

# 动画大作业 - 结题文档

姓名: 张馨月

学号: 2022012199

## 一、项目概述

本项目实现了一个麋鹿模型的骨架绑定与动画系统，支持命令行批量导出和GUI交互两种运行模式。

### 核心功能：

- 自主实现的蒙皮权重计算（反距离加权 + 区域分割）
- 命令行批处理 + 图形化交互界面
- 实时动画预览与手动姿态调整
- 动画视频导出

项目地址: [https://github.com/Fiorina-moon/CA\\_project](https://github.com/Fiorina-moon/CA_project)

### 完成内容：

- 基本要求：权重计算、视频导出
- 附加要求：GUI交互界面

### 交付文件：

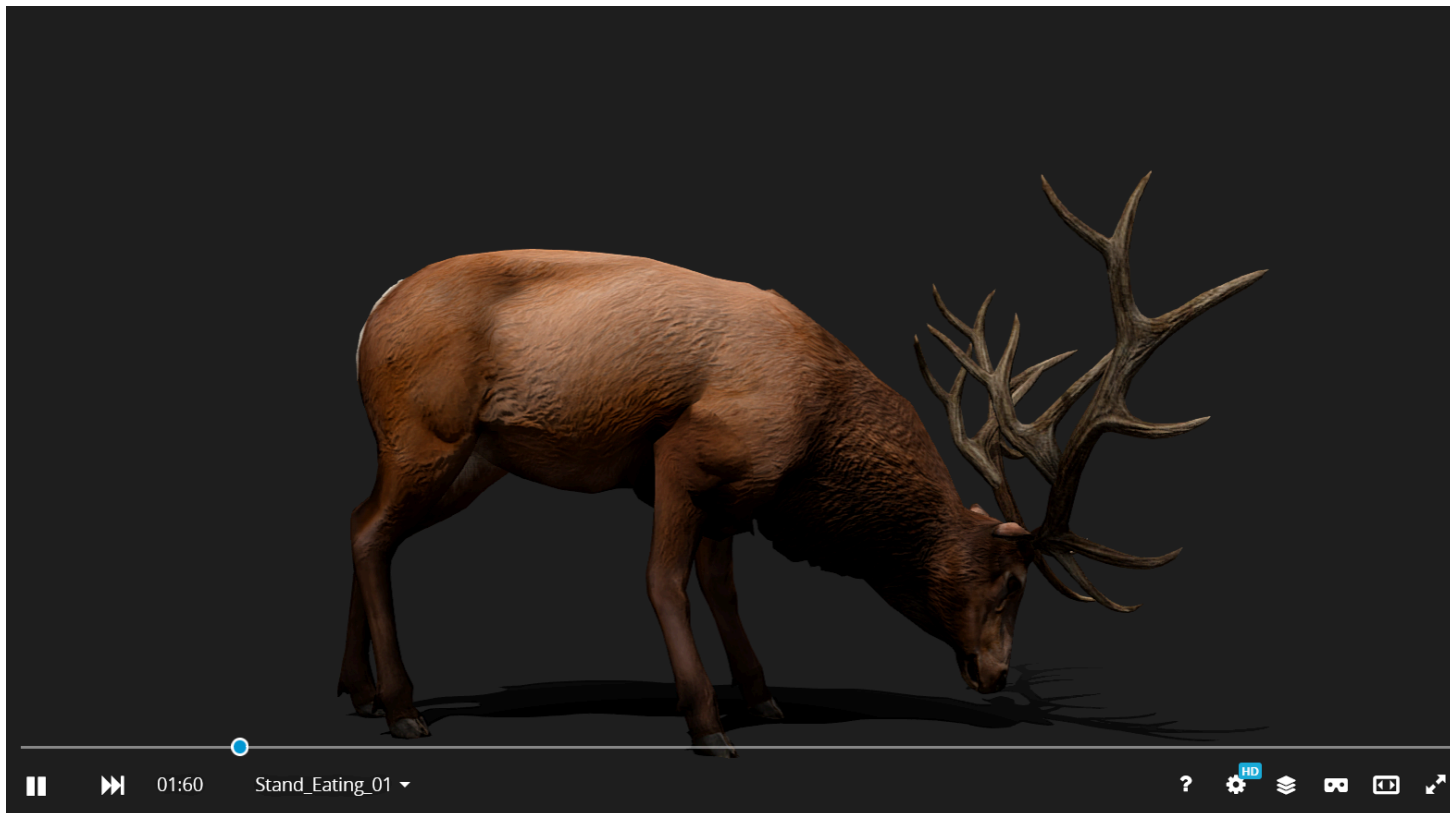
