Fortnite 预告片

为快速工作流创建实时开发流程

关于中文翻译

- 动画行业相关流程术语中文习惯叫法并不统一，因此保留英文原文
- 部分Unreal引擎相关概念术语以及编辑器工具菜单沿用英文，不做翻译

目录

Fortnite预告片

为快速工作流创建实时开发流程

在2017年7月，Epic 发布了Fortnite，一个玩家需要拯救世界范围激烈风暴侵袭后幸存者的游戏。玩家可以利用搜集的材料建造建筑，组装武器来抵御和对抗怪物和敌人。至2017年8月，游戏已经有超过一百万的玩家

2016年底，为了准备游戏的发布，Epicgames 开始为Fortnite制作一段3分钟长的预告片。目标是制作一段具有离线渲染品质的实时渲染动画短片，并且在制作过程中可以即时互动和浏览。

这个短片的目的是展现游戏的趣味性和独特美术风格，介绍故事意图以及可玩性元素，比如收集，建造和防御。

图 1: Fortnite 短片场景

这个文档概述了Fortnite预告片使用虚幻引擎的制作过程，以及最大化协作和创意的同时最小化制作时间的流程

虚幻引擎最初是一个设计用来做游戏开发的工具，但也整合了许多动画制作流程中所需的工具-版本及资源管理，绑定，动画，编辑，实时预览，修改的发布等等。她的实时渲染能力为开发者提供了即时反馈，从而实现更快的迭代及更好的效果

流程开发

流程创建

在为预告片创建流程前，Epic的团队首先考虑到的是
目标以及工作量

这个片子包含了6 Sequences[ ] й 130个独立的镜头，以下是具体的目标

- 最终的影片需要能够在UE4里实时运行播放
- 帧率24
- 大概三分钟长度
- 实时渲染动画的视觉品质至少和预渲染一样高

预渲染 й 以下分列了实时渲染 任务

- 环境角色设计
- 故事板
- 使用什么工具（软硬件）
- Sequence[ ]拆分
- 资源的制作和收集
- 粗略运动布局设定
- 声音和对话的录制和制作
- 动作的捕捉或制作
- 多团队资源协同编辑
- 场景建筑物件等制作
- 根据Rough layout[粗略布局设定]创建动画
- 灯光
- 渲染
- 后期
- 在目标分辨率和帧率下测试输出
- 给其他资深团队审阅
- 最终输出

如何在工作流中组织这些任务以及选择使用什么工具来完成这些任务会对最终产品的品质，完成的效率或者说是否能够完成产品有重大的影响

我们的目标就是简化流程，制作高品质产品，减少制作时间（压力）。为此我们执行以Unreal Engine 4为中心工作流程

线性工作流和实时工作流的比较

片子制作的早期阶段，团队便寻求改进传统流程的最好方法

传统的线性动画工作流程就像装配线装配产品，任务是按顺序执行的；然而新的更整体化的流程为非线性及平行方式处理和发布数据做了优化

线性工作流的缺点：

- **修改限制.** 作为产品，片子需要被拆分成许多不同的需求，每个需求对最终的表现都至关重要。每次需求改进都可能涉及到要少许调整个别动画，一组灯光等等。在线性流程中这样的修改在后期很花时间，以至于有的时候不愿去做修改。修改即品质，也就是不得不品质对于时间做了妥协
- **后期.** 传统走线性流程的工作室一般片子会渲染输出各种图层以便后期在其他工具里做混合编辑。这样的确为最后效果提供了很大的自由，但也很费时费钱。不少后期都是外包给其他团队做。

为了最小化这些限制，我们需要为Fortnite预告片制定一个新的工作流。利用UE4为工具，开发一种实时工作流来避免线性工作流程中问题，最大化美术品质，同时保证制作过程平顺

- **交互型的创意过程.** 整个制作流程都是在UE4里呈现的，从资源的获取到动画，特效，灯光，编辑的过程，后期等等。评审的内容可以是其中的一块或者整个片子作为一个整体，来决定哪些需要修改。完了以后美术可以方便地在Seqencer里做迭代修改，修改的结果都是即时呈现。使得创意过程变得即时和可交互
- **方便编辑.** Sequencer 可以即时地创建，修改，录制或删除镜头，调整顺序等。而且它还是制作中的编辑中心以非线性的方式整合镜头，动画等等
- **更快速地输出.** 整个后期可以在UE4里完成，而不需要再输出到外部工具合成

UE4的实时渲染工作流需要一些方式来提高视觉品质，节省时间。一下列了一些Fortnite预告片中用到的一些选项：

- 重用已有游戏内的资源，提升品质，节省资源制作时间
- 利用ABC文件来输出面部动画提高品质
- 使用全动态灯光
- 短片中对象属性优先级可覆盖性调整，这样可逐镜头对灯光对象做覆盖性的参数调整

实时流程细看

利用UE4引擎的实时流程的创建是为了解决在传统动画开发中的各种问题，这些问题和美术品质，技术考量以及效率息息相关

以下是传统线性流程和实时流程各个组成上的对比表

	线性流程	实时流程
渲染方式	离线线性渲染	实时渲染
主要制作工具	各种DCC工具	Unreal Engine
工作流	部门很少平行工作	更多平行工作
数据组织管理	分散	中心化
开发模式	拉版本	按需获取及推送
命名规则	严格的	比较自由
版本资源控制	手动	自动管理更新迭代
输出形式	分层	直接输出最终画面
资源要求	可以非常高精度	需要优化

表格 1: 线性和实时工作流程比较

渲染方式

直到最近离线预渲染和实时渲染在最终品质上的差别还是可以察觉到的。最近几年随着软硬件的提升，实时渲染的改进，如果处理的好，在视觉品质上已经可以达到离线渲染的品质

传统流程在不同阶段做多次多批次的渲染输出，实时渲染就不需要多阶段多批次，所有阶段批次的渲染就融合到一个新的单一流程中

传统流程中需要做些修改需要大量时间重新渲染，然后才能有视觉反馈；新流程修改实时可见并应用，立刻可以收到反馈，并继续接下去的工作

实时渲染的所做立刻所得的优势消除了传统流程中由于渲染时长和多阶段多批次所产生的各种问题

主要的创作工具(DCC 对比 Unreal Engine)

DCC (数字内容创建) 软件工具集如Maya, 3ds Max, Cinema 4D 和 Blender等等一般情况下都是用来在线性流程下做其特定工作，比如用来建模，贴图，绑定，动画；但他们不具有整合及管理所有这些数据功能

然而UE4可以用来把所有这些在一个统一的框架下做整合及管理。美术可以在其上并行工作

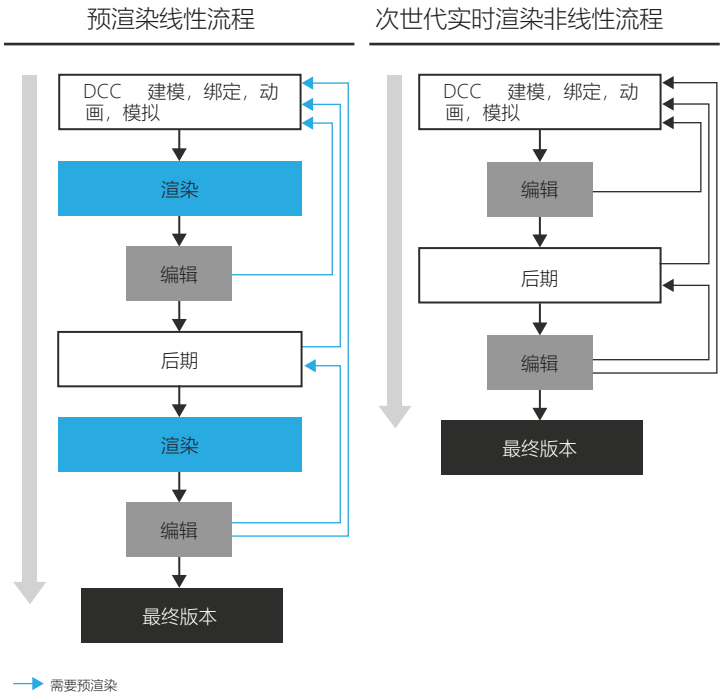


图 2: 使用预渲染和实时渲染的流程比较图

工作流

尽可能多并行化工作

为实现这个目的有两个关键的因素。第一利用UE4的Level[关卡]和SubLevel[子关卡]功能。管理者可以把不同性质的内容分到不同关卡中，那样不同部门（工种）就可以单独自由地工作于其相关关卡而不会影响其他部门以及主关卡的工作。当某部门的修改交了，就自动推送更新，所有工作在项目上的成员都及时看到更新。在这个过程中不会影响各个部门手头的工作，以达到并行化工作。比如灯光关卡，场景关卡，角色动画关卡等等

第二个因素使得并行工作变得可能就是实时渲染的即时性，避免了长时间等待结果，团队可以及时看到效果即时做出调整

高效的数据管理也是并行化工作流的前提因素，这可以避免很多不同部门，比如美术和程序之间对于数据控制权的相互等待，甚至浪费时间编辑过时的数据发生不得不重做的情况。

数据组织

传统流程中文件拥有不同的命名规则被存储在不同的文件服务器和地点，这对于文件组织结构以及文件指向脚本的设计需要很小心。通常都有个叫Home的主目录，但也会引起疑惑

实时工作流中，数据被组织到一起。UE利用UnrealGameSync（UGS）一个UE外的前端图形化工具来同步和编译UE项目

被批准上传的数据被统一推送到这个数据中心，其他使用者就可以利用一个统一单一的界面来访问

虽然这个系统是为游戏开发设计的，但这种数据中心化统一管理的方式也很适合动画片制作

开发模式

在产品开发中，美术在工作中经常性地更新已有文件，或者覆盖老文件，创建新文件；同时其他美术需要即时知道哪些文件被更新，锁定，并且得到更新过的文件

为了实现这个工作流，任何这个流程中的文件管理系统必须设计符合以下需求：

- 当一个美术在一个特定文件上工作时，需要一个机制来告诉其他美术这个文件正在被修改，谁在修改，以避免多个开发者在同一个文件做重复劳动
- 当修改完成，并上传。文件已经被修改这个信息需要被记录下来。（别人就可以了解这个文件已被更新）
- 其他如果需要使用这些文件的美术需要能够方便地得到这些文件的历史版本和最新版本

线性动画产品一般是依靠拉版本的数据获取形式。比如美术需要哪些文件来构建场景，需要的数据文件往往通过一个手动维护的更新列表，或者一些有限制的自定义版本控制脚本来实现

手动的方式依靠美术和监管以口头和书面记录改动的方式来跟踪变动，这些信息被存储在一个独立或者说和制作工具分离的表单里。这种手动跟踪变动并更新信息的方式很费时，也容易出错，完全依赖于美术和监管记录的及时性和准确性。手动版本控制对于小团队，小项目可能没问题，但对于大团队很快就会变得错误百出。

大工作室往往会写些自己的脚本为美术使用的工具（DCC，比如Max Maya等）来增加文件管理和跟踪的功能。虽然优于手动方式但也会带来一些问题：

- 新脚本必须是逐DCC工具单独写，而且需要随着DCC的更新而更新
- DCC的脚本语言对于写一些其功能相关的功能比较方便。比如建模，动画等，但对于版本的管理控制比如需要锁定文件编辑这种机制的实现并不友好。如果说脚本程序员可以通过检测文件名的方式控制访问，但一旦美术不遵守命名规则，这个方式就不起效了

简单的说，基于DCC脚本实现的版本控制会好于手动方式，但有诸多限制，也容易出错

一些工作室使用Shotgun（一个跟踪和评审工具集）作为部分的版本控制方法。然而Shotgun虽然擅长于产品进度的跟踪，但对于资源和版本的管理有诸多限制

在这个过程中，严格的汇报和评审规则经常是和创意过程反其道而行的。比如一个资源的改进行为因为美术疏于记录而被错过，没有被整合到最终版本中。（顺便说一下，这也不能怪美术，因为他们是因为其艺术才能而被雇佣的，而不是管理技能。修改后每次的记录，报告整理需要有一定的项目管理技能）

作为对比，实时流程中使用的获取和推送系统利用了ACID数据库交换原则来管理文件的访问和修改

利用这套系统，上面提到的三个需求就可以完全满足，避免了手动或者基于DCC脚本文件管理方案带来的各种限制和问题

版本和源代码管理

相对于传统线性工作流通常以来手动，VersionlessMasters[1]或者复杂的版本选择脚本而言，实时工作流使用单一的界面来控制所有的资源文件包括场景和引擎的代码

虚幻引擎本身就是一个可自定义工具，程序员可以为其增加功能来改进效率，实现自定义行为。除了为资源管理提供可靠的获取和推送系统，UGS还提供了引擎代码修改的同步和分发工具

所有资源及代码控制是通过Perforce P4V来完成，这是个行业领先的版本管理系统。UGS扮演了其封装角色，提供了P4V和UE的无缝数据管理体验

UGS是设计用来在设计师，程序员，美术等之间降低开销，加速迭代

UGS特定的功能包括：

- 新增修改能够及时分发到整个库中，从而允许其他用户同步其修改，并在本地编译代码匹配其修改。用户可以为其他开发者在其修改中增加说明或者标记版本好坏
- 编译过的编辑器版本能够被打包，自动同步并解压缩给需要的美术
- 引擎本身相关文件和美术资源可以分开同步。这样的可选性可以降低不需要的同步时间
- 程序员个人如果正在修复一个不工作的版本时可以做标记来告知团队
- 文件活跃日志和递交修改列表能够一起被显示在一个列表里
- 版本编译可以自定义使用制定的工具来做

[1] Versionless Master是一个特定版本的复制，或者是这个特定版本的链接。它的意义在于得了某个Versionless Master版本确保了美术每次打开项目所有的数据文件都是这个版本相关的，而不会总是最新版本。这可以来做一些Staging的版本，比如关键节点版本。

