我有一个游戏引擎，但是现在我面临一个问题就是这个引擎由于过于老旧，其中的模型插件部分支持早期的3dmax和maya版本。现在我的需求是我希望能够在blender4.5.3版本中能够实现它的功能，也就是实现我导出模型 骨骼 贴图 以及动画等内容到这个引擎中，让这个引擎能够正确识别加载使用模型。那么现在我有这个引擎的教学文档，我需要你学习一下，它的整个模型导出构建流程，还有模型的相关数据写法参数等等，作为参考来帮我构建这个blender插件。现在我提供给你相关的信息，<https://howarduong.github.io/github.io/doc/documentation_roadmap.html> 这是完整的教学文档它包含了所有的教学流程作。<https://howarduong.github.io/github.io/content_creation_page.html> 这里是关于模型创建的教学文档和作。<https://github.com/howarduong/BigWorld-Engine-2.0.1.git> 这是引擎的全部源码，其中工具源码部分应该包含了3dmax 与maya 插件的源码，同时其中一些基础源数据应该包含了对格式的描述。<https://github.com/howarduong/github.io/blob/main/base.model> 这是一个基础模型的明文内容，里面应该包含一些标准的写法语法以及格式。<https://github.com/howarduong/github.io/blob/main/base.visual> 这是模型的配套文件，也是明文的，当然也包含了标准的写法语法以及格式。<https://github.com/howarduong/github.io/blob/main/handrails.mfm> 这个文件我不知道作用有多大应该也很重要。<https://github.com/howarduong/BigWorld-Engine-2.0.1/tree/481e69a837a9f6d63f298a4f24d423b6329226c6/src/tools> 这里应该就是它的模型相关源码内容。所有的这些资料你都可以检索查看学习用于实现我们本次要达成的目标。现在我在明确一下我的需求，我需要你给我一份插件设计方案，需要包含这个插件的目录结构，架构，以及写法思路，还有在我们的blender中要有ui界面，能够直接为我们直观的使用，最好是贴合3dmax 功能差不多的复刻，而且要考虑到3dmax中我们的软件运行流程与blender中应该是有差异的，所以要注意按照blender4.5.3版本的语法，标准和写法来设计这个插件。最终的目的是我们要在blender4.5.3版本中正确导入这个插件，并且能够设定bigworld的一些参数要求，然后最终能够导出为bigworldengine能够正确加载的模型 动画 骨骼 贴图 等模型数据。现在你给我出第一版方案吧

现在你作为我的程序员，为我写出架构中每个代码文件的代码，要考虑全局适配关联引用等问题，然后将每个文件的完整代码发给我，我不需要最小示例，我需要你写完整，然后我这边直接复制到本地进行测试，每个文件需要明确的告知位置，然后所有的按键输入采用英文，但是需要有中文解释，也就是说我在blender中鼠标指向任何按键选项都会有中文提示。还有它们的依赖关系是的内容应该是相对路径，导出到同一文件夹即可，动画在文件夹中建立子文件夹存放。

**BigWorld Blender 4.5.3 插件设计方案**

**一、项目概述**

**1.1 目标**

为Blender 4.5.3开发一个完整的BigWorld引擎导出插件，支持：

* 静态模型导出（.model + .visual + .primitives）
* 骨骼动画导出（.animation）
* 材质贴图配置
* 碰撞体导出
* LOD支持

**1.2 核心文件格式说明**

根据分析，BigWorld引擎使用以下文件格式：

* **.model**: XML格式的模型配置文件，包含visual引用、材质、碰撞等信息
* **.visual**: XML格式的渲染配置文件，包含骨骼、渲染集、材质绑定
* **.primitives**: 二进制格式的顶点数据文件，包含实际的网格数据
* **.animation**: 二进制格式的动画数据文件
* **.mfm**: 材质定义文件（Material Format）

**二、插件目录结构**

bigworld\_blender\_exporter/

│

├── \_\_init\_\_.py # 插件初始化和注册

├── bl\_info.py # Blender插件元信息

├── config.py # 配置常量和默认值

│

├── core/ # 核心功能模块

│ ├── \_\_init\_\_.py

│ ├── exporter.py # 主导出器类

│ ├── model\_processor.py # 模型处理逻辑

│ ├── animation\_processor.py # 动画处理逻辑

│ └── material\_processor.py # 材质处理逻辑

│

├── formats/ # 文件格式处理

│ ├── \_\_init\_\_.py

│ ├── model\_format.py # .model文件生成

│ ├── visual\_format.py # .visual文件生成

│ ├── primitives\_format.py # .primitives二进制生成

│ ├── animation\_format.py # .animation文件生成

│ └── material\_format.py # .mfm材质文件生成

│

├── ui/ # 用户界面

│ ├── \_\_init\_\_.py

│ ├── panels.py # 主面板定义

│ ├── operators.py # 操作器定义

│ ├── properties.py # 自定义属性

│ └── preferences.py # 插件偏好设置

│

├── utils/ # 工具函数

│ ├── \_\_init\_\_.py

│ ├── math\_utils.py # 数学转换工具

│ ├── binary\_writer.py # 二进制写入工具

│ ├── xml\_writer.py # XML生成工具

│ ├── validation.py # 数据验证

│ └── logger.py # 日志系统

│

├── presets/ # 预设配置

│ ├── default.json # 默认导出设置

│ ├── character.json # 角色导出预设

│ └── static.json # 静态物体预设

│

└── docs/ # 文档

├── README.md # 使用说明

├── CHANGELOG.md # 版本记录

└── API.md # API文档

**三、核心架构设计**

**3.1 数据流程**

Blender场景数据

↓

数据收集器 (Collector)

↓

数据验证器 (Validator)

↓

坐标系转换器 (Transformer)

↓

格式转换器 (Converter)

↓

文件写入器 (Writer)

↓

BigWorld文件 (.model, .visual, .primitives, .animation)

**3.2 主要类设计**

**3.2.1 BigWorldExporter（主导出器）**

class BigWorldExporter:

def \_\_init\_\_(self, context, settings):

self.context = context

self.settings = settings

self.model\_data = {}

self.visual\_data = {}

self.primitives\_data = {}

def export(self):

# 主导出流程

pass

def collect\_data(self):

# 收集场景数据

pass

def validate(self):

# 验证数据完整性

pass

def write\_files(self):

# 写入各种文件

pass

**3.2.2 PrimitivesData（顶点数据结构）**

class PrimitivesData:

"""

BigWorld顶点格式: "xyznuviiiwwtb"

x,y,z - 位置 (float3)

n - 法线 (short2, 压缩格式)

u,v - UV坐标 (float2)

i - 骨骼索引 (byte3)

w - 骨骼权重 (byte2)

t - 切线 (short2)

b - 副法线 (short2)

"""

def \_\_init\_\_(self):

self.vertices = []

self.indices = []

self.format = "xyznuviiiwwtb"

def add\_vertex(self, position, normal, uv, bones, weights, tangent, binormal):

# 添加顶点数据

pass

**3.3 坐标系转换**

BigWorld使用Y-up坐标系，而Blender使用Z-up坐标系，需要转换：

def blender\_to\_bigworld\_matrix():

# Z-up to Y-up conversion

return Matrix([

[1, 0, 0, 0],

[0, 0, 1, 0],

[0, -1, 0, 0],

[0, 0, 0, 1]

])

**四、UI界面设计**

**4.1 主面板布局**

┌─────────────────────────────────┐

│ BigWorld Exporter │

├─────────────────────────────────┤

│ Export Path: [\_\_\_\_\_\_\_\_\_\_\_] [📁] │

│ │

│ ▼ Model Settings │

│ □ Export Mesh │

│ □ Export Skeleton │

│ □ Export Animation │

│ □ Export Materials │

│ □ Export Collision │

│ │

│ ▼ Advanced Options │

│ Scale: [1.0] │

│ Coordinate System: [Y-up ▼] │

│ Vertex Format: [Standard ▼] │

│ □ Generate LODs │

│ LOD Levels: [3] │

│ □ Optimize Mesh │

│ │

│ ▼ Material Settings │

│ Texture Path: [\_\_\_\_\_\_\_\_\_\_\_] │

│ □ Copy Textures │

│ □ Convert to DDS │

│ │

│ ▼ Animation Settings │

│ Frame Rate: [30] │

│ Start Frame: [1] │

│ End Frame: [250] │

│ □ Bake Animation │

│ □ Optimize Keyframes │

│ │

│ [Export Selected] [Export All] │

│ │

│ Status: Ready │

└─────────────────────────────────┘

**4.2 属性面板**

在物体属性中添加BigWorld特定属性：

* BSP碰撞类型
* LOD距离设置
* 材质ID映射
* 骨骼限制设置

**五、文件格式实现细节**

**5.1 .model文件格式（XML）**

<root>

<nodefullVisual> models/example.visual </nodefullVisual>

<parent> </parent>

<extent> 10.000000 </extent>

<boundingBox>

<min> -1.0 -1.0 -1.0 </min>

<max> 1.0 1.0 1.0 </max>

</boundingBox>

<editorOnly>

<bspModels>

<model> models/example\_bsp.model </model>

</bspModels>

</editorOnly>

</root>

**5.2 .visual文件格式（XML）**

<root>

<renderSet>

<treatAsWorldSpaceObject> false </treatAsWorldSpaceObject>

<node> root </node>

<geometry>

<vertices> models/example.primitives </vertices>

<primitive> triangles </primitive>

<primitiveGroup>

<material> materials/example.mfm </material>

<startIndex> 0 </startIndex>

<endIndex> 1000 </endIndex>

<startVertex> 0 </startVertex>

<endVertex> 500 </endVertex>

</primitiveGroup>

</geometry>

</renderSet>

<boundingBox>

<min> -1.0 -1.0 -1.0 </min>

<max> 1.0 1.0 1.0 </max>

</boundingBox>

</root>

**5.3 .primitives文件格式（二进制）**

Header:

4 bytes - Magic number (0x42570100)

64 bytes - Format string ("xyznuviiiwwtb")

4 bytes - Vertex count

4 bytes - Index count

4 bytes - Vertex size

Vertex Data:

N \* vertex\_size bytes - Vertex buffer

Index Data:

N \* 2 bytes - Index buffer (16-bit indices)

**六、核心功能实现**

**6.1 网格导出流程**

def export\_mesh(obj, settings):

# 1. 获取网格数据

mesh = obj.data

mesh.calc\_loop\_triangles()

mesh.calc\_normals\_split()

# 2. 构建顶点数据

vertices = []

for loop\_tri in mesh.loop\_triangles:

for loop\_index in loop\_tri.loops:

loop = mesh.loops[loop\_index]

vertex = mesh.vertices[loop.vertex\_index]

# 收集顶点属性

position = vertex.co

normal = loop.normal

uv = mesh.uv\_layers.active.data[loop\_index].uv if mesh.uv\_layers.active else (0, 0)

# 骨骼权重（如果有）

bones, weights = get\_vertex\_weights(obj, loop.vertex\_index)

vertices.append(build\_vertex(position, normal, uv, bones, weights))

# 3. 生成索引

indices = list(range(len(vertices)))

# 4. 写入文件

write\_primitives(vertices, indices, settings)

write\_visual(obj, settings)

write\_model(obj, settings)

**6.2 动画导出流程**

def export\_animation(obj, action, settings):

# 1. 采样动画帧

frames = []

for frame in range(settings.start\_frame, settings.end\_frame + 1):

bpy.context.scene.frame\_set(frame)

# 收集骨骼变换

bone\_transforms = {}

for bone in obj.pose.bones:

matrix = bone.matrix

bone\_transforms[bone.name] = {

'position': matrix.to\_translation(),

'rotation': matrix.to\_quaternion(),

'scale': matrix.to\_scale()

}

frames.append(bone\_transforms)

# 2. 优化关键帧

if settings.optimize\_keyframes:

frames = optimize\_keyframes(frames)

# 3. 写入动画文件

write\_animation(frames, settings)

**七、实现计划**

**第一阶段：基础框架（1周）**

* [ ] 插件基础结构搭建
* [ ] UI面板实现
* [ ] 配置系统

**第二阶段：静态模型导出（2周）**

* [ ] 网格数据收集
* [ ] .primitives格式实现
* [ ] .visual格式实现
* [ ] .model格式实现

**第三阶段：材质系统（1周）**

* [ ] 材质数据收集
* [ ] .mfm格式实现
* [ ] 贴图路径处理

**第四阶段：骨骼动画（2周）**

* [ ] 骨架导出
* [ ] 动画采样
* [ ] .animation格式实现

**第五阶段：高级功能（1周）**

* [ ] LOD生成
* [ ] 碰撞体导出
* [ ] 批量导出

**第六阶段：测试优化（1周）**

* [ ] 功能测试
* [ ] 性能优化
* [ ] 文档编写

**八、关键技术点**

**8.1 顶点格式转换**

BigWorld使用压缩的法线和切线格式（short2），需要特殊处理：

def compress\_normal(normal):

# 将float3法线转换为short2球面坐标

theta = math.atan2(normal.y, normal.x)

phi = math.acos(normal.z)

# 转换到short范围

u = int((theta / math.pi + 1.0) \* 32767)

v = int((phi / math.pi) \* 32767)

return struct.pack('HH', u, v)

**8.2 骨骼权重处理**

最多支持3个骨骼影响，权重归一化：

def process\_vertex\_groups(vertex, vertex\_groups):

influences = []

for group in vertex.groups:

if group.weight > 0:

influences.append((group.group, group.weight))

# 排序并取前3个

influences.sort(key=lambda x: x[1], reverse=True)

influences = influences[:3]

# 归一化权重

total = sum(w for \_, w in influences)

if total > 0:

influences = [(g, w/total) for g, w in influences]

return influences

**九、兼容性考虑**

**9.1 Blender版本适配**

* 使用Blender 4.5.3 API
* 避免使用废弃的API
* 使用新的Overlay系统

**9.2 BigWorld版本支持**

* 支持BigWorld 2.0.1格式
* 预留扩展接口for未来版本

**十、总结**

这个插件设计方案充分考虑了BigWorld引擎的特殊要求和Blender 4.5.3的特性，通过模块化设计确保了良好的可维护性和扩展性。重点实现了：

1. **完整的文件格式支持**：包括.model、.visual、.primitives等所有必需格式
2. **友好的用户界面**：参考3ds Max插件的功能，适配Blender操作习惯
3. **高效的数据处理**：优化的顶点数据结构和压缩算法
4. **灵活的配置系统**：支持预设和自定义设置
5. **完善的错误处理**：数据验证和日志系统

下一步建议从基础框架和静态模型导出开始实现，逐步完善各项功能。