float4 | x is orthographic camera’s width, y is orthographic camera’s height, z is unused and w is 1.0 when camera is orthographic, 0.0 when perspective. |
| unity\_CameraProjection | float4x4 | Camera’s projection matrix. |
| unity\_CameraInvProjection | float4x4 | Inverse of camera’s projection matrix. |
| unity\_CameraWorldClipPlanes[6] | float4 | Camera frustum plane world space equations, in this order: left, right, bottom, top, near, far. |

·Time

Time is measured in seconds, and is scaled by the Time multiplier in your Project’s Time settings. There is no built-in variable that provides access to unscaled time.

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| \_Time | float4 | Time since level load (t/20, t, t\*2, t\*3), use to animate things inside the shaders. |
| \_SinTime | float4 | Sine of time: (t/8, t/4, t/2, t). |
| \_CosTime | float4 | Cosine of time: (t/8, t/4, t/2, t). |
| unity\_DeltaTime | float4 | Delta time: (dt, 1/dt, smoothDt, 1/smoothDt). |

·Lighting

Light parameters are passed to shaders in different ways depending on which Rendering Path  
 is used, and which LightMode Pass Tag is used in the shader.

Forward rendering (ForwardBase and ForwardAdd pass types):

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| \_LightColor0 *(declared in UnityLightingCommon.cginc)* | fixed4 | Light color. |
| \_WorldSpaceLightPos0 | float4 | Directional lights: (world space direction, 0). Other lights: (world space position, 1). |
| unity\_WorldToLight *(declared in AutoLight.cginc)* | float4x4 | World-to-light matrix. Used to sample cookie & attenuation textures. |
| unity\_4LightPosX0, unity\_4LightPosY0, unity\_4LightPosZ0 | float4 | *(ForwardBase pass only)* world space positions of first four non-important point lights. |
| unity\_4LightAtten0 | float4 | *(ForwardBase pass only)* attenuation factors of first four non-important point lights. |
| unity\_LightColor | half4[4] | *(ForwardBase pass only)* colors of of first four non-important point lights. |
| unity\_WorldToShadow | float4x4[4] | World-to-shadow matrices. One matrix for spot lights, up to four for directional light cascades. |

Deferred shading and deferred lighting, used in the lighting pass shader (all declared in UnityDeferredLibrary.cginc):

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| \_LightColor | float4 | Light color. |
| unity\_WorldToLight | float4x4 | World-to-light matrix. Used to sample cookie & attenuation textures. |
| unity\_WorldToShadow | float4x4[4] | World-to-shadow matrices. One matrix for spot lights, up to four for directional light cascades. |

Spherical harmonics coefficients (used by ambient and light probes) are set up for ForwardBase, PrePassFinal and Deferred pass types. They contain 3rd order SH to be evaluated by world space normal (see ShadeSH9 from UnityCG.cginc). The variables are all half4 type, unity\_SHAr and similar names.

Vertex-lit rendering (Vertex pass type):

Up to 8 lights are set up for a Vertex pass type; always sorted starting from the brightest one. So if you want to render objects affected by two lights at once, you can just take first two entries in the arrays. If there are less lights affecting the object than 8, the rest will have their color set to black.

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| unity\_LightColor | half4[8] | Light colors. |
| unity\_LightPosition | float4[8] | View-space light positions. (-direction,0) for directional lights; (position,1) for point/spot lights. |
| unity\_LightAtten | half4[8] | Light attenuation factors. *x* is cos(spotAngle/2) or –1 for non-spot lights; *y* is 1/cos(spotAngle/4) or 1 for non-spot lights; *z* is quadratic attenuation; *w* is squared light range. |
| unity\_SpotDirection | float4[8] | View-space spot light positions; (0,0,1,0) for non-spot lights. |

·Lightmaps

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| unity\_Lightmap | Texture2D | Contains lightmap  information. |
| unity\_LightmapST | float4[8] | Scales and translates the UV information to the correct range to sample the lightmap texture. |

·Fog and Ambient

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| unity\_AmbientSky | fixed4 | Sky ambient lighting color in gradient ambient lighting case. |
| unity\_AmbientEquator | fixed4 | Equator ambient lighting color in gradient ambient lighting case. |
| unity\_AmbientGround | fixed4 | Ground ambient lighting color in gradient ambient lighting case. |
| UNITY\_LIGHTMODEL\_AMBIENT | fixed4 | Ambient lighting color (sky color in gradient ambient case). Legacy variable. |
| unity\_FogColor | fixed4 | Fog color. |
| unity\_FogParams | float4 | Parameters for fog calculation: (density / sqrt(ln(2)), density / ln(2), –1/(end-start), end/(end-start)). *x* is useful for Exp2 fog mode, *y* for Exp mode, *z* and *w* for Linear mode. |

·Various

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| unity\_LODFade | float4 | Level-of-detail fade when using LODGroup. *x* is fade (0..1), *y* is fade quantized to 16 levels, *z* and *w* unused. |
| \_TextureSampleAdd | float4 | Set automatically by Unity for UI  only based on whether the texture being used is in Alpha8 format (the value is set to (1,1,1,0)) or not (the value is set to (0,0,0,0)). |

◆Shader Reference - Writing vertex and fragment shaders - Shader variants and keywords

可以使用shader keywords控制编译一个shader的不同版本，不同版本的shader定义了不同的keywords。使用以下语句定义keywords：

#pragma multi\_compile

#pragma multi\_compile\_local

#pragma shader\_feature

#pragma shader\_feature\_local

·shader\_feature声明的shader版本若不被调用则不会被unity最终build，multi\_compile无此限制。

·#pragma multi\_compile \_\_ KEYWORD1可以产生两个版本，一个定义KEYWORD1，另一个没有，这样可以省下一个keyword名额。使用shader\_feature可以省略\_\_，即只定义一个keyword以达到相同效果。

·C#代码中操作关键字：

Shader.EnableKeyword: enable a global keyword

Shader.DisableKeyword: disable a global keyword

CommandBuffer.EnableShaderKeyword: use a CommandBuffer to enable a global keyword

CommandBuffer.DisableShaderKeyword: use a CommandBuffer to disable a global keyword

Material.EnableKeyword: enable a local keyword for a regular shader

Material.DisableKeyword: disable a local keyword for a regular shader

ComputeShader.EnableKeyword: enable a local keyword for a compute shader

ComputeShader.DisableKeyword: disable a local keyword for a compute shader

·还有一些built-in multi\_compile指令，相当于生成一些预设的shader变种，并且可以用#pragma skip\_variants来跳过一些变种：

multi\_compile\_fwdbase - compiles all variants needed by PassType.ForwardBase. The variants deal with different lightmap types, and the main Directional Light’s shadows being enabled or disabled.

multi\_compile\_fwdadd - compiles variants for PassType.ForwardAdd. This compiles variants to handle Directional, Spot or Point Light types and their variants with cookie Textures.

multi\_compile\_fwdadd\_fullshadows - same as multi\_compile\_fwdadd, but also includes ability for the lights to have real-time shadows.

multi\_compile\_fog - expands to several variants to handle different fog types (off/linear/exp/exp2).

◆Shader Reference - Writing vertex and fragment shaders - GLSL Shader programs

略

◆Shader Reference - Writing vertex and fragment shaders - Shading language used in Unity

Unity最初使用CG，后改用HLSL。CGPROGRAM会自动添加include文件以兼容。

·HLSL在DX9之前和DX10之后使用不同的纹理采样方式，为了便于跨平台统一，建议使用unity或相关第三方包装的类型和方法来处理纹理，它们会自动根据平台进行转换。

◆Shader Reference - Writing vertex and fragment shaders - Shader compilation Target Levels

◆Shader Reference - Writing vertex and fragment shaders - Shader data types and precision

◆Shader Reference - Writing vertex and fragment shaders - Using sampler states

sampler state实际上就是带有属性配置功能的采样器。有两种方式声明：

·SamplerState Name[Index]{ [state\_name = state\_value;] ... }; //注意Name要遵从sampler + TextureObjectName的惯例，来进行绑定

·使用inline sampler state，是通过unity的命名惯例来配置sampler，只要名称中含有以下单词，不必用下划线分隔

Sampler names recognized as “inline” sampler states (all case insensitive):

——“Point”, “Linear” or “Trilinear” (required) set up texture filtering mode.

——“Clamp”, “Repeat”, “Mirror” or “MirrorOnce” (required) set up texture wrap mode.

Wrap modes can be specified per-axis (UVW), e.g. “ClampU\_RepeatV”.

——“Compare” (optional) set up sampler for depth comparison; use with HLSL SamplerComparisonState type and SampleCmp / SampleCmpLevelZero functions.

◆Shader Reference - ShaderLab Syntax

## 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 |