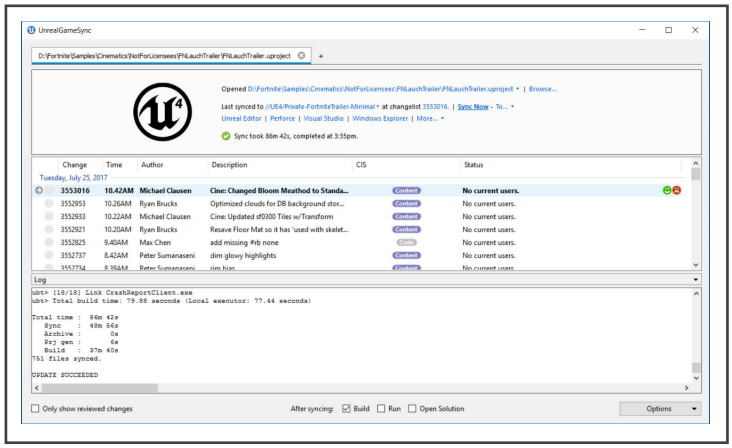


图 3: 使用 UnrealGameSync源代码管理

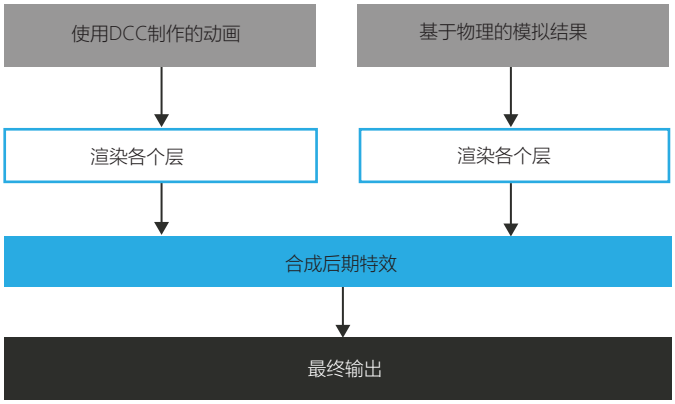
输出目标

传统走线性流程的工作室一般片子会渲染输出各种图层以便后期在其他工具里做混合编辑。这样的确为最后效果提供了很大的自由，但也很费时费钱。

作为比较，实时流程中动画，特效，编辑，后期等等都被汇总呈现在一个单一界面里。无论是上游的资源修改，还是下游的参数调整，结果都会被即时呈现在引擎编辑界面里

UrenalEgnine被设计为输出最终结果。实时特性使得在Sequencer中的交互及修改不会影响上游部门的工作。整个基于物理模拟过程及后期都直接在引擎中实现，从而减少外部合成的需求

传统线性后期方式



Final Pixel Output Approach

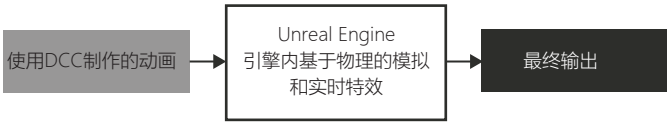


图4: 一次性最终输出方式和传统方式在后期制作上的区别

制作中的概念/术语

动画电影制作借鉴了很多真人电影制作的概念。其中之一是把产品开发过程分成前期制作，中期制作和后期制作三个阶段。对于真人电影这个拆分以实际拍摄过程为界，拍摄过程是中期制作阶段，之前是前期，之后是后期

其中一个需要把三个阶段清晰的区分开的原因是预算开销考虑。比如前期没有准备好的情况下，演员就需要等待，浪费金钱。同样如果在后期需要重新组织和拍摄一些镜头也会极大增加开销

同样原因，动画电影也遵循这个明确的分法，前，中，后期。而且以动画制作为中心的中期制作阶段。设定角色是前期的工作，拼地图及完成是中期制作的一部分。任何不遵循这个线性结构的修改都会增加开销和时间

虽然这个流程可以完成成品，但给团队也增加了很大的压力来确保当前阶段的工作的确是"完成了"，才能继续接下去的阶段

反过来，实时流程就没这么死板需要顺着流程顺序来做，而是并行，一些原本是中期或者后期的工作如果需要可以放到更早的时间来做。实时渲染以及数据储存和获取的统一化途径使得这些工作可以按需在时间上做前后调整，来节约时间和提高成品质量

Sequence[幕]拆分

另外一个动画电影从真人电影制作借鉴的概念是Sequence[幕]拆分

在开始拍摄前，真人电影的导演为了助于组织故事流以及拍摄细节会把故事脚本拆分成一系列更小的结构。这种做法也从真人电影很好的应用到了动画电影制作过程中

Sequence[幕] - 故事脚本会根据一个符合逻辑的时间间隔被拆分成大块的幕。通常一幕的拆分发生在一个地点的变换或者次情节的转换时

Beat[节拍]- 每幕又会被拆分成更小的块，叫做节拍。每一节拍可以是一组连续完整的行为表演，分界可以是这组表演的结束，切换到一个不同的环境或者视角，或者一系列针对一幕的自然拆分

Shot[镜头] - 每个Beat又会拆分为独立的Shots，一个Shot包括一个特定的摄像机设置（静态或者移动）和其中一部分的脚本表现。在不同摄像机中的同一个脚本表现属于不同的Shot，在同一摄像机中的不同脚本表现也属于不同的Shot

Scene[场景]- 表演发生的环境。同一场景可以使用在许多不同的Shot和Sequence中

Take[镜次] - 同个Shot的多个版本复制。和其复制的Shot拥有一样的摄像机设置，和拍摄的表演内容。但做了一定的调整表现不同的运动或者情感等。这个经常用来做几个Take来让导演做选择，或者尝试

Cut [剪辑]-Cut可以有多个意思。在真人电影拍摄中，通常一个cut指编辑阶段一个镜头开始起始到被剪断之间。因为一个镜头往往包含不需要的部分，比如导演喊action，演员进入屏幕等。一个被剪辑过的shot称为一个cut，这个剪辑的行为叫做cutting[剪辑]。

Cut也可指一个编辑过的特定版本。比如针对不同观众同一部片子的G-rated和R-rated cut版本，还比如那些只在DVD里附带的导演剪辑版本

拆分Sequence(幕)是需要在前期阶段执行的一个管理任务，来帮助组织工作

前期制作步骤

动画项目的前期步骤按顺序基本包括以下

Design[设定] - 项目最开始美术设计环境，角色，武器，场景物件等

Story Board [故事版]- 接下来是故事版，把整个故事使用一些列图片的形式呈现出来。一个故事版可以像一本漫画书一样，每张图片表现一个单独的表演，可以附带一些对话来说明情节。故事版也常常作为一些手绘的卡片贴在办公室墙上，以便随时可以移动和评审，来帮助完善故事。

Story Reel [故事样片]- 使用一些编辑工具，把故事版的图片整到一起，按设计的时间线节奏播放，可以加入一些声音，对话来帮助表达。用来更好的评审故事，也可以叫做样片

Sequence [故事写作概念：幕]拆分 - 前面已经提到，把故事拆分成幕和镜头来组织工作

Rough Layout [粗设]- 粗设是一个粗略的布局设定短片，使用近似的动画，声音等等来替代Story Reel中的图片。在制作后期每个RoughLayout中的章节将会被最终品质的动画序列帧替换掉而输出成品

每个步骤—设计，故事版，样片，粗设，在往下个阶段进行前都需要被导演和团队成员一起评审过

线性流程 与 实时流程 前期制作的比较

虽然两者在前期制作上很多流程是保持一致的，但也有一些不一样的地方使得实时流程更加高效

线性流中整个设计必须在前期制作阶段完成。实时流程中，粗设步骤完全可以贯穿于整个产品制作的周期中，设计可以和制作本身并行，随时可以实现改进的设计。

线性流中，粗设是通过手动key动画的方式经过多轮的评审和调整定下来的。Seqence[幕]的拆分必须在粗设之前完成，这样关键帧动画的工作才能交给美术去做。另外更多的关键帧需要在中期制作中完成

实时流中，动捕数据可以用来制作一个交互性的粗设。这个过程是实时被评审和敲定的。脚本拆分为Sequence[幕]会在mocap[动捕]前完成，但拆分成Shot[镜头]的工作可以在导演决定使用哪些mocap后做。这个方式还包括提供已经被敲定的关键帧动画，来节省在中期的评审时间。

Fortnite 流程

一系列我们使用到的工具及其应用范围，所有这些和Unreal Engine一起被纳入产品制作流程

- Autodesk® Maya® – 低模，UV，动画
- Autodesk 3ds Max® - 角色和硬边物体建模
- Pixologic ZBrush® - 雕细节
- Modo™ - 重拓扑工具
- Adobe® Photoshop® and Allegorithmic® Substance Painter - 贴图绘制
- Allegorithmic® Substance Designer - 材质和贴图制作
- xNormal™ - 场景物件的法线烘培
- Autodesk Motionbuilder® - 动捕数据处理（数据来自Vicon动捕硬件）
- Blender® - 动态破碎模拟
- Adobe Premiere® – 最终动画序列图的编辑
- Autodesk Shotgun™ - 制作过程跟踪管理
- Vicon® Blade™ - 动画捕捉

数据转换文件格式

- Autodesk FBX® - 场景模型，肢体绑定/动画
- Alembic - 面部绑定/动画

数据序列化和整合的工具:

Unreal Game Sync (UGS) - 资源整合及版本控制

Sequencer - 作为一个影片的制作中心，在一个非线性的编辑器中整合各个部门动画，场景等资源使其序列化

Sequencer代替了很多外部程序，提供了对影片的编辑工具，动画控制等等。UGS和Sequencer一起在UE4内建立了一个类似"工作室"的氛围

接下去的图表展示了为Fortnite 预告片开发的制作流程

尽管Sequencer中就可以实时的播放片子，每天可以在指定的屏幕上一起评审进程。但最终我们还是每晚都把整个片子渲染成1080P的PNG文件（最终需求分辨率），为创意和制作团队评审。这样可以让美术和指导在细节上评审每一个镜头，给出记录和反馈意见。编辑团队会在Premiere中整合这些PNG序列，团队就可以对片子中每个镜头的时长和节奏做出评审。时间上，由于主要因为文件存盘时间，渲染/存盘过程一般会花费一个小时，合成过程大概2-4个小时

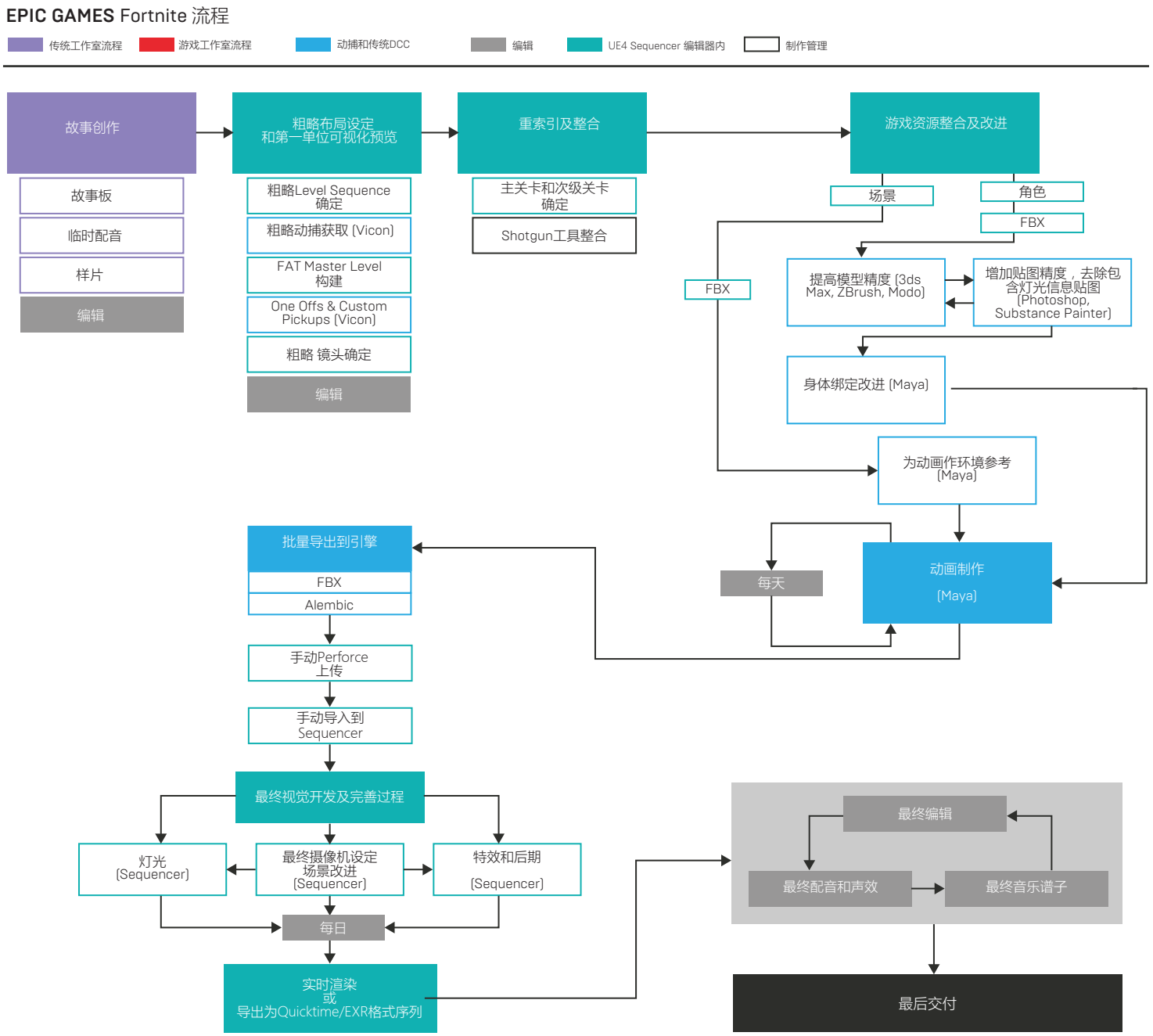


图 5: Fortnite短片流程图

前期制作

前期制作

对于Fortnite的前期制作，我们的优势是已经有了一些游戏内资源和动画，绑定都可以重复利用。 所以我们省掉了概念图设计阶段，直接进入故事和布局设计

故事创作

Fortnite也是从传统故事板开始画起。然后把这些故事板图片扫描导入Premiere Pro里串成一个故事样片。故事样片用来作为之后粗设在分镜头和节奏上的参考。故事板概念美术还会绘制气氛图来表现图片的细节场景气氛等

图 6: 在短片中使用的一些游戏角色资源

图 7: 故事板

Sequence[幕]的拆分

前期团队把Fortnite影片看作一个3分钟左右的短片，把它拆分成相应的幕，节拍，镜头

动画发生在三个场景中，每个场景发生两组不同的连续行为活动，这样使得这些活动和场景的切换点成为了逻辑上的拆分点把整个动画拆分为6个幕

在这个时候幕没有被再拆分成更小的节拍和镜头。之后更细的拆分会在粗设的过程中自然的拟定，这是随着初步动作捕捉被设计完成而同时完成的

Rough Layout[粗设]

