介绍：

本指南是 Paradox Interactive 游戏中所有 3D gfx 的完整指南。大多数示例都来自我的 HOI4 mod：Old World Blues。未来的更新可能包括 2D gfx，但这只是作为 Clausewitz 引擎的一般指南，Clausewitz 引擎是 EU4、HOI4 和 Stellaris 的 3D 支柱。该指南将跟踪 OWB 中的 protectron 资产的整个过程。整个指南将使用 Blender 3D、Notepad ++ 和 GIMP。任何问题都可以通过 https://discord.gg/2W8JpRP 在 discord 上发送给我

任何和所有错误或过时的信息都可以 pm 给我，以便我修复它。谢谢！

指向 blender 插件 github https：/lgithub.com/ross-g/io_pdx_ mesh 的链接转到 “Releases” 部分下载 zip，然后在 blender 中使用安装插件 ->select zip。

第 1 部分：建模

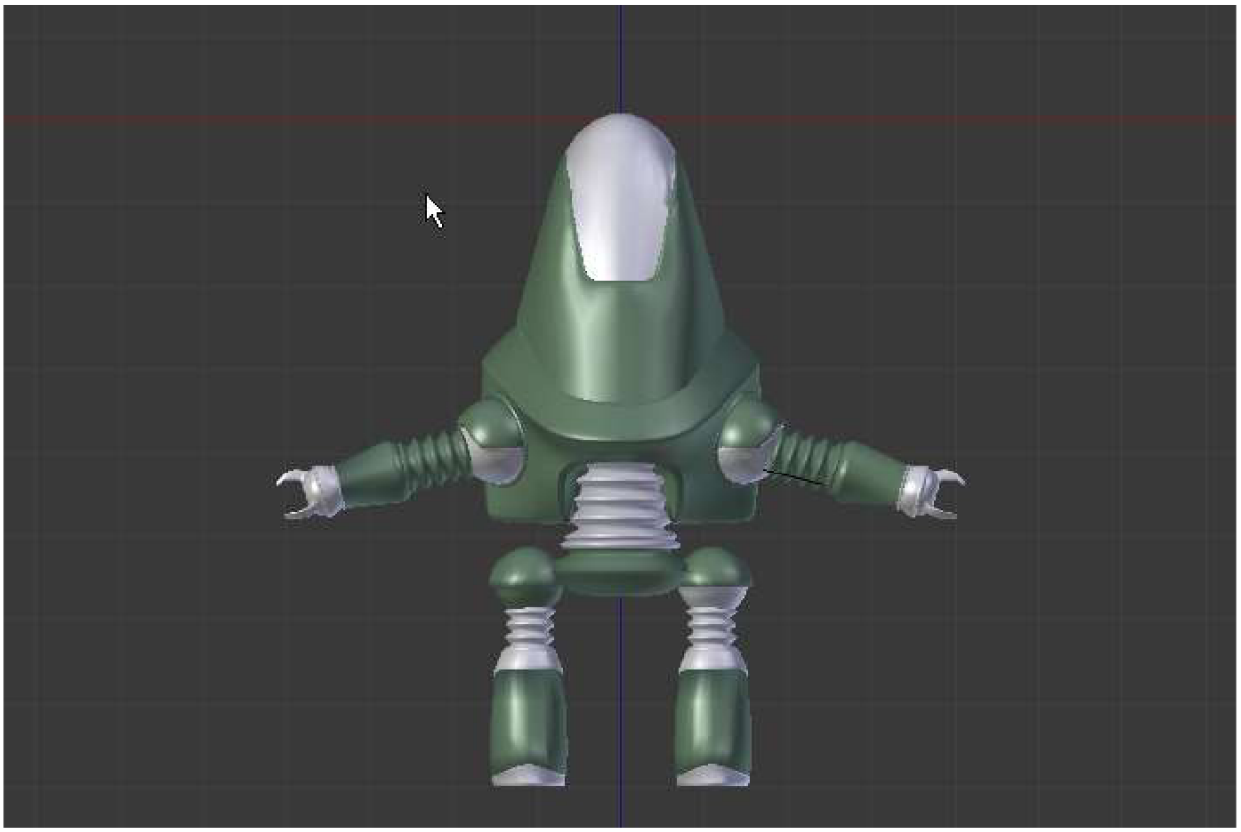
本指南的第一部分 ill indopth 涵盖了对资产进行建模和准备各种方法，这些方法详细说明了添加到游戏中，同时可以有效利用用户资源。

Highpoly / 详细网格

向游戏（任何游戏）添加内容的第一步是完成网格的高多边形或详细版本。这部分将是一个多数理论。

了解此过程最重要的一点是，有许多方法，每种方法都提供不同级别的产品。每个都需要不同的技能，我更喜欢顶点作，即使用作为 3D 模型基础的各种顶点来作它们，而不是添加填充步骤，例如数字雕刻。我发现在我的项目中，顶点作比雕刻更容易，因为我制作了更多坚硬的表面对象，并且没有很棒的绘图板供我使用。

这是我坐下来几个小时后使用参考图像后的保护器，显示了正交投影（无透视）中的正面和侧面突面。



为了快速了解生产过程，它使用镜子、细分和次表面修改器进行建模，管子是弯曲成曲线的阵列修改器。

这是过程中最主观的步骤。

Lowpoly / 游戏就绪

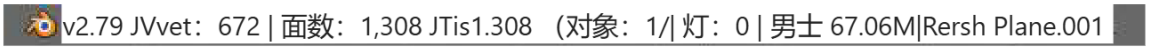
低多边形网格是我和 PDX 社区中另一位杰出的 3D 艺术家（《星际迷航：新视野》的作者）之间的一些辩论。在游戏中使用高多边形和详细网格会消耗更多资源，但可以产生更高质量的视觉效果，正如预期的那样。在难以判断低多边形网格不是高多边形网格的地方取得平衡是最好的，但更耗时。

要创建低多边形网格，高多边形网格需要重新拓扑，或者使用统一、优化的几何体在其上制作“蒙皮”。有多种工具可以做到这一点，但我总是手工完成，从最大的块开始，然后锻炼。这需要很长时间，但我总是以我想要的结束，并且可以将我的顶点数量完全优化为我想要的任何值。

这个过程也是主观的，所以这是一个最终结果：

为了比较，之前的顶点计数：

之后：

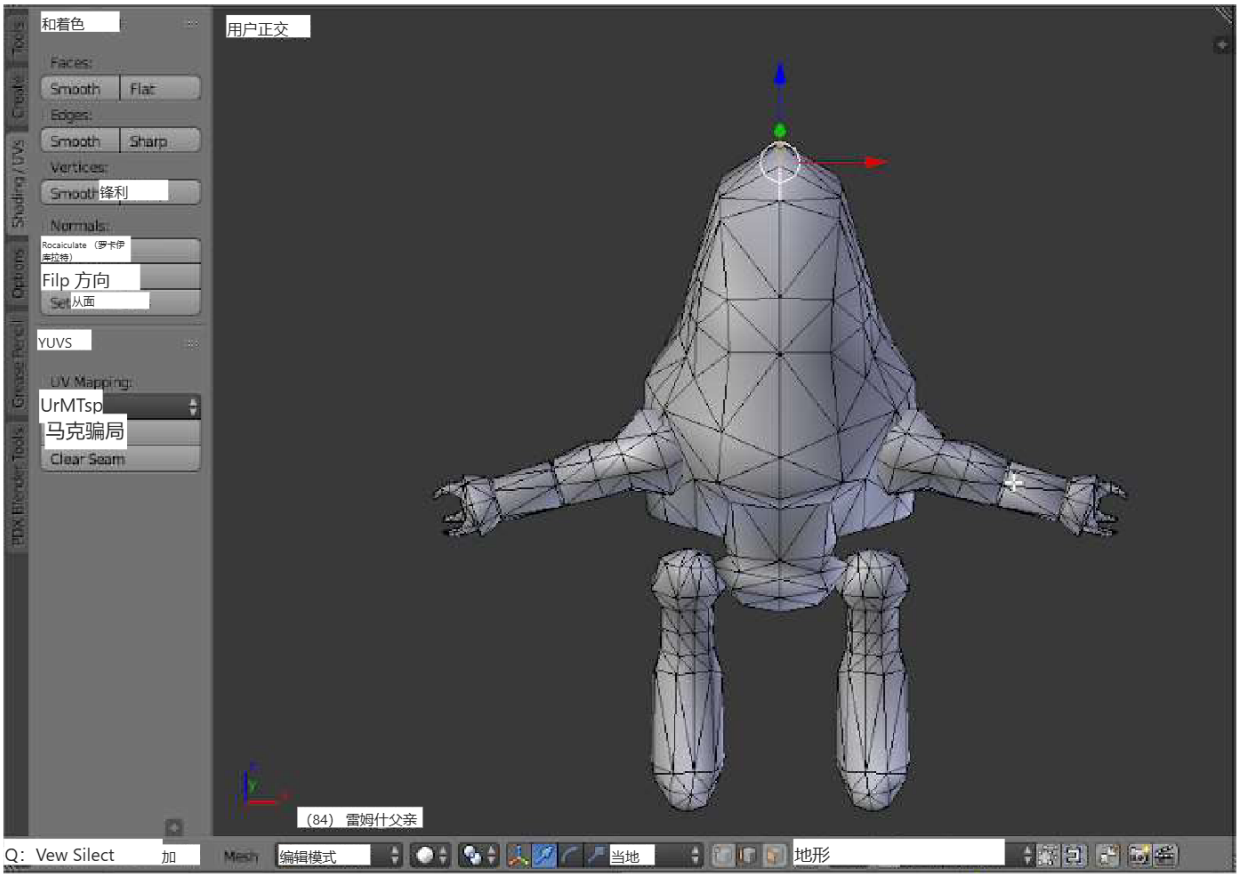


巨大的性能节省，虽然屏幕上有一个，影响会非常小，但是屏幕上有 1000 个，这就是 60 fps 和崩溃之间的区别。

UV 展开：纹理的第一步

应用纹理是在任何游戏中创建漂亮的 3D 资产的最必要步骤，在本部分的结尾处将很明显。AUV 是一组 2D 坐标，用于将 3D 顶点映射到 2D 平面上。回想一下小学，这就是复杂的“网”是什么，除了不仅仅是制作绝对比例的广义投影（50 像素正方形上的 1 个像素是等效的 50 个单位高的面上的 1 个单位）。我们创建复杂形状的投影，但这些形状不是（纹理拉伸），拉伸是 100% 可以避免的，但通过避免 i，我们在 3D 中的两个相邻块在 2D 中不相邻的地方创建接缝，这会使纹理变得困难。取得平衡很重要。在不太明显的地方添加接缝，并添加足够的接缝以最大限度地减少拉伸。一个技术娴熟的 3D 艺术家几乎可以有 O 个拉伸和隐藏的接缝，但一个技术娴熟的纹理艺术家可以同时处理拉伸和接缝，所以如果一个部分必须弥补另一个部分，就应该取得平衡。此外，值得注意的是，较低的多边形网格更难消除这两个问题。

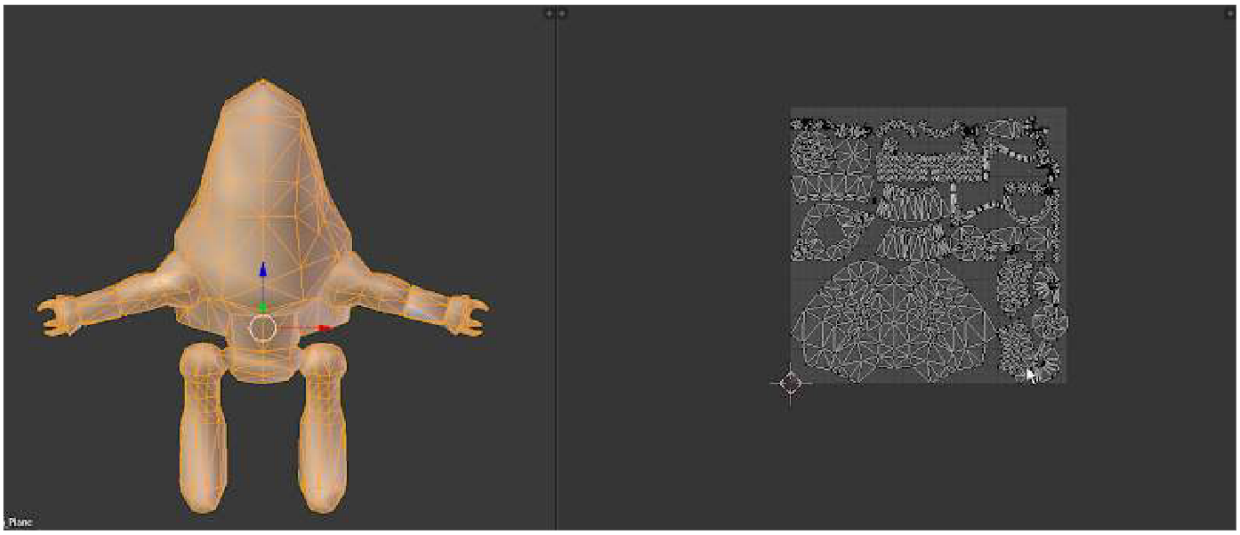
要进行 UV 展开，请在低多边形模型上进入编辑模式。舔 边缘选择 模式，然后选择要接缝的边缘。



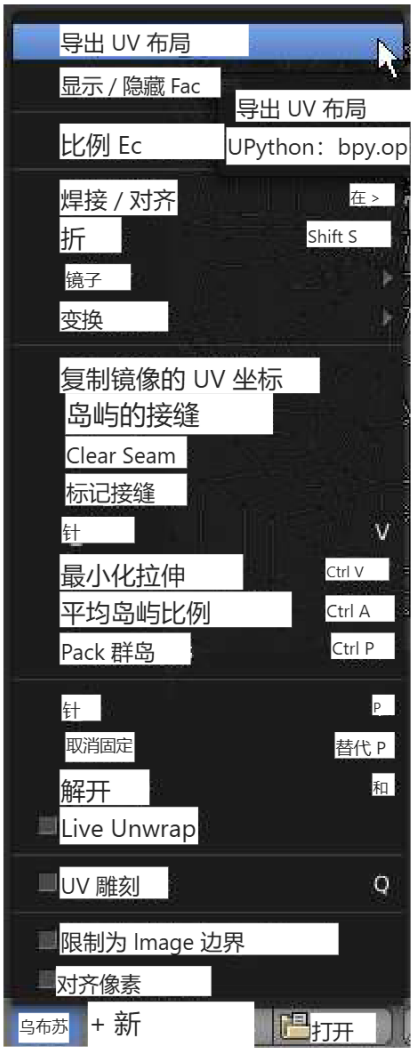
然后按 mark seams。

标记完所有接缝后，按 A 直到选择模型，然后按 U 并选择 unwrap （展开）。您也可以忽略标记每个边接缝，并使用 Smart UV Project。它做得不错，我在这里使用了它，因为拉伸不是一个大问题，因为模型大部分已经被分割成材料。

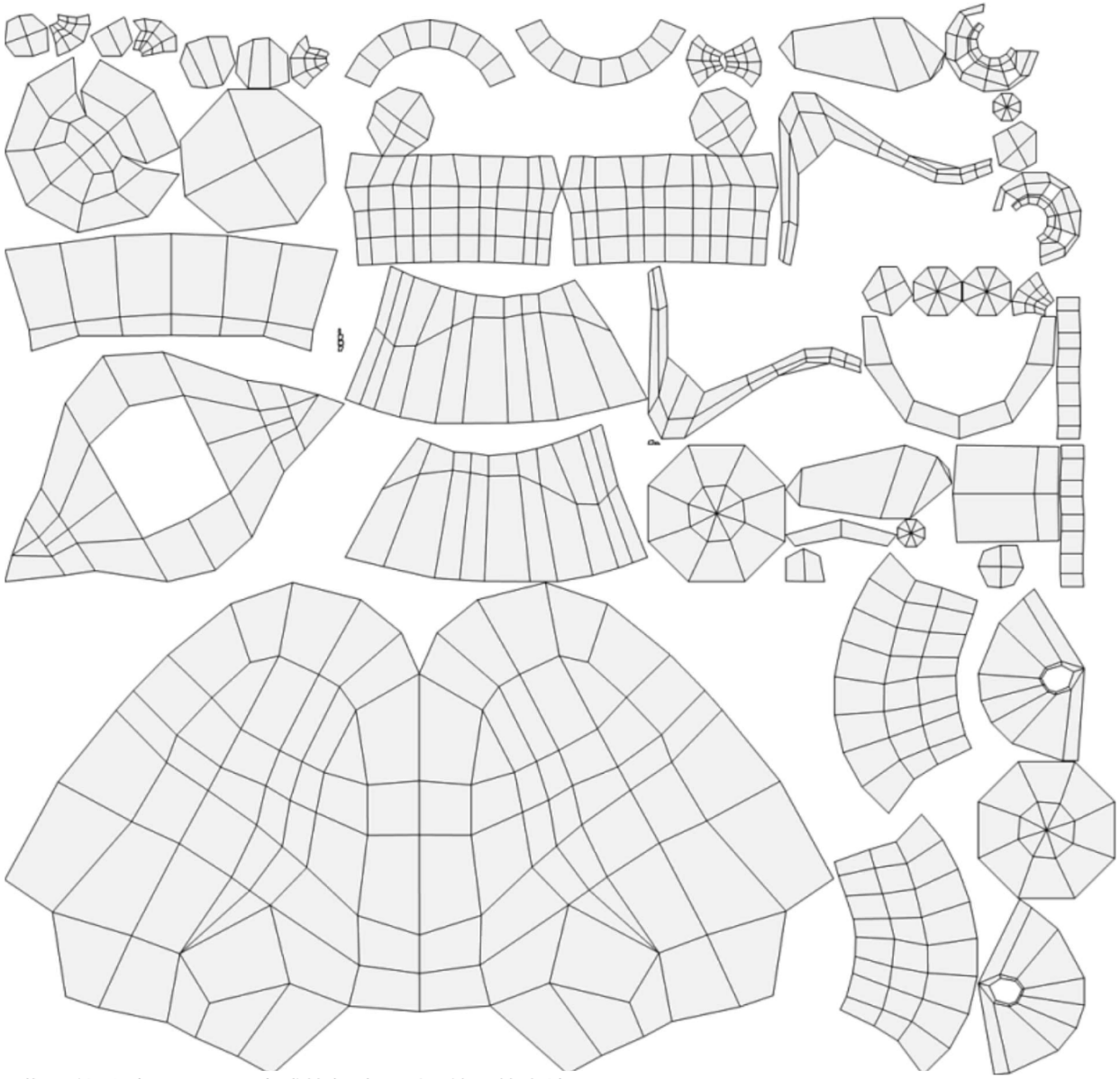
这是我的 UV 贴图：



建议将此 UV 贴图另存为稍后用于创建纹理。



要获取此图像，请执行以下作：

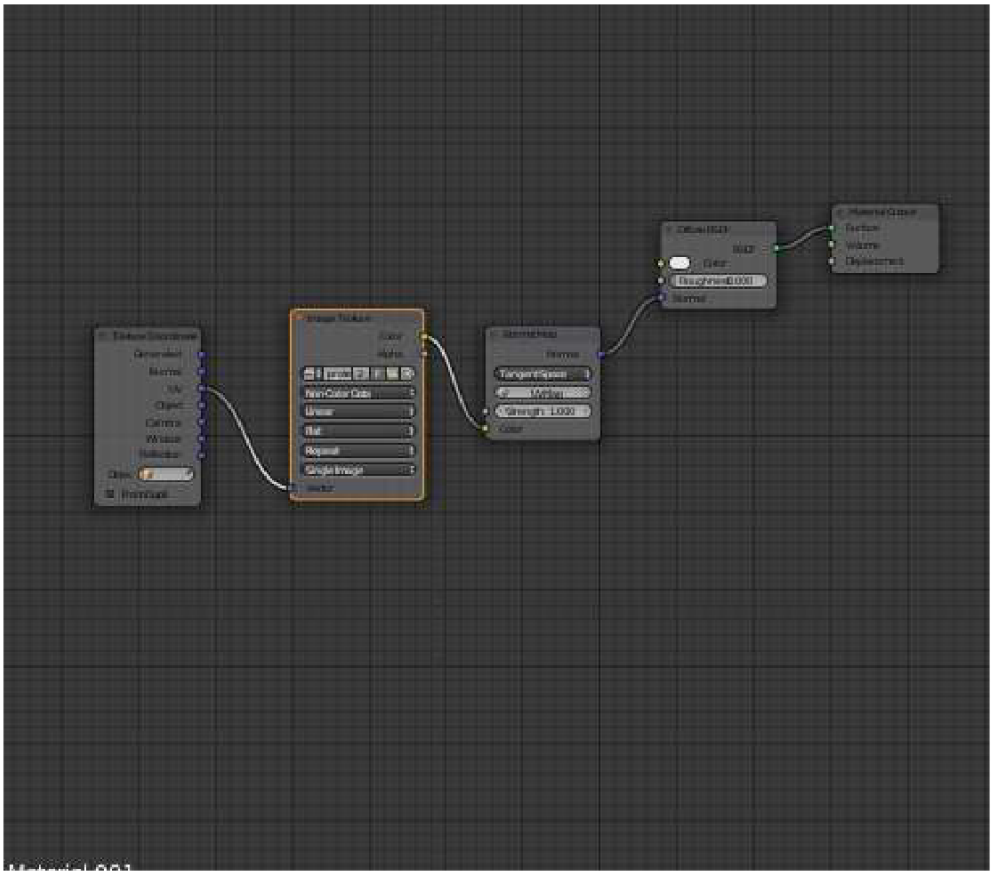


（您可以看到 Smart UV 生成的各种不利于纹理的小片段）

法线映射：将 Low Like High 设为

为了确保 lowpoly 网格尽可能接近 highpoly，在游戏中获取它的下一阶段是对其进行法线贴图。这使用三个颜色通道 RGB 制作一个 2D 矢量映射，为显卡制作一组标准化的指令，以 “推送” 外观。我不是它如何运作的基础专家，但它会产生漂亮的结果。这是一个 “基础” 法线贴图，然后我会添加稍后在纹理中绘制的任何材质的纹理。

切换到 Cycles Render。使用图像纹理节点在低多边形上制作材质，然后添加所需分辨率的空白纹理。选择节点。



首先，选择所有 highpoly 部分，然后按住 Shift 键并右键单击选择 lowpoly 部分。重要的是，它只是一个对象。

在 Camera properties （摄像机属性） 面板（右侧）中，向下滚动到 Bake （烘焙）。

复制这些设置。

解释：

Space： Tangent - 制作切线空间图，使用模型的切线来定义每个像素的颜色及其矢量。

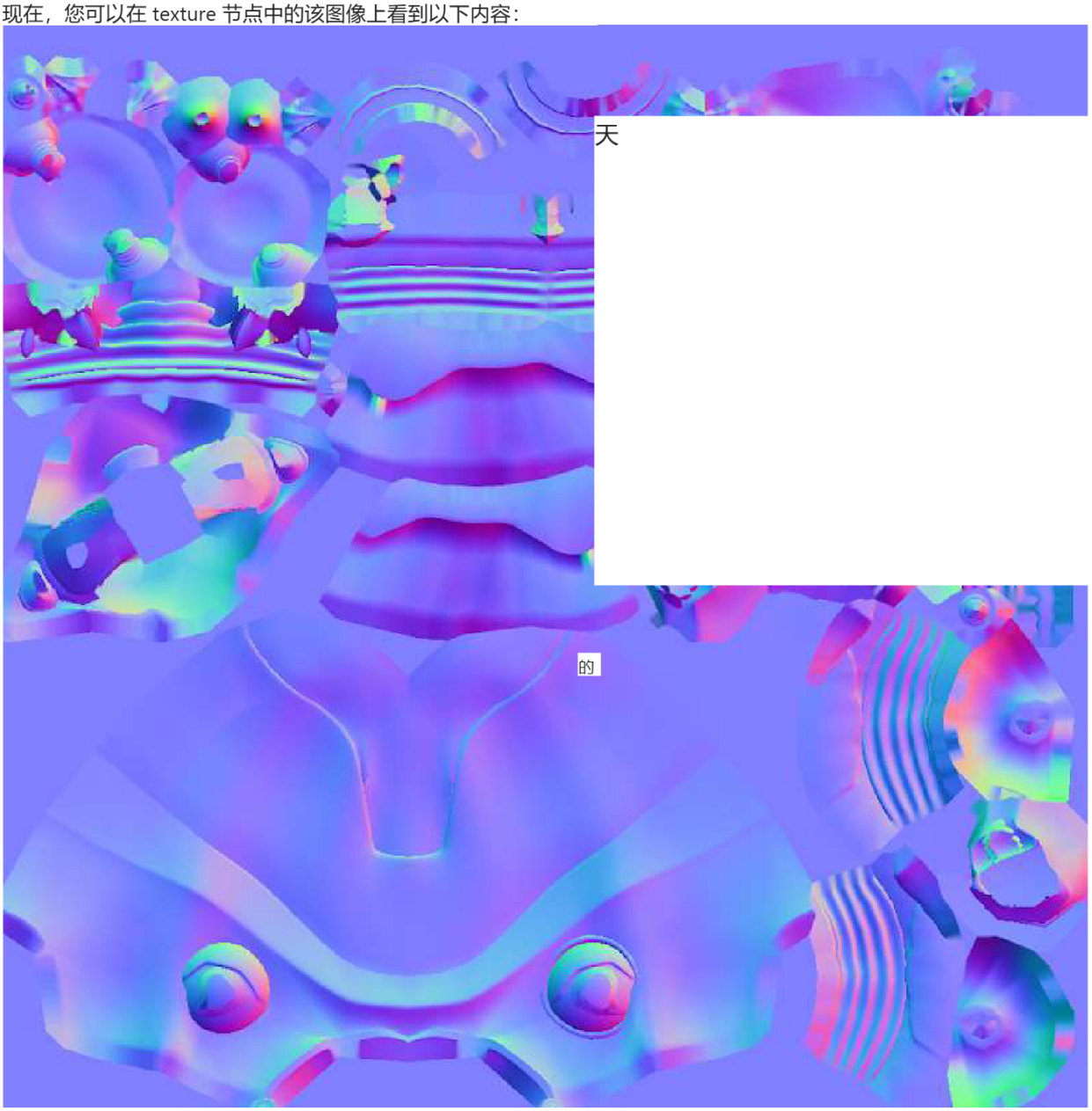
margin：4px - UV 周围的边距也要烘焙边缘的颜色，有助于确保没有间隙。

Selected to Active（选定项到活动项）：VERY IMPORTANT - 使选定部分烘焙到 LAST 选择项上

Clear：这只是一个好的做法，如果烘焙多个连续部分（不同的低多边形网格），请取消选中此项。

Swizzle：别管这个，它定义了 xyz 如何映射到 rgb。

烤！
现在，您可以在 texture 节点中的该图像上看到以下内容：



纹理

这一步对于制作 fuly feged 资产来说是最重要的，它与您现在屏幕上的空白白色形状相得益彰。

免责声明：我不是 OWB 的纹理专家，并发送我们的纹理师创建纹理所需的一切，但我确实了解这个过程。

要制作纹理，请将上图所示的 UV 布局导入到照片编辑器中。在 t 下创建一个新图层，并使 UV 布局透明。在 UV 布局下导入法线。

您可以开始纹理化。这也是非常主观的，所以我只能解释克劳塞维茨是如何解释纹理的。您只需将每个通道重新映射到下面的通道，或使用 Photoshop 导出器（此处未详细介绍）。

Clausewitz 纹理集指南：

输入纹理：

漫反射 / 反照率 - 原始颜色数据，仅着色材质特定的阴影。

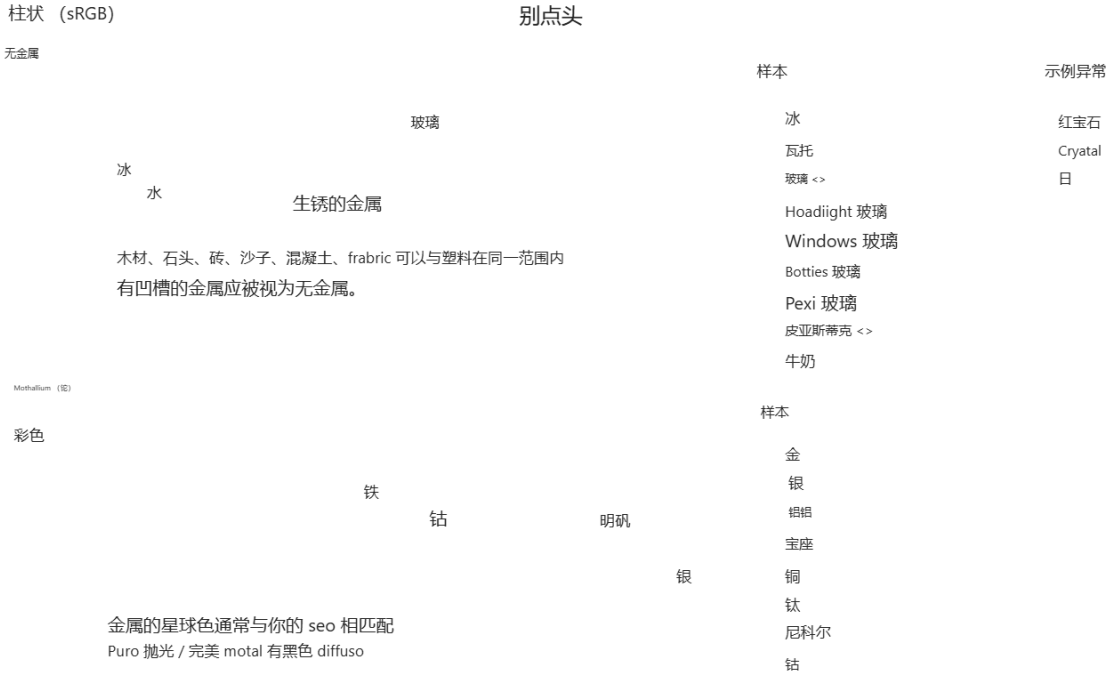
正常 - 上面解释了，给出了显卡的更多详细信息

Specular - 细节与光的交互

镜面反射 - 对象的反射率

金属度 - 与其他材料相比，物体是金属的金属金属与光有特殊的相互作用。

粗糙度 - 物体的粗糙程度，这会影响散射光的方向，而镜面反射会影响吸收。这使得闪亮的粗糙塑料和类似纸的东西之间存在差异。



以下是镜面反射贴图的指南：

输出贴图（通过 Stellaris Wiki）：

弥漫 性：

R- 漫反射 R

G- 扩散型 G

B- 扩散 B

A- 不透明度 （Alpha）

正常：
R- 正常 R
G- 正常 R
B- 自发光
A- 普通 G
（实际法线是 2D，而不是我们导出的 3D，因此蓝色通道丢失了）

镜面：
R - 蒙版（星辰中的自定义帝国颜色等）
G- 镜面反射
B- 金属度
A - 光泽度（粗糙度反转，又名 1 粗糙度为 0 光泽度）

保护层的输出纹理：
弥漫 性：
正常：
镜面：

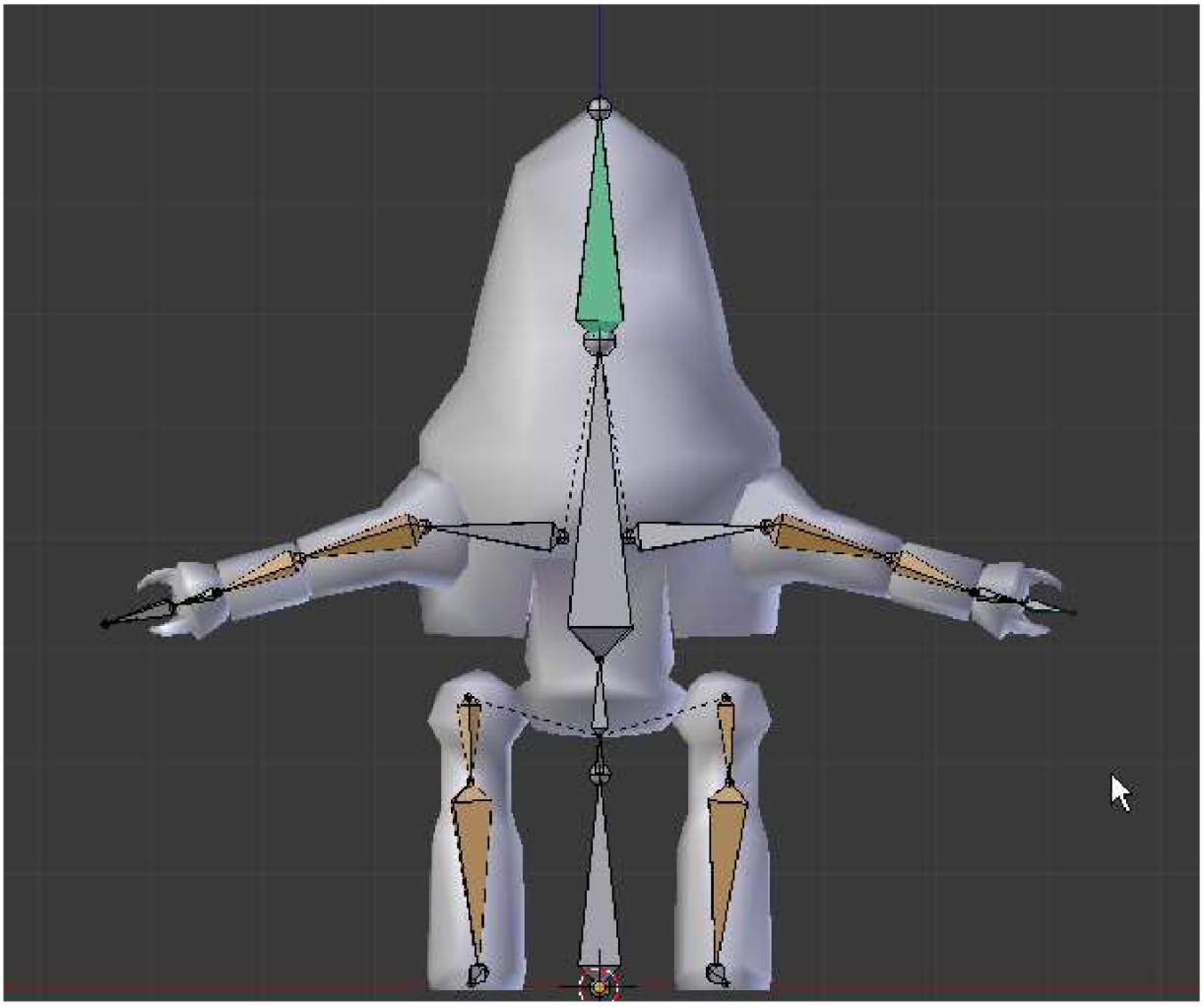
第 2 部分：升降装置

这就是我们让模型跳舞的地方！索具是一个极其复杂的话题，因此与一个人可以做的事情相比，这将非常简短。唯一要记住的重要一点是没有形状键，没有驱动程序，也没有零长度的骨骼。Shape 键采用两个网格状态并在它们之间进行插值。例如，带有微笑形状键的中性状态面孔。驱动程序使用骨骼位置更改 blender 中的一些设置值。克劳塞维茨两者都不支持。

FK/Deform Rig（FK / 变形装置）：

FK 装备是实际移动网格的装置。网格直接绑定到这个 rig，而 rig 是我们最终导出的。两条规则：确保每个骨骼都有一个父骨骼，除了一个名为 Root 的骨骼。没有头部和尾部相同的骨骼（零长度的骨骼）。

索具的基本作是创建一个骨架，并在脚之间的底部添加一根骨骼。按 E 键从前一个骨骼的尖端挤出骨骼。这会在新骨骼和旧骨骼之间创建一个可以弯曲的连接处。在模型的每个自然关节处创建一个关节，直到得到如下结果：



重要的是要确保骨骼在模型的正面和侧面与自然关节所在的关节完全对齐。除了颜色，我们稍后会谈到这些。

确保在某些模型的手端或背面添加 “nodes”，它们被用作事物的连接点！关于 Stellaris 的说明：Stellaris 使用定位器来定位火炮和飞船部分。这些目前无法通过我用来暴露的技术添加到我的知识中，通过即将推出的插件可能会改变这一点（导出部分中的解决方法）。

我将骨骼组织成骨骼组，以便跟踪它们，并更好地从正交视图制作动画（如果它们是相同的颜色，则很难区分左右）。

接下来，我们需要将此网格加权到我们的新骨骼集！选择网格，然后选择对象模式下的骨架。按 Ctrl-P 并选择 With Automatic Weights（使用自动权重）。这将创建顶点组，将每个顶点分配给与骨骼同名的组。每个顶点的权重介于 0 和 1 之间，以影响该骨骼。这可以通过选择网格并进入 Weight Paint 模式来更改。这个过程可能很挑剔，而且有比我更好的教程来涵盖这一点。

IK：反向运动学

下一步是创建一些控制器骨骼来控制我们未来的 IK 或反向运动装备。反向运动学使用类似于手的控制器骨骼来移动整个手臂。可以这样想：你不要移动你的肩膀，然后是你的上臂，然后是你的肘部，然后是你的下臂，然后是手腕，然后是拿起玻璃杯的手，你把你的手移到玻璃杯上！如果你认为，你确实必须移动所有这些，但只考虑

关于移动你的手。这就是 IK 的作用，您移动手，其余的必要工作就会随之完成！

因此，对于像这样的双足角色，我做了一些基本控制：
手
肘部
Look target（允许您将头部移动到 “arget”'empty 中，给人一种专注的印象
脚
膝盖

一种普遍需要避免的控制是 heel-roll rig，它们包含零长度的骨骼，但在 blender 的系统中没有，骨骼可以被父系但分开。在 Clausewitz 中，骨骼是首尾相连的，这使得 heelroll 的骨骼长度为零！

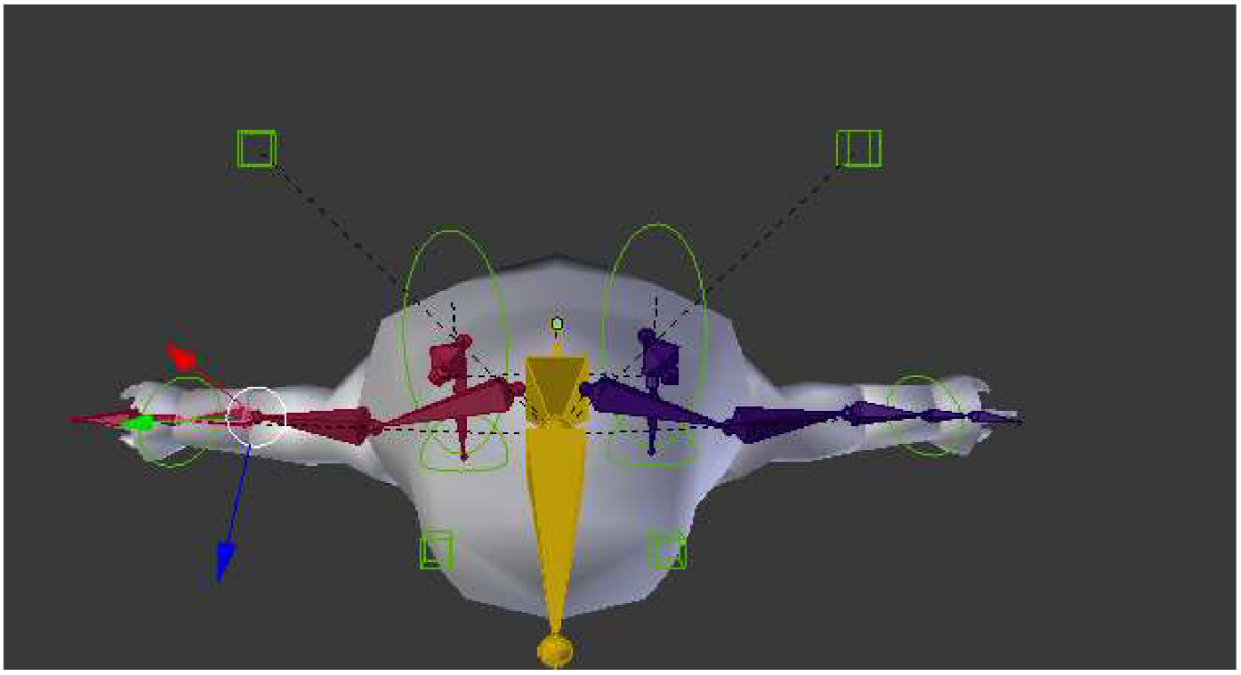
这是我的 IK 控制器

使它们发挥作用
手：
选择前臂骨骼，添加骨骼约束 Inverse Kinematics。从下拉列表中选择手部 IK 控制器（选择骨架 > 打开骨骼选择菜单）

将链长度设定为刚好低于肩部
检查拉伸
确保 lkK 控件的头部接触前臂的尾部
选择手部骨骼，添加复制旋转，选择 IK 手部控制器。

脚步相同

肘部：
选择上臂骨骼，添加 Inverse Kinematics。选择肘部 IK
链条长度到刚好低于肩部（通常为 1）
确保在编辑模式下，弯头位于弯头接头的后面：



对膝盖重复此作，但在关节前面

看：
选择头部。Add track to.Select look IK 目标
收件人：仅限 Z

您现在拥有了一台功能齐全的 IK 设备！试试看！

第 3 部分：动画

动画是主观的，所以我会用最笼统的术语给出说明。

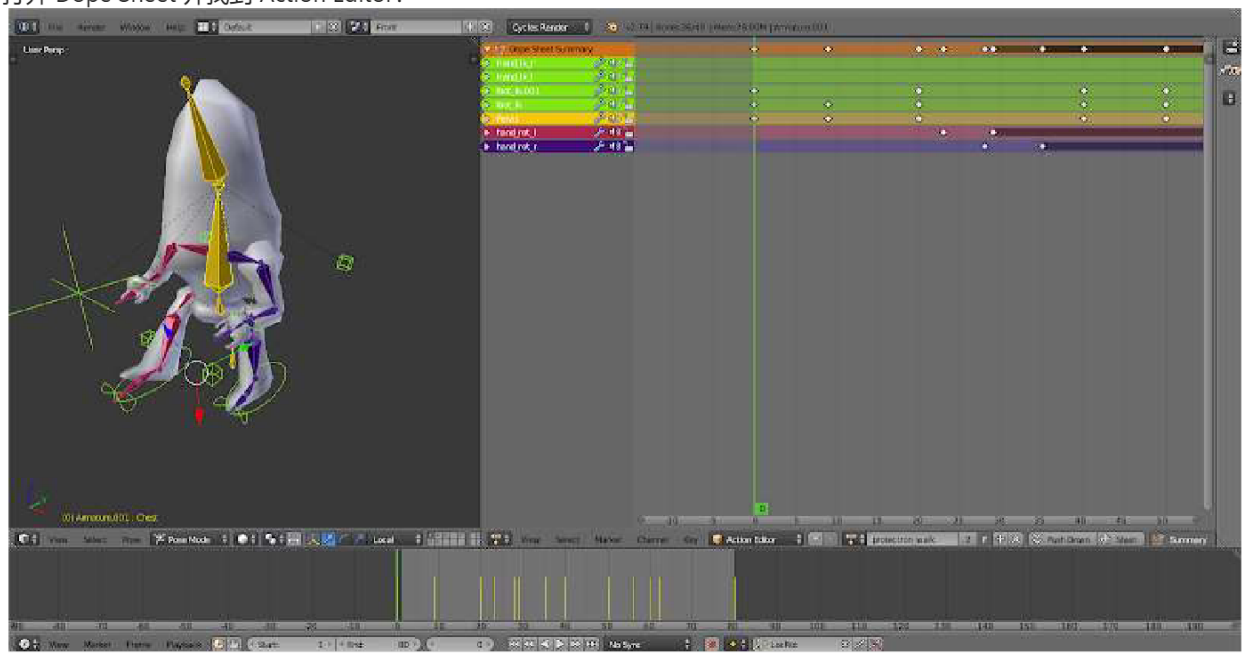
首先，找到 rig 的起始位置。您想要的任何位置的第一帧。在时间线下找到下拉菜单并选择 “LocRot”。在 3D 视窗的右侧，将旋转设置更改为 Quaternion WXYZ，这将有助于确保导出的内容与您所看到的完全一致，因为 PDX 动画格式使用 WXYZ。

据我所知，缩放是有效的，但很少建议更改某些内容的缩放，除非你打算创建具有大量流的动画，但不会无缝循环。（举个例子，我推荐《英雄联盟：命运的转折》）。我们将制作这些动画，因为它们不是为了讲故事，而是不断循环，不会产生大量的视觉闪光和混乱。

要设置关键帧，请选择一个骨骼并按 I。Boom！您正在路上。

在制作所需帧的动画后，将其约束到这些帧。如果动画在第 30 帧结束，请确保 Start 为 1 且 End 为 30，而不是更大或更小。

打开 Dope Sheet 并找到 Action Editor：

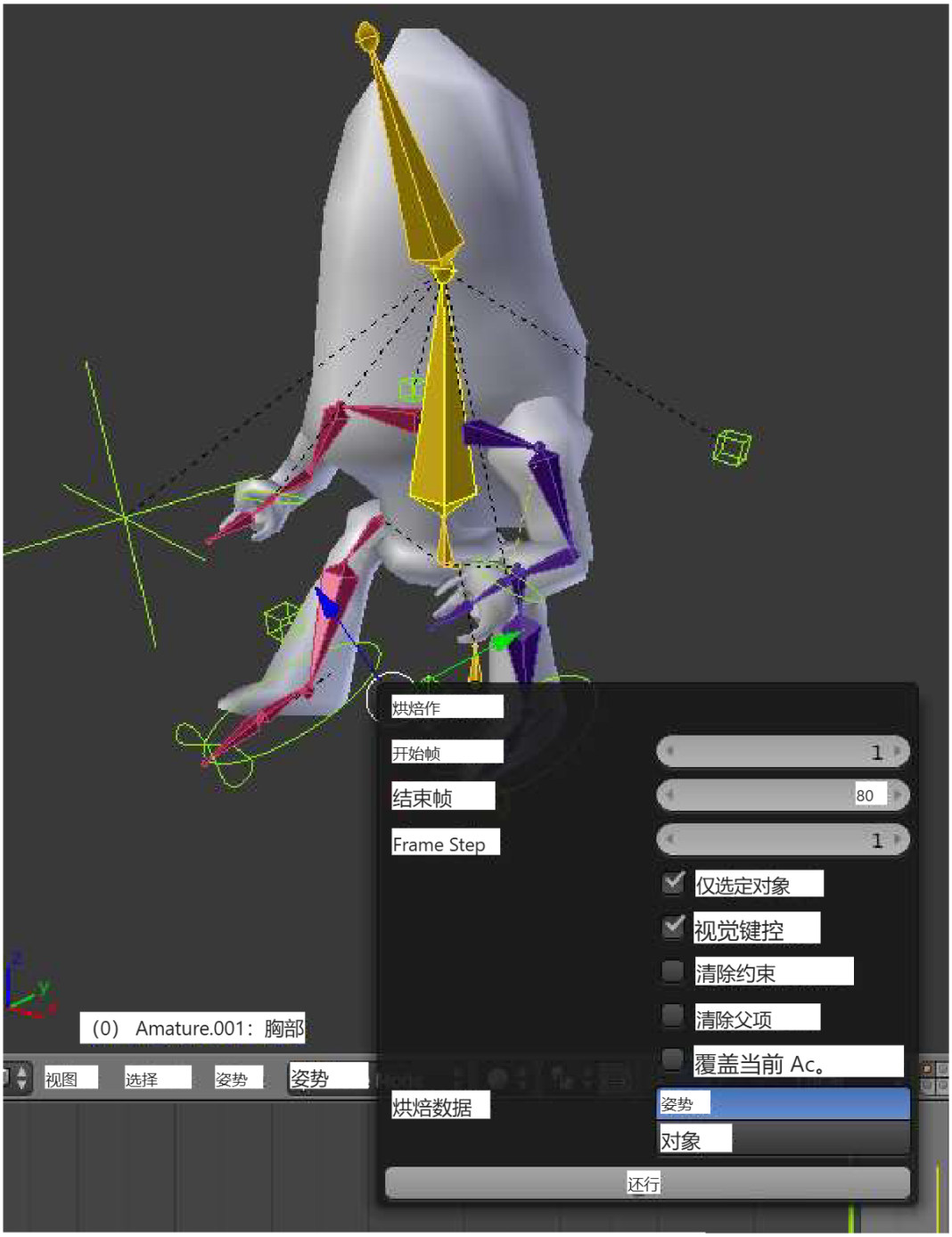


在这里，我制作了一个基本的行走循环，您想在时间轴上方的右侧命名您的作，并确保按大写的 F 保存它。

准备导出动画

第一步是将动画格式化为导出到 Clausewitz 引擎中，而不会导致其崩溃。首先，我们要烘焙动作，这会记录每个骨骼移动的每一帧的位置，而不仅仅是关键帧。

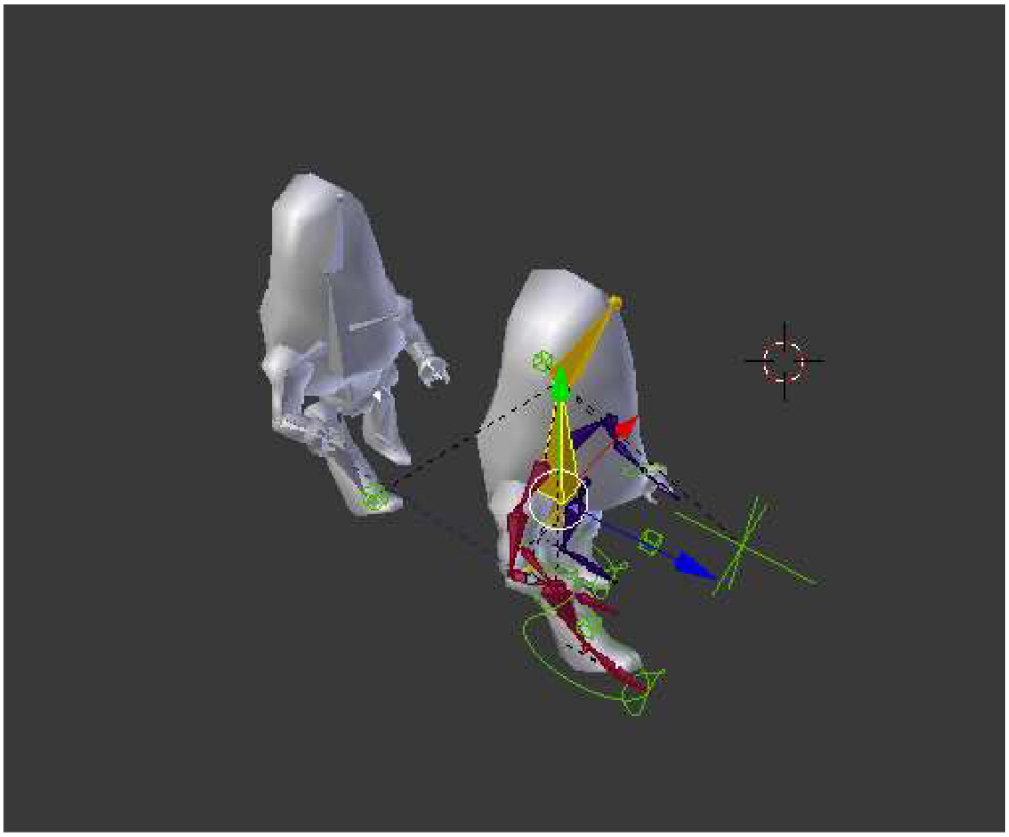
将光标置于 3D 视窗中，然后按空格键。导航到 Bake Action 并复制我的设置。



确保同时单击小 F 并命名您的新作！

在包含新动作的 Action 编辑器中，按 Shift-O 对关键帧进行采样，然后每帧放置一个关键帧，即使骨骼没有移动。

接下来，我们需要制作一个导出装备。这将使我们的 rig 正常运行，并且不会严重崩溃。在对象模式下，使用 Shift-D 复制您拥有的装备。右键单击并进入编辑模式。删除所有 IK 骨骼。再次烘焙您的动作，但这次选中 Clear constraints and Overwrite current action，这将删除 IK 的所有痕迹。这似乎有悖常理，但是一旦所有动画在 IK 绑定上烘焙过一次，您就可以转到动作编辑器，并在姿势模式下使用新的导出绑定选择烘焙动画，效果很好！



这是我的 lk 绑定（fron）和后面的烘焙绑定（bake rig），两者都使用了烘焙动画。

Al 准备开始将我们美丽的模型放入游戏中！

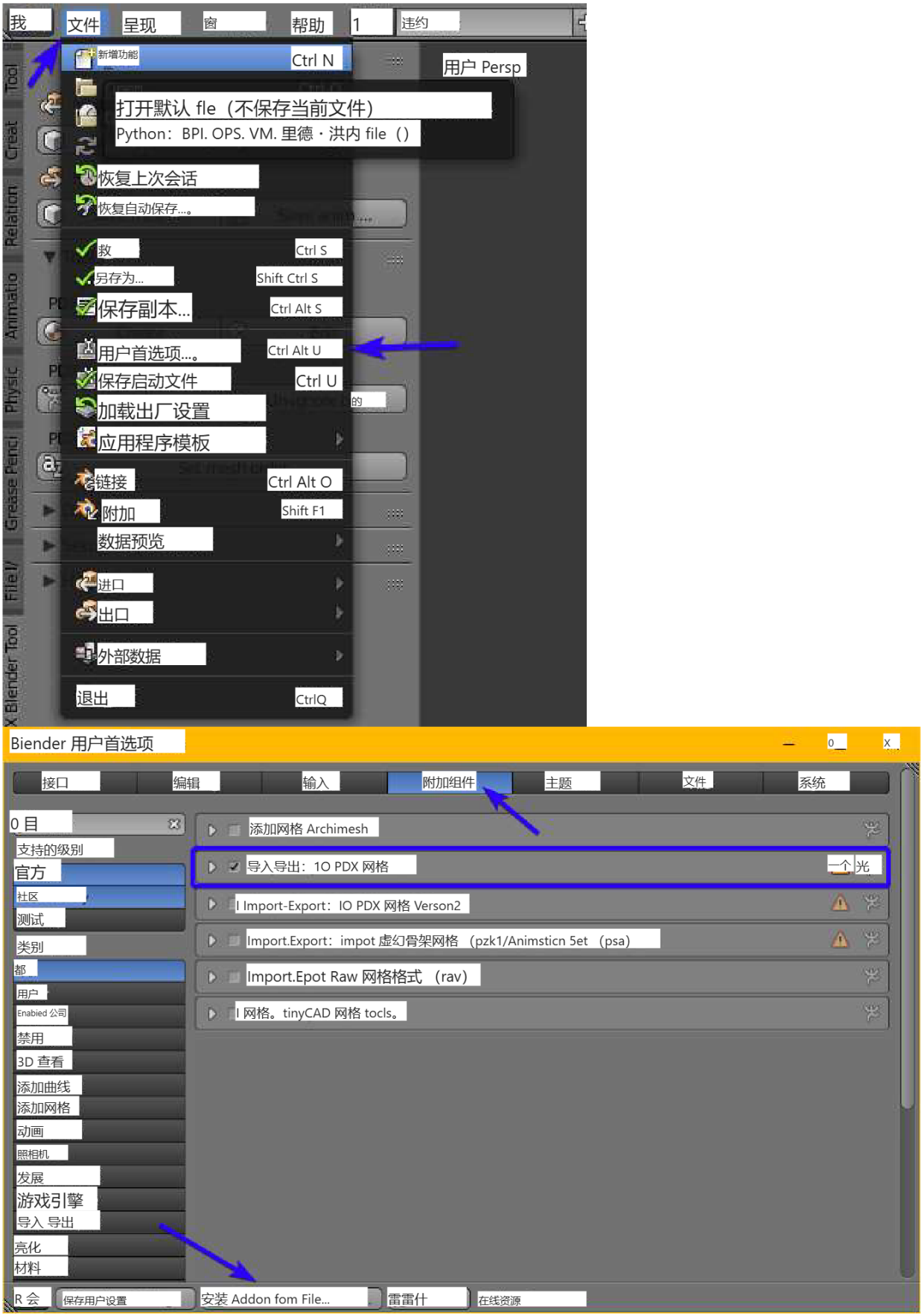
第 4 部分：导出

截至 2019 年 5 月，我们现在可以使用 blender 中的插件直接导出为 PDX 网格和 PDX 动画文件！这还有一个好处，即允许在文件中应用着色器对象、网格索引和更复杂的绑定设置。

插件安装

您可以在此处添加插件： https：/q/ithub.com/ross-qlio pdx mesh

以下是安装说明



PDX 材质创建器

导出的第一步是创建要导出的 PDX 材质。为此，我们转到 PDX Material > Create。给它一个你能认识的名字，然后给它分配一个着色器。这些着色器可以在底部的 “（game）ygfx/fx/pdxmesh.shader” 文件夹中找到。通常，您需要 PdxMeshAdvanced，但有时需要其他着色器或自定义着色器。确保将新材质分配给您的网格。

可以通过选择名称，然后单击左侧面板中的 edit 来编辑此材质。

导出网格

要导出网格，请确保已选择网格。确保它应用了您的材质。我不喜欢每个文件只有一个导出，否则您可能会得到一些意外导出。

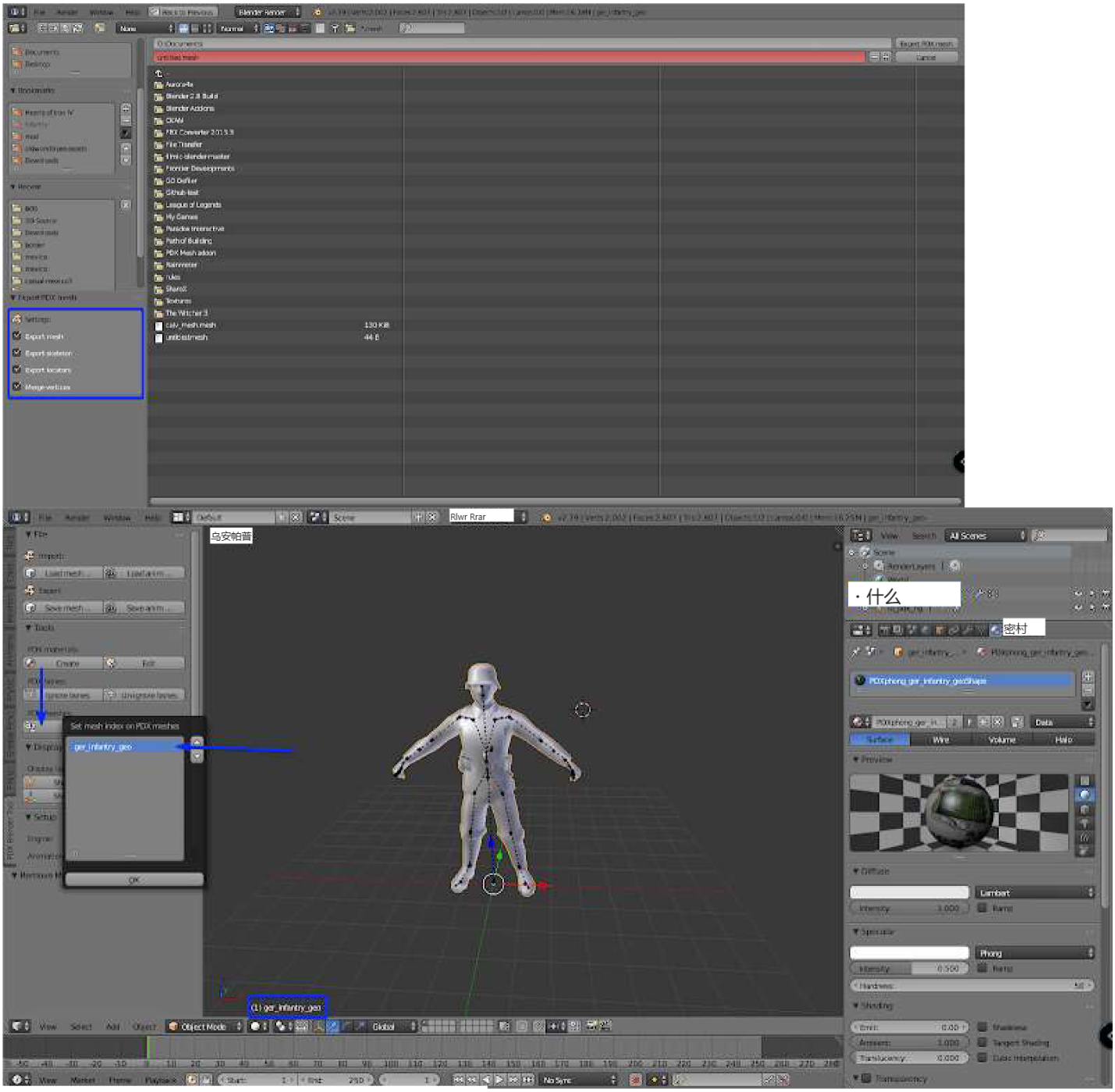
现在是索引网格的时候了！在此处使用这个方便的按钮为您的网格编制索引（从 0 开始），以便您稍后可以在 .gfx 文件中引用它们。

您可以在大纲视图中或通过选择对象在左下角查看对象的名称。

确保您的方向如下，以获得最佳结果。
应用所有变换和旋转 （Ctr-A），因此对象的局部位置是世界原点，有 n 次旋转。面对一切，使 Y - 为 FORWARD 且 Z+ 向上。

对于平面网格（肖像），您需要确保您的法线朝向 Y，并应用了变换！

现在我们只需单击 export mesh，并选择网格（不是骨架）。

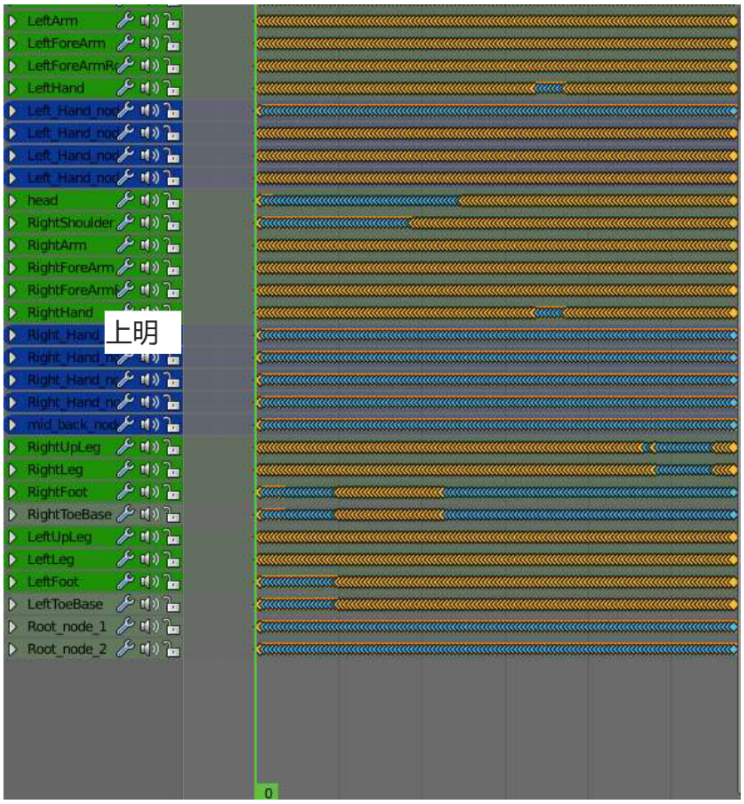


我强烈建议你把所有这些都选中。

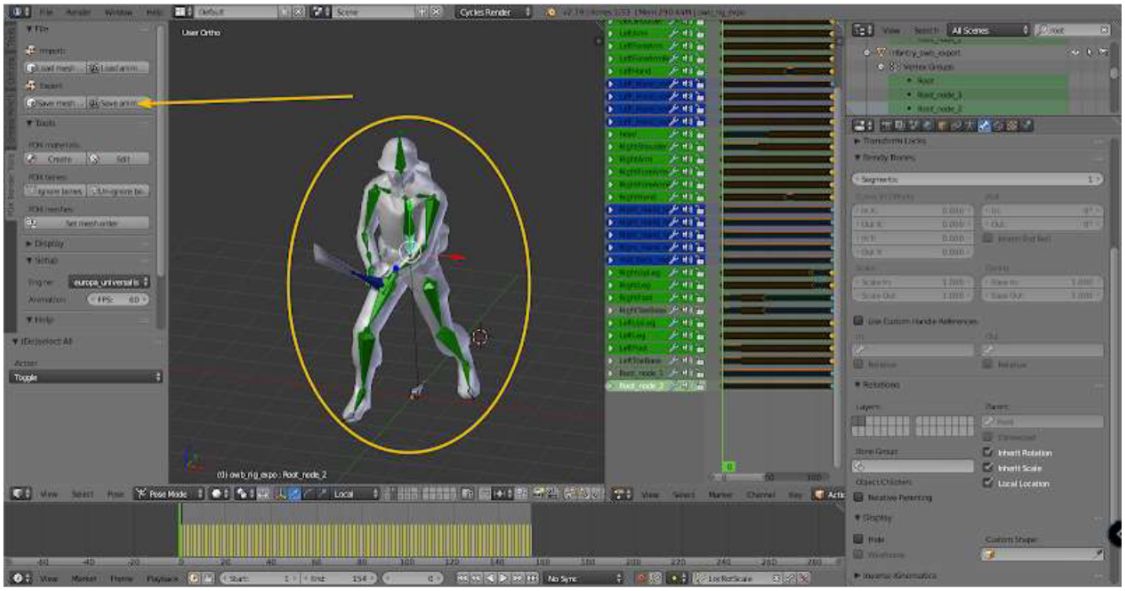
要导出带有骨架的网格，请确保您的网格是骨架的父子对象，骨架修改器位于以骨架为目标的网格上，并使用顶点权重作为选项。我更喜欢将根骨骼朝后，但只要网格以中性姿势面向 Y-，就没关系。

导出动画

动画也很容易导出。这里最重要的两件事是确保你的装备上没有变换，并且你的关键帧是烘焙的。



现在，我们可以转到我们的 rig，其中只有变形骨骼，没有要导出的约束。在 pose 或 object 模式下选择骨架。确保将时间轴设置为动画长度。减去一帧，所以你不会得到 Loop-Seam（由于你的第一个和最后一个关键帧相同，因为你有 2 个完全相同的帧）。按下保存动画，然后砰！它就是好用！确保仔细检查您的动画速度（默认为 15 fps）。我更喜欢 60 fps，因为它可以制作出非常流畅的动画。游戏可以进行更高的插值以考虑刷新率（如果需要）。



第 5 部分：勾搭 - Up

.mesh 文件

纹理位置仅从与 .mesh 本身相同的目录中读取！

.gfx 文件

.gfx 是你的小模型在模型事物的悖论数据库中的 ife。这里我们声明：animations、shader、textures、.mesh 和 scale。它也用于其他一些不值得一提的事情，例如

它们似乎从来没有相关性。

```
objectTypes = {
   pdxmesh = {
            name = "protectron_mesh"
            file = "gfx/models/OWBEntity/robots/protectron.mesh"
            scale = 0.45

            animation = { id = "idle"
                  type = "protectron_idle_anim" }
            animation = { id = "idle2"
                  type = "protectron_idle_2_anim" }
            animation = { id = "attack"
                  type = "protectron_attack_anim" }
            animation = { id = "move"
                  type = "protectron_walk_anim" }
            animation = { id = "retreat"
           type = "protectron_retreat_anim" }


            meshsettings = {
                    name = "jorodoxShape"
                    index = 0
                    texture_diffuse = "protectron_diffuse.dds"
                    texture_normal = "protectron_normal.dds"
                    texture_specular = "protectron_specular.dds"
                    shader = PdxMeshAdvanced
            }
   }
}
```

所以让我们逐步完成这个：
首先，您有 .gfx 文件的标题

```
objectTypes = {
```

我们总是需要这个在顶部，在底部关闭

```
pdxmesh = {
            name = "protectron_mesh"
            file = "gfx/models/OWBEntity/robots/protectron.mesh"
            scale = 0.45
```

这里我们有我们的对象类型、一个 pdx 网格、这个 GFX 对象的内部名称、来自 your.mod 目录的 ABSOLUTE 文件路径（在我的例子中是 oldworldblues）和比例。

```
  animation = { id = "idle"                    type = "protectron_idle_anim" }
            animation = { id = "idle2"
                  type = "protectron_idle_2_anim" }
            animation = { id = "attack"
                  type = "protectron_attack_anim" }
```

接下来，我们声明此网格的动画。有两部分，ID 和类型。ID 是在 .asset 文件内的实体块中引用的内容，用于实际播放动画。Type 是动画的内部寄存器，它也在 a.asset 文件中声明。保持这两者的不同很重要，ID 是通用的，类型是唯一的名称。例如，我的所有模型都使用同一组 ID，因此我确切地知道每个频率下哪个动画是做什么的，但我的类型始终是 model_animation_anim。

```
meshsettings = {
                    name = "jorodoxShape"
                    index = 0
                    texture_diffuse = "protectron_diffuse.dds"
```

```
            texture_normal = "protectron_normal.dds"
            texture_specular = "protectron_specular.dds"
            shader = PdxMeshAdvanced
        }
```

这些是在 .mesh 中找到的网格设置，您可以在此处进行编辑！重要提示：如果使用 jorodox 工具导出，则名称不应更改，索引是导出中的网格，并且仅在将多个网格导出到一个文件时才重要。这就是您之前设置的名称和索引很重要的地方！如果文件中只有一个对象，则可以省略这两行。纹理声明可以采用任何文件路径，但需要注意的是：它是相对于我们之前声明的 .mesh 文件路径的。例如：our.mesh 在 mod/gfx/models/mymodelmesh.mesh 下，如果我们把纹理放在 mod/gfx/models/mymodel/textures/ 中，就可以把它们说成 /textures/texture.diffuse

.asset 文件：动画

这是对 a.asset fle 怪物的第一次介绍。首先，我们将触摸 aniations，因为这是我需要的 .gfx。

.asset 文件类型上没有标头

```
animation = {
    name = "protectron_idle_anim"
    file = "robots/protectron_idle_anim.anim"
}
```

这是 animation 块。我们有名称，它在 .gfx 中被引用为 type（还记得吗？和文件。该路径是 RELATIVE 的，用于声明动画的 .asset 文件。

实体

这是将完全自定义模型添加到任何 Paradox 游戏中最复杂的部分。它处理动画、动作、网格、附件、作品。将尽我所能分解每个部分，但此文件的各个部分特定于 HOI4。动画状态在其他游戏中可能不起作用，但通用文件将说明这一点：我在 Stellairs 中有我的动力装甲模型，带有动画：

到我们实体的完整代码中：

```
entity = {
    ###Name - entity name, structed: unitname_entity
```

```
    name = "power_armour_entity"
    ####paradox mesh - inside a .gfx file (powerarmor_meshes.gfx)
    pdxmesh = "powerarmor_t51_mesh"

    ####Animations and triggers

    ###state to be in when nothing else is triggered
    default_state = "idle"
    ####state to use when attacking an enemy
    ###second attack state (shooting gun)
    state = { name = "attack"                    animation
= "attack" animation_blend_time = 0.0  animation_speed = 1.0  looping
= no chance = 3 next_state = "attack" propagate_state = { rifle1 = "idle" } }
    state = { name = "attack"                    animation
= "shoot" animation_blend_time = 0.0  animation_speed = 1.0 looping
= no chance = 1 next_state = "attack" }
    ###defensive state
    state = { name = "defend"                    animation
= "shoot" animation_blend_time = 0.0  animation_speed = 1.0 }
    ####supporting another attacking unit
    state = { name = "support_attack"    animation
= "shoot" animation_blend_time = 0.0  animation_speed = 1.0 }
    ###movement on map with no enemies
    state = { name = "move"                          animation = "move"
 animation_blend_time = 0.0 animation_speed = 1.0 propagate_state =
{ rifle1 = "idle" } }
    ###retreat "limp"
    state = { name = "retreat"            animation="retreat" }
    #### death state (despawns after this)
    state = { name = "death"    animation="retreat"    }

    state = { name = "idle" animation="idle1" chance = 6 propagate_state =
{ rifle1 = "idle" } }
    state = { name = "idle" animation ="idle2" chance
= 2 next_state="idle1" propagate_state = { rifle1 = "idle" } }
    ### training state, normally many of these
    state = { name = "training" animation="idle1" chance = 6 propagate_state =
{ rifle1 = "idle" } }
    state = { name = "training" animation ="idle2" chance
= 2 next_state="idle1" propagate_state = { rifle1 = "idle" } }


    attach = {     name = "rifle1"    Right_Hand_Node = "minigun_pa_entity" }

    #attach = {     name = "rifle1"    hand_r = "minigun_pa_entity" }
    #attach = {     name = "rifle2"    hand_l = "laser_rifle_entity" }
    #attach = {     name = "rifle2"    hand_l = "minigun_pa_entity" }
    #attach = {     name = "rifle3"    Root = "laser_rifle_entity" }

}
```

这个文件已经有注释了，但我还是要解释核心功能：

名字：

名称描述如下：

Identifier （子单元类型） （视觉级别）实体（变体编号）

标识是图形区域性（在图形区域性 txt 中声明，并在国家 fief hoi4 中声明）或 hoi4 的国家 / 地区标记，例如德国的 GER。

Subunit type 在您的单元文件中定义。您也可以在此处使用 Sprite 类型（重型、中型、轻型坦克）。对于飞机，您可以使用设备名称

视觉级别与 hoi4 设备级别相关联，并在设备的每个级别中声明。我只使用一个级别，因此我省略了视觉级别，因为级别 1（从 1 开始）不需要数字，但级别 2 需要

xxx_2_entity。
变体编号可让您提供更多风格。例如，为不同的迷彩选项添加更多实体供您的士兵使用。

Stellaris 遵循类似的格式，但在图形文化（标识符）之后，部件名称会列出，如您正在制作的飞船文件内所示。EX mammalian_battleship_bow_04_entity。

Pdx_mesh：
这是我们在 .gfx 文件中声明的名称，我们更早地做到了！

克隆：

```
clone = "light_robot_entity"
```

此选项将继承克隆的任何实体的所有代码，从而允许新实体内的任何信息覆盖它。当您需要为不同国家 / 地区重新绘制单位的纹理，但希望保留动画而不是复制和粘贴巨大的代码块 60 次时，这非常有用。

动画剖析：
动画是使用 “state” 制作的。这些在你正在修改的游戏中是硬定义的，所以可能需要一些研究。您可以创建自定义状态，我将在本节的子部分中解释如何执行此作。

首先，我们有 default_ 状态。这是实体在任何时候加载时都应该生成的任何内容，无论是缩放、地图视图、生成，还是其他任何方式。

```
    state = { name = "attack"                    animation
= "attack" animation_blend_time = 0.0  animation_speed = 1.0  looping
= no chance = 3 next_state = "attack" propagate_state = { rifle1 = "idle" } }
```

所以这是一个状态。我们有几个组成部分

名称 - 这在很大程度上是硬编码的，但我们可以从硬编码的状态链接自定义状态
Animation - 我们在 .gfx 中定义的动画 ID
Animation blend time （动画混合时间） - 这提供了混合前一状态和此状态的动画的时间（以秒为单位）。

Animation speed - 动画的十进制速度
循环 - 这将作动画，但大多数状态都是连续的，因此此选项可能无关紧要，最好使用它

Chance - 这允许给定状态的随机动画。例如，这里：我们有一个正在运行的动画，它偶尔会开枪，而不是只是站着不动地放着单位。
Next state - 在此状态之后要移动到的状态。
传播状态 - 这会强制任何附件过渡到给定状态。例如，在上面，我们强制附加到 rifle1 的任何实体变为空闲状态，在这种情况下，它会强制转轮机枪停止射击，或者它可能会继续下一个动画循环。

Event：这是一个声明，它在 state 的指定时间执行某项作。

```
event = { time = 1.6 node = "muzzle"  particle
= "heavy_tank_attack_barrel_particle" light = "mg_muzzle_flash" keep_particle
= yes sound = { soundeffect = heavy_armour_fire  }  }
```

Time - 执行某项作的时间（以秒为单位）
Node - 这是一个定位器对象，可以放置在编辑器（Blender 或 Maya）中，也可以手动创建
Particle - 在节点上生成的粒子效果
Light - 在节点上生成的光（照亮地图上的其他对象）
Keep_particle - 在状态结束时保留粒子、EX shells、fire 等。
声音 - 不言自明
Entity - 此时生成给定的实体。Keep 粒子将在 kep_particle = no 时将其消失

附加实体：

根据游戏的不同，可以通过两种方式将 Enties 附加到其他实体。首先是通过定位器。这些对象与定位器对象一起放置在 Maya 中，或者手动编码，如下所示：

```
locator = { name = muzzle position = { 0 0 0 } rotation = { 0 0 0 } }
```

这是不言自明的。位置和旋转基于对象的局部轴以三个欧几里得变换的形式给出。我从来都不太确定哪个是哪个，因为从 blender 导出使用不同的坐标系，但我相对确定 X 是正确的 ef，Y 是向上的，Z 是 "向前" 和 "向后"。旋转在同一轴上以度为单位。如果你用它做一些奇怪的事情，这将是万向节锁定的，所以你可能需要使用两个附加在下面方法的实体。我们基本上是从对象的原点输入一个转换矩阵，因此知道原点在哪里很重要！

附加是另一种方法，它使用关节本身。attach={name ="rifle1" Left_Hand_node="protectron_aser_un_")

名称 - 任意到您需要的内容

XXXX = 这是骨骼的名称，在本例中 Left Hand_node 是我在 blender 中的装备中的骨骼

"Xxx" 这是您要附加的实体。

不幸的是，没有旋转，所以你的骨骼必须是正确的方向！

自定义状态和传播的魔力：

可以这么说，不能强迫实体进入自定义的动作状态。自定义状态允许进行一系列作。例如，如果你想制作一把 stellaris、charge、fire、recoil 和 return 的枪，使用 3 种状态比一个长长的动画和一堆事件要容易得多。为什么？由于每个动画和计时都可以单独更改，因此请允许添加 CHANCE。Chance 允许动画链的进展发生变化，从而比其他方式更能为视觉效果增添趣味。自定义状态的创建方式与普通状态类似，但您必须使用 next_state = "custom_ state" 来触发它们，而硬编码状态只会触发。

传播和自定义状态的魔力在 Protectron 中被用来使其在 Old World Blues 中充满激情。我使用一个带有位置的空网格，附加到手上。然后，它有一个自定义状态，当我需要激光粒子时，我会传播它。它触发我的激光对象，该对象在手部生成另一个实体，该实体被设置为使激光粒子正常工作。

您的自定义模型现在可以在游戏中使用！祝贺

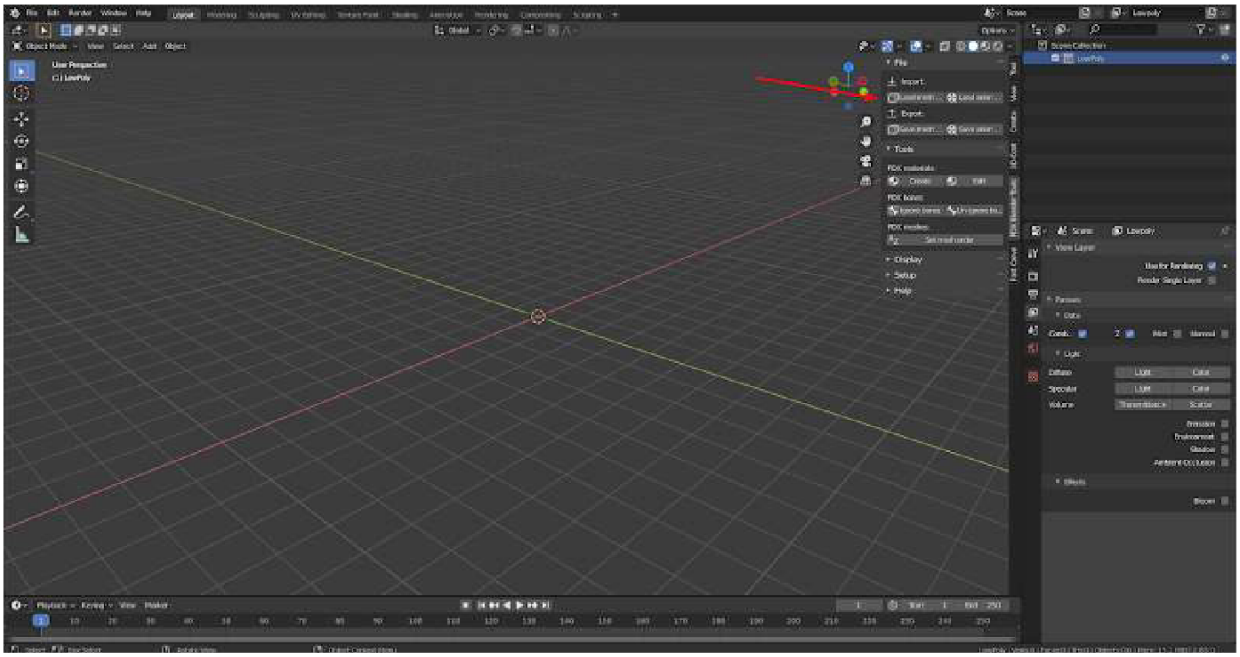
第 6 部分：粒子

一旦您模糊地了解了系统的工作原理，添加和编辑粒子（特别是在 HOI4 和 Stellaris 中）就相当容易了。这项工作中使用的最重要的工具是 PDX Particle Editor。它与 HOI4 和 Stellaris 捆绑在一起，我没有 EU4 或 CKII，所以我无法测试它们。要打开编辑器，请将启动选项设置为 -editor ONLY。启动时，它将全屏打开，并且可能不会响应。给它一点时间，它必须加载所有图形才能让您能够编辑它们。在此编辑器中，您可以调用实体、网格和粒子。可以检查纹理等的网格，可以检查粒子、动画和状态的实体。Particles 允许您完全更改和保存新粒子。

Particle Asset 文件

第 7 部分：其他

导入 Vanilla 网格文件



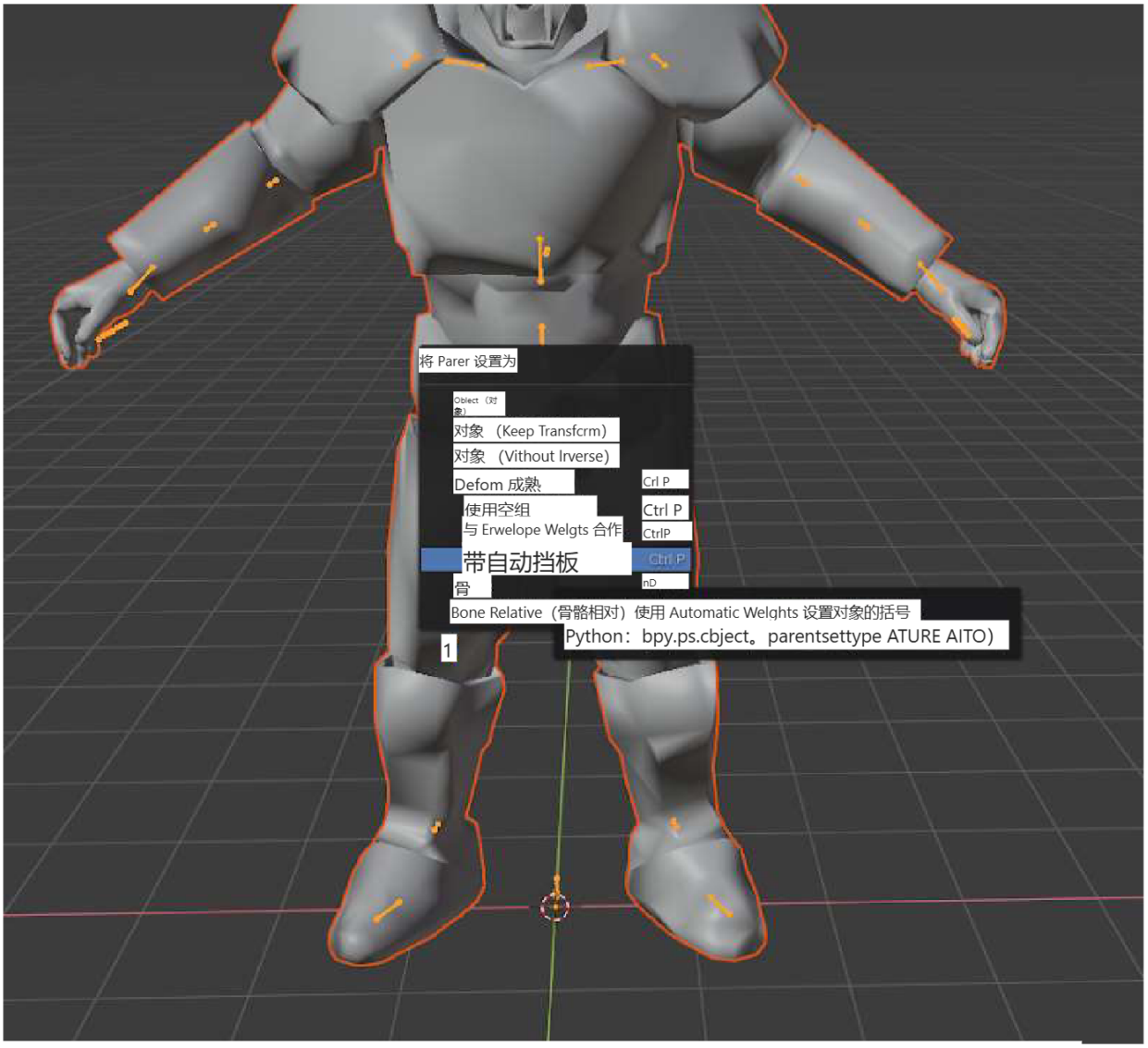
使用 PDX io 插件，可以轻松地从用于导出的同一菜单中导入网格。导入时，网格包含几个部分：

每个网格都附带一个着色器，其中存储了 Clauswitz 的数据。AI Iso 将网格与顶点组 / 权重和装备一起嵌入。

将新网格导出到 Vanilla 骨架上

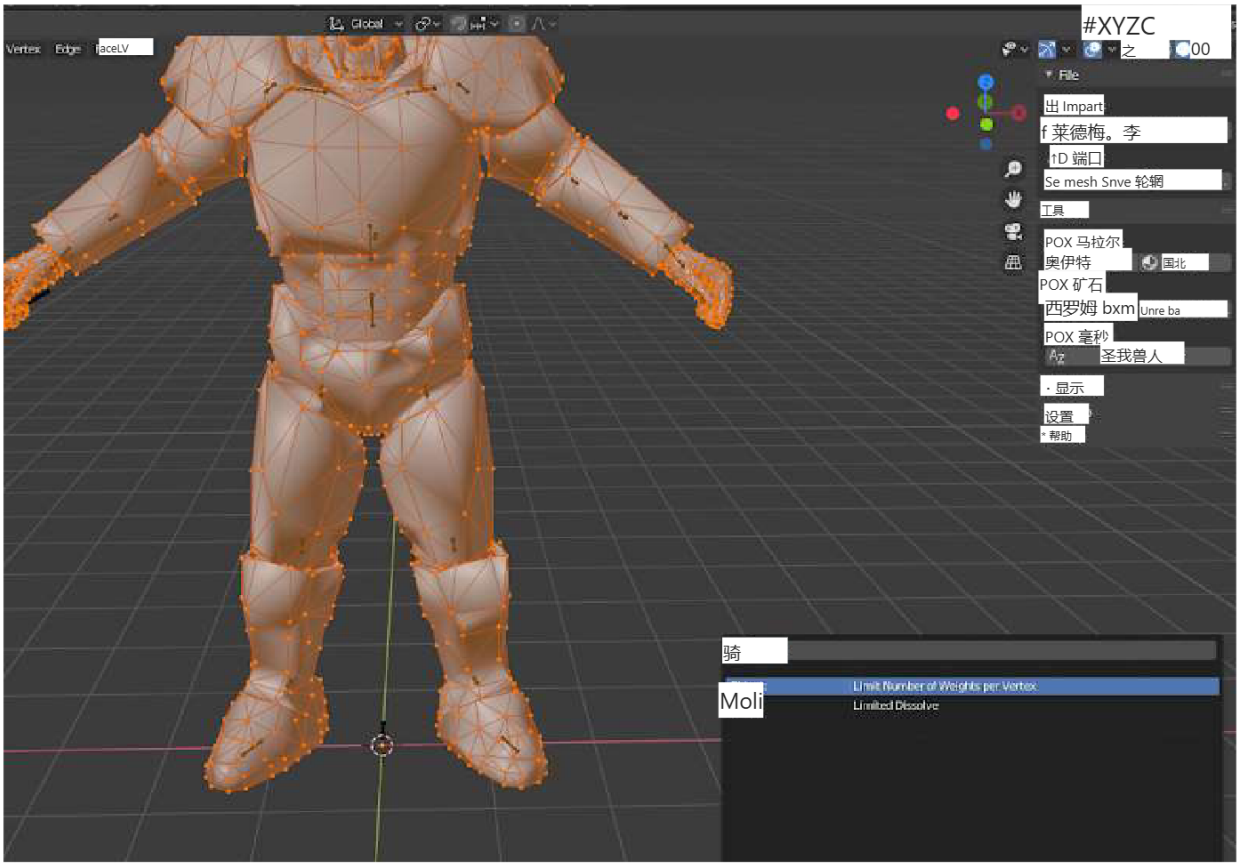
在开始之前，请确保网格没有指南开头提到的任何错误。检查流形几何体（itte 或无孔）、三角化几何体（或所有四边形）以及没有垂直的非流形面。

要将新网格导出到原版骨架上，首先必须将新网格对象设置为现在无网格骨架的父级。

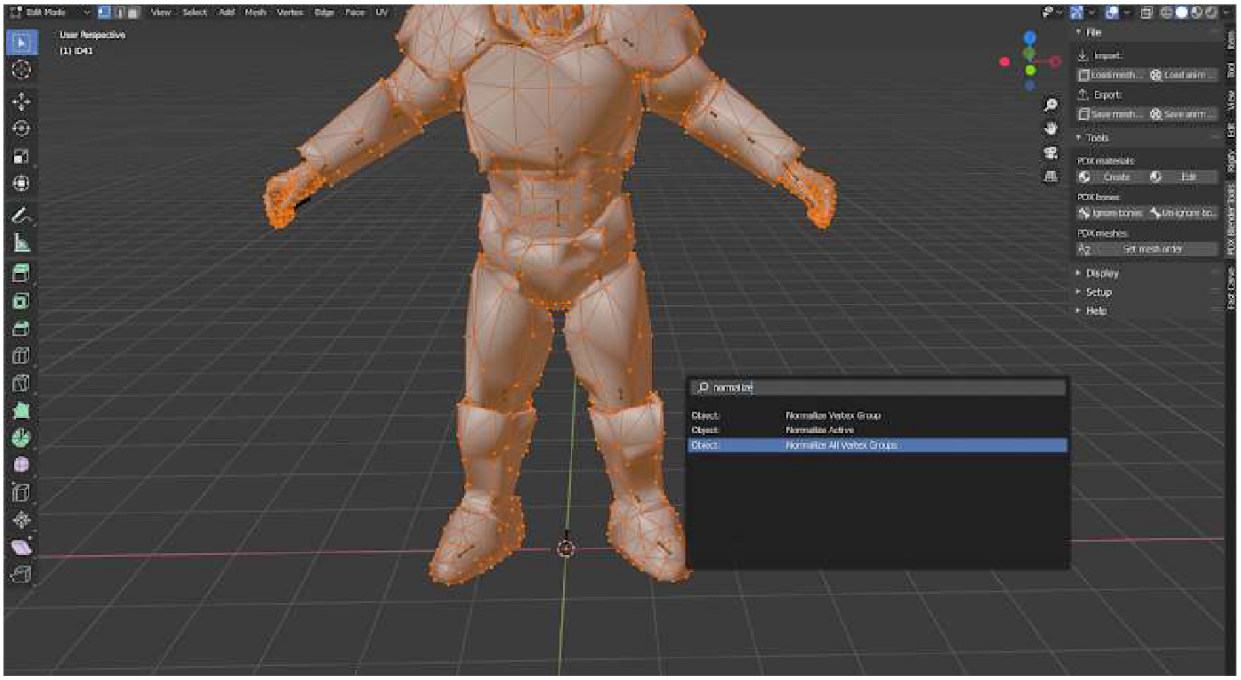


确保网格处于选中状态，并且骨架是活动选择（橙色高亮显示）。

结果就是这样，为装备上的每个骨骼创建一个顶点组。接下来，我们要选择装备并按 Tab 键进入编辑模式



然后，我们希望将对任何顶点的影响量限制为 3。为此，请使用 A 选择 all，然后按空格键并搜索 limit。选择 Limit number of weightes per vertex（限制每个顶点的权重数）。选中后，将设置更改为 3。保持网格处于选中状态，然后我们想要对所有顶点组进行规格化，因此顶点上的权重为 1。



就是这样！按照 第 5 部分 中的步骤完成作，您的模型现在已正确加权到基本游戏网格。