- exe/：可执行文件
- code/：源代码
- doc/：文档

## 二、前期准备

### 2.1 模型获取

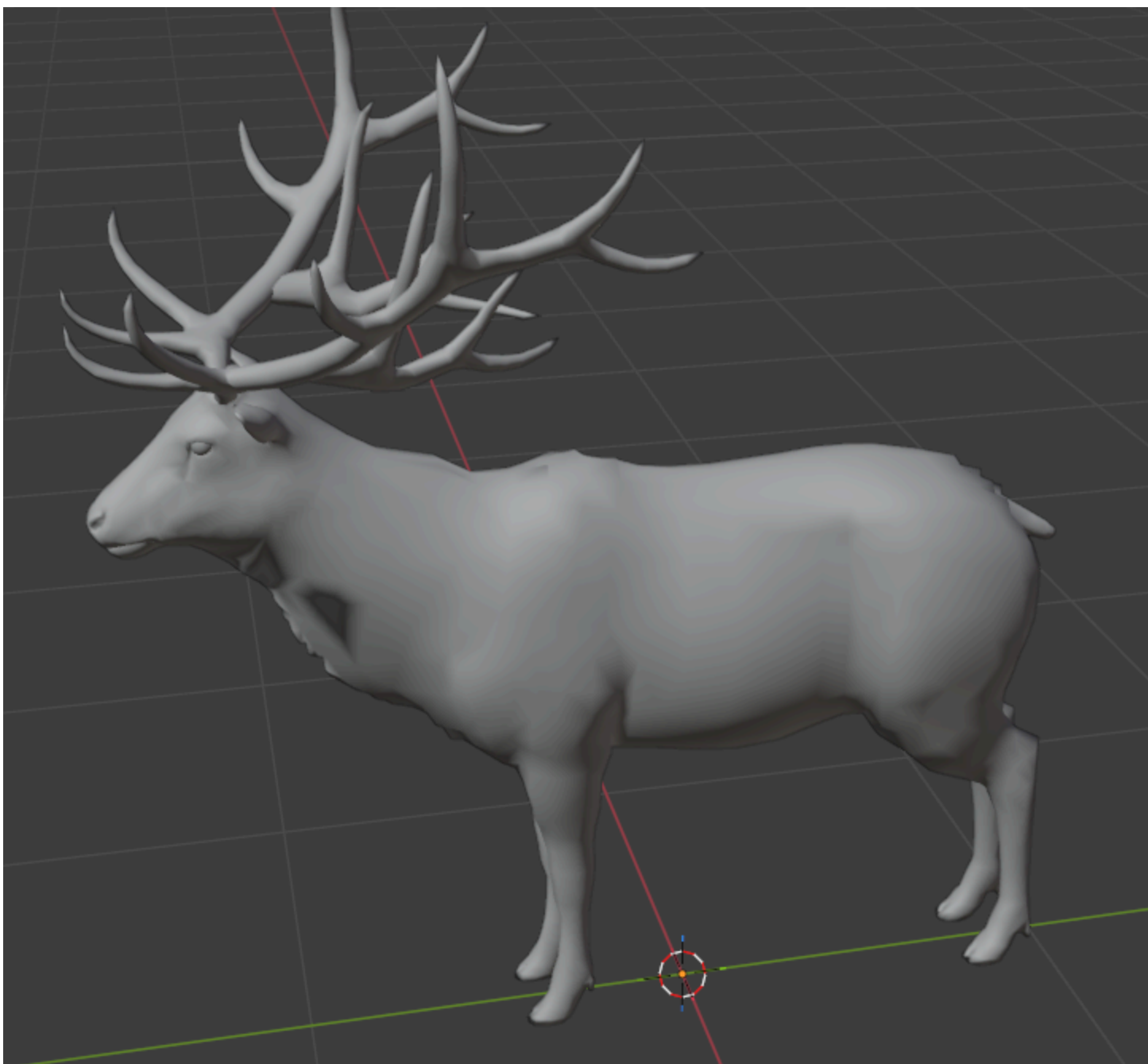
从Sketchfab下载了Realistic Animated Elk 3D Model (WildMesh 3D)，链接：

<https://sketchfab.com/3d-models/realistic-animated-elk-3d-model-free-download-787834f9caa2474d9f1814b807c072d7>



下载格式为.glb。因为下载模型包含骨架等其他信息，需要将其转换为mesh文件。采用的方法是：导入到Blender中，手动删去骨架之后再导出为.obj格式。获得模型文件 `data\models\elk.obj`