为了加快粗设的过程，Fortnite团队绕开了传统的粗设方式，而是通过利用动画捕捉的过程来拟定出大致[blockout[2]]的动作，为导演和摄影师更多的创意空间，节省产品制作周期

在传统的粗设过程中，布局主管拿到Story reel[故事样片]，然后许多3D美术开始在不同的软件工具利用逐步关键帧（关键帧间变化较大，并且之间不插值，可理解成定格动画）的方式创建粗略[blockout[2]]的角色动画以及架设摄像机位置。美术通常要花费数日，甚至数周的时间来构建粗设镜头内容，然后扔回给编辑整合到幕中来替代之前故事样片中的故事板图片，然后让导演评审，如果有任何需要修改需要再次发回美术

使用UE4引擎来制作粗设的方式提供了缩短这个苦力活的过程。使用Seqencer，动画捕捉数据可以很快的被导入，预览，编辑，重新捕捉，直到导演满意为止。创建一个被导演通过的粗设短片的过程从几天/周缩短到了几小时

Fortnite团队把这种方式叫做First Unit Previs[第一单位可视化预览]；这种称呼结合了电影行业对主摄像团队称呼的术语（First unit[第一单位]）和这种在实际拍摄前对复杂动作的可视化方式（可视化预览）

所有动作都是使用Vicon Blade通过MotionBuilder捕捉的，然后在Maya里修整后导入到UE4的Seqencer中

First Unit Previs[第一单位可视化预览]

通过结合应用各种技术方式，第一单位可视化预览加速了传统手key动画创建粗设的过程。因为导演和摄影师有更多的自由度亲自去概念化心中的想法，设计过程变得更像拍电影的方式

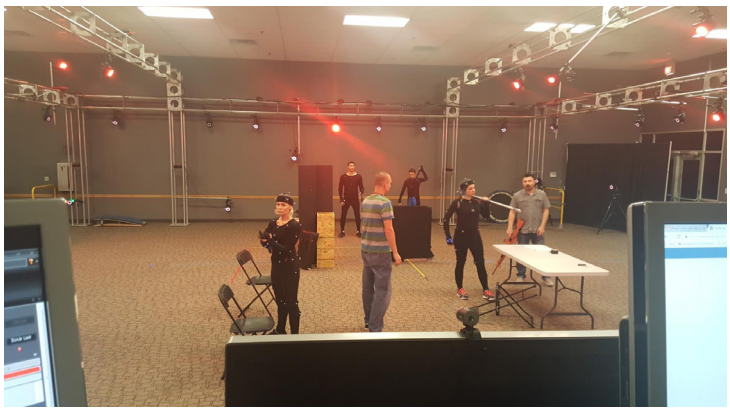


图 8: 动捕阶段

在动捕前，先有个大概的场景布局已经创建好作为占位，比如地面，建筑等。Fortnite影片的故事版里要求有一个破烂的汉堡店的房子，因此我们使用游戏中汉堡店美术资源，做了破损处理使用在这里，帮助交代情节

block[2] 或 blockout[2]：为了定义演员表演中的移动。这是一个来自早期真人电影制作的术语，利用一些木头的占位符或者替代物的移动来代表演员的移动方位，每块木头代表一个演员。这种定义演员移动的行为和结果都被叫做占位

每个场景都包括一些低精度的游戏角色模型和绑定来测试动捕的动作。一开始就有这些资源就让动画捕捉的开销降到最低。

在Fortnite第一单位可视化预览阶段，演员根据脚本表演，动作被捕捉成多个长动画版本。每个版本的表演动作通过应用到绑定的虚拟角色上放在虚拟场景里被预览评审 。

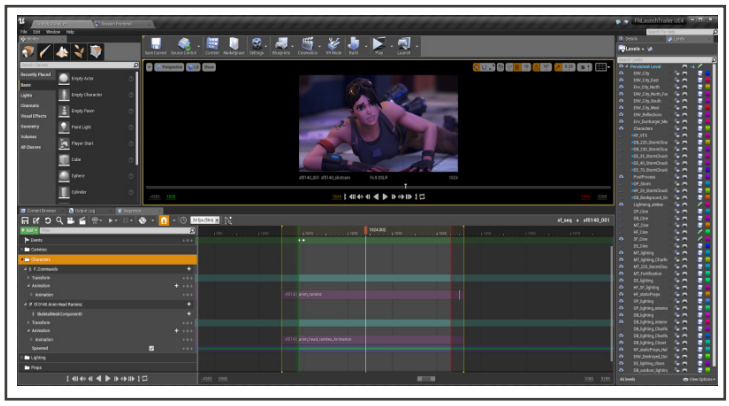


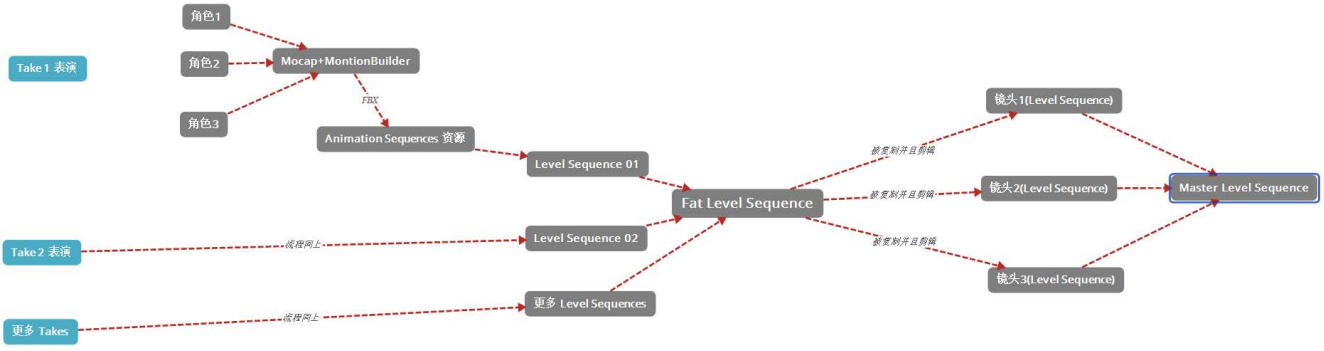
图 9: 动捕数据被应用于游戏角色资源来进行分析评审

通过分析这些表演来决定哪个最合适表现影片。最合适的那段长动画将作为交互式粗设的基础，之后导演和其他相关的成员可以评审，重新捕捉，截取动作直到满意

Sequencer 步骤

每幕被导演认可的捕捉表演通过Motionbuild后以FBX格式导入引擎，应用到骨骼模型上，作为动画轨迹放置到一个单独的Level Sequence中。
这幕长动画的其他版本（Takes）Level Sequence又被组织到一个高级别的Level Sequence中，这个Level Sequence被从称为Fat Sequence。
Fat Sequence在这个阶段不断增长，又会被复制后剪辑成不同的Level Sequence（镜头级别的level）

Epic在动捕过程中同时捕捉摄像机位置，但如果没有被捕捉，摄像机也可以在UE4里使用Cine摄像机来放置



清理Rough Layout[粗设]

在动捕和导演评审过程中，为了让这个过程尽快结束，关卡和幕被很快宽松的创建出来。在这个快速推进的过程中很少关注精细的管理组织任务

一段初步的设计被通过了，接下去需要一个重组的阶段，就是重新标记镜头，重定义关卡以及把幕整合成更有序的方式

定义Level[关卡]

在UE中你能看到交互的物体都存在于关卡中。一个关卡由一些列的模型，体，灯光，脚本程序组合一起来给玩家期望的视觉体验。UE4中的关卡有大有小，可以是非常巨大的地形到只有很少的一些物件。

关卡是在Level[关卡]编辑器里创建和组织的。默认情况，关卡编辑器里总是有一个母关卡，她是幕的基础关卡，任何直接放在这个关卡中的物件将会和所有次关卡一起呈现

一般母关卡可以是空关卡，然后在它下面挂一些子关卡，这些关卡可以是场景，特定的基于幕或者镜头相关的特效，跨不同幕和镜头的角色，场景布置，物件，灯光组架设，蓝图，后期等等。

子关卡可以被看作其他DCC软件里层的概念。当一个子关卡被激活为当前关卡时，所有的编辑工作都会被保存到这个子关卡中。另外子关卡还可以在任何时候被隐藏或显示

Fortnite关卡组织包括

主关卡

- 场景子关卡
- 角色子关卡
- Cine 子关卡
- 灯光子关卡
- 蓝图子关卡[3]
- 后期子关卡

了解更多关于如何定义子关卡，在关卡间增加移动物件，设置可见性，加卸载方式的信息请参考UE4关于多关卡管理的文档

[3] 在Unreal Engine中，蓝图是一种使用基于节点以可视化工具编写的脚本容器。使用蓝图可以不需要编写代码就能控制游戏内或者影片行为；比如一个蓝图可以实现当角色走近一扇门时触发开门的行为

Level Sequences

关卡中元素的运动和动画是通过Level Sequence这种由许多动画轨迹组成的对象来承载的 。一条动画轨迹可以是角色的骨骼动画，位移/放缩/旋转动画，声音变化，或任何独立元素的可动画属性。一条轨迹甚至也可以包含一整个镜头

在UE4里，一个Level Sequence被当作一个关卡中的一个单独对象。一个Level　Sequence可以被比作一个电影场景各个元素动画，运动及组织管理的容器及编辑器。这个编辑器类容器被作为一个对象处理

图 10: Add Level Sequence 选项

因为Level Sequence是独立的资源，它也可以被嵌套到另一个Level Sequence中

默认一个Level Sequence是自上而下构建的，因此低级别的Level Sequence或者Track中的属性动画会覆盖高级别的。这样符合传统电影制作者流程习惯，就是在镜头级别做的修改直接覆盖掉幕级别的统一属性修改

Fortnite Level Sequences

Fortnite的Level Sequences遵循以下结构

一个最高级别的Level Sequence来容纳整个短片

- 第一幕的Level Sequence
  - Shot[镜头] A Level Sequence
    - 各个这个镜头内需要的动画轨迹(包括摄像机动画轨迹，灯光动画轨迹，角色，这个镜头内需要的音效等)
  - 镜头B Level Sequence
    - 同上，镜头B内需要的各个动画轨迹
  - 更多的镜头 Level Sequence
  - 第一幕Level Sequence的可见性动画轨迹
- 第二幕，第三幕……次级结构同上
- 主音效轨迹
- 各个Seqences间或者整个短片头尾的融合特效轨迹
- 最高级别关卡可见性轨迹

顶级Level Sequence和其他次级别的Level Seuqnece一样也有关卡可见性轨迹，另外加上其中的声音和融合可以从最高级别来控制整个影片

以下是Fortnite短片Level Sequences的拆分示意图。其中第一幕Level Sequence被继续拆分演示，其他幕Level Seuqence类似，不再做具体分解

如下：顶级Level Sequence被命名为Launch Trailer 包含了这个短片所有的6幕，每一幕使用一个独立的Level Sequence来容纳。每幕Level Sequence中又容纳了此幕所需的多个镜头 Level Sequence，每个镜头Level Sequence中又包含这个镜头所需的摄像机动画轨迹，角色骨骼动画轨迹，特效轨迹，灯光设置轨迹等等

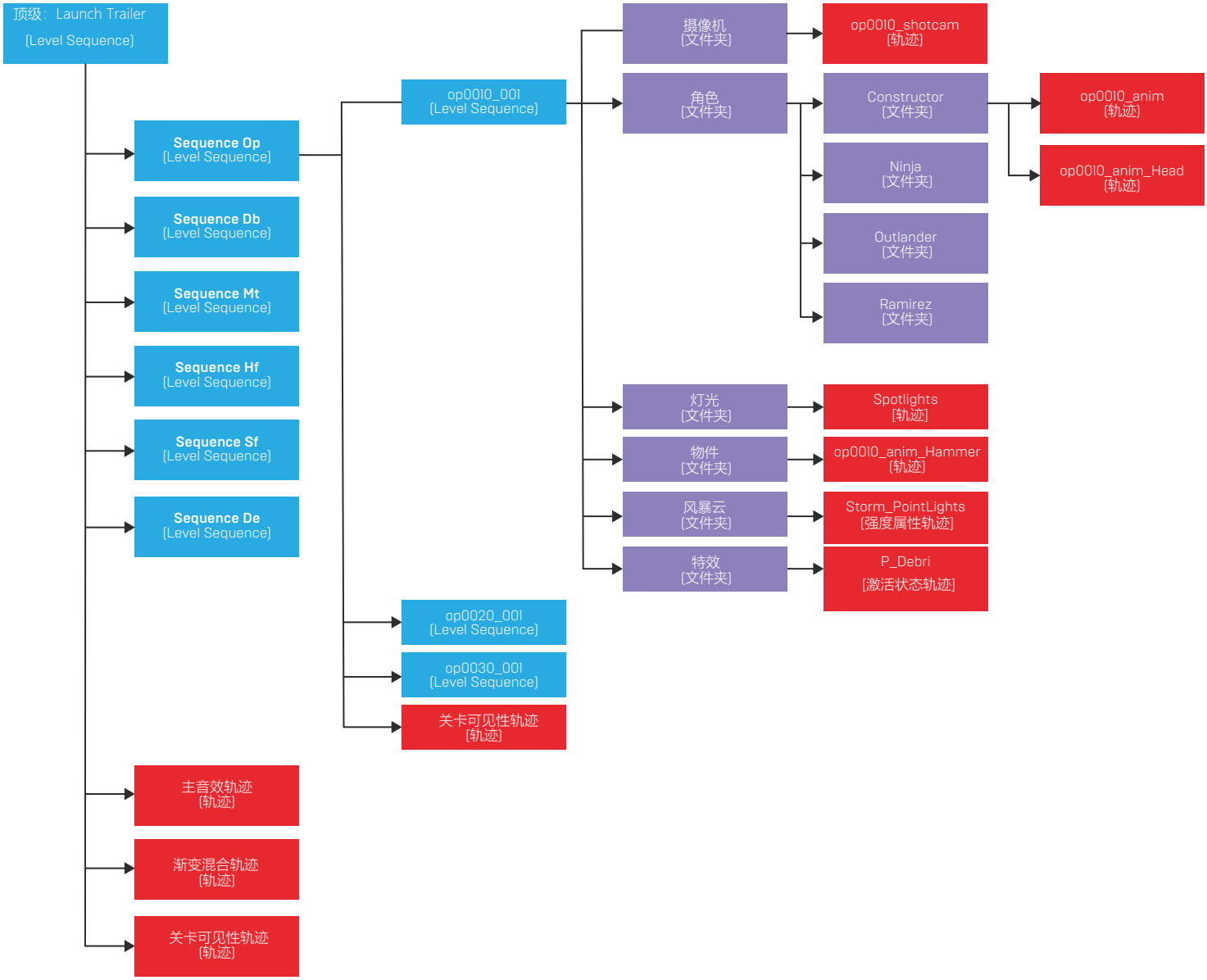


图 11: Fortnite短片Level Sequence及其动画轨迹层级结构图

文件和文件夹的组织管理

在Sequence编辑器中可以创建文件夹来组织管理轨迹

同一级别的同类型轨迹会被放置到一个文件夹下，如上图在镜头op0010_001这个镜头Level Sequence中为所有角色动画轨迹建立了一个叫Characters(角色)的文件夹，在这个文件夹中又分别为每个Character建立了相应的文件夹分别来容纳其所有角色动画轨迹，比如Constructor这个角色文件夹中容纳了它的头部和身体骨骼动画轨迹

命名规则

- 顶级Level Sequence: *Launch Trailer*
- 幕Level Sequence: *OP_Seq, HF_Seq*, 等.
- 镜头Level Sequence: *OP_0010, OP_0020*, 等.

命名规则使用字母缩写可以更加容易地在UE4里管理这些幕和镜头的Level Sequence

OP_Seq: OP为幕名缩写，Seq代表其级别是幕

OP_0010：OP为幕名缩写，0010是第几个Shot(镜头，一般4位数；后面还可以再跟_XX, XX代表这个镜头的不同Takes[镜次](上面提到了是一个镜头复制后的少许调整版本；比如OP_0010_02代表OP这幕第0010个镜头第02Take

关卡可见性

关卡可见性在UE里是很重要的概念。上面提到了顶级Level Sequence和Shot(镜头 Level Sequence都有其各自的关卡可见性动画轨迹

关卡的显示与否直接和这些可见性动画轨迹相关联。当关卡是可见时，关卡里的所有元素，包括场景，灯光，物件，特效后期等等都被可见，当然还包括这个关卡中的Shot[镜头] Level Sequence对象

尽管关卡可见性轨迹可以被添加到任何Level Sequence中，但Epic选择了只在顶级和幕级别的Level Sequence中添加。通常情况下，Shot[镜头] Level Sequence中并不需要添加，因为其关卡在特定Shot中总是要被显示。如果有在一个镜头中一些元素需要被显示和隐藏的需求，可以通过对物件本身可见性属性做动画轨迹来实现

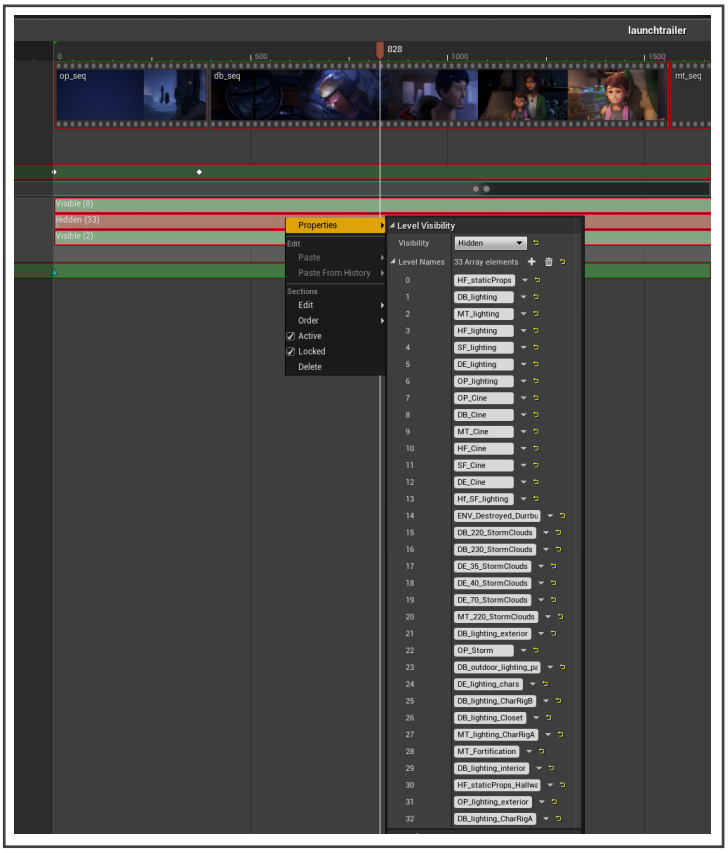


Figure 12: 当前幕的关卡可见性

事实上Fortnite短片里在镜头Level Sequence中也使用到了非常少量的关卡可见性轨迹，主要用来隐藏在当前镜头无法看到或会影响到当前镜头的对象，增加效率
对于关卡可见性轨迹的操作总体上思路是从高级别Level Sequence到低级别Level Sequence逐步做加法；即在高级别仅显示必须总是可见的对象关卡，在低级别逐渐添加需要的关卡

关卡管理

要了解一个资源属于哪个关卡可以在World Outline窗口中右上角切换到Level标签来查看

如果Sequencer编辑器窗口是打开的，World Outline中也会显示物件和哪个Sequencer有关联。World Outline以层级组织方式像是场景中所有的对象

当播放顶级 Level Sequence时，World Outline中的物件一会消失一会出现，这即反映了关卡可见性状态的动态切换

更多关于Level Sequence以及Sequencer 编辑器的信息参见Unreal Engine文档Sequencer Overview部分

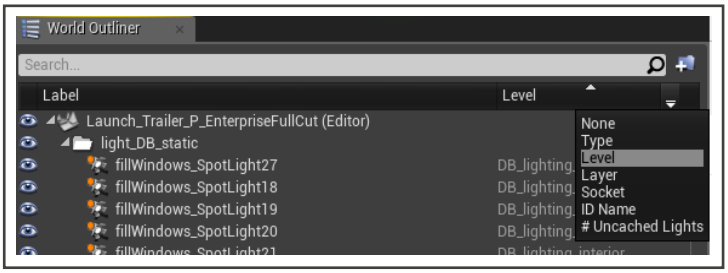


图 13: World Outliner

中期制作阶段

中期制作

拼场景

当使用外部DCC工具制作的模型，动画，特效，等等完成后，需要在UE4中整合这些资源。不同于传统CG流程在DCC中整合这些资源，我们需要在UE4中最终化和拼接这些资源。资源导入UE4后美术可以分别在动画，角色，场景上同时做调整和最终化的编辑

我们通过两种格式的数据来导入DCC制作的资源

- FBX – 模型，动画绑定
- Alembic – 非常复杂无法使用骨骼动画表现的点动画

Alembic和FBX存储不同类型数据。UE中整合的Alembic存储了烘培的点位置信息，也可以看作一系列的Morph Target及相应的绑定信息。而FBX保留了模型及其绑定蒙皮设定。即通过FBX保存的绑定可以回到DCC中再编辑，然而Alembic中的绑定无法导出再做修改了

对游戏资源的改进

鉴于Fortnite游戏和短片之间巨大品质改进的需求，Epic需要为最终短片的品质改进游戏资源，如模型，贴图，绑定等等。同时还要保持内存使用上的轻量化以便可以实时播放

动画测试

首先测试之前提到的在第一单位可视化预览阶段动画捕捉的数据在游戏模型和绑定上是否工作良好。这些原始的绑定是在ART工具里完成，ART是Maya的一个动画和绑定插件，里面有很多绑定的便利功能，模型通过ART绑定完毕后最终导入UE4工作

经过预览测试，团队很快发现有两方面需要改经

- **肢体绑定**– 模型的精度需要增加来提供模型细节点变形动画的可能性。现有游戏内模型的绑定可以提供基本的短片角色绑定需求，但还需要在ART里加强
- **脸部绑定** – 现有的脸部模型和绑定对于短片动画的需求是不够的。除了要重建更高精度的角色头部模型以外还需要使用Maya自带的绑定工具做脸部绑定，然后导出为Alembic格式来支持其丰富的点变形动画

模型精度

对于Fortnite短片，Epic决定使用比游戏更高精度的模型，不仅仅是角色还有场景环境物件。更高精度的模型增加动画表现，在加面的同时也带来一些其他的考量

- 工具可以自动加面，但新增加的多边形必须手动调整使其更好地表现结构和动画
- 加面的同时也需要同步增加贴图分辨率
- 增加精度的同时会增加回放动画时内存的开销。必须非常谨慎的增加精度，以免由于超出内存无法回放
- 角色绑定需要同时调整甚至重做来匹配更高精度的模型及增加动画真实性
- 高精度角色在短片中实时的变形动画和游戏里的非常不同，这意味着低模上的绑定和关键帧也需要被调整

图14: 拆分角色的头，肢体，服饰

因为对于角色头部新的绑定和动画方式的需要，头部模型网格线被布置成高精度通用拓扑结构。尽管每个角色头部张的不同，但网格拓扑结构是类似的。这种不同角色类似网格拓扑结构方式缩短了绑定和动画的时间，因为类似绑定可以被通用到所有角色头部上

每个角色头部模型大约有185,000个三角面

建模

为创建Fortnite短片中的模型，我们把游戏中使用的模型导出到Max, ZBrush, Modo 和 Maya进行接下去的工作

角色模型

当开始角色模型调整时，美术先把每个角色模型拆分成许多独立的模块。技术动画师会建议怎么拆对于今后的绑定和动画最有利

然后增加所有身体模块的模型精度以便为之后更好的点变形动画和物理计算做准备

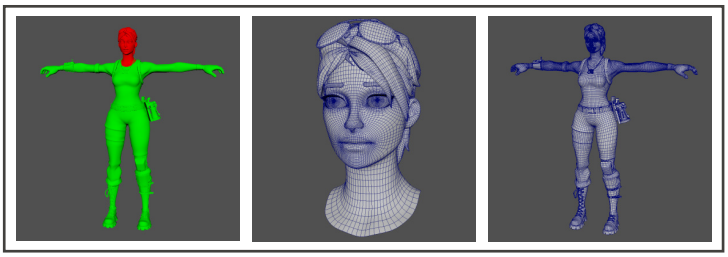


图 15: 高精度角色身体和头部模型

场景模型

目标也是在Fortnite已有资源的基础上改进及扩展，但同时要保证实时运行效率

对于场景物件，我们先拿一些简单的小物件，增加其精度（模型面数），调整满意后作为最终精度的范本和标准来为其他所有物件改进的方向的目标作参考

场景资源使用FBX格式导入UE4

材质和贴图

游戏内的材质和贴图和FBX一起从引擎导出。一些已经包含光影信息的贴图需要在短片中被去掉

许多贴图需要增加分辨率来匹配提高精度后的模型。Fortnite团队使用Substance Designer基于新的高精度模型的拓扑结构来改进现有的贴图

Substance Designer提供了多种优于传统技术的方式创建多张独特的2D贴图。Substance Designer 可以同时使用像素和向量图信息作为材质的基础，包括使用从模型导入的信息，来增加AO以及模型结构本身的变化做为基础层，应用于此模型的多个材质中

贴图做完后，角色身体部分的贴图可以随着FBX从Maya中导出（模型被导出成FBX，模型的贴图会被同时复制到导出的目录下）。因为脸部绑定和动画是和身体是两部分，所以从Maya导出为Alembic，贴图也会随之一起导出。

作为Alembic导出的模型，Maya中的材质和贴图命名需要遵循一定的规范。在Maya中指定给模型相应面次层级的Shading Group名字必须和UE4中子材质的名字相同

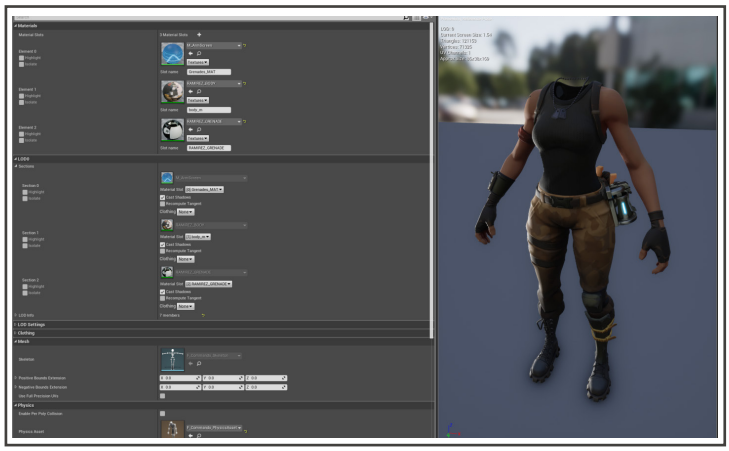


图 16: 给角色身体指定材质

对于FBX导出，Maya的材质节点被指定给模型的物体层级

图 17: 匹配Maya中Shading Group的名字和Unreal Engine中材质的名字

对于Alembic (Shader Group) 和 FBX (Material Node), 材质必须在面次层级被指定，而不是模型网格层级

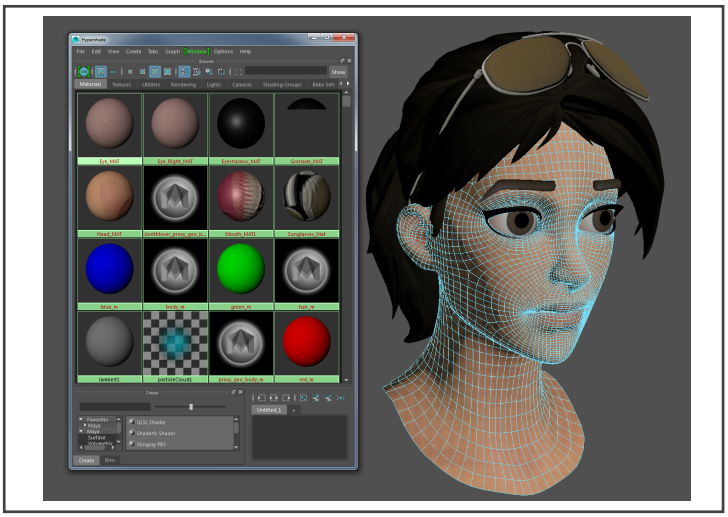


图 18: 在Maya中的面部材质

对于实时的考量

因为要高效实时渲染，对于模型面数贴图尺寸始终需要小心。随着产品的制作，很快东西就越来越多超出预算

团队为了优化，删除了一些游戏场景中对于短片故事情节无关的物件。另外当摄像机角度确定后，删除了模型背面等相机内无法看到的三角面。优化和品质提升需要同行！

绑定和动画

因为脸部使用的是Maya/Alembic 流程，而身体是ART/FBX流程，所以最终身体和脸部的绑定是两种不同的流程

身体是公司内部团队使用ART做的绑定。项目周期考量，脸部绑定被外包给第三方。每个角色被单独自定义绑定，然后和我们给的身体绑定整合后一起发还给我们

我们拿到返回的身体加脸部绑定后又会把他们发给另一个第三方的团队在Maya里制作动画。除了绑定我们还会发给他们粗略的动画发生的场景文件

当动画制作完毕后，外包动画公司发还给我们其Maya文件和动画视频文件。我们把动画导入UE4预览，评审；公司内部的动画师根据修改的短片剪辑还会进一步修改

身体绑定

游戏角色的绑定可以提供绝大部分动画需求，但为短片更高的保真度我们在游戏绑定上增加了额外的Joint骨点来提供更丰富的细节变形动画

骨骼层级和绑定控制器会被自动分开，以便干净地导出到UE4中（UE4不需要控制器）。这步没有ART工具也可以在Maya手动完成

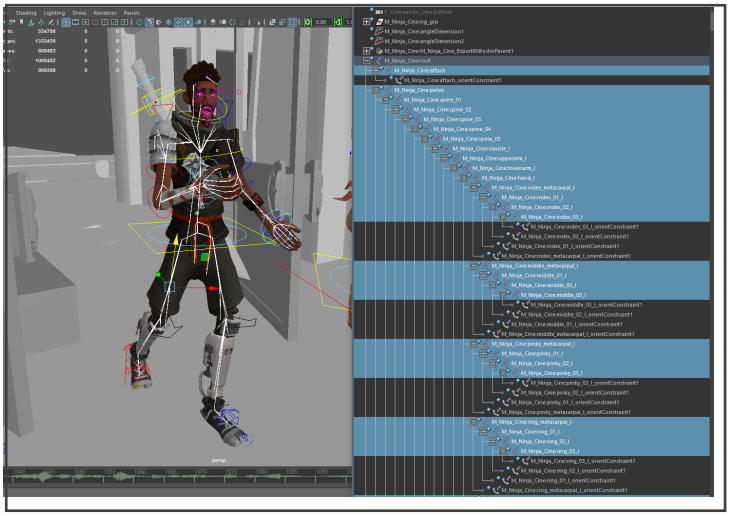


图 19: 骨骼层级和绑定被自动分开

身体动画

身体表演来自粗设阶段的动捕数据和手工关键帧动画的结合

粗设中的粗略场景也会从UE4中导出，然后导入Maya中为制作动画提供环境参考

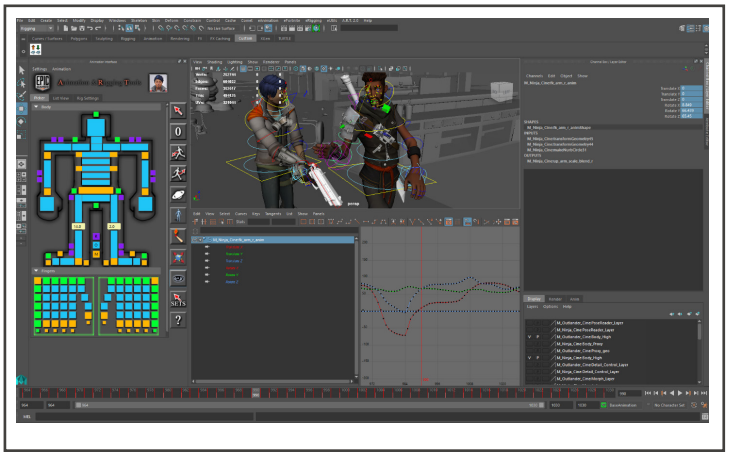


图 20: Maya中参考模型

动画师使用通用的动画技术基于镜头来打磨和细化动作。ART工具一定程度上简化了流程。动画做完后通过FBX格式被重新导回UE4中

动画和绑定工具集 ART

除了提供一个骨骼创建，定位，蒙皮等一系列初始化的绑定功能外，ART还提供了包括身体部件选择器，导入动捕和动画数据选项，导出FBX， 调整姿态，镜像，空间置换设置等等功能。以下描述了其中三个标签页面和右边按钮的一些功能

身体部件选择器

选择器标签页提供动画师快速选中需要对身体哪些部位绑定的功能。

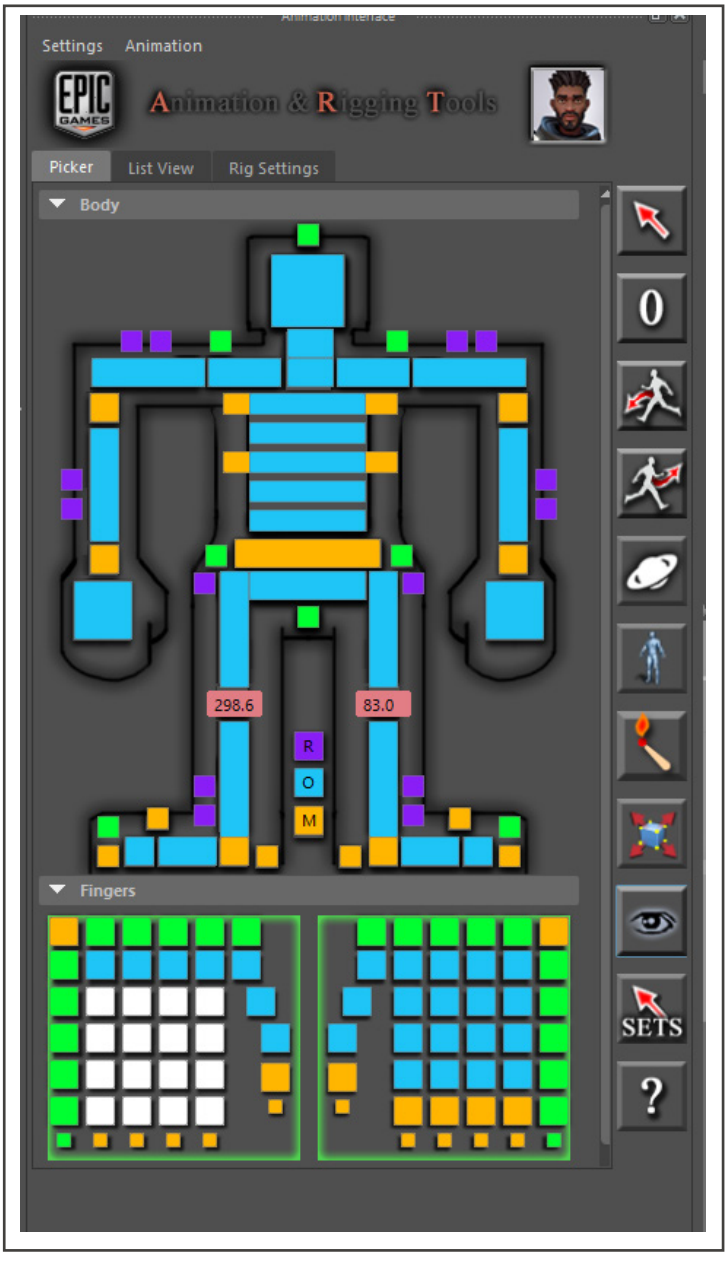


Figure 21: ART Body Picker

列表视图和绑定设定

List View 和 Rig Settings 标签界面包括了选择，导入导出，空间转换，绑定可视性和设置创建工具

图 22: ART List View and Rig Settings

Pose 编辑器

Pose 编辑器用来存储和加载自定义姿态

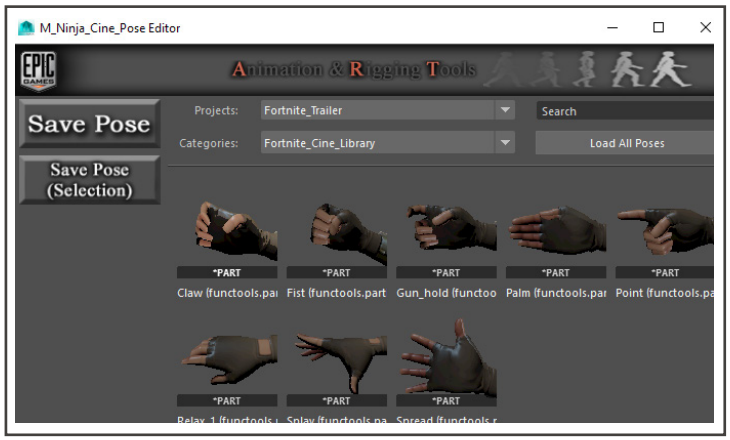


Figure 23: ART Pose Tool Window

面部绑定和动画

测试下来最终我们决定使用不同于身体的工作流来做面部的绑定和动画，在Maya里使用自带的绑定工具而不是ART工具，导出为Alembic格式

团队使用Maya的Lattices工具来制作面部网格变形动画。Lattice是类似一种虚拟的网格框，带有一些顶点。可以编辑这些顶点来改变Lattice的形状。角色模型上包含在Lattice框范围内的点会受到Lattice形状的影响，跟随其变动

Lattices是用来修改一簇模型顶点位置很好地工具。由于ART工具不支持Lattices，所以团队直接使用Maya自带绑定工具绑定模型

团队还希望使用Blend Space，ART也不支持这个。Blend spaces是同一模型不同造型的多个版本，现成的例子就是一个角色脸部模型不同极限表情的各个版本。当动画的时候，可以使用blend spaces在这些表情模型之间混合出最终表情

最终的脸部绑定包含了201个骨点，Lattices和Blend spaces。所有的角色头部模型都是同样的网格拓扑结构，同样的面部绑定，制作时重用绑定设定就成为可能

Blend Spaces 可以导入导出为FBX格式文件，但变形挤压包括Lattices的修改不行。因此无法使用FBX作为面部绑定和动画的导出格式。另外模型的点往往会被许多根绑定的骨骼，Blend Spaces混合，Lattices的变形同时所影响；由于FBX格式限制了最多一个顶点受到8种影响，但是对于面部这是不够的。Alembic就不受这个限制，因此它对于面部绑定和动画的导出是一个更好的选择

面部绑定由一些被约束到控制器的权重骨点组成，它们使用权重混合设定驱动关键帧，来动画眼球和面部肌肉群等。控制这些权重骨点的控制器又被成组约束到更高一级的控制器来实现更大面积的动画的驱动，比如移动整个下巴来实现张嘴动画。这样要比一个个单独调节下一级控制器要方便很多

面部绑定控制直接在3D模型面部进行控制，相比于虚拟控制面板界面上调整的方式直观很多

所有面部表情动画都是在Maya里使用手工关键帧动画实现

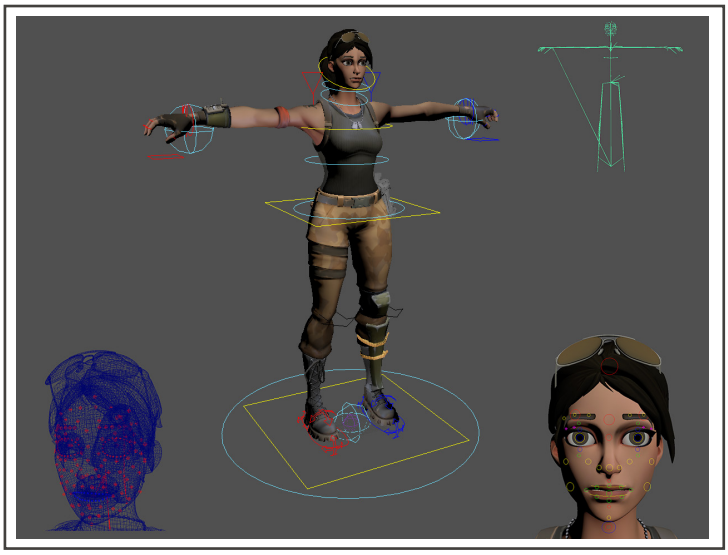


图 24:面部和身体绑定控制器示意图

图 26: 面部表情

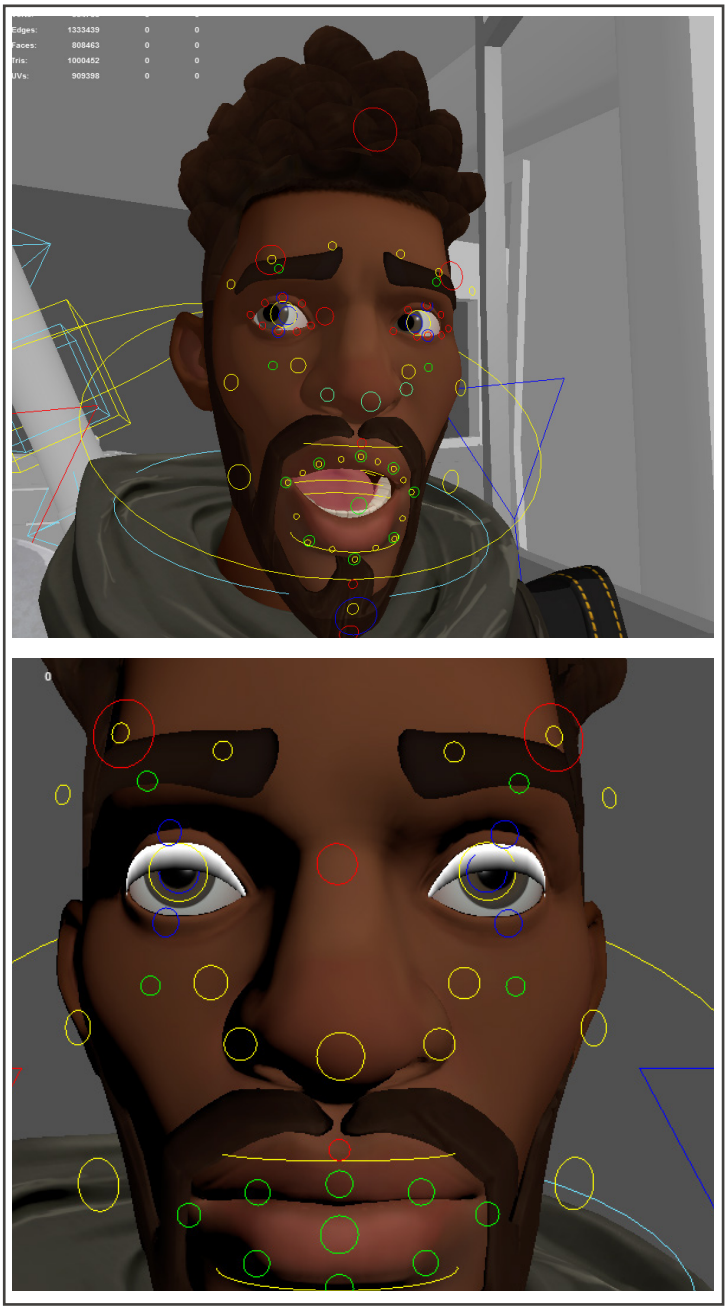


图 25: 模型相应位置的面部控制器

导出为FBX/Alembic

对于角色身体和场景物件动画修改和更新每次使用FBX导出，面部绑定和动画为Alembic

可以使用Maya内置导出方式，也可以使用ART带的FBX导出器

从Maya里导出FBX

图里展示了一些从Maya导出FBX的设置

所有动画导出时需要把控制器，约束等等的动画都烘培到骨点上，包括之前提到的为高精度短片模型所增加的哪些变形和控制，都需要最终只反映到骨骼动画上，UE4仅认骨骼动画。可以在Maya中通过Edit>Keys>Bake Simulation来实现烘培动画的操作

点击烘培， 烘培所有动画到骨点上

选择骨点，设置需要导出的动画范围，导出为FBX

更多信息参见Unreal Engine文档FBX动画流程主题

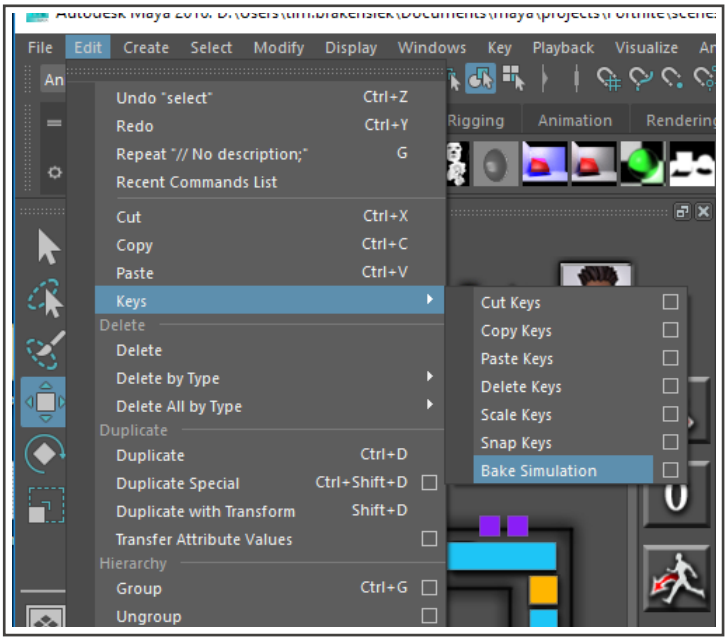


图 27: Maya中 Bake Simulation menu 选项

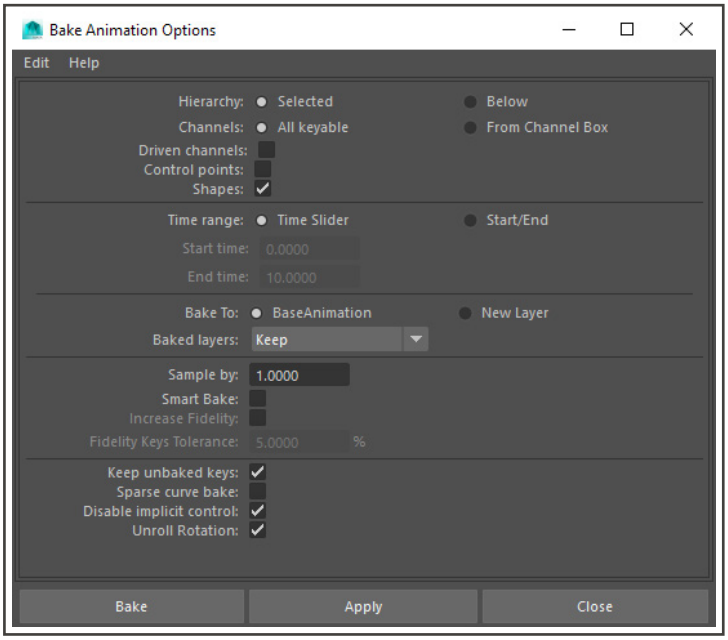


图 28: Maya 中 Bake Animation 选项对话框

通过ART导出FBX

这个演示了直接从ART中导出Fortnite短片第DB0137镜头中动画的步骤

在场景和绑定都被加载的情况下，点击Export Motion按钮打开Export Motion对话框

在对话框中设置选项然后点击Export FBX。这个操作自动生成一个绑定的副本，烘培所有动画到副本上，然后导出为FBX。这替代了在Maya里需要手动完成的工作

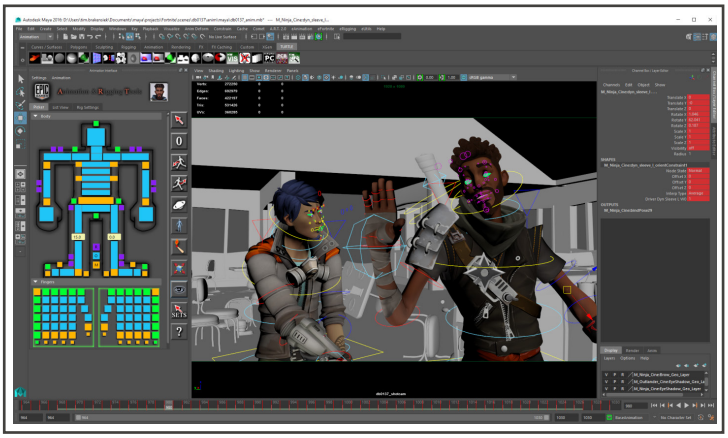


图 29: Maya中在场景中使用使用*ART*

图 30: Export Motion按钮

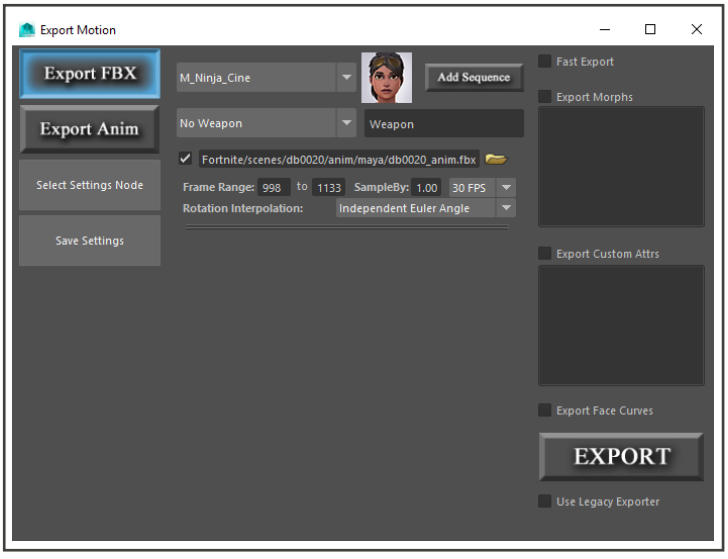


图 31: 使用ART导出动画对话框

从Maya里导出Alembic

从Maya里导出头部绑定和动画前，模型需要被三角面化，这是用了Maya的三角面化（Triangulate）工具

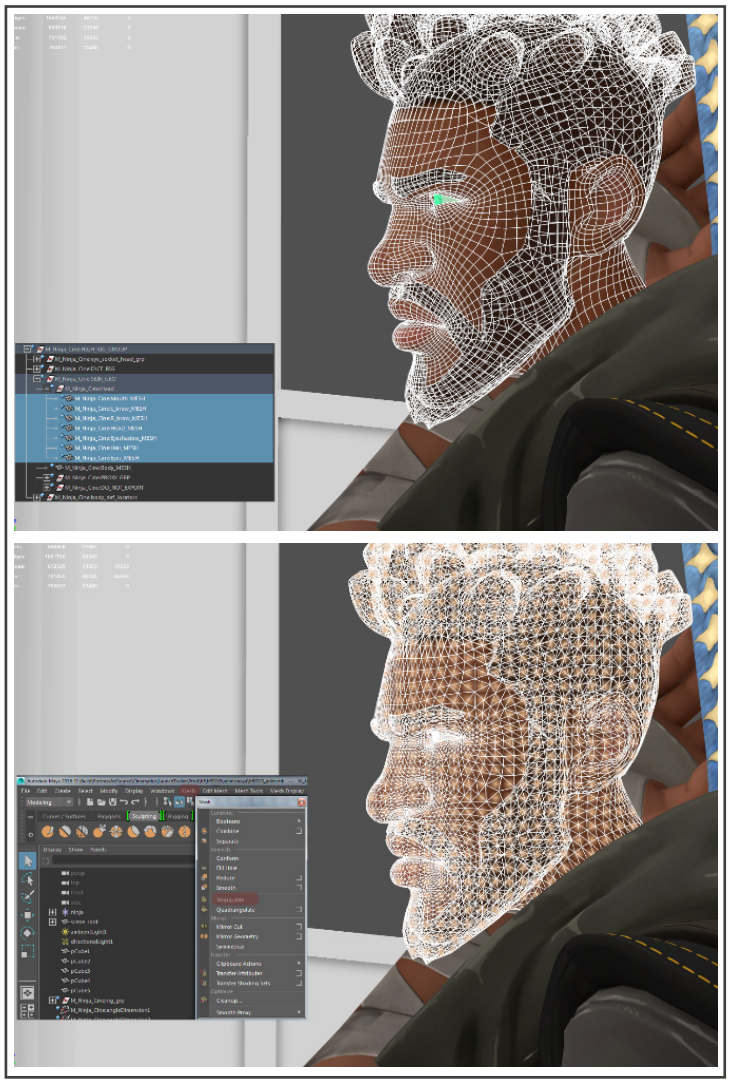


图 32: 三角面化前后的面部模型

导出Alembic文件，选中模型，然后从Maya菜单中选择 Cache > Alembic Cache > Export Selection to Alembic

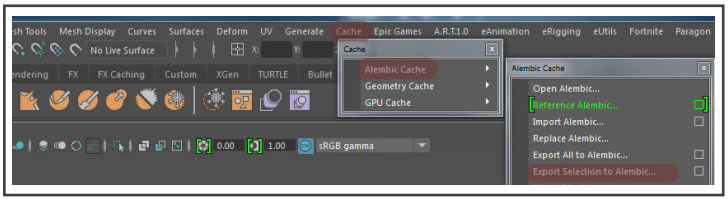


图 33: 导出选中的为Alembic格式文件

Alembic的导出选项设置窗口

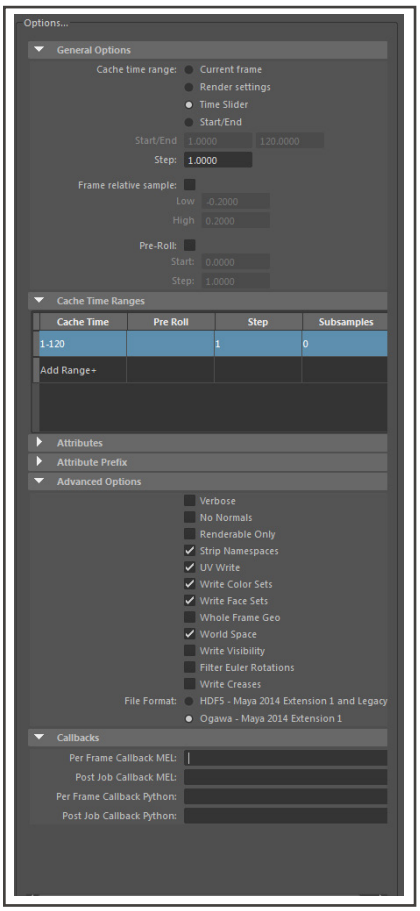


图 34: Alembic 导出选项对话框

自定义 Alembic/FBX Export工具

头身每次导出为不同格式的数据，重复次数多了也是很麻烦的，Epic内部开发了一个可以在一个界面同时导出FBX/Alembic的工具

这个工具叫Cinematic Exporter，目前还没包含在ART工具里。在这里提到这个工具为了说明在开发中是可以利用一些自定义的工具可以加速开发流程，比如Python写的Maya导出小工具

从Maya菜单中可以调出Cinematic Export选项界面，左边是场景中可以导出节点列表，选择需要导出的节点。

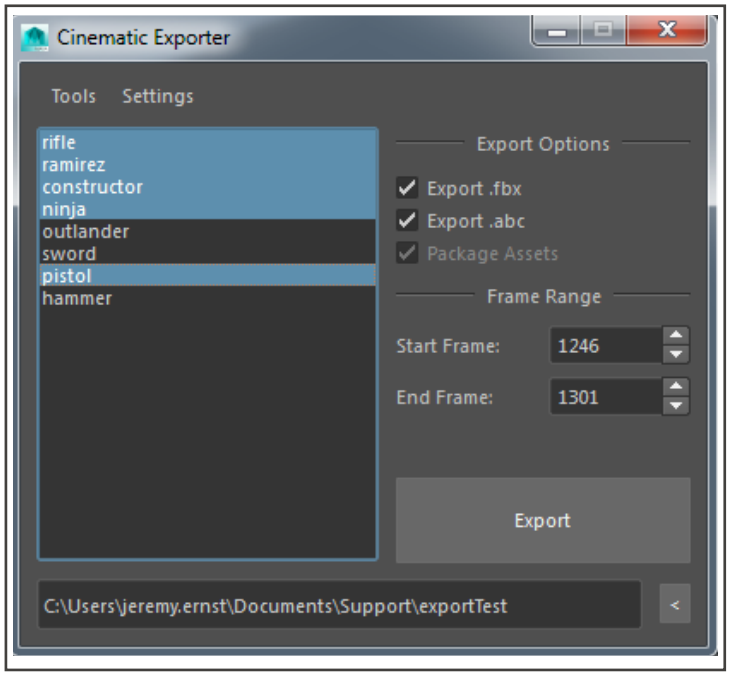


图 35: Cinematic Exporter 对话框内节点选择

默认情况下，会同时导出两种格式，但也可以单选。还可以设定需要导出的时间段，默认是当前动画轴范围

需要设定一个导出目的路径。Fortnite项目流程中，这是一个Perforce工作空间内路径

点击Export， 接着就会应用之前已经设定好的FBX和Alembic导出选项自动在后台执行导出的运算操作。这期间并不影响美术继续在Maya里工作

导出的数据被放置在一个命名过的文件夹内方便之后导入UE4使用

为了让Cinematic Exporter能够识别到那些需要导出的绑定目标，Epic的技术动画师创建了一个带脚本的导出Node，这是一个包含了导出数据信息的空物体，被放置在所有包含有需要导出的绑定目标的场景中。当打开Cinematic Exporter时，它会根据场景中找到的这个导出node，遍历其中的可导出绑定目标，如角色的头，身体，动画的场景物件等等

如下图，Export nodes是使用内部开发的Maya工具来定义哪些为Alembic，哪些为FBX，以及指定他们相关的一些信息。这些信息就是用来给上面提到的Cinematicf Exporter识别需要导出的绑定目标

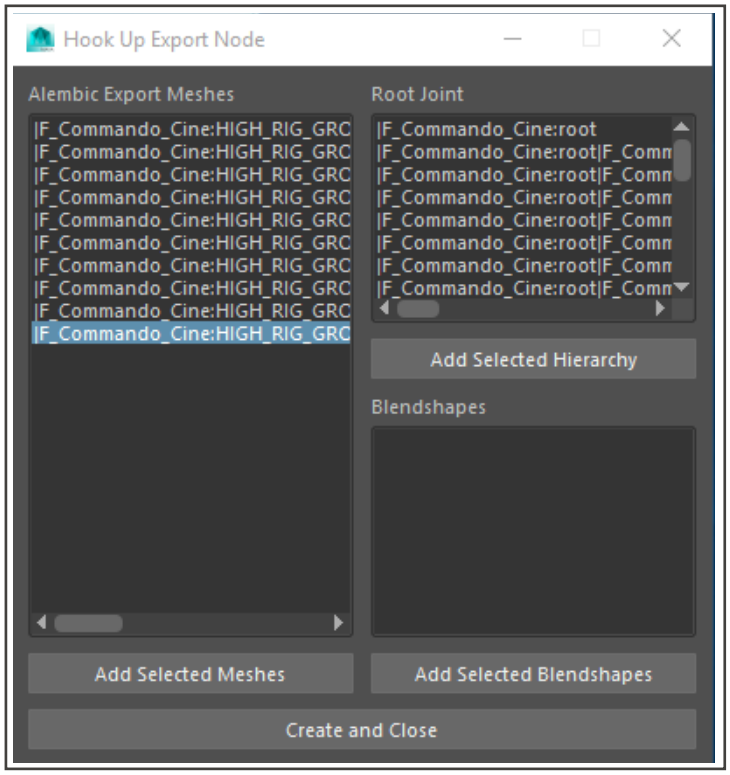


图 36: 设置 Alembic和 FBX 导出节点

导出的数据被有条理的按文件夹组织以便更方便地导入

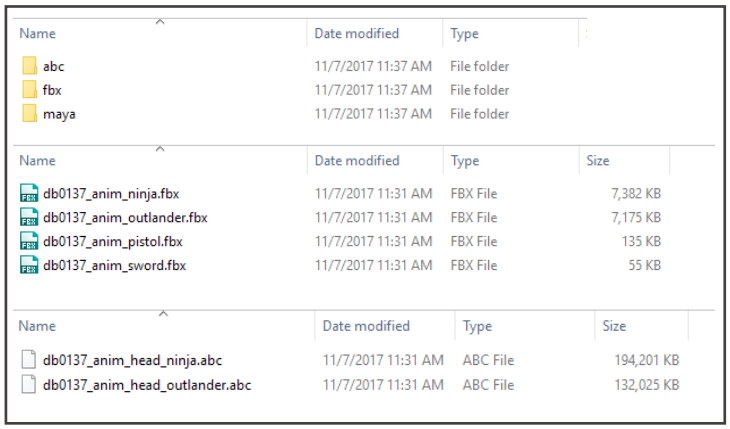


图 37: 被导出文件的组织管理

导入到UE4

最后一步是把两种文件FBX/Alembic（ABC）导入到UE4中

导入FBX到UE4

身体和场景物件及动画通过UE4的FBX导入工具直接导入

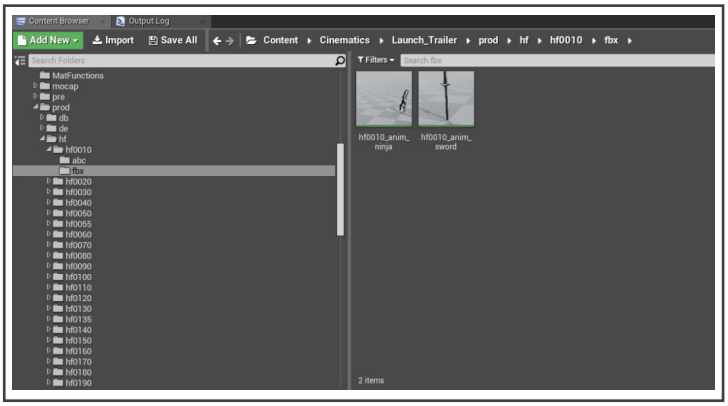


图 38: 将FBX文件导入UE4前选中相应的FBX文件夹

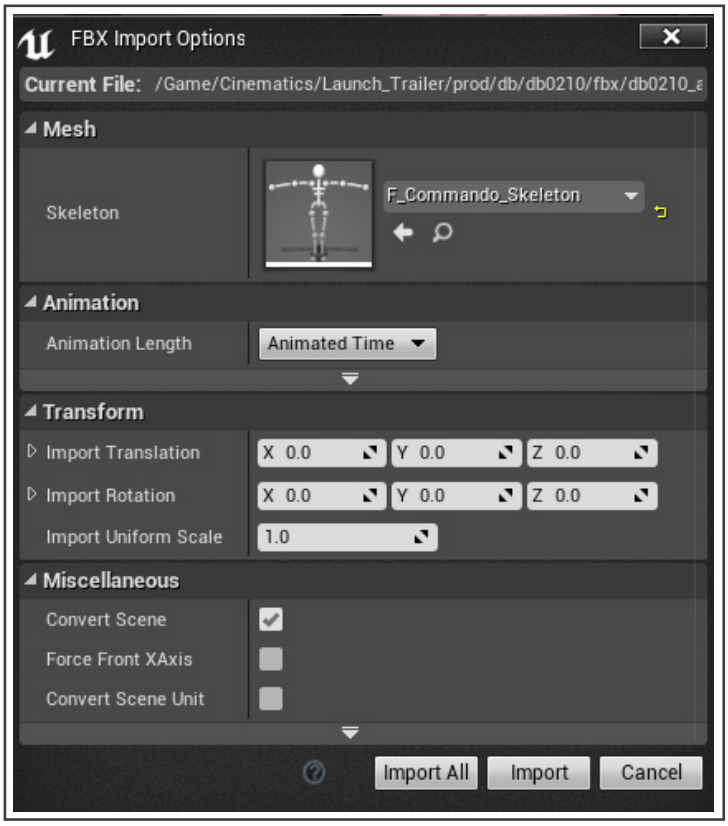


图 39: Unreal Engine FBX 导入选项对话框

导入Alembic到UE4

Alembic文件使用UE4的ABC导入工具导入为使用PCA压缩方式生成的Morph target

Alembic导入选择导入为Skeletalmesh选项

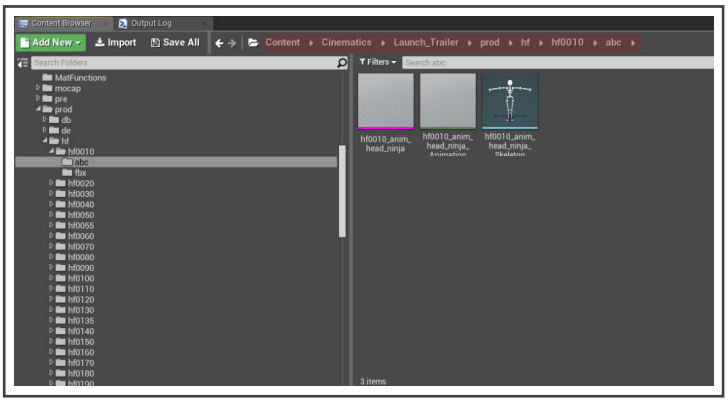


图 41: 将ABC文件导入UE4前选中相应的ABC文件夹

图 40: 导入动画应用于角色

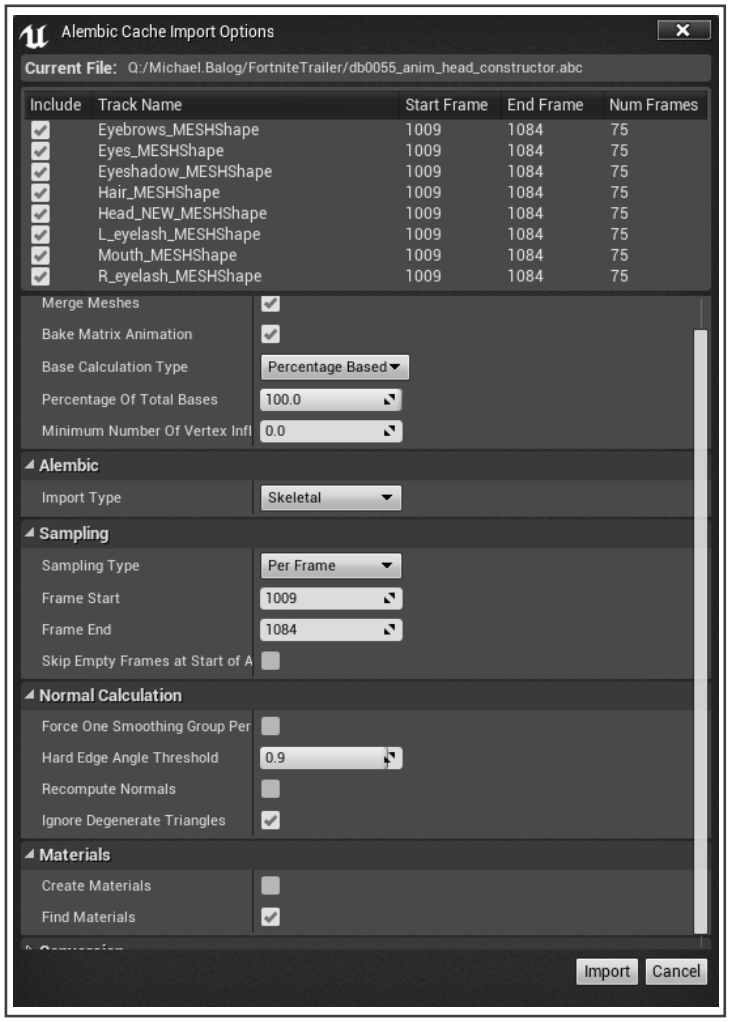


Figure 42: Alembic import options

在Morph target预览窗口可以预览导入Alembic后生成的Morph target数据

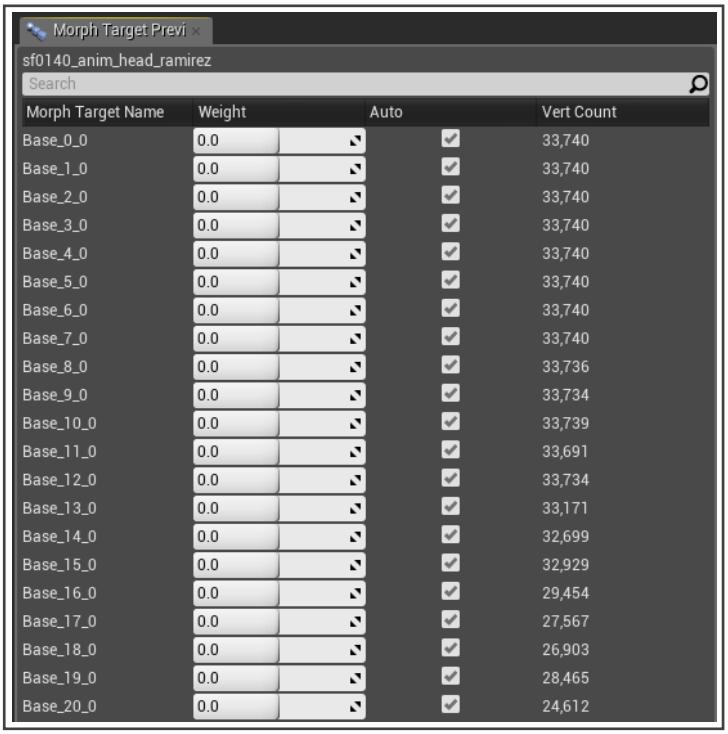


Figure 43: Morph Target Preview window

Morph target的运算量很大，最好使用GPU来处理，所以为了更好的回放需要把Use GPU for Computing morph targets（使用GPU运算Morph target）选项打开， 在Project Settings里设置 *Edit > Project Settings > Rendering > Optimizations*.

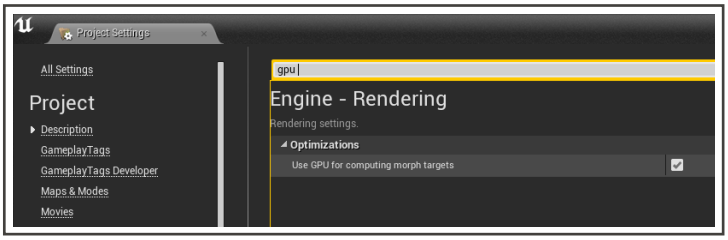


图 44: 使用 GPU 计算 morph targets 选项

PCA Compression(压缩方式)

当动画导出为Alembic时，ABC文件在每一帧存储了每个点的位置信息。对于一个15万三角面模型的2分钟动画，这个数据量非常大，实时分析以及回放这么大数据并不可行

Unreal Engine 导入ABC时使用 Principal Component Analysis (PCA) compression压缩方式来减少数据量的同时保持高品质的动画。在导入时，会抽取一个平均模型姿态以及一定数量较平均姿态有差异的模型变换姿态。通过姿态差异分析后决定哪些帧的姿态会被抽取为变化目标姿态来作为Morph targets，最终被存储的数据仅仅只是这些被抽取的姿态帧。这个过程把原本极大数据量的点动画提炼成更小的可管理的数据组

回放时，Unreal Engine加载这些Morph target数据到内存，然后每帧做实时地混合计算

使用这种方式，Fortnite的脸部动画可以使用ABC格式高效实时地在引擎中播放。AnimDynamics作为另一个UE4的动画特性为角色在原来动画的基础上添加了次级动画，一些细节物理动画比如身上的挂件摆动，飘带飞扬，还有头发和布料甩动，全都是实时通过AnimDynamics运算产生叠加于主动画上的次级动画。（Fortnite里没有特别飘逸的长发和服饰，未使用Apex Cloth）

灯光

本来Fortnite短片打算用烘培灯光的解决方案来提高渲染和回放效率。很多已有的和改进过的游戏资源也已经在贴图里烘培进了灯光信息。但烘培灯光的方案对于导演要自由快速的调整灯光有很大局限

因此Fortnite选择放弃烘培方式而采用全动态灯光加动态阴影的照明方案。逐镜头灯光的调整变得非常便利

图 45: 在Unreal Engine中实时回放Alembic面部动画

优先级覆盖

Unreal Engine 能够通过高优先覆盖方式修改场景中已有灯光和物件动画属性。这使得美术可以高效地在高层级Level Sequence中创建灯光组合，然后在低层级Shot的level Sequence中修改并覆盖其属性

属性优先级特性简化了自定义物件和灯光的放置，根据实际情况方便地在低层级中修改灯光位置和参数等。

灯光组和Spawnables临时可生成性

Fortnite实行灯光组和可生成灯光来为短片作为照明方案。两种方式都证明是高效的，但创建灯光组为打光计划提供了更好组织性。最终，最好的方式是构建一套专门的灯光组给整体影片的照明，然后在此基础上需要时添加可生成临时灯光

Distance Field Ambient Occlusion（DFAO距离场环境光遮蔽）

Fortnite是全动态的照明，对于间接光遮蔽，比如天光的遮蔽（阴影）是一种软阴影，我们使用了一种叫做距离场环境光遮蔽的动态软阴影方式。

首先需要了解模型距离场，它是一种包围在模型周围立方形的3D贴图。贴图定义了包围模型的立方体空间内到最近模型表面的距离。所有模型表面外立方体内的点都为正距离，模型内部的点为负距离。这些贴图是模型导入后就预计算好的并存储于模型文件中，贴图的分辨率决定距离场的精度。

通过距离场的信息光线首先可以快速做球形追踪，加速计算；其次使得投射软阴影成为可能

动态天光利用模型距离场可以快速的进行光线计算而产生中型尺度上（这相对于屏幕空间SSAO小尺度而言）的环境光阴影

更多关于距离场信息请参见文档

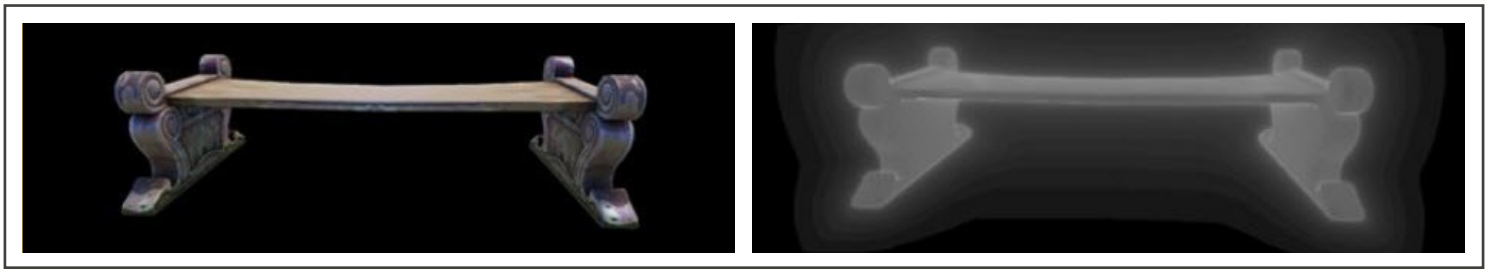


图 46: 原始模型（左）和距离场信息显示（右）

特效和后期

特效和后期

传统动画流程需要有专门的渲染农场来渲染后期特效，作为单独的渲染输出层，在如After Effect或者Nuck的后期软件里合成。虚幻引擎避免这个步骤，提供直接引擎内后期特效合成渲染

对于这部短片，Fortnite团队使用UE4中FFT Bloom渲染特性来增加真实的光晕特效，并使用其Cinemetric Tonemapping[电影色调映射]后期功能来调色

这些烟状的风暴和敌人死亡特效值得特别关注。在Fortnite里，敌人总是从这种紫色风暴中出现，所以短片一个主要的目标要强化敌人角色和这种风暴的关联性的表现

图 47: 敌人怪物从风暴中出现

为达成这个目的，我们利用实时体积化渲染方式来表现更加生动的特效以及直接在UE4里调节效果，而不是从外部工具模拟完了导入

敌人死亡特效

短片中有超过25中不同的怪物死亡特效。由于ESRB[娱乐软件分级限制]，这些死亡效果不能看上去太血腥。Epic于是设计了一种死亡后迅速沙化并消散的效果来避免血腥暴力，同时也增强了风暴和敌人之间的关联性

一般像这种效果往往使用外部的模拟软件计算完了然后导入使用；但由于摄像机角度，角色动画的修改变换，每次更新都需要重新计算模拟，再次导入，迭代效率差。我们决定完全在UE4制作和模拟这种效果，利用材质，BP以及引擎强大的渲染特性实现了这种全实时引擎内流体模拟特效，加速了迭代效率

这种沙化流体特效根据需求还得受环境灯光和物理力影响，并且是实时的，这更加具有挑战

图 48: 怪物受到甩动的大锤击毙后的特效

敌人死亡瞬间被体素化，这利用了一些小手段把角色模型转化为与其同样造型的体积贴图。模型的运动方向在体素化过程中也被捕捉并加入模拟中。美术还结合速度，加上一些扰流噪波贴图以及模型法线信息

为了提高运算效率，照明仅使用了单次散射的方式，即灯光照射到体积特效后经过单次散射反弹回来。对于实时渲染，对于体积特效多次散射照明计算开销相当大。不过我们利用了模糊单次散射结果的方式来模拟多次散射的效果

我们使用颜色而不仅仅是亮度来表现阴影的明暗，因为这种体积特效被照射后在不同的厚度区域会体现不同的颜色。这种颜色的差异更强化了我们需要的风格化视觉目标，即那种粉里带紫的烟雾效果

每个特效的模拟过程是用过Sequencer来控制的。模拟在特效开始前一帧被触发，为了提前得到运动方向。接下去的那些属性比如速度，密度都可以利用关键帧动画在时间轴上被记录。相对于离线模拟再导入的方式，迭代效率大增

压力迭代期间为了效率使用半分辨率渲染。模拟计算利用了GPU加速的Navier-Stokes-based流体解算器。那些怪物本身就是特效的发生器，它们的密度，速度会提供给实时模拟计算需要的信息。这些信息可以在Sequencer编辑器中的轨迹中方便的做关键帧动画

Blueprint也被利用来把模拟的结果转化基于某个特定摄像机的Flipbook[4]（动画序列图）。它可以作为某些无法使用实时模拟表现的镜头的后备方案；比如一个镜头里出现大量怪物被同时杀死产生如此特效的情况

在一些关键镜头里，利用一些不可见的模型来产生人为的运动和速度，更好控制效果。体积贴图实际是一种假体积，是一系列2D切片模型堆叠而成3D化

3D模拟则使用标准的raymarch shader来渲染

风暴

风暴造型是利用一种程序化距离场层来定义的，使用3D 动画Flowmaps来驱动其动画。这些Flowmaps是在UE4里利用BP工具在VR里绘制的

在VR里绘制Flowmaps要比使用鼠标光标来得直观的多。云的增大缩小动画则是通过动画距离场层的阈值来实现

所有的体积风暴云和敌人死亡特效一样使用Raymarch shader渲染，但使用的是Flowmaps替代了全流体模拟。为了实现不同形状的云，VR中Flowmap的绘制是一种不断反复的交互迭代过程

图 49: Fortnite短片中云的形状

[4]在UE4中，Flipbook是一系列2D序列图片，按一定的速度连续播放产生动画效果。这种方式要比实时模拟快

最终形状是程序化的层和VR里实时手绘结合的结果。最终Flowmaps上速度的体现也是手绘的速度叠加噪波贴图后综合的结果。程序化的造型变化更多用在需要许多云膨胀动画的镜头中，美术可以通过参数逐镜头的控制其膨胀大小

体积雾

一些镜头需要一种暴风即将来临的感觉，但又不需要前面提到的体积特效。这里我们就用到了UE4自带的体积雾

因为体积雾使用体素反平方分布函数，远距离很快就失去细节。所以它更适用于全场景相对柔和的雾效。远距离还是放置了一些使用raymarch shader的物体来增加特定形状的体积雾效

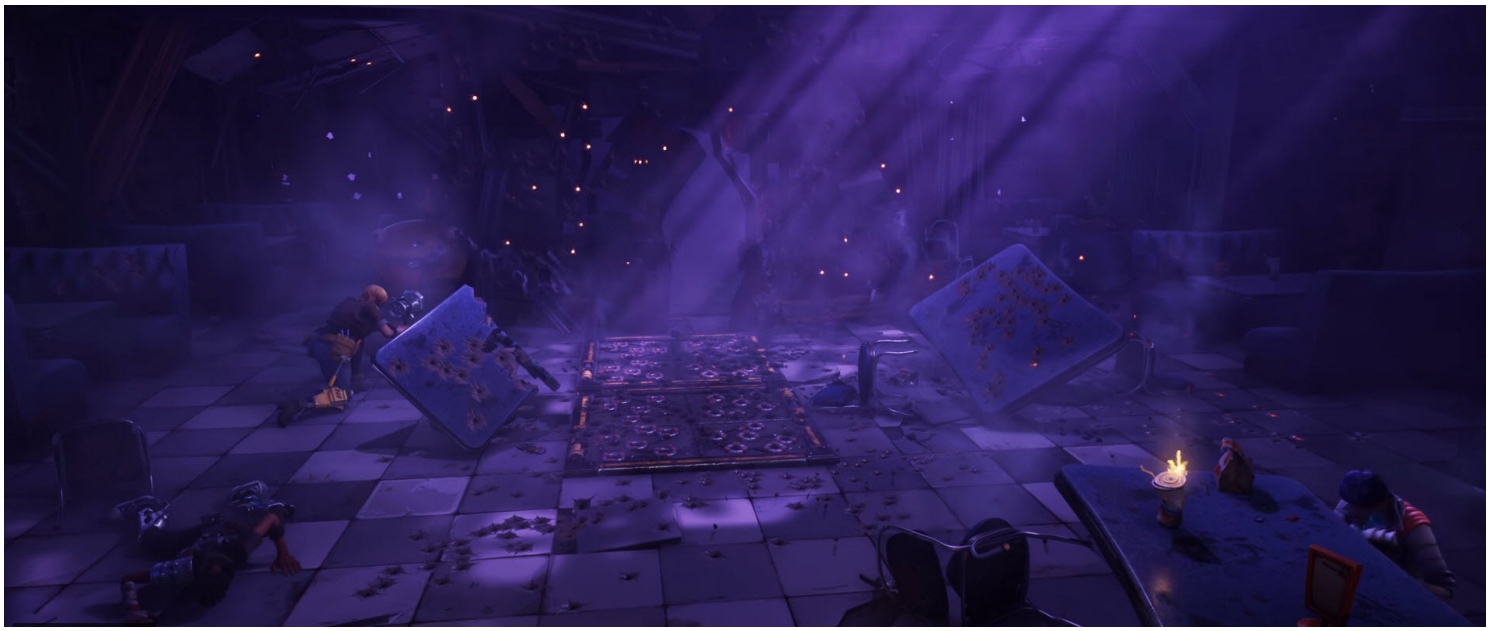
场景中的体积雾可以接受阴影，还附上了特殊的材质增加细节[扰动地尘埃等]，其中包括应用了上面提到的假体积贴图。

Unreal Engine 使得EPIC通过展示在实时渲染中电影品质的特效模拟的可能性来突破行业的界限。在实时环境下创建所有这些特效使得我们能够实现在传统离线渲染环境下所不能达到的品质和整合能力

最终输出

当所有制作完成后，每帧会从UE4被输出为不压缩1080P PNG格式文件，然后再使用Adobe Media Encoder和PNG Video codec 转成QuickTime（MOV）。使用Adobe Premiere CC编辑合成声音

最终产品交付格式为使用 PNG video codec编码的 Quicktime (MOV)



产品数据&信息

项目相关数据

制作时间表

- 立项: 2016年8月
- 开始已有游戏资源的改进: 2016年11月
- 预研开始于： 2017年1月
- 开发: 2017年2-4月
- 基础幕的灯光 January 2017
- 镜头灯光: March 2017
- 特效开始: April 2017
- 最终调整交付: May 2017

团队组成

• Michael Clausen - Sr. Cinematic Designer and Co-Director
• Gavin Moran - Creative Cinematic Director and Co-Director
• Andrew Harris - Studio CG Supervisor
• Michael Balog - Director Animation Technology
• Ryan Brucks - Principal Technical Artist
• 3 Senior Cinematic Artists
• Approximately 7 to 10 Unreal Engine lighting, effects, and technical artists
• External Animation Team - Steamroller Studios, FL
• Editorial Team

目标平台配置

PC 配置

- PC - i7 (第7代，或更高)
- 64 GB 内存
- SSD 硬盘
- 1080 GTX / 1080 Ti / P6000

关于此文档

作者

Brian J. Pohl
Tim Brakensiek
Simone Lombardo

感谢合作

Michael Clausen
Gavin Moran
Michael Balog
Brian Brecht
Andrew Harris
Ryan Brucks
Sebastien Miglio

编辑

Michele Bousquet

中文翻译及编辑

李文磊
Wenlei Li