次世代手游的探讨

Speaker Kevin BY
Art Director & iDreamsky

游戏开发者大会•中国
GAME DEVELOPERS CONFERENCE CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
SHANGHAI, CHINA · OCTOBER 25-27, 2015

- 第一部分 次世代手游的探索和制作

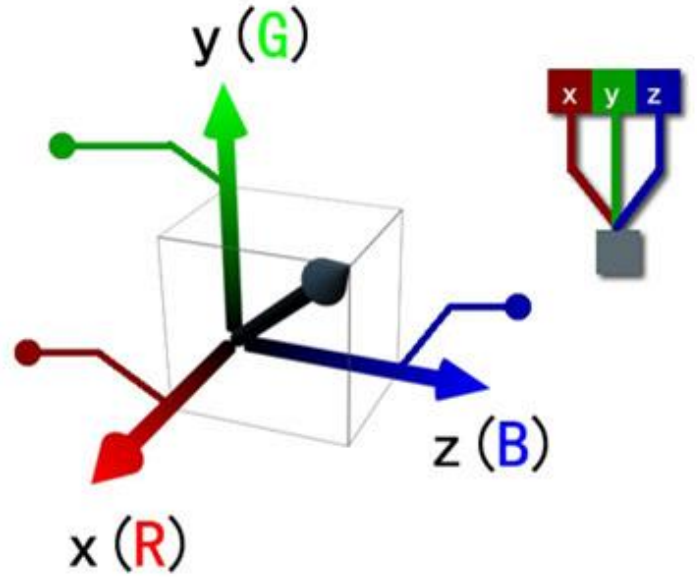
  - 1.Shader
  - 2.Mesh
  - 3.Animation
  - 4.Lightmap

1.Shader

- 1.1Normal Map（法线贴图）
- 1.2Specular Mapping(高光贴图)
- 1.3Cube Map（环境反射贴图）
- 1.4自研Shader

•1.1Normal Map（法线贴图）

法线贴图就是记录了一个需要进行光影变换的贴图上的各个点的凹凸情况的贴图，显示芯片根据这个贴图的内容，来实时的生成新的有过光影变化的贴图，从而实现立体效果。



借色彩之值存法线之向，巧妙的存储

学过初中物理的朋友一定还都记得，表示光线射向平面的角度时通常使用光线和该点法线角度来表示。这也就意味着，如果我们将一个贴图上所有点的法线记录起来的话，就不难再利用这些信息实现后期的假的凹凸效果了。

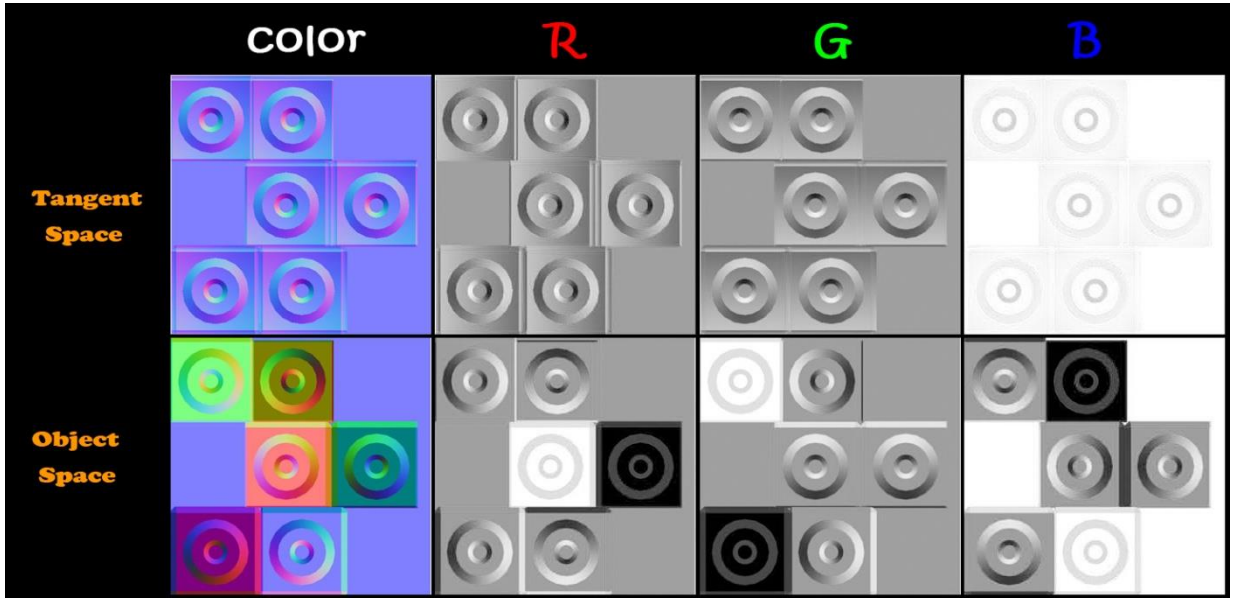
记录这些法线的载体就被我们称为法线贴图。为什么称之为贴图呢？我们知道，一条法线是一个三维向量，一个三维向量由X、Y、Z等3个分量组成，于是人们想出了一个聪明的方法，就是以这3个分量当作红绿蓝3个颜色的值存储，这样的话就生成一张新的贴图了，这就是法线贴图的来历。

•1.1Normal Map（法线贴图）

Object-Space的优点

A. 实现简单，更加直观。我们甚至都不需要模型原始的normal和tangent等信息，也就是说计算更少。生成它也非常简单，而如要要生成Tangent-Space Normal Map的话，由于它的tangent是和UV方向相同，因此想要效果比较好的Normal Map的话要求UV Map也是连续的。

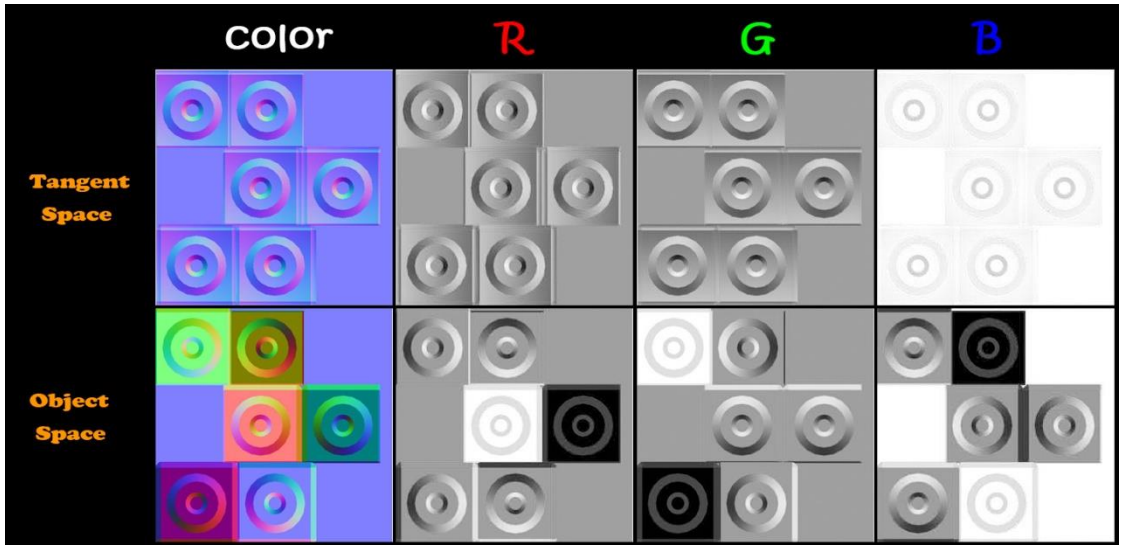
B. 在UV缝合处和尖锐的边角部分，可见的突变（缝隙）较少，可以提供平滑的边界。这是因为Object-Space Normal Map存储的是同一坐标系下的法线信息，因此在边界处通过插值得到的法线可以平缓变换。而Tangent-Space Normal Map中的法线信息则依靠UV的方向和三角化结果，可能在边缘处或尖锐的部分会造成更多可见的缝合迹象。



•1.1Normal Map（法线贴图）

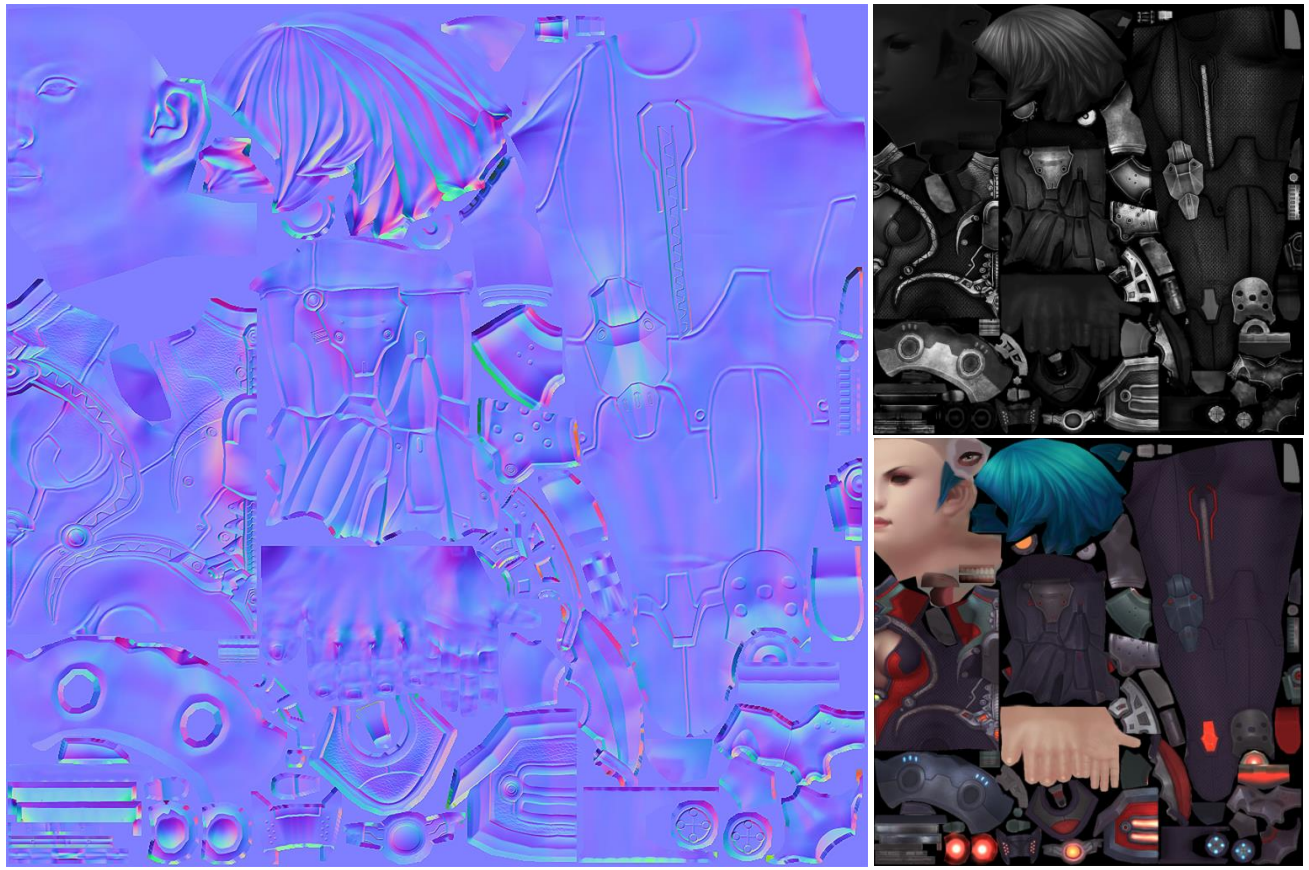
Tangent-Space的优点

A. **自由度很高。**Object-Space Normal Map记录的是绝对法线信息，仅可用于创建它时的那个模型，而应用到其他模型上效果就完全错误了。而Tangent-Space Normal Map记录的是相对法线信息，这意味着，即便把该纹理应用到一个完全不同的网格上，也可以得到一个合理的结果。
B. **可进行UV动画。**比如，我们可以移动一个纹理的UV坐标来实现一个凹凸移动的效果，但使用Object-Space Normal Map会得到完全错误的结果。原因同上。这种UV动画在水或者火山熔岩这种类型的物体会会用到。
C. **可以重用Normal Map。**比如，一个砖块，我们可以仅使用一张Normal Map就可以用到所有的六个面上。
D. **可压缩。**由于Tangent-Space Normal Map中法线的Z方向总是正方向的，因此我们可以仅存储XY方向，而推导得到Z方向。而Object-Space Normal Map由于每个方向都是完全可能的，因此必须存储三个方向的值，不可压缩。



•1.1Normal Map（法线贴图）

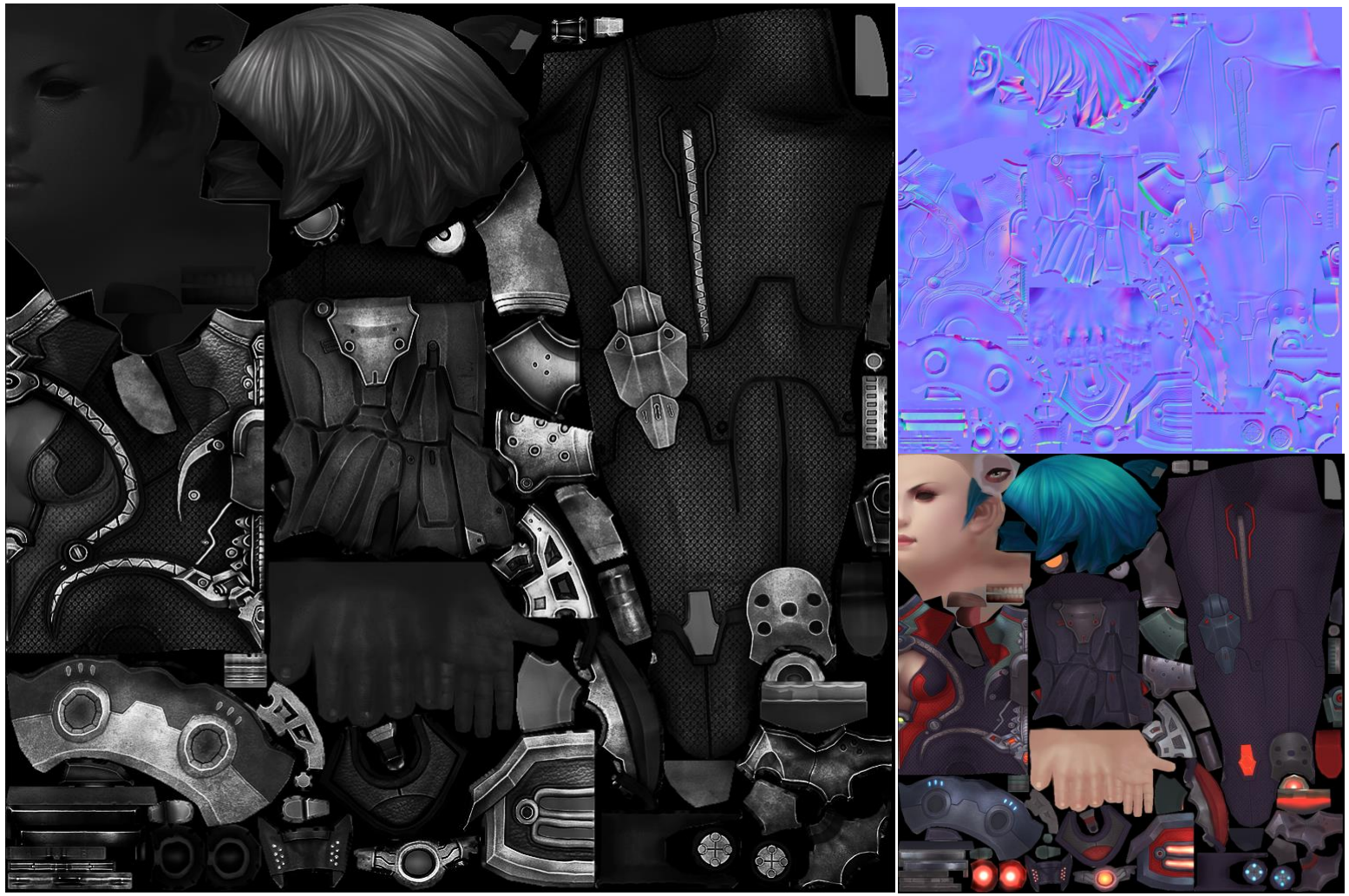
一般情况下我们会使用ZBrush\Maya Mental ray、CrazyBump、NDo2、海龟渲染器…等软件生成Tangent-Space Normal Map。



•1.2Specular Mapping(高光贴图)

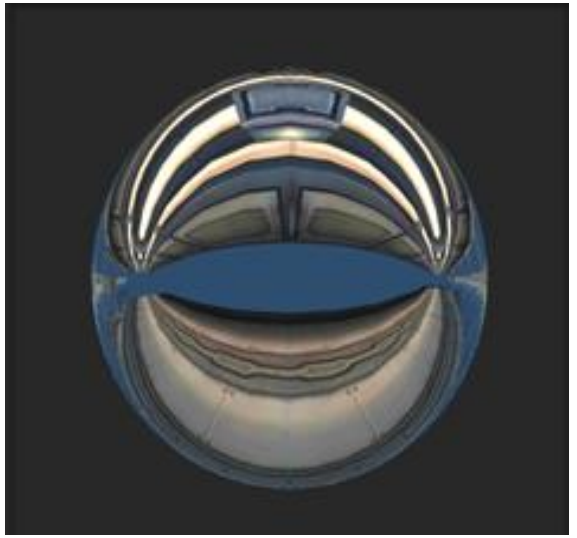
高光贴图在定义上是针对某特定的角度范围反光，而不是全范围的漫射光。基本上高光的亮度是取决于面的法线方向、摄像机和光源的平均方向.

除此之外，高光贴图还可以反映不同的材质，例如金属的反光范围较小，比较接近全漫射光，而且高光还可以体现结构的光滑程度。



- 1.21.3Cube Map（环境反射贴图）

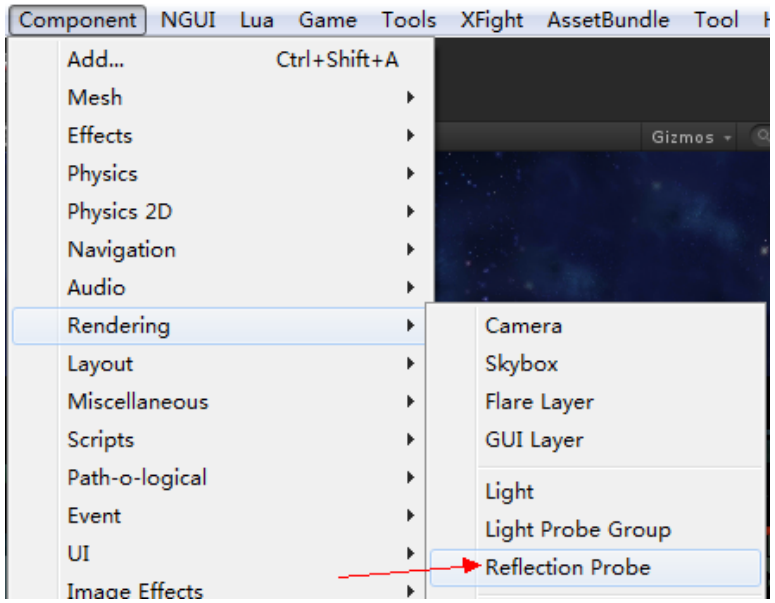
为了在游戏中表现出真是的反射，我们不可能采用真实的物理学计算来模拟，因为代价实在是太大了，我们的手机也承受不了。我们会使用Cubemaps，在Shaders中创建反射的效果。

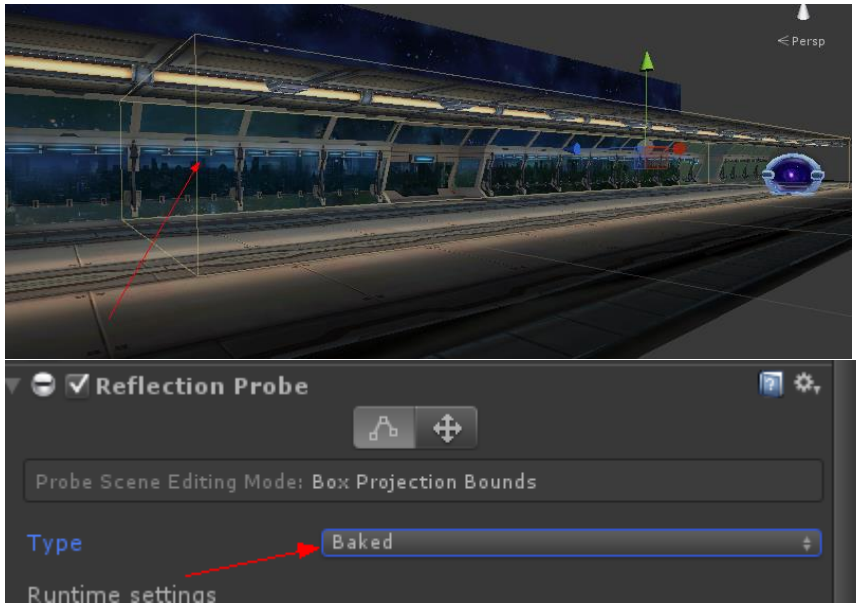


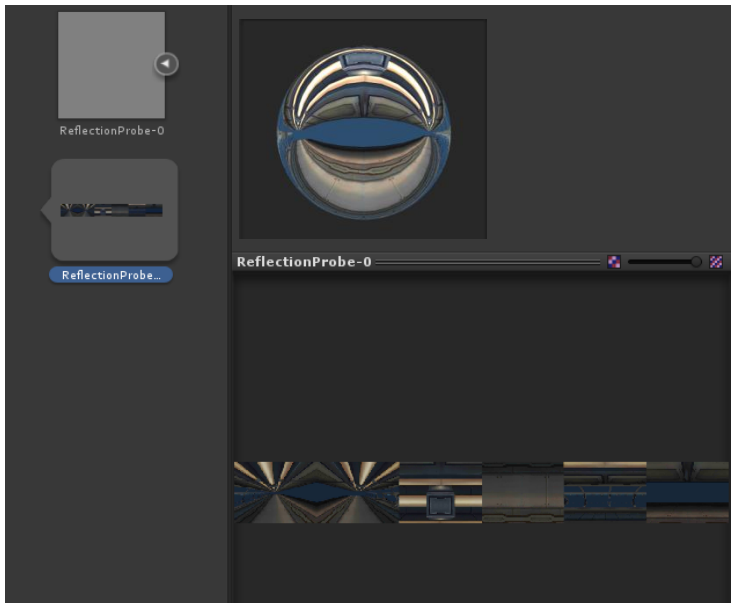
•1.21.3Cube Map（环境反射贴图）

Unity5问世后，我们可以利用新功能Reflecttion Probe来制作我们场景的CubeMap。下面介绍一下大致的方法。

1.建一个空物体，给物体添加一个Reflection Probe Component。
2.将Reflection Probe的Type设置为Baked，并且在场景中调整尺寸到合适的位置。
3.在Inspector里调整各个参数，点击Bake按钮，生成CubeMap，在文件夹里即可找到，可以提供给我们的Shdaer做反射通道使用。





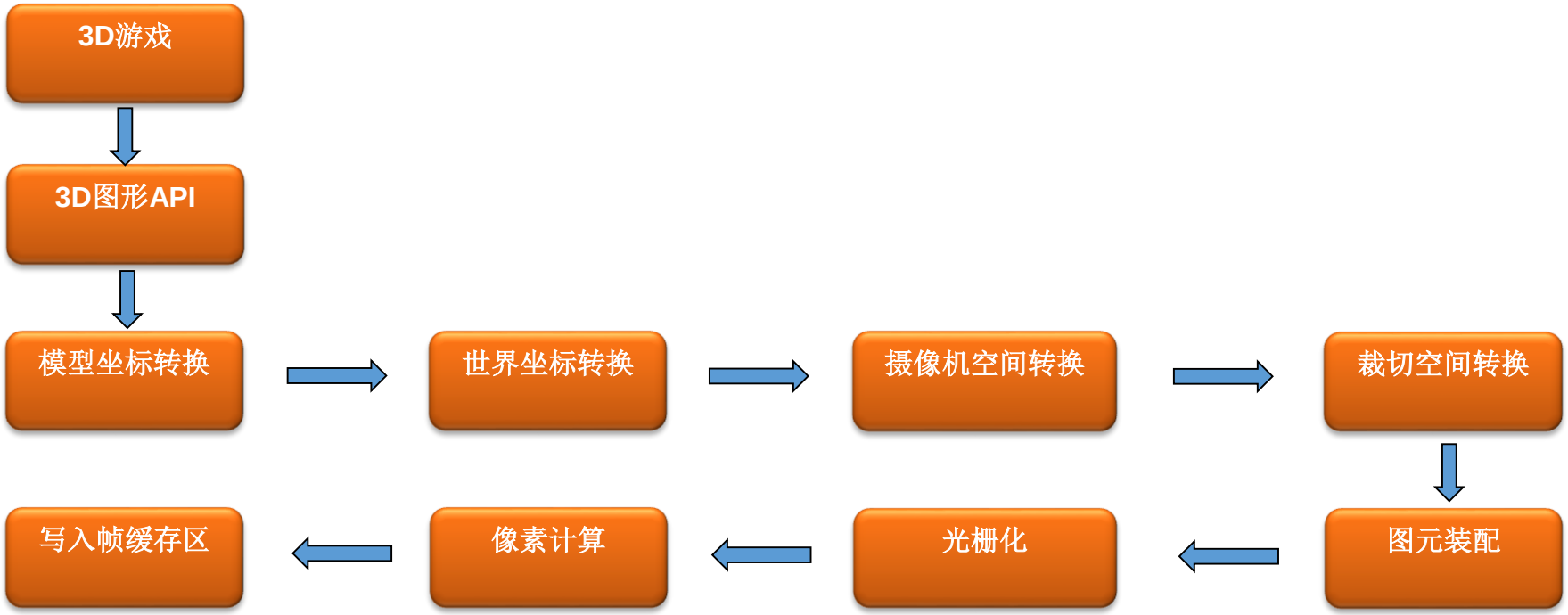


•1.21.3Cube Map（环境反射贴图）

Unity5问世后，我们可以利用新功能Reflecttion Probe来制作我们场景的CubeMap。

•1.4自研X-Shader

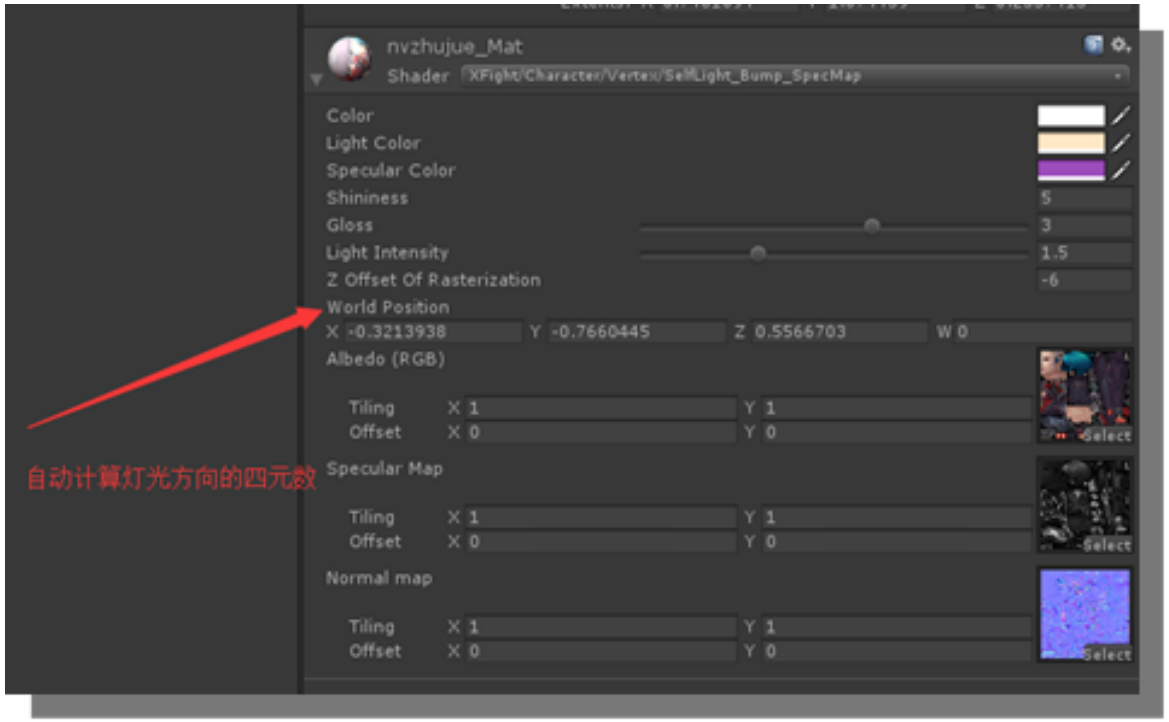
所谓GPU的渲染管线，听起来好像很高深的样子，其实我们可以把它理解为一个流程，就是我们告诉GPU一堆数据，最后得出来一副二维图像，而这些数据就包括了”视点、三维物体、光源、照明模型、纹理”等元素。
在各种图形学的书中，渲染管线主要分为三个阶段：应用程序阶段、几何阶段、光栅阶段。



•1.4自研X-Shader

在手机游戏中使用实时灯光是奢侈的，由于给CPU、GPU的压力较大，所以在很多机器上不仅会有帧率降低的问题，而且还会造成手机发热、费电问题严重，严重影响了用户的游戏体验。既想在游戏中实现次世代游戏的画面效果，例如角色的高光法线效果等，又想提高游戏的运行效率，那么如何解决这个问题呢？

游戏中的场景模型我们会使用LightMap技术（后面会讲到），会有表现真实的阴影效果，所以不必使用实时灯光。但是我们的角色模型在游戏中是动态的，我们想了一个优化办法：给角色模型使用的shader添加一些参数，用来模拟灯光的参数，使其参与到shader里的计算，不再使用Unity3D中的灯光。



•1.4自研X-Shader

Unity3D引擎里灯光无非就是就是一些参数集合的对象，比如灯光位置、灯光强度、灯光颜色等等这些信息，我们可以自己利用这些算法的原理，来优化我们的shader，摒弃一些浪费的计算方式，从而在游戏中不再使用实时灯光，提高游戏的运行效率。

我们通过UnityCG.cginc文件，可以找到计算在Object Space下灯光方向计算的函数：

```
// Computes object space light direction
inline float3 ObjSpaceLightDir( in float4 v )
{
        float3 objSpaceLightPos = mul(_World2Object, _WorldSpaceLightPos0).xyz;
        #ifndef USING_LIGHT_MULTI_COMPILE
                return objSpaceLightPos.xyz - v.xyz * _WorldSpaceLightPos0.w;
        #else
                #ifndef USING_DIRECTIONAL_LIGHT
                return objSpaceLightPos.xyz - v.xyz;
                #else
                return objSpaceLightPos.xyz;
                #endif
        #endif
}
```

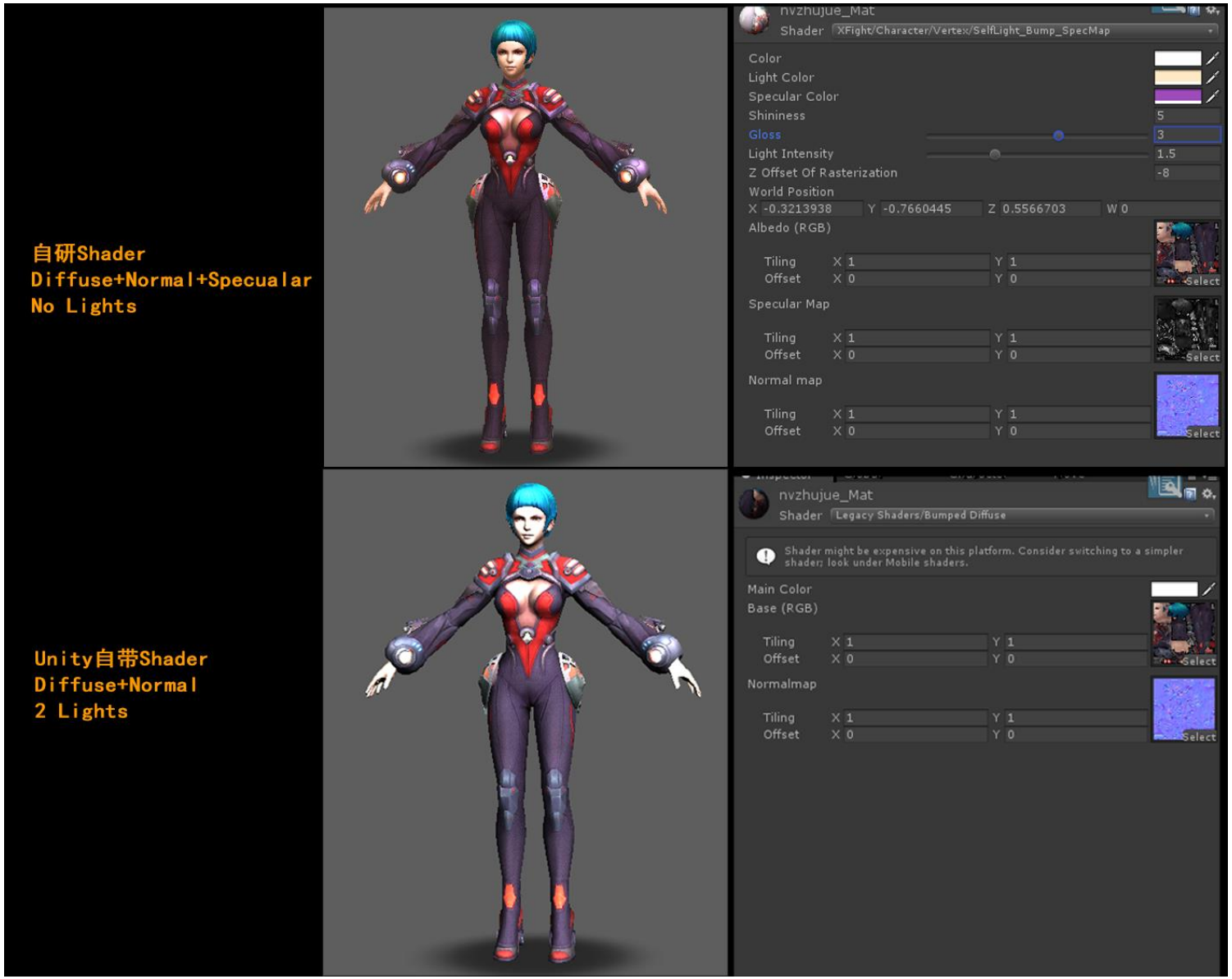
•1.4自研X-Shader

因为我们要在shader中模拟一个灯光，灯光方向我们通关一个四元数来设置，这样可以自由灵活独立控制每一个角色的灯光效果（方向、颜色等）：

```
XFight_Character_V2F vert_XFight_Character(appdata_full v,float4 WPos)
{
        XFight_Character_V2F o;
        o.pos = mul(UNITY_MATRIX_MVP,v.vertex);
        o.uv = v.texcoord.xy;
        o.normal = normalize(v.normal);
        float3 objSpaceLightPos = mul(_World2Object,WPos).xyz;
        float3 objCamPos = ObjSpaceViewDir(v.vertex);
        o.lightDir = normalize(-objSpaceLightPos.xyz);
        o.cameraDir = normalize(objCamPos);
        o.lightColor = ShadeVertexLights(v.vertex,v.normal);
        _ZOffsetOfRasterization *= 0.0001;
        o.pos.z += _ZOffsetOfRasterization;
        return o;
}
```

•1.4自研X-Shader

项目中针对角色的不同效果以及不同的人Render Path使用不同的Shader，如图所示。



•1.4自研X-Shader

Maya 与Unity3D Shader 表现一致

使用unity3D引擎开发游戏，有一个问题就是如何使引擎中自己开发的shader表现与3D美术资源制作软件（我们使用的是Maya)的shader表现实现统一，并且如何美术制作人员可以方便将Maya的材质参数传递给Unity，不需要在Unity中进行2次编辑。

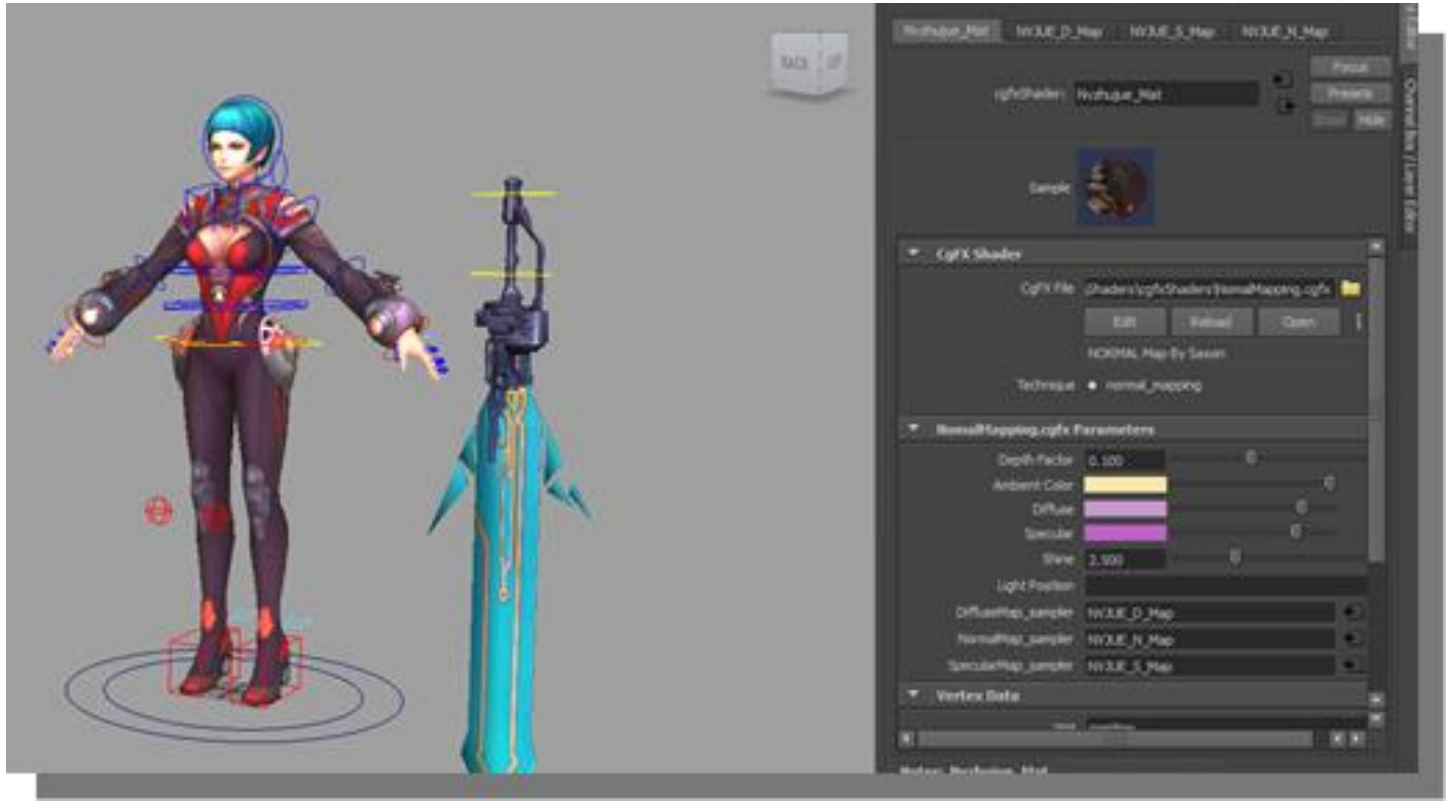
我们在Maya里开发了和在Unity中效果1：1的cgfx shader。

```
float4 normal_map(
    v2f IN,
    uniform sampler2D texmap,
    uniform sampler2D reliefmap,
    uniform sampler2D specmap) : COLOR
{
    float2 uv=IN.txcoord;
    float4 normal=tex2D(reliefmap,uv);
    normal.xyz-=0.5;
    //normal *= 2;

    // transform normal to world space
    normal.xyz=normalize(normal.x*IN.tangent-normal.y*IN.binormal+normal.z*IN.normal);

    // color map
    float4 color=tex2D(texmap,uv);

    // specular map
    float4 specMap = tex2D(specmap,uv);

    // view and light directions
    float3 v = normalize(IN.vpos);
    float3 l = normalize(IN.lightpos.xyz-IN.vpos);

    // compute diffuse and specular terms
    float att=saturate(dot(l,IN.normal.xyz));
    float diff=saturate(dot(l,normal.xyz));
    float spec=saturate(dot(normalize(l-v),normal.xyz)) * specMap.x;

    // compute final color
    float4 finalcolor;
    finalcolor.xyz=ambient * color.xyz + (color.xyz * diffuse.xyz * diff)  +  specular.xyz * pow(spec,shine);
    finalcolor.w=1.0;

    return finalcolor;
}
```

•1.4自研X-Shader

Maya 与Unity3D Shader 表现一致

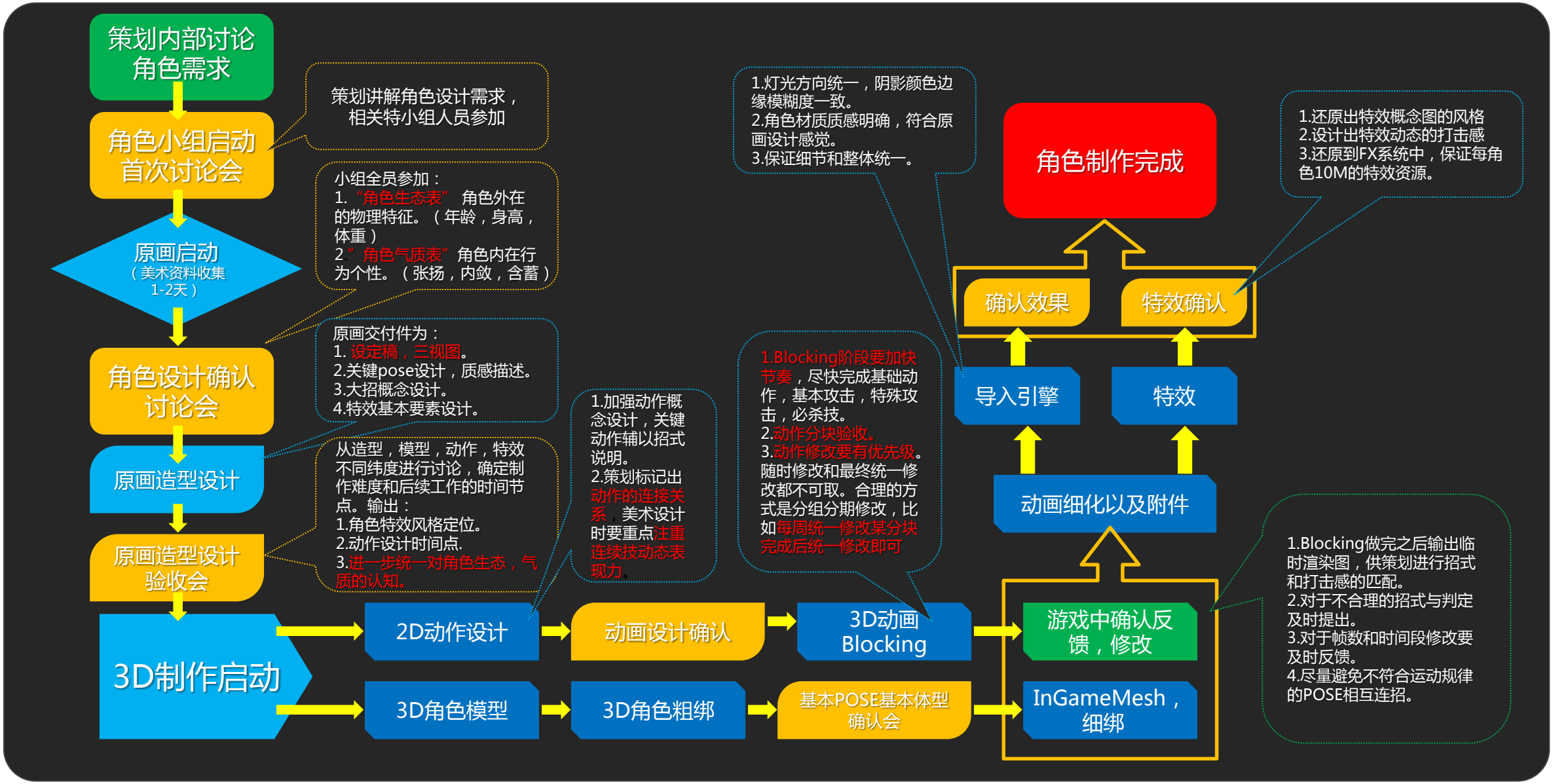
当角色模型的美术制作人员在Maya中调整好角色材质球的美术效果后，通过角色模型导出工具（后面会讲到）可以将材质的参数一同导出，在Unity中传递参数即可。



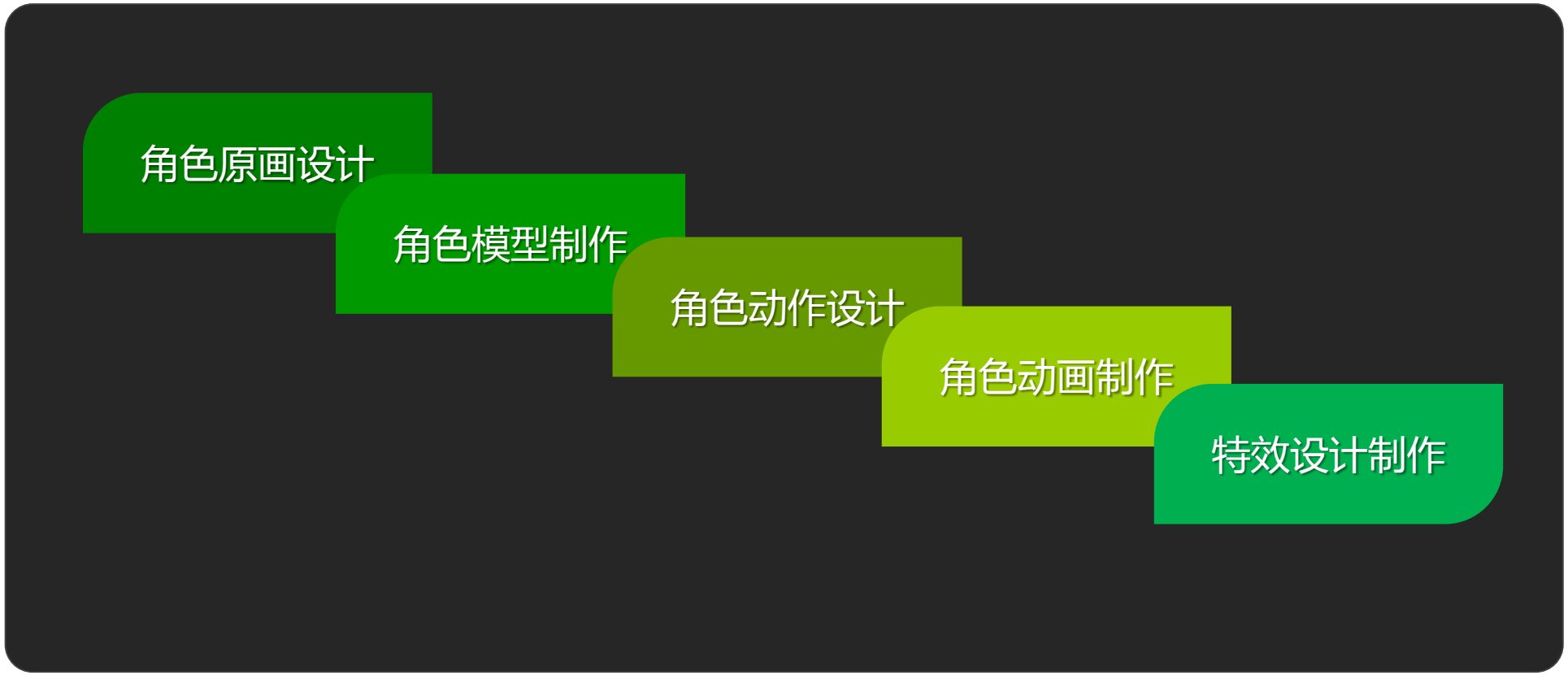
2.Mesh

- 2.1角色的制作流程
- 2.2高模制作
- 2.3INGAME MESH

•2.1角色的制作流程

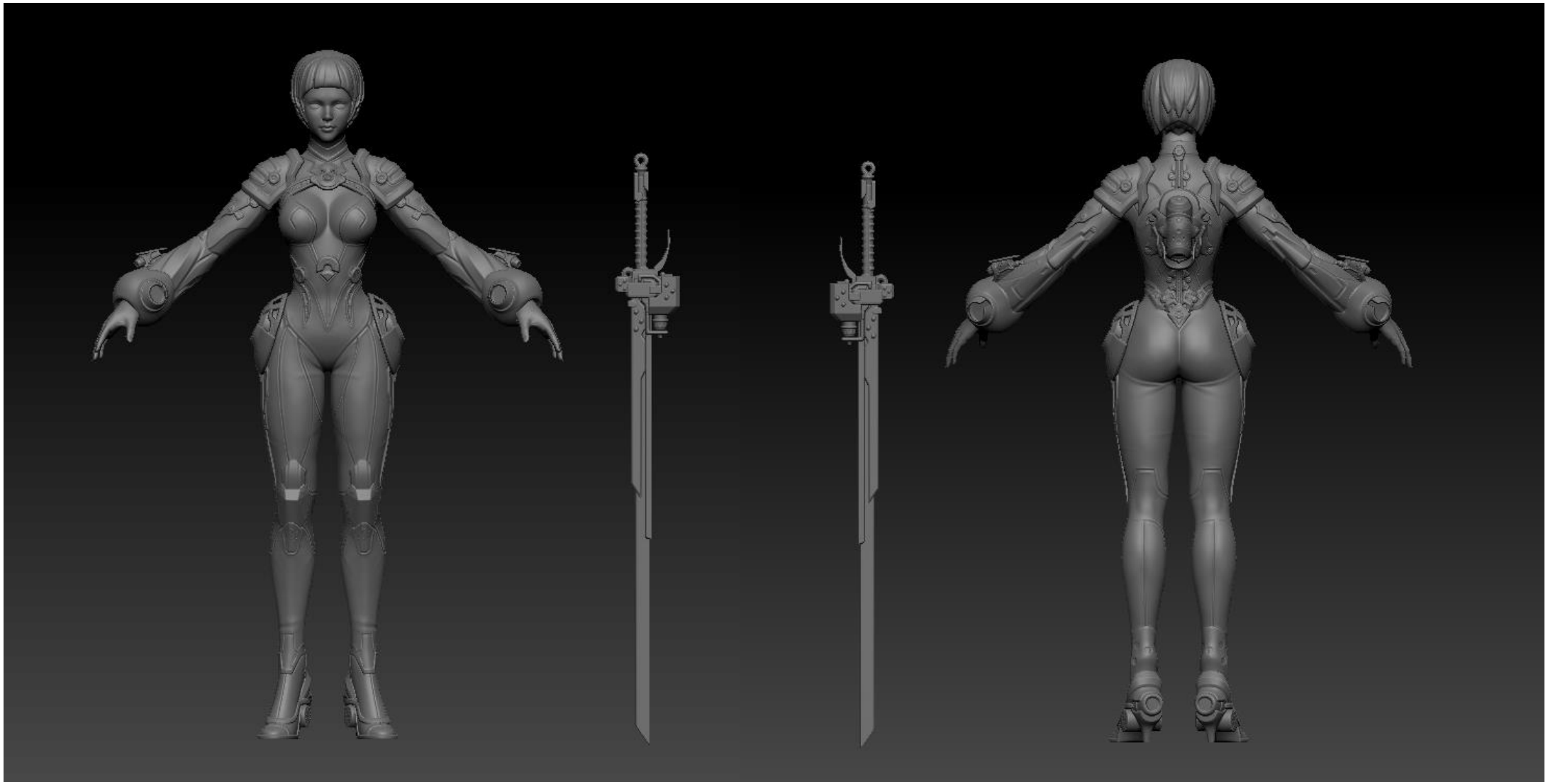

策划内部讨论
角色需求
策划讲解角色设计需求，
相关特小组人员参加
角色小组启动
首次讨论会
小组全员参加：
1.“角色生态表” 角色外在的物理特征。（年龄，身高，体重）
2”角色气质表”角色内在行为个性。（张扬，内敛，含蓄）
原画启动
（美术资料收集
1-2天）
角色设计确认
讨论会
原画交付件为：
1. 设定稿，三视图。
2.关键pose设计，质感描述。
3.大招概念设计。
4.特效基本要素设计。
原画造型设计
从造型，模型，动作，特效不同纬度进行讨论，确定制作难度和后续工作的时间节点。输出：
1.角色特效风格定位。
2.动作设计时间点.
3.进一步统一对角色生态，气质的认知。
原画造型设计
验收会
3D制作启动
2D动作设计
1.加强动作概念设计，关键动作辅以招式说明。
2.策划标记出动作的连接关系，美术设计时要重点注重连续技动态表现力。
动画设计确认
3D动画
Blocking
1.Blocking阶段要加快节奏，尽快完成基础动作，基本攻击，特殊攻击，必杀技。
2.动作分块验收。
3.动作修改要有优先级。随时修改和最终统一修改都不可取。合理的方式是分组分期修改，比如每周统一修改某分块完成后统一修改即可
3D角色模型
3D角色粗绑
基本POSE基本体型
确认会
游戏中确认反馈，修改
InGameMesh，
细绑
1.Blocking做完之后输出临时渲染图，供策划进行招式和打击感的匹配。
2.对于不合理的招式与判定及时提出。
3.对于帧数和时间段修改要及时反馈。
4.尽量避免不符合运动规律的POSE相互连招。
动画细化以及附件
导入引擎
特效
确认效果
特效确认
角色制作完成
1.灯光方向统一，阴影颜色边缘模糊度一致。
2.角色材质质感明确，符合原画设计感觉。
3.保证细节和整体统一。
1.还原出特效概念图的风格
2.设计出特效动态的打击感
3.还原到FX系统中，保证每角色10M的特效资源。

- 2.1角色的制作流程



- 2.2高模制作

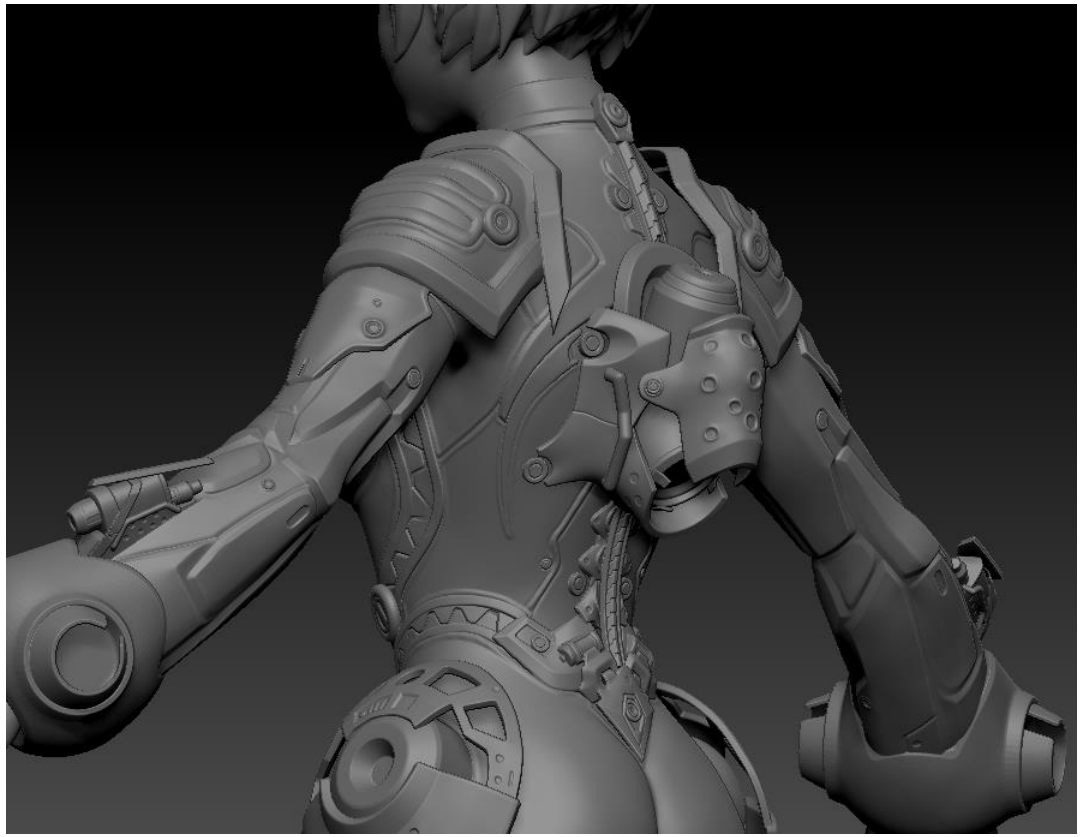
- 2.2高模制作

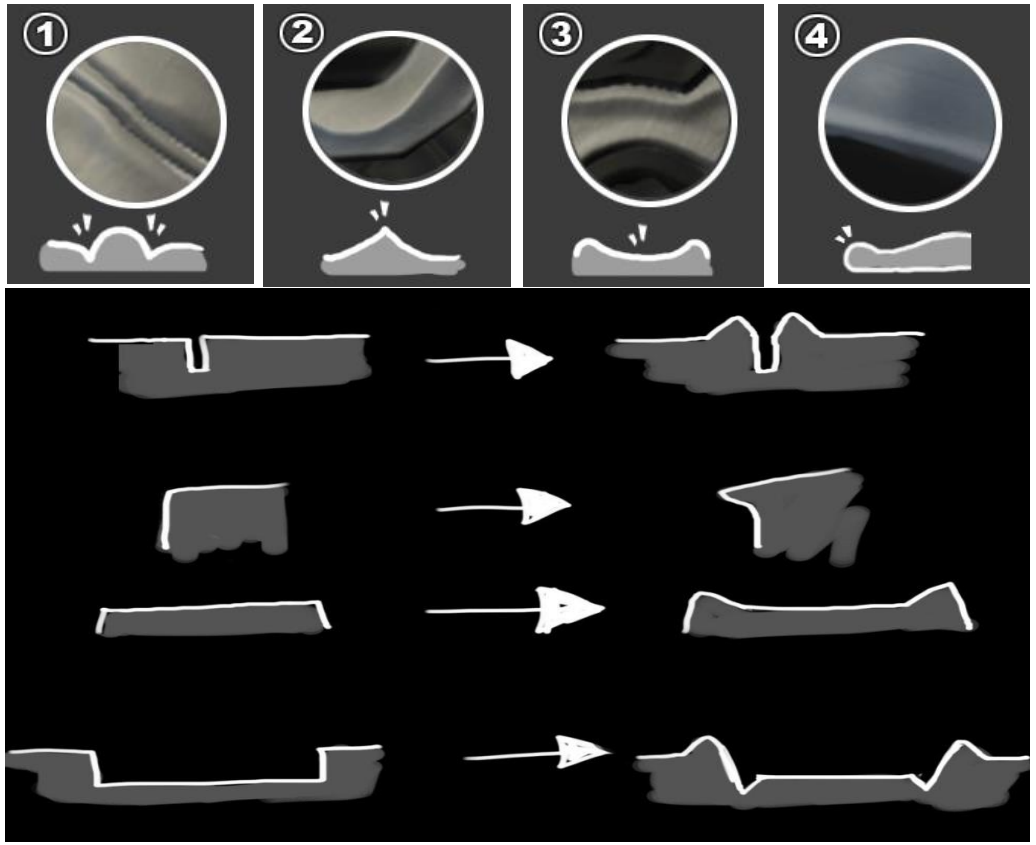


•2.2高模制作

如果是盔甲以及硬皮质的质感的衣服制作的要求如下。

下图是制作原则的总结；



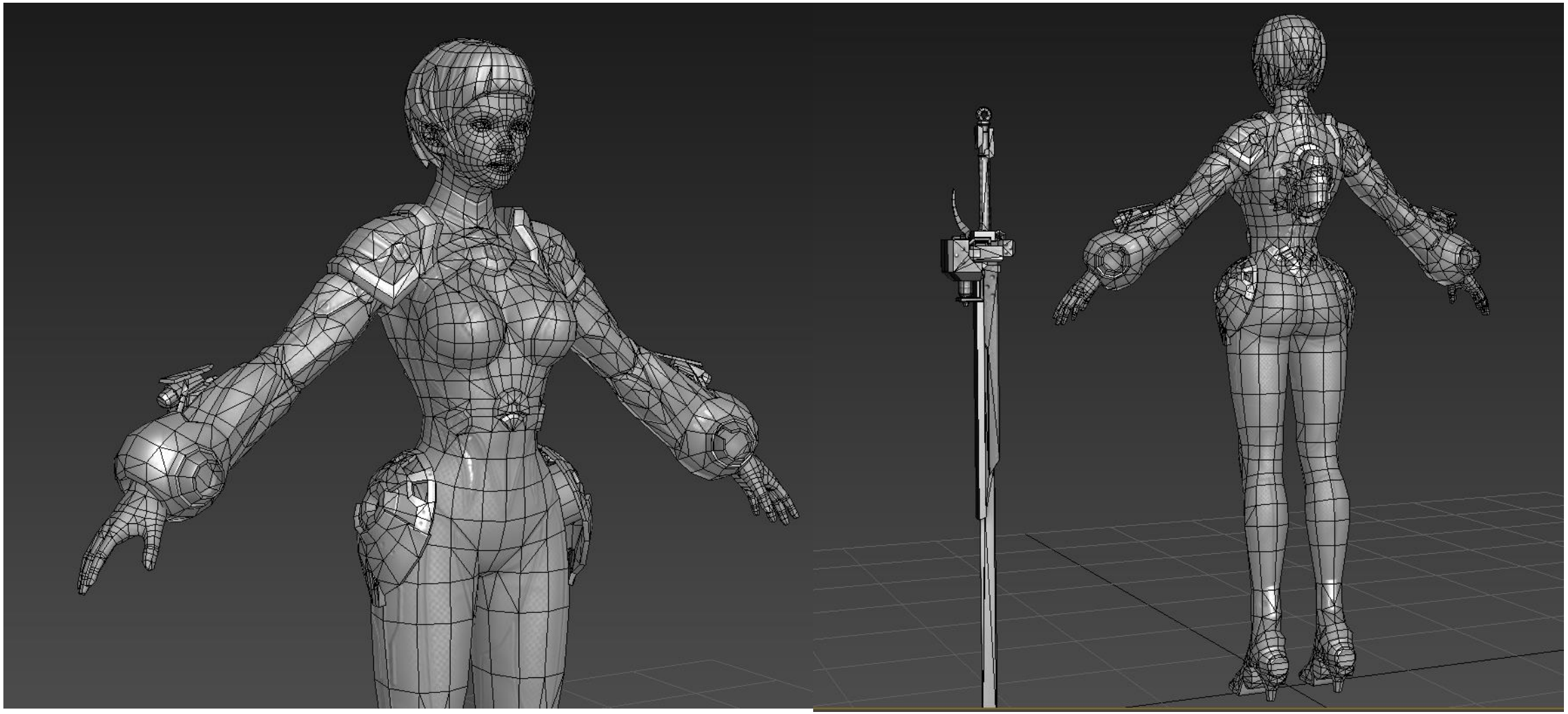


•2.2高模制作

高模制作的原则

像软皮夹克，牛仔裤，布质感的面料要保证结构准确，细节清楚，衣褶的节奏表现分明，有美感。饰物的细节要清楚。花纹要合理要大转折，注意把握疏密的节奏。结合原画设计部分的大转折大色块的要求。

- 2.3INGameMesh



- 2.3INGameMesh

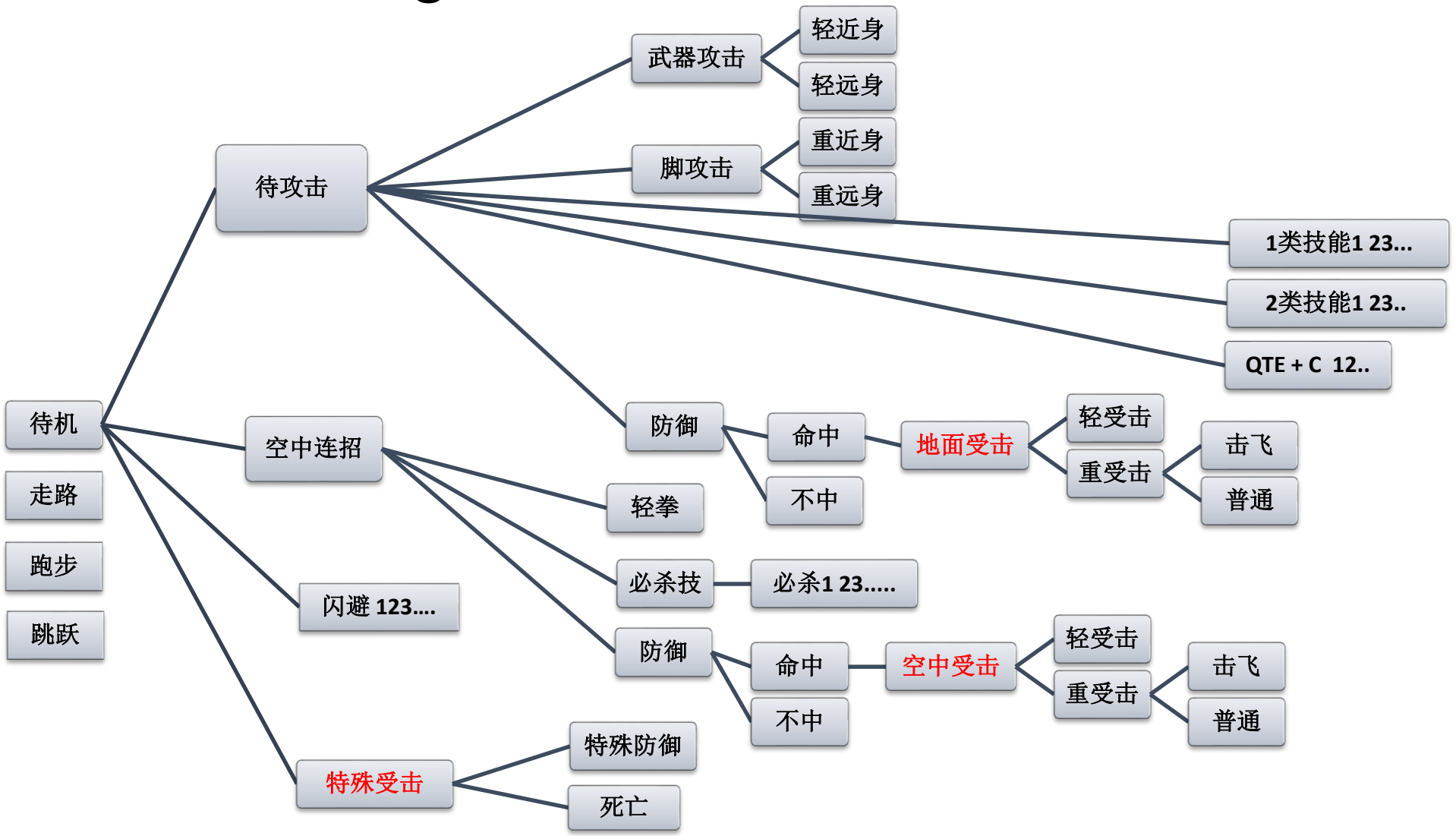
3. Animation

- 3.1 Animation Design
- 3.2 Animation Rigging

Animation Design

要获得高品质动画，必先了解你要做内容和它们之间的逻辑关系。特别是动作类游戏动作和动作之间要结合流畅的连招和相关类似的一些攻击特性。所以必须要先规划一个层级严谨的Animation Tree,先要搞清楚所有的跳转关系。通过跳转关系我们可以清楚的看到所有的动作的数量和大概时长。一方面可以给策划设计动作提供一个设计原则，另一方面可以让动画师将动作很好的分类，并分清主次。

Animation Design



Animation Design

受击（通用所有角色，配合更精准的攻击部位，并延为展招为各个式体系）
表演类（角色性格描述）
基本类 （走跑跳 ，配合烟尘特效）
普通攻击类（通用特效）
特殊攻击类(特效)
必杀超杀类（QTE+镜头特写）

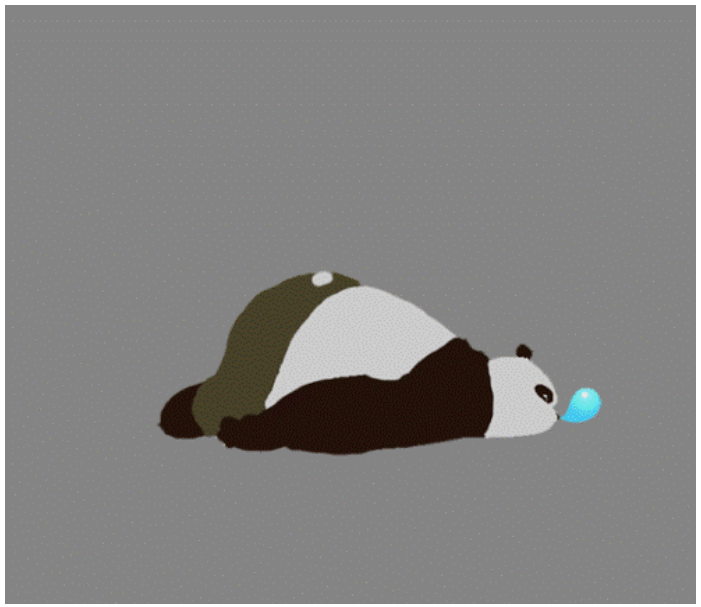
Animation Design

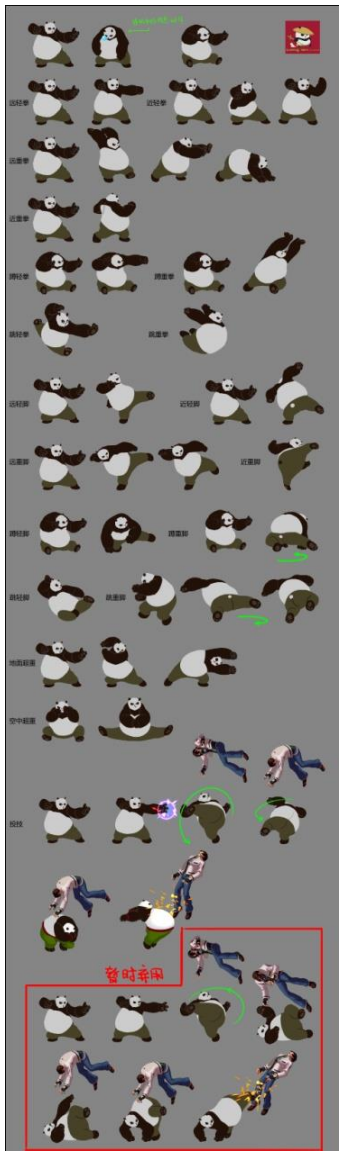
在游戏中的角色的动作都是有限的，如何在有限的动作表现中，一方面实现策划对于攻击判定的要求，另一方面又要体现角色本身鲜明的个性。这个点需要斟酌和权衡，而这个时候的动作的设计者需要被约束，约束的选择是，受击必须是所有角色通用的动作。而受击的表现决定了攻击动作攻击的部位，这样的话受击就会显的尤为重要，特别是精细化的动作游戏。必须是受击反推攻击的原则来设计动作表现。

动作名称	动作命名	动作类别	帧数
受击防御类			
往左受击（重）	nvzhujue@hit_standing_down	受击类	001-012
中段受击（中）	nvzhujue@hit_standing_middle	受击类	001-012
往右受击(轻)	nvzhujue@hit_standing_side	受击类	001-012
后仰受击	nvzhujue@hit_standing_straight	受击类	001-012
防御	nvzhujue@block_standing	受击类	001-012
原地死亡	nvzhujue@die	受击类	001-028
空中受击（仰）	nvzhujue@knock_back_up	受击类	001-007
落地动作（仰）	nvzhujue@fall_down_up	受击类	001-008
空中受击（俯）	nvzhujue@knock_back_bottom	受击类	001-007
落地动作（俯）	nvzhujue@fall_down_bottom	受击类	001-008
空中受击（吹飞）	nvzhujue@knock_back_repel	受击类	001-008
破防	nvzhujue@broken	受击类	001-012

Animation Design

我们之所以会记住某一些优秀的电影或游戏角色，除了他们张相和衣着之外，他们的言行举止也是我们重要的记忆点，特别是动作，最直接的刻画了人物性格最关键部分。而体现这些角色的性格动作的最关键的特征部分就是角色POSE，或者角色剪影。而这些是需要我们在前期需要设计师来进行设计的。





Animation Rigging

Animation系统我们选择了使用Maya作为主要的开发工具。 Maya节点式的结构，以及提供强大的API，为我们开发出相关工具提供了很多便利。

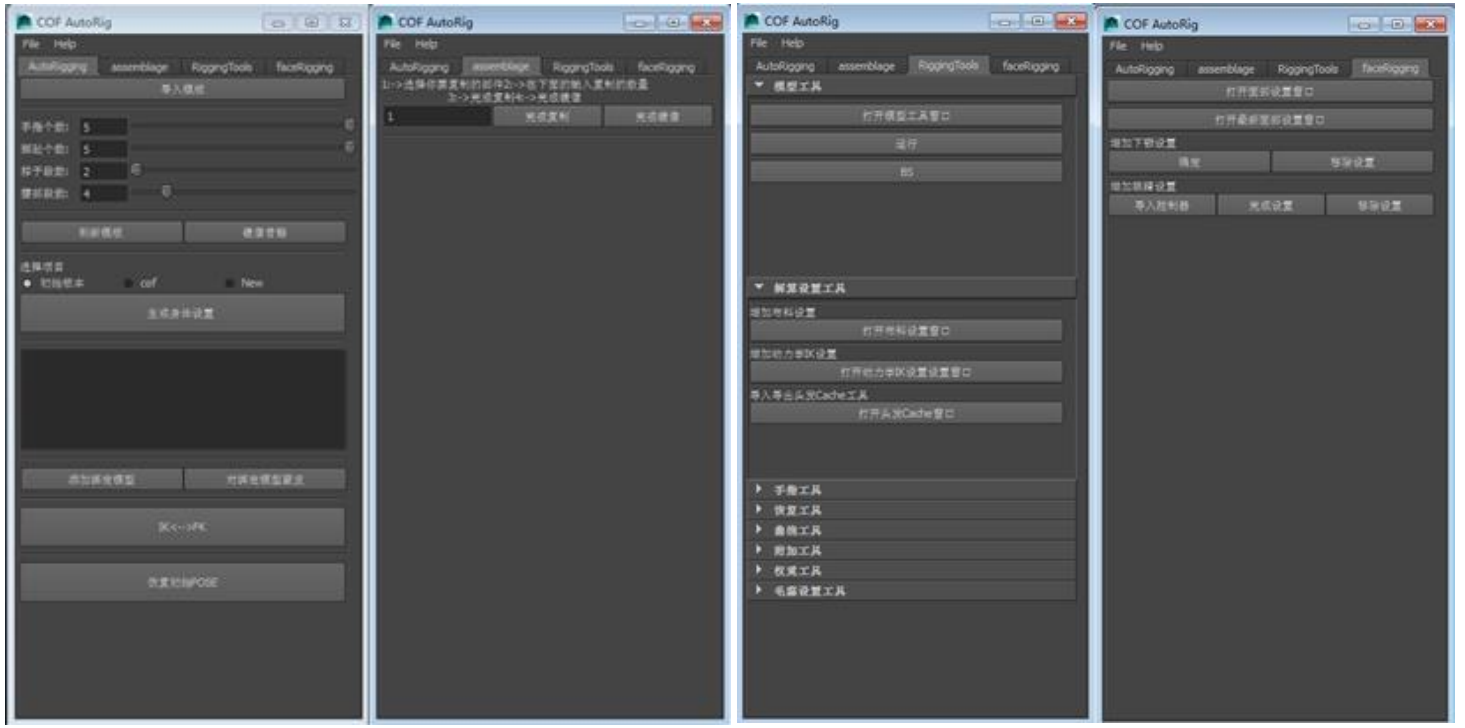
Maya的角色绑定工作还是比较有难度的，偏技术性强，如果想实现复杂的动画效果，对绑定的要求也会很高。如果让美术制作人员手动去绑定一个角色，可能要一周，并且手动操作无法避免的增大了出现错误的可能性。

所以，我们开发了 Auto Rigging Tools自动绑定工具。

Animation Rigging

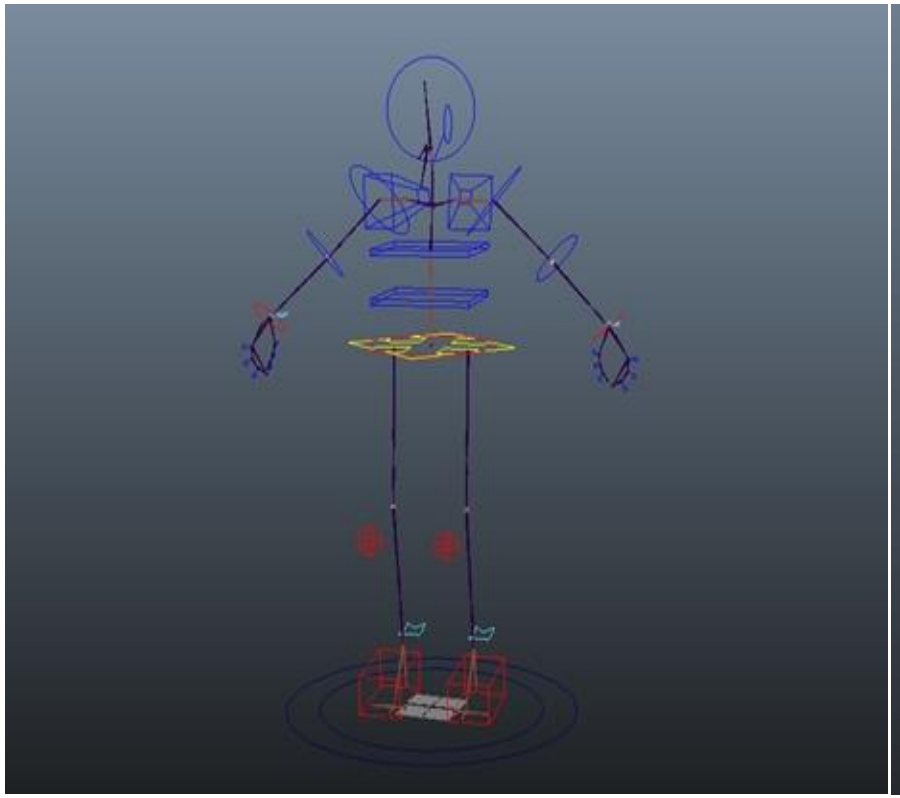
自动绑定工具使用Python开发，Maya从Maya8.5开始支持Python，Python是一种面向对象、解释型计算机程序设计语言，Python语法也很简洁清晰。 该工具提供了可能遇到的各种情况，比如不同的手指个数、脚趾个数、脖子段数、腰部段数、骨骼数量的精细度等等，并且解决了各种技术难题，例如IK（反响动力学）和FK（前向动力学）之间的无缝切换、表情动画的自动化实现、布料解算系统、头发解算系统等。

首先看一下我们工具的UI。



Animation Rigging

通过工具可以给不同形态的角色进行骨骼定位，只需简单的3个步骤，即可自动生成一套手动操作需要一周的绑定，如上图所示。并且自动生成了各种形象的nurbs curve控制器，方便动画师进行各种动画的实现。



Animation Rigging

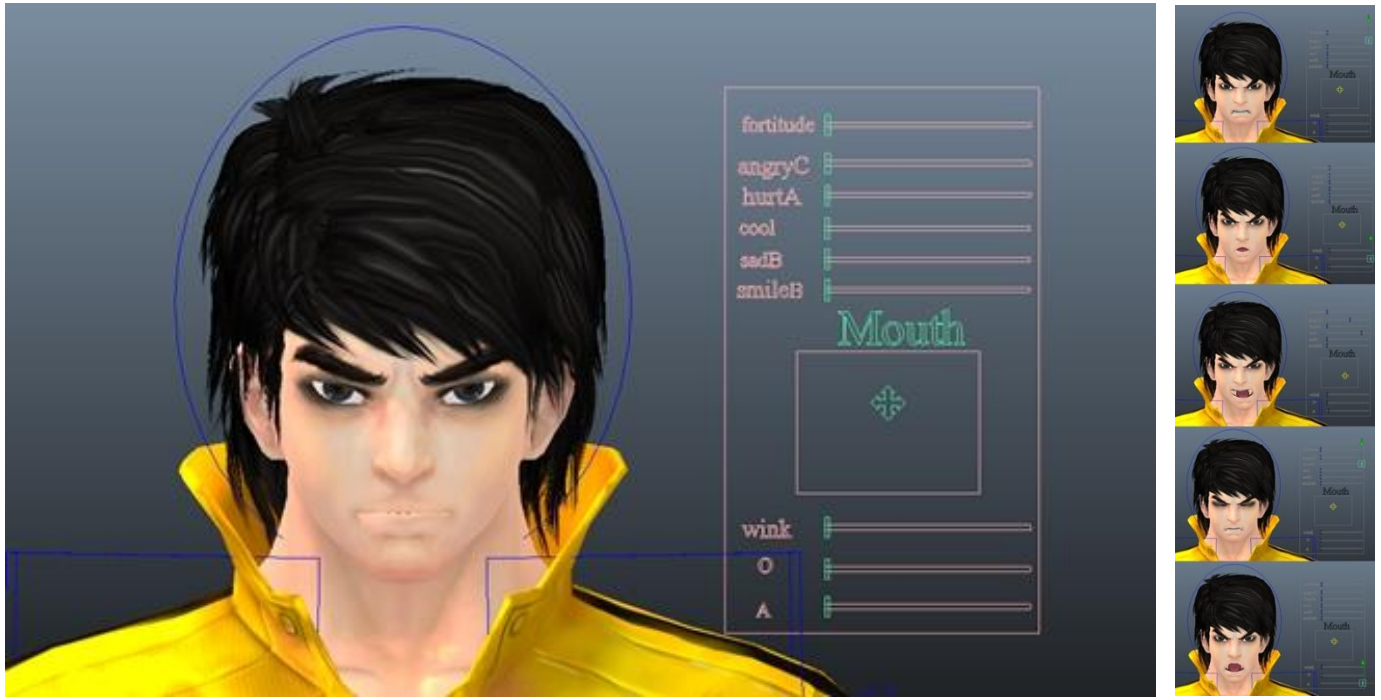
现在一个核心的问题是，游戏引起里参与蒙皮的骨骼数量不能太多，否则严重影响运行效率，我们会在优化骨骼的数量，一般角色在30个骨骼左右。
同样也是使用Python开发了针对于游戏骨骼优化的工具。

AutoCreatJoint

```
import maya.cmds as cmds

class AutoCreatJoint(object):

    def __init__(self):

    def copyJoint(self):
        for oriJnt in self.jntInfoDict.keys():
            for attr in self.lockAttr:
                cmds.setAttr(oriJnt + '.' + attr,l = False,k = True)
            newJnt = cmds.duplicate(oriJnt,rr = True,po = True,n = self.jntInfoDict[oriJnt])[0]
            cmds.parent(newJnt,w = True)
            cmds.select(cl = True)

    def createJntStr(self):

    def createConstrain(self):
        for oriJnt in self.jntInfoDict.keys():
            cmds.parentConstraint(oriJnt,self.jntInfoDict[oriJnt],mo = True,w = 1)
            #cmds.scaleConstraint(oriJnt,self.jntInfoDict[oriJnt],mo = True,w = 1)
        cmds.parentConstraint('Lf_Arm1_Jnt','L_Twist',mo = True,w = 1)
        cmds.parentConstraint('Rt_Arm1_Jnt','R_Twist',mo = True,w = 1)
        cmds.select(cl = True)

    def controlerLockAttr(self):
        cmds.setAttr('waist_Ctrl.ikfk_switch',l = True,k = False,cb = False)
        cmds.setAttr('waist_Ctrl.second_vis',l = True,k = False,cb = False)
        cmds.setAttr('waist_FK1_ctrl.tx',l = True,k = False,cb = False)
        cmds.setAttr('waist_FK1_ctrl.ty',l = True,k = False,cb = False)
        cmds.setAttr('waist_FK1_ctrl.tz',l = True,k = False,cb = False)
        cmds.setAttr('waist_FK2_ctrl.tx',l = True,k = False,cb = False)
        cmds.setAttr('waist_FK2_ctrl.ty',l = True,k = False,cb = False)
        cmds.setAttr('waist_FK2_ctrl.tz',l = True,k = False,cb = False)
        cmds.setAttr('LfArm_UpArm_FK.twistMiddlement',l = True,k = False,cb = False)
        cmds.setAttr('RtArm_UpArm_FK.twistMiddlement',l = True,k = False,cb = False)
        cmds.setAttr('Lfleg_leg_FK.twistMiddlement',l = True,k = False,cb = False)
        cmds.setAttr('Rtleg_leg_FK.twistMiddlement',l = True,k = False,cb = False)
        cmds.setAttr('Lf_shoulder.Arm_follow',0)
        cmds.setAttr('Rt_shoulder.Arm_follow',0)

        needHideCtrls = ['Lf_thumb2_Up','Lf_index2_Up','Rt_thumb2_Up','Rt_index2_Up']
        for ctrl in needHideCtrls:
            cmds.setAttr(ctrl + '.visibility',l = False)
```

Animation Rigging

对于一款次世代格斗游戏来说，我们绝不满足于传统手机游戏的效果。我们决定将表情动画加入到我们游戏的主角当中去。一开始我们想到的是使用骨骼来制作表情动画，可是处于性能上的考虑，这可能使我们的角色骨骼数量翻倍，降低了运行效率。
幸运的是Unity支持BlendShape变形器，这为我们制作表情提供了很好的解决方案。下面我将介绍一下如何在Maya中制作BlendShape表情绑定。

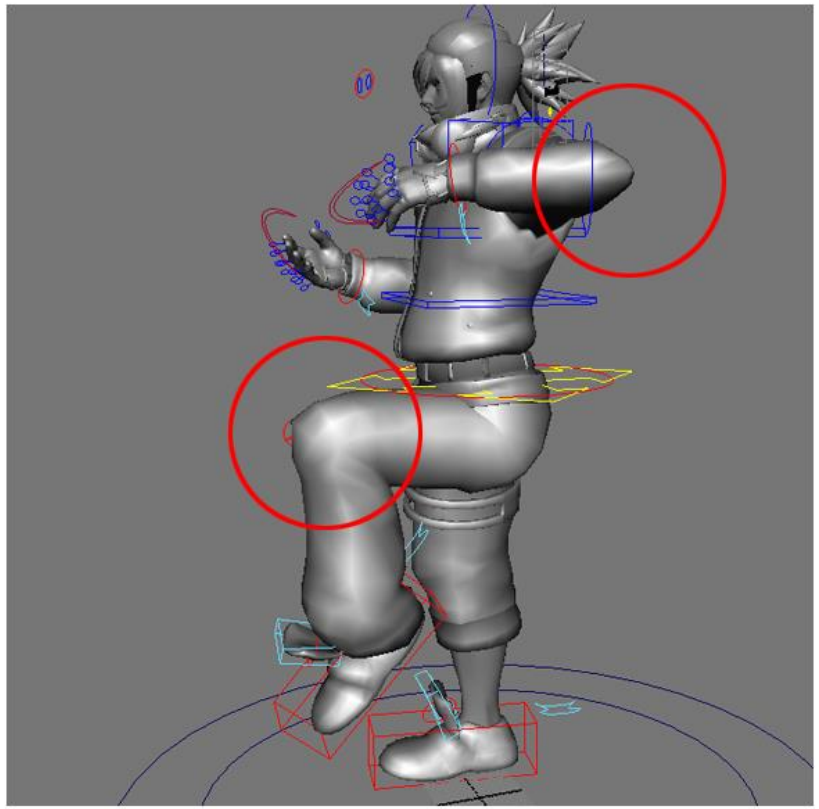


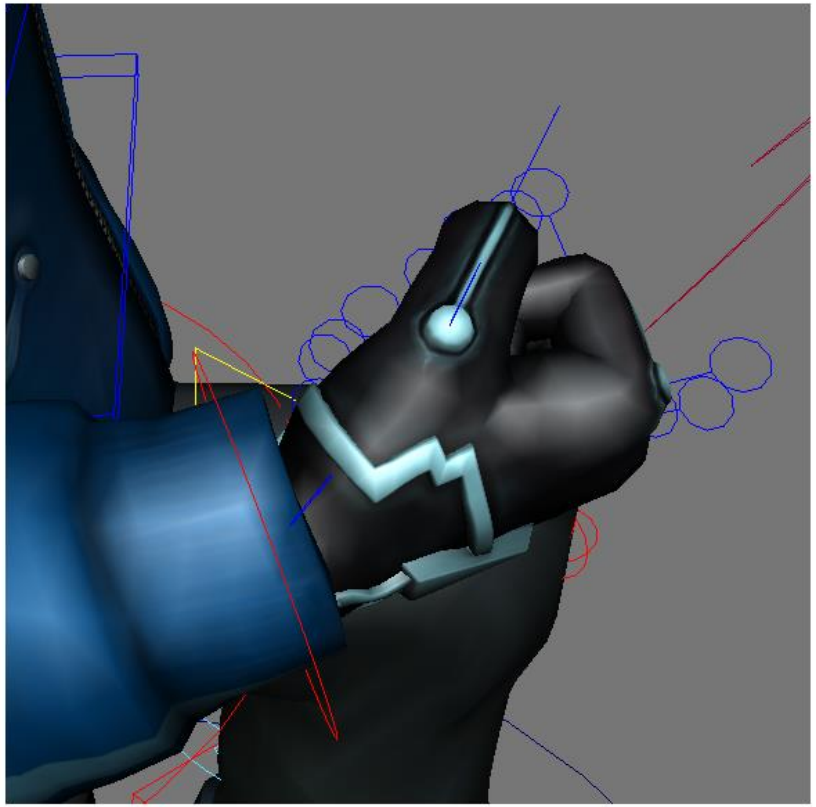
为了尽可能的达到效果和性能上的平衡，我们一个角色的表情目标体控制在8个左右。这几个不同的目标体通过我们的控制器面板实现各种融合，以达到不同的表情效果。

Animation Rigging

当然我们还使用Unity支持BlendShape变形器的方便，为极限动作设计的一种纠正方法，主要是使用驱动关键帧的mesh点的位移来实现骨骼的纠正。

实现了漂亮的手肘和近乎完美的拳头。





4.Lightmap

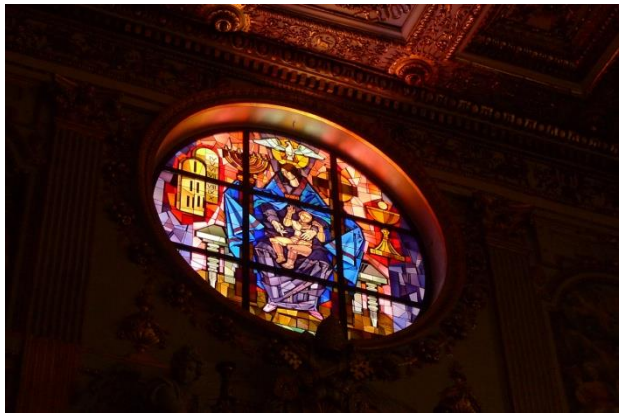
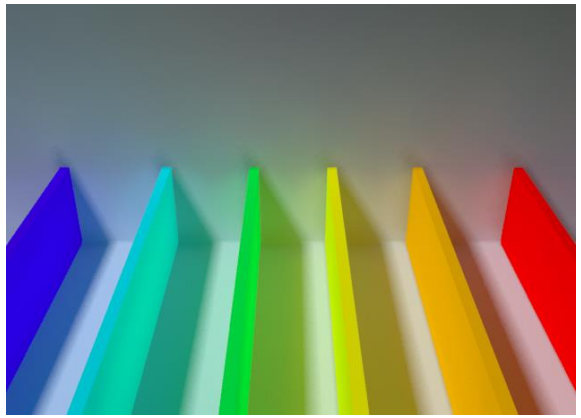
- 4.1Ligthmap的意义和效果
- 4.2Lightmap的使用技巧、方法
- 4.3如何获得优质的Ligthmap（印象派艺术）：

•4.1Ligthmap的意义和效果

首先我们要了解光线与介质：

1.我们所生活的这个客观世界，是存在于**介质**之中的。例如：空气、水、尘埃、和雾。
2.我们所存在和生活的这个世界的光线，因为介质的原因，存在**衰减**。
3.光线透过介质，会产生**漫反射**。
4.万物本身都带有自己的固有色，同时又因为光线的漫反射，自身的色彩也会受到别的物体的影响。



•4.1Ligthmap的意义和效果

光线本身就很美，而LightMap的主要作用就是通过，贴图的方式记录游戏中的光线的信息。

•4.2Lightmap的使用技巧、方法

光线本身就很美，而LightMap的主要作用就是通过，贴图的方式记录游戏中的光线的信息。

在Unity3D中，我们使用LightMapping技术对场景进行灯光烘培，这样做的好处就是把所有的灯光效果、参数通过把这些数据烘培到一张贴图上，不用使用实时灯光就可以实现照明效果，一方面减少了性能上的开销，另一方面让整个场景有非常真实的、柔和的阴影效果，使画面更加真实生动。

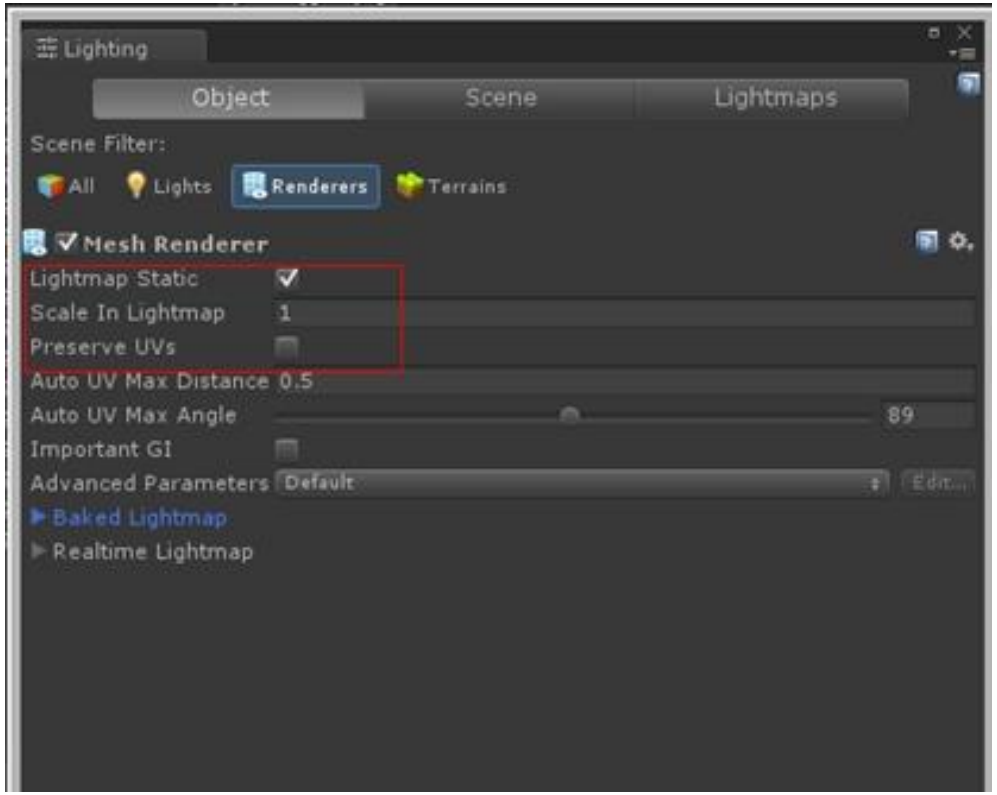
Unity5.X版本采用的是PowerVR Ray Tracing和Enlighten技术，所以Lighting Window 面板里的参数差别就很大。

由于手机游戏会压缩优化最终的包体大小、优化内存大小等，在我们的游戏中，我们对场景进行灯光烘培，优化各个参数，最终只使用一张光照贴图使场景变得丰富、真实。

下面我们详细介绍一下在Unity3D中使用LightMap制作出次世代效果的技巧和方法。

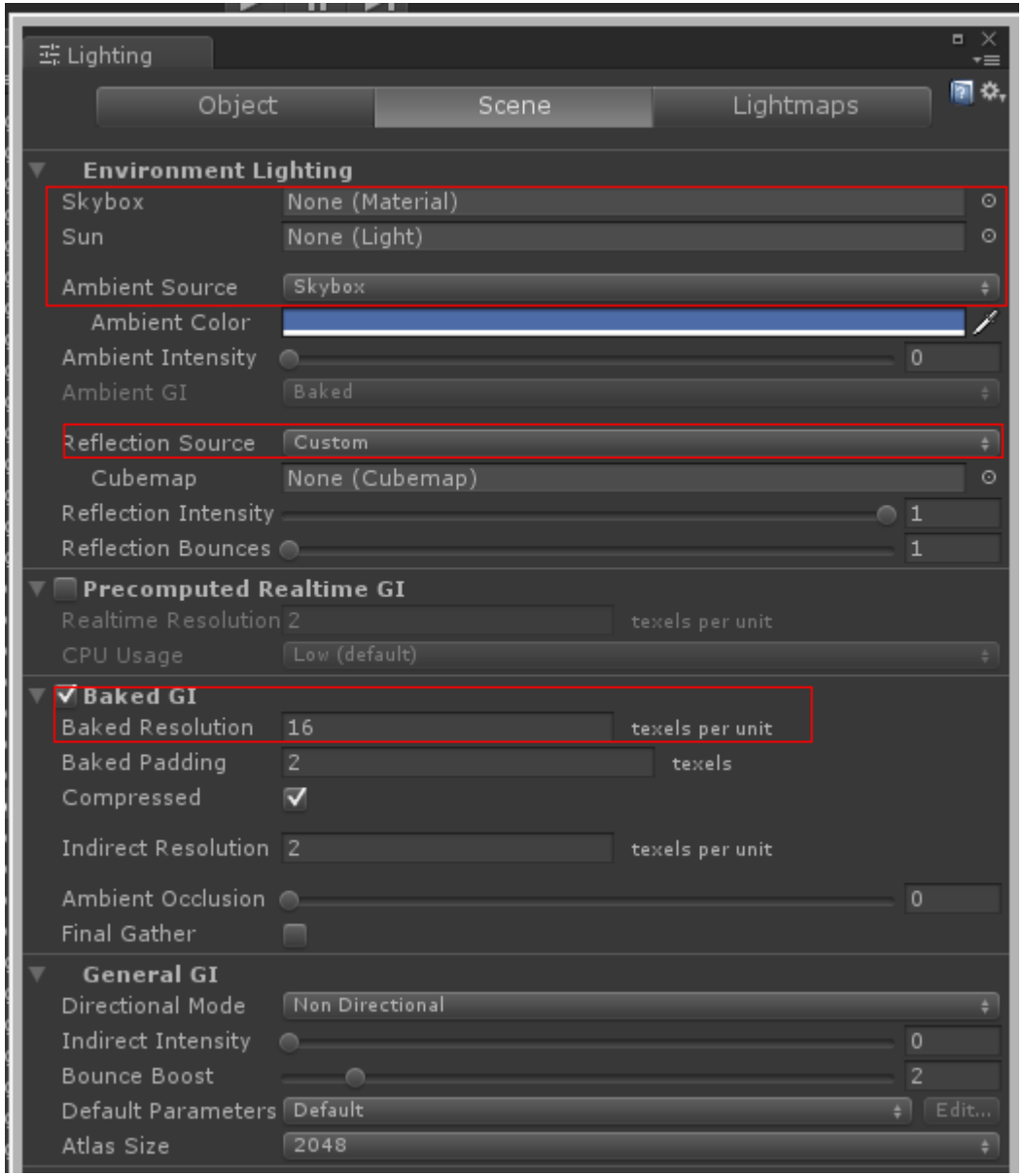
•4.2Lightmap的使用技巧、方法

1. Lightmap Static：必须把烘焙的物体设置成静态
2. Scale in Lightmap：数值的大小与该物体的表面积有关，这个值影响烘焙时间，值越大时间越长。这个值也影响光影图的数量
3. Preserve UVs： 如果物体没有在3d制作软件里展好UV，那么这里必须勾选。



•4.2Lightmap的使用技巧、方法

1.Skybox：会直接影响你的环境光
2.Ambient Source：环境源.
3.Ambient Intensity: 环境光的强度值，也就是环境光的亮度值
4.Reflection Source：反射源，这是Unity4.X里没有的。烘焙后会在场景文件的同名文件夹中多出一个名为 LightmapSnapshot的文件。
5.Realtime Resolution：实时分辨率，这个物体所产生的GI对其它物体的影响程度，值越大影响越大。
6.Baked Resolution的参数值调成1后，阴影的边变的很虚了
7.Indirect Intensity：间接强度，这个和unity4.x里的参数有些像，可以简单的理解为光的反射强度。
8.Bounce Boost：反弹强度，同一单位面积内增大光的反射数量，相比较来说，这个值增大会直接影响渲染时间。
9.Default Parameters：默认参数，这里还可以自己创建自己的参数。一般情况在预烘焙时选择最低设置。
10.Atlas Size:烘焙图的大小，以前4.X版本只能通过脚本来调整，现在好了。



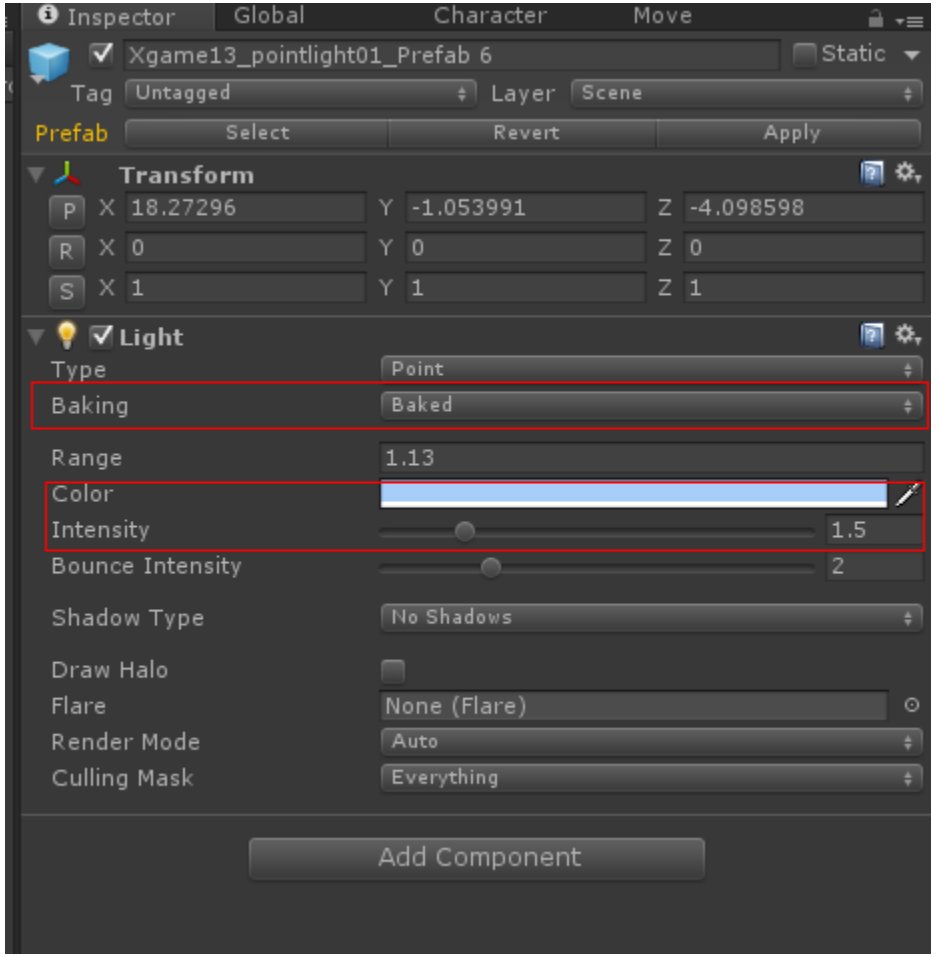
•4.2Lightmap的使用技巧、方法

这里需要注意的是Baking 选项里要设置成Baked，否则烘焙不起作用。

剩下的两个参数，基本和Unity4.X的参数功能相同。

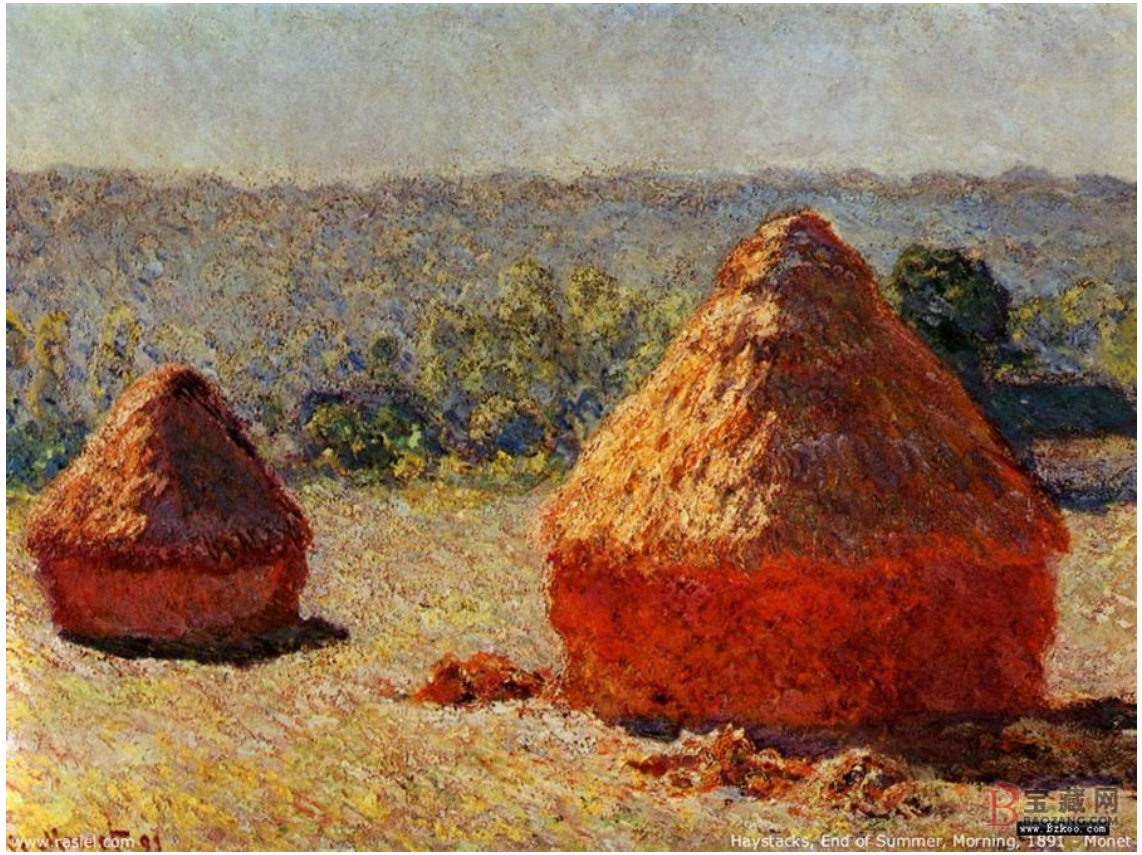
最后说一下，下面这几个参数一定要注意，它们影响烘焙速度和光影图数量。“Scale In Lightmap 、Realtime Resolution、Baked Resolution、Atlas Size”。总体来讲要想提高烘焙速度，就从这Scale In Lightmap 、Realtime Resolution、Baked Resolution三个参数入手基本就可以了。

这几个参数配合着调整，比如烘出多张图时，可以降低使Baked Resolution，或者提高Atlas Size等方法使其LightMap只烘在一张图上，哪怕尺寸大也没关系，可以通过图片优化设置对其进行优化。



4.3如何获得优质的Ligthmap（印象派艺术）

在这之前我们先要说说印象派，印象派是基于对光线的科学认知之后，对于美术界产生的深远影响。从印象派之后，人们对于画面的理解更进了一步。所以我们在讨论游戏美术的画面的时候是绕不开印象派的。印象派对于色彩的理解是从感性到理性的一个认知。



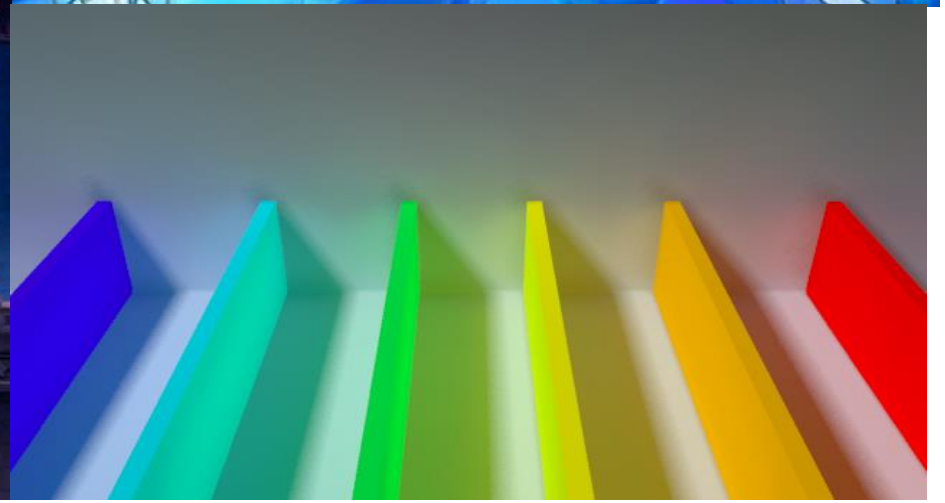
- 4.2Lightmap的使用技巧、方法

大家知道著名的动画电影工作室PIXAR的作品，他们在90年代末出现的作品跟，别的公司之间，除了剧情的优势之外，在场景美术方面也看得出来具备了很大的不同。因为他们熟知印象派对于场景中的光线、色彩、漫射、衰减以及阴影处理方面的这些方法论。



•4.2Lightmap的使用技巧、方法

所以，我们在制作Lightmap的时候不得不去考虑这些客观存在的自然现象，我们可以通过大量的灯光和各种各样的有色彩的光线去烘焙模拟环境色、阴影、漫反射等等自研现象，来丰富我们游戏场景的颜色。从而通过一张简单的Lightmap获得在场景中丰富的色彩和阴影。



- 第二部分 次世代手游如何优化
  - 1.CPU方面的优化
  - 2.GPU方面的优化

次世代手游美术资源的优化

为什么要优化

科技发展日新月异，手机硬件也在不断升级优化，这使得我们可以在移动端上实现次世代游戏的画面效果。由于还是收到一些硬件限制的瓶颈，我们不可能像之前制作主机游戏那样“肆意浪费、随心所欲”，必须要优化处理各种美术资源，以便在不同的平台、不同的硬件终端上有一个好的游戏体验。
根据我们游戏的一些经验，我将一些美术资源的优化点和大家分享一下。

次世代手游美术资源的优化

性能瓶颈有哪些

首先，我们得了解，影响游戏性能的因素哪些，才能对症下药。对于一个游戏来说，有两种主要的计算资源：CPU和GPU。它们会互相合作，来让我们的游戏可以在预期的帧率和分辨率下工作。CPU负责其中的帧率，GPU主要负责分辨率相关的一些东西。

总结起来，主要的性能瓶颈在于：

CPU

- *过多的Draw Calls*
- *复杂的脚本或者物理模拟*

GPU

- *填充率：图形处理单元每秒渲染的像素数量。*
- *像素的复杂度：比如动态阴影，光照，复杂的shader等等*
- *几何体的复杂度：顶点数量*
- *GPU的显存带宽*

次世代手游美术资源的优化

涉及的优化技术

对于CPU来说，限制它的主要是游戏中的Draw Calls，那么什么是Draw Call呢？ DrawCall是CPU调用底层图形接口。比如有上千个物体，每一个的渲染都需要去调用一次底层接口，而每一次的调用CPU都需要做很多工作，那么CPU必然不堪重负。但是对于GPU来说，图形处理的工作量是一样的。所以对DrawCall的优化，主要就是为了尽量解放CPU在调用图形接口上的开销。上面说到过，我们想要绘制图像时，就一定需要调用Draw Call。例如，一个场景里有水有树，我们渲染水的时候使用的是一个material以及一个shader，但渲染树的时候就需要一个完全不同的material和shader，那么就需要CPU重新准备顶点数据、重新设置shader，而这种工作实际是非常耗时的。如果场景中，每一个物体都使用不同的material、不同的纹理，那么就会产生太多Draw Call，影响帧率，游戏性能就会下降。其他CPU的性能瓶颈还有物理、布料模拟、粒子模拟等，都是计算量很大的操作。

而对于GPU来说，它负责整个渲染流水线。它会从处理CPU传递过来的模型数据开始，进行Vertex Shader、Fragment Shader等一系列工作，最后输出屏幕上的每个像素。因此它的性能瓶颈可能和需要处理的顶点数目的、屏幕分辨率、显存等因素有关。总体包含了顶点和像素两方面的性能瓶颈。

了解了上面基本的内容后，下面涉及到的优化技术有：

- ***CPU优化***
  1. ***减少Draw Call数量***
- ***GPU优化***
  1. ***顶点数优化***
  2. ***像素优化***
  3. ***带宽优化***

1.CPU的优化

1.1减少Draw Call数量

次世代手游美术资源的优化

减少Draw Call数量

1.批处理（Batching）

最常见的就是通过批处理（Batching）了。从名字上来理解，就是一块处理多个物体的意思。那么什么样的物体可以一起处理呢？答案就是使用同一个材质的物体。因此，对于使用同一个材质的物体，它们之间的不同仅仅在于顶点数据的差别，即使用的网格不同而已。我们可以把这些顶点数据合并在一起，再一起发送给GPU，就可以完成一次批处理。

Unity中有两种批处理方式：**一种是动态批处理，一种是静态批处理**。

对于动态批处理来说，好消息是一切处理都是自动的，不需要我们自己做任何操作，而且物体是可以移动的，但坏消息是，限制很多，可能一不小心我们就会破坏了这种机制，导致Unity无法批处理一些使用了相同材质的物体。对于静态批处理来说，好消息是自由度很高，限制很少，坏消息是可能会占用更多的内存，而且经过静态批处理后的所有物体都不可以再移动了。

首先来说动态批处理。

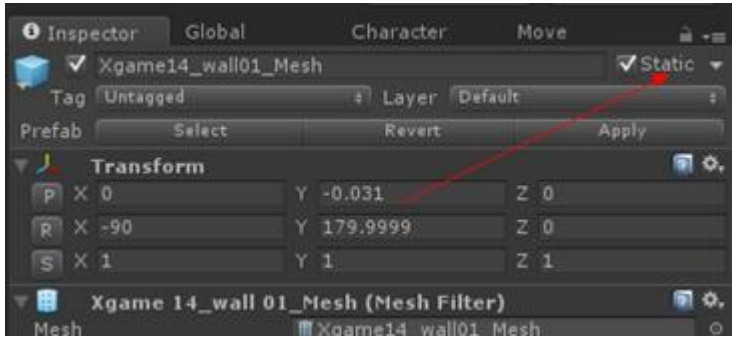
Unity进行动态批处理的条件是，物体使用同一个材质并且满足一些特定条件。Unity总是在不知不觉中就为我们做了动态批处理。动态批处理虽然自动得令人感动，但它对模型的要求很多：

1. 顶点属性的最大限制为900。
2. 物体都必须需要使用同一个缩放尺度（可以是(1, 1, 1)、(1, 2, 3)、(1.5, 1.4, 1.3)等等，但必须都一样）。但如果是非统一缩放（即每个维度的缩放尺度不一样，例如(1, 2, 1)），那么如果所有的物体都使用不同的非统一缩放也是可以批处理的。
3. 使用lightmap的物体不会动态批处理。多passes的shader会中断批处理。接受实时阴影的物体也不会批处理。

次世代手游美术资源的优化

减少Draw Call数量

动态批处理的条件这么多，因此Unity提供了另一个方法，静态批处理。方法很简单，只需把物体的“Static Flag”勾选上。
有一些小提示可以使用：



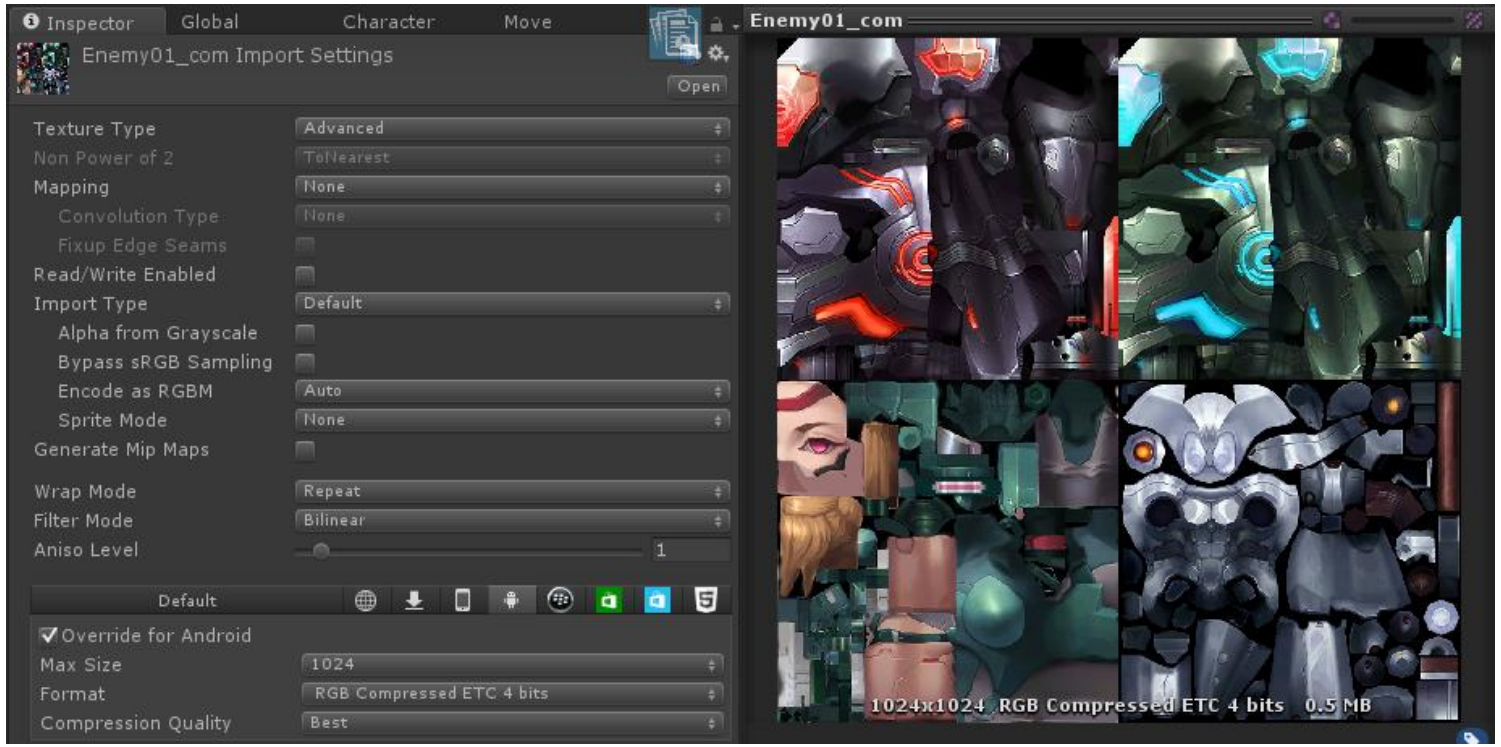
1.尽可能选择静态批处理，但得时刻小心对内存的消耗。
2.如果无法进行静态批处理，而要使用动态批处理的话，那么请小心上面提到的各种注意事项。例如：
2.1尽可能让这样的物体少并且尽可能让这些物体包含少量的顶点属性。
2.2不要使用统一缩放，或者都使用不同的非统一缩放。
3.对于游戏中的小道具，可以使用动态批处理。
4.对于包含动画的这类物体，我们无法全部使用静态批处理，但其中如果有不动的部分，可以把这部分标识成“Static”。

次世代手游美术资源的优化

减少Draw Call数量

合并纹理

虽然批处理是个很好的方式，但很容易就打破它的规定。例如，场景中的物体都使用Diffuse材质，但它们可能会使用不同的纹理。因此，尽可能把多张小纹理合并到一张大纹理（Atlas）中是一个好主意。例如我们将多个敌兵的贴图合并到一个图集中。具体内容后面我们会讲到。



2.GPU的优化

- 2.1 顶点优化
  - 2.1.1 优化Mesh，减少顶点数量
  - 2.1.2 LOD（Level of detail）技术
  - 2.1.3 遮挡剔除（Occlusion culling）技术
- 2.2 像素优化
  - 2.2.1 控制绘制顺序
  - 2.2.2 警惕透明物体的使用
  - 2.2.3 尽量避免实时光照
- 2.3 带宽优化
  - 2.3.1 使用Texture Atlas
  - 2.3.2 关于Generate Mip Maps
  - 2.3.3 优化纹理尺寸大小、压缩格式

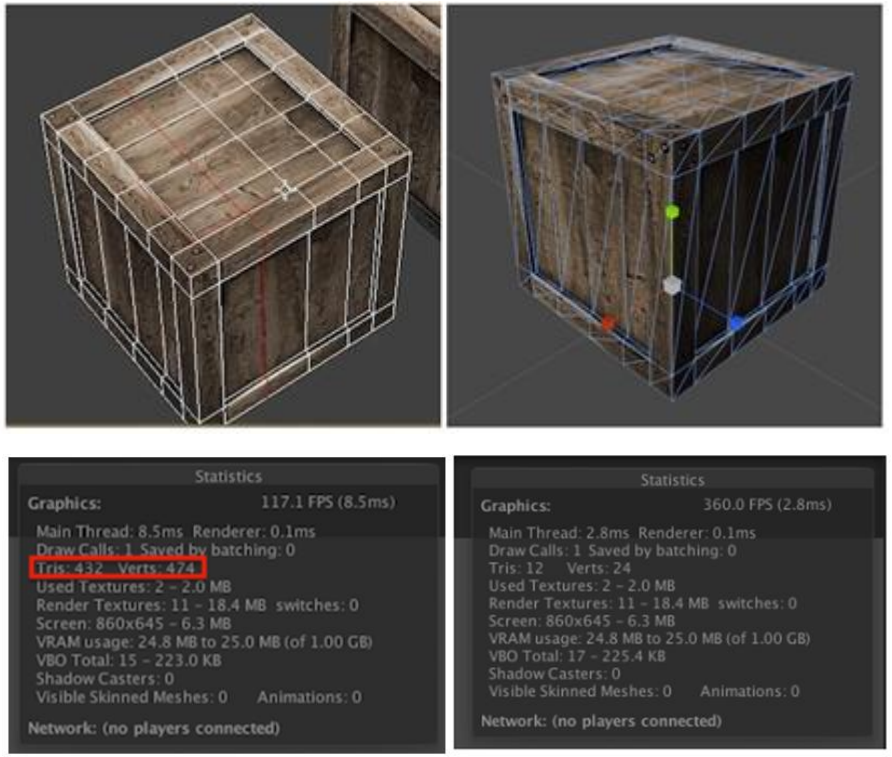
次世代手游美术资源的优化

顶点优化——优化Mesh，减少顶点数量

3D游戏制作都由模型制作开始。而在建模时，有一条我们需要记住：尽可能减少模型中三角形的数目，一些对于模型没有影响、或是肉眼非常难察觉到区别的顶点都要尽可能去掉。例如在下面左图中，正方体内部很多顶点都是不需要的，而把这个模型导入到Unity里就会是右面的情景：

在Game视图下，我们可以查看场景中的三角形数目和顶点数目。可以看到一个简单的正方形就产生了这么多顶点，这是我们不希望看到的。

同时，尽可能重用顶点。在很多三维建模软件中，都有相应的优化选项，可以自动优化网格结构。最后优化后，一个正方体可能只剩下8个顶点。



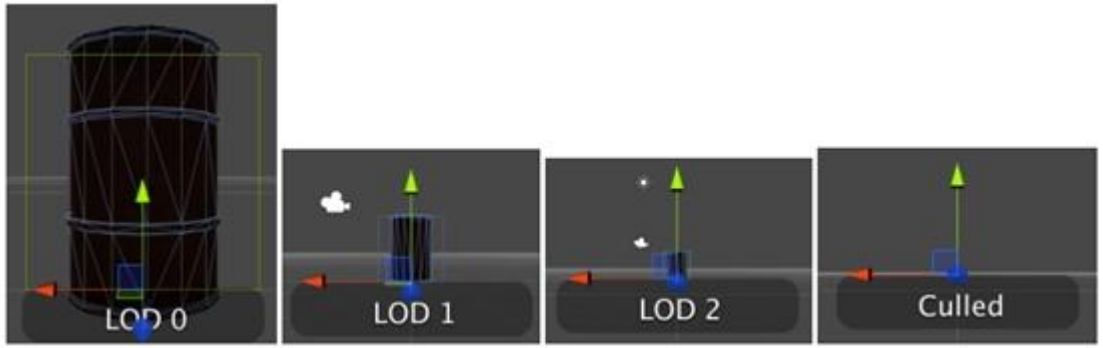
次世代手游美术资源的优化

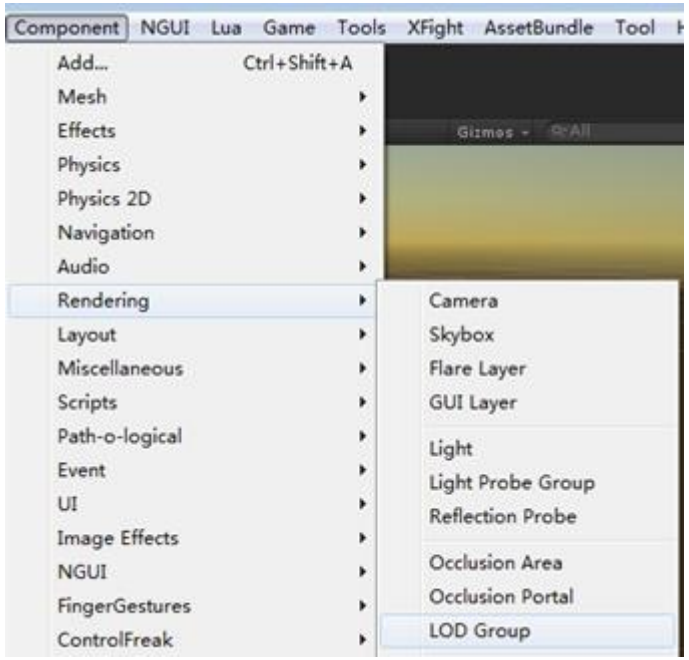
顶点优化—— LOD（Level of detail）技术

LOD技术有点类似于Mipmap技术，不同的是，LOD是对模型建立了一个模型金字塔，根据摄像机距离对象的远近，选择使用不同精度的模型。它的好处是可以在适当的时候大量减少需要绘制的顶点数目。它的缺点同样是需要占用更多的内存，而且如果没有调整好距离的话，可能会造成模拟的突变。

在Unity中，可以通过LOD Group来实现LOD技术。

通过上面的LOD Group面板，我们可以选择需要控制的模型以及距离设置。下面展示了油桶从一个完整网格到简化网格，最后完全被剔除的例子：

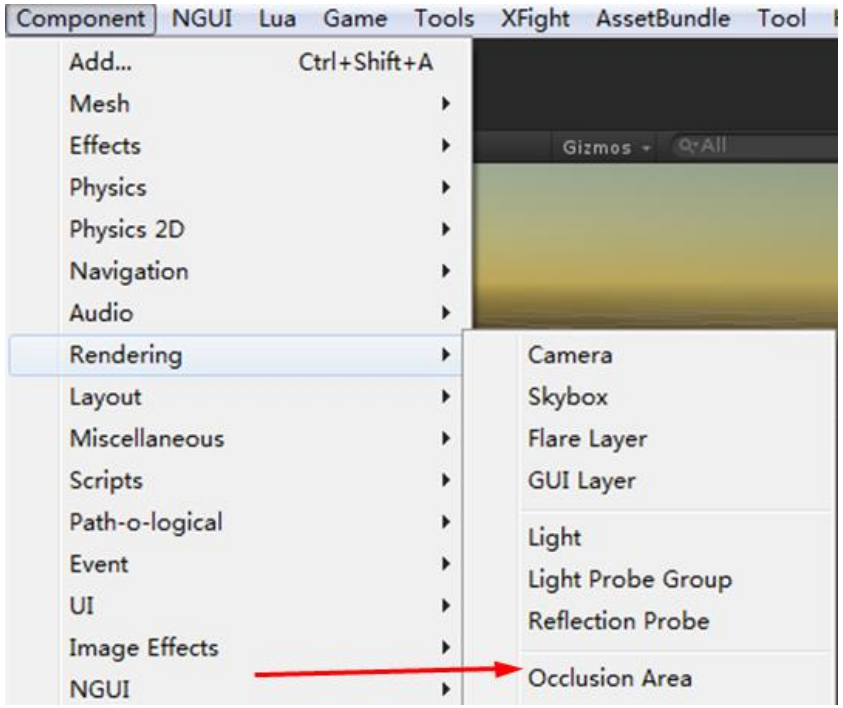




次世代手游美术资源的优化

顶点优化——遮挡剔除（Occlusion culling）技术

遮挡剔除是用来消除躲在其他物件后面看不到的物件，这代表资源不会浪费在计算那些看不到的顶点上，进而提升性能。



次世代手游美术资源的优化

像素优化——控制绘制顺序

像素优化的重点在于减少overdraw。之前提过，overdraw指的就是一个像素被绘制了多次。关键在于控制绘制顺序。

Unity还提供了查看overdraw的视图。当然这里的视图只是提供了查看物体遮挡的层数关系，并不是真正的最终屏幕绘制的overdraw。也就是说，可以理解为它显示的是如果没有使用任何深度检验时的overdraw。这种视图是通过把所有对象都渲染成一个透明的轮廓，通过查看透明颜色的累计程度，来判断物体的遮挡。

右图，红色越是浓重的地方表示Overdraw越严重。

需要控制绘制顺序，主要原因是为了最大限度的避免overdraws，也就是同一个位置的像素可以需要被绘制多变。在PC上，资源无限，为了得到最准确的渲染结果，绘制顺序可能是从后往前绘制不透明物体，然后再绘制透明物体进行混合。但在移动平台上，这种会造成大量overdraw的方式显然是不适合的，我们应该尽量从前往后绘制。从前往后绘制之所以可以减少overdraw，都是因为深度检验的功劳。

在Unity中，那些Shader中被设置为“Geometry” 队列的对象总是从前往后绘制的，而其他固定队列（如“Transparent”“Overla”等）的物体，则都是从后往前绘制的。这意味这，我们可以尽量把物体的队列设置为“Geometry”。而且，我们还可以充分利用Unity的队列来控制绘制顺序。例如，对于天空盒子来说，它几乎覆盖了所有的像素，而且我们知道它永远会在所有物体的后面，因此它的队列可以设置为“Geometry+1”。这样，就可以保证不会因为它而造成overdraws。

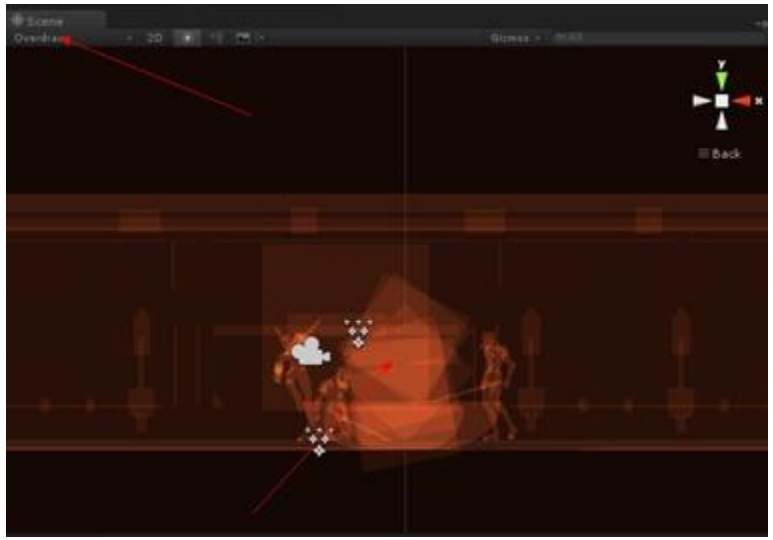
次世代手游美术资源的优化

像素优化——警惕透明物体的使用

对于透明对象，由于它本身的特性决定如果要得到正确的渲染效果，就必须从后往前渲染（这里不讨论使用深度的方法），而且抛弃了深度检验。这意味着，透明物体几乎一定会造成overdraws。如果我们不注意这一点，在一些机器上可能会造成严重的性能损失。例如，对于GUI对象来说，它们大多被设置成了半透明，如果屏幕中GUI占据的比例太多，而主摄像机又没有进行调整而是投影整个屏幕，那么GUI就会造成屏幕的大量overdraws。

因此，如果场景中大面积的透明对象，或者有很多层覆盖的多层透明对象（即便它们每个的面积可以都不大），或者是透明的粒子效果，在移动设备上也会造成大量的overdraws。这是应该尽量避免的。

对于上述GUI的这种情况，我们可以尽量减少窗口中GUI所占的面积。如果实在无能为力，我们可以把GUI绘制和三维场景的绘制交给不同的摄像机，而其中负责三维场景的摄像机的视角范围尽量不要和GUI重叠。对于其他情况，只能说，尽可能少用。当然这样会对游戏的美观度产生一定影响，因此我们可以在代码中对机器的性能进行判断，例如首先关闭所有的耗费性能的功能，如果发现这个机器表现非常良好，再尝试开启一些特效功能。



次世代手游美术资源的优化

像素优化——尽量避免实时光照

实时光照对于移动平台是个非常昂贵的操作。如果只有一个平行光还好，但如果场景中包含了太多光源并且使用了很多多Passes的shader，那么很有可能会造成性能下降。而且在有些机器上，还要面临shader失效的风险。例如，一个场景里如果包含了三个逐像素的点光源，而且使用了逐像素的shader，那么很有可能将Draw Calls提高了三倍，同时也会增加overdraws。这是因为，对于逐像素的光源来说，被这些光源照亮的物体要被再渲染一次。更糟糕的是，无论是动态批处理还是静态批处理，对于这种逐像素的pass都无法进行批处理，也就是说，它们会中断批处理。

我们看到很多成功的移动游戏，它们的画面效果看起来好像包含了很多光源，但其实这都是骗人的。

次世代手游美术资源的优化

带宽优化——Texture Atlas

使用Texture Atlas可以帮助减少Draw Calls，而这些纹理的大小同样是一个需要考虑的问题。在这之前要提到一个问题就是，所有纹理的长宽比最好是正方形，而且长度值最好是2的整数幂。这是因为有很多优化策略只有在这种时候才可以发挥最大效用。

右图是Unity中查看纹理参数可以通过纹理的面板。我们尽量把一个场景单独使用的模型贴图，合并到一起，有经验的美工在分UV、贴图制作是就已经充分考虑到了这一点。

次世代手游美术资源的优化

带宽优化——Generate Mip Maps

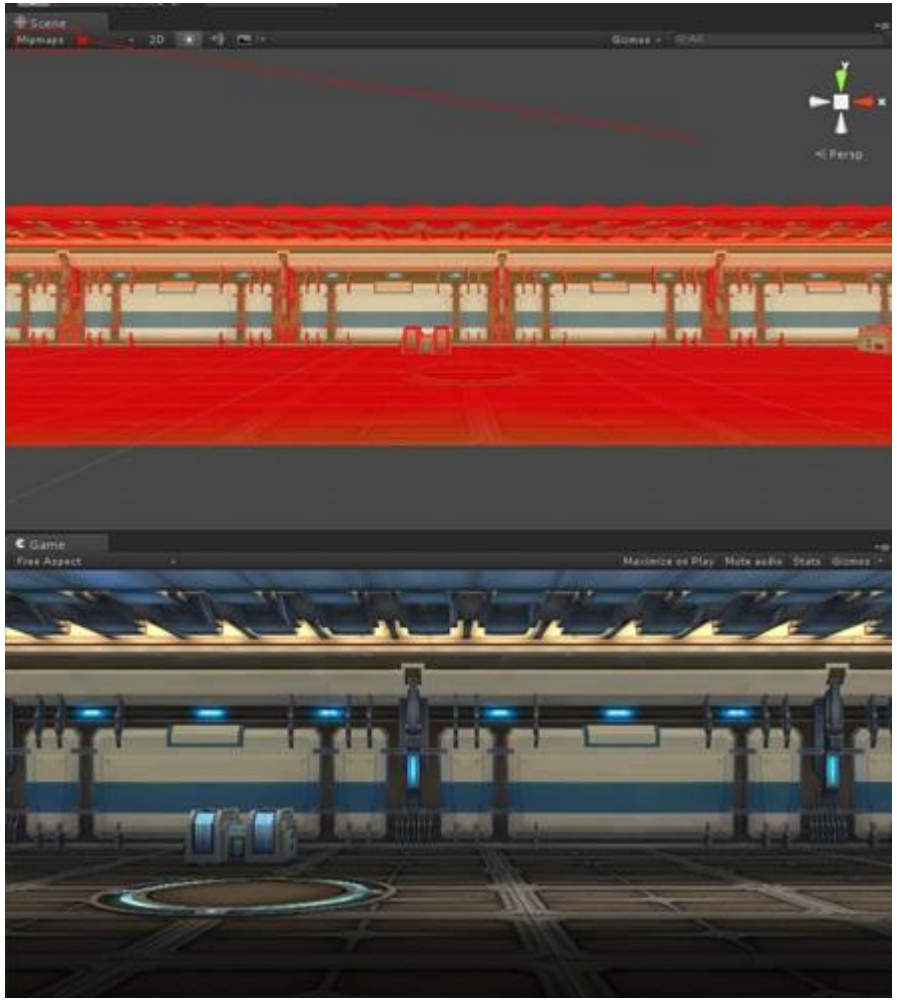
Generate Mip Maps会为同一张纹理创建出很多不同大小的小纹理，构成一个纹理金字塔。而在游戏中可以根据距离物体的远近，来动态选择使用哪一个纹理。

这是因为，在距离物体很远的时候，就算我们使用了非常精细的纹理，但肉眼也是分辨不出来的，这种时候完全可以使用更小、更模糊的纹理来代替，而这大量可以节省访问的像素的数目。但它的缺点是，由于需要为每一个纹理建立一个图像金字塔，因此它会需要占用更多的内存。

在我们的项目中，因为摄像机在Z轴方向是不动的，所以我们为了降低内存使用，没有使用MipMaps。

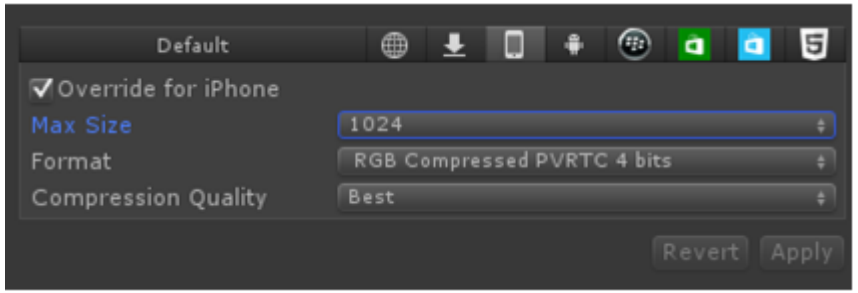
例如上面的例子，在勾选“Generate Mip Maps”前，内存占用是0.5M，而勾选了“Generate Mip Maps”后，就变成了0.7M。除了内存的占用以外，一些时候我们也不希望使用MipMaps，例如GUI纹理等。我们还可以在面板中查看生成的Mip Maps：

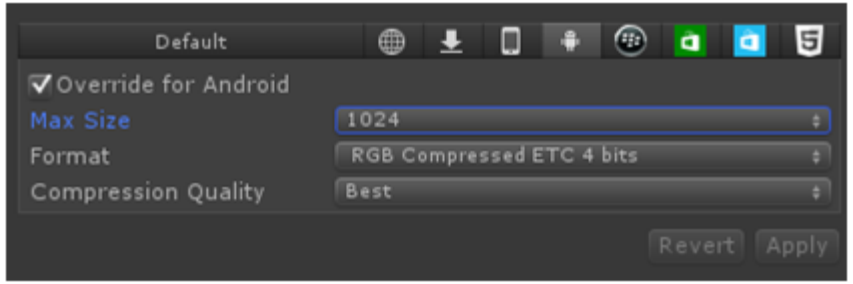
Unity中还提供了查看场景中物体的Mip Maps的使用情况。更确切的说是，展示了物体理想的纹理大小。其中红色表示这个物体可以使用更小的纹理，蓝色表示应该使用更大的纹理。

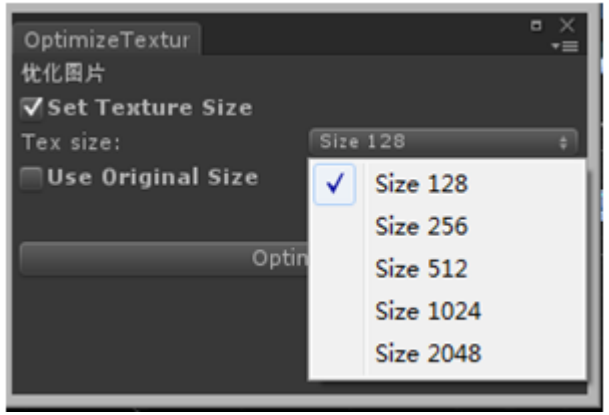


次世代手游美术资源的优化

带宽优化——优化纹理尺寸大小、压缩格式







纹理尺寸大小：
纹理尺寸大小通过MaxSize设置，它决定了纹理的长宽值，如果我们使用的纹理本身超过了这个最大值，Unity会对其进行缩小来满足这个条件。
这里特别说明一点，所有纹理的长宽比最好是正方形，而且长度值最好是2的整数幂，例如64、128、256、512、1024等。这是因为有很多优化策略只有在这种时候才可以发挥最大效用。

压缩格式：
压缩格式通过Format设置，Format负责纹理使用的压缩模式。通常选择这种自动模式就可以了，Unity会负责根据不同的平台来选择合适的压缩模式。而对于GUI类型的纹理，我们可以根据对画质的要求来选择是否进行压缩。我们还可以根据不同的机器来选择使用不同分辨率的纹理，以便让游戏在某些老机器上也可以运行。例如在Android平台上，我们的压缩模式会采用RGB Compressed ETC 4 Bits，在IOS平台上，我们采用的压缩模式是RGB Compressed PVRTC 4 Bits。

开发贴图自动优化工具：
关于贴图的优化，上面的一些操作，会让美术制作人员感到反感，我们做了一个很简洁的工具。它自动调整贴图size、format、Mip Maps选项等，只需在Project View里选择需要优化的帖图文件，点击按钮即可。