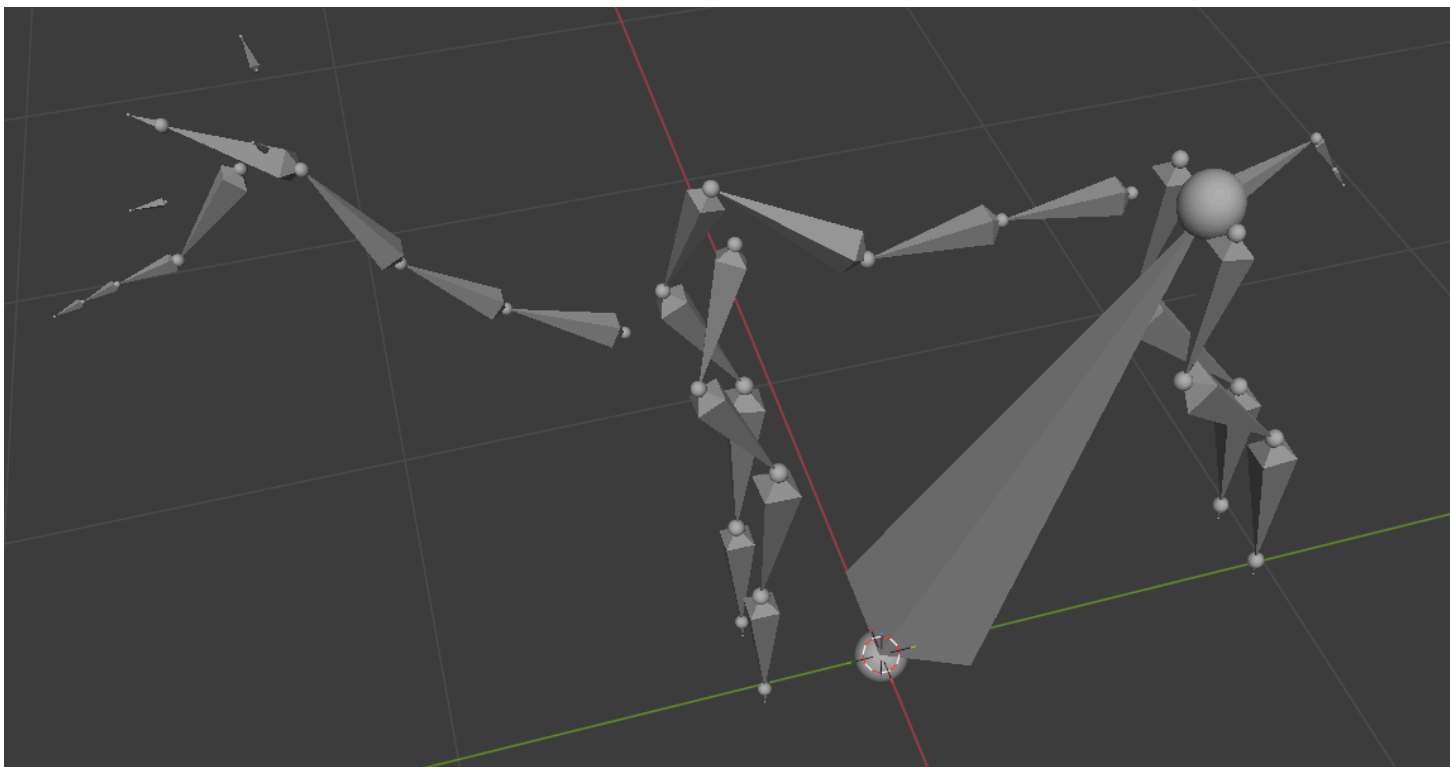
模型效果：



## 2.2 骨架生成

在Blender的控制台中输入脚本导出原始模型文件的骨架获得骨架文件 `data\skeleton\skeleton.json`。由于导出骨架距今时间以及过久，并且当时尝试了多种导出骨架的方法，使用的导出脚本已经丢失。

骨架效果如下：



查看骨架文件可得，模型共有38个关节，37条骨骼，命名基本可读。

```
{
  "hierarchy": {
    "RigRoot_01": "_rootJoint",
    "RigPelvis_02": "RigRoot_01",
    "RigLBleg1_03": "RigPelvis_02",
    "RigLBleg2_04": "RigLBleg1_03",
    ...
  }
}
```

## 三、运行环境与依赖

### 3.1 系统要求

- 操作系统: Windows 10/11
- Python: 3.10.19

## 3.2 依赖库

```
numpy>=1.19.0
PyOpenGL>=3.1.5
PyQt5>=5.15.0
opencv-python>=4.5.0
scipy>=1.7.0
glfw>=2.5.0
```

## 四、项目架构

### 4.1 目录结构

```
CA_project/
├── src/
│   ├── core/           # 网格、骨架数据结构
│   ├── animation/      # 关键帧、插值、播放控制
│   ├── skinning/       # 权重计算、LBS变形
│   ├── rendering/      # OpenGL渲染、视频导出
│   ├── ui/             # PyQt5界面
│   └── utils/          # 数学、几何工具函数
├── data/
│   ├── models/         # .obj模型
│   ├── skeleton/       # .json骨架
│   ├── animations/     # .json动画
│   └── weights/        # .npz权重
├── output/
│   ├── videos/         # 导出视频
│   └── frames/         # 临时帧
├── main.py             # 命令行入口
├── ui_main.py          # GUI入口
└── requirements.txt
```

### 4.2 模块设计

#### **src/core/ - 核心数据结构**

主要文件：

- `mesh.py`：定义 `Mesh` 类，存储顶点、面片、法向量等网格数据
- `mesh_loader.py`：实现 OBJ 格式文件的解析和加载
- `skeleton.py`：定义 `Skeleton`、`Bone`、`Joint` 类，表示骨架层级结构
- `skeleton_loader.py`：从 JSON 文件加载骨架数据

**职责：**

- 提供网格和骨架的数据表示
- 负责从文件加载数据到内存对象
- 提供基本的数据访问接口

## **src/animation/ - 动画系统**

**主要文件：**

- `keyframe.py`：定义关键帧数据结构，存储每帧的关节变换
- `interpolation.py`：实现关键帧之间的插值算法（线性插值、球面插值）
- `animator.py`：动画播放控制器，管理动画时间轴和帧更新

**职责：**

- 读取和管理动画数据
- 计算任意时刻的骨架姿态
- 驱动骨架变换更新

## **src/skinning/ - 蒙皮系统**

**主要文件：**

- `weight_calculator.py`：实现蒙皮权重计算算法（IDW + 区域约束）
- `deformer.py`：执行线性混合蒙皮（LBS）变形

**职责：**

- 计算顶点与骨骼的绑定权重
- 根据骨骼变换对网格进行蒙皮变形

## **src/rendering/ - 渲染模块**

**主要文件：**

- `renderer.py`：基于 OpenGL 的 3D 渲染器
- `camera.py`：相机控制（视角、投影、缩放）

- `frame_exporter.py`：将渲染结果导出为图像帧
- `video_export.py`：批量渲染并合成视频（使用 MoviePy）

职责：

- 将网格数据渲染到屏幕或离屏缓冲区
- 提供相机交互控制
- 导出静态图像和动画视频

## src/ui/ - GUI模块

主要文件：

- `main_window.py`
- 提供交互式用户界面
- 集成渲染视图和控制组件
- 处理用户输入并调用底层模块

## src/utils/ - 工具模块

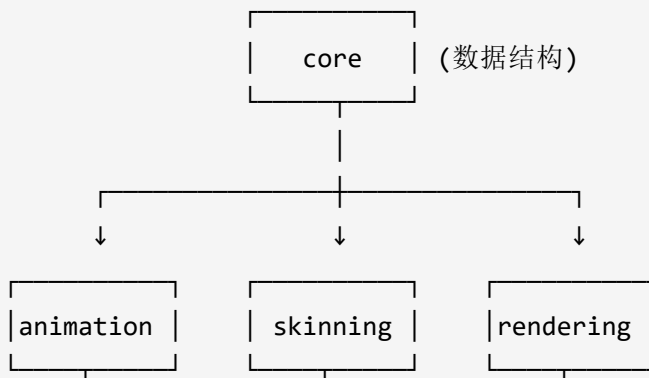
主要文件：

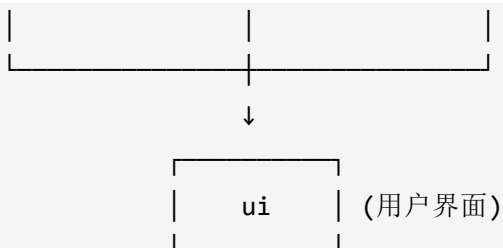
- `math_utils.py`：向量/矩阵运算、四元数转换
- `geometry.py`：点到线段距离、包围盒计算
- `file_io.py`：通用文件读写工具

职责：

- 提供底层数学和几何计算函数
- 封装常用的文件操作

## 4.3 模块间依赖关系





## 五、运行流程

### 5.1 安装依赖

```
pip install -r requirements.txt
```

### 5.2 命令行模式

列出所有动画：

```
python main.py list
```

计算蒙皮权重：

```
python main.py compute --max-influences 4
```

导出动画视频：

```
python main.py export walk_circle --angle 90 --fps 30 --duration 5
```

执行流程：

1. 解析命令行参数
  - ├ 动画名称：walk\_circle
  - ├ 视角：90度
  - └ 帧率：30 FPS

2. 加载资源

- └ OBJLoader.load(elk.obj) → Mesh对象
- └ SkeletonLoader.load(skeleton.json) → Skeleton对象
- └ load\_weights\_npz(weights.npz) → 权重矩阵

### 3. 初始化动画系统

- └ SkinDeformer(mesh, skeleton, weights)
- └ Animator(skeleton)
- └ animator.load\_clip(animation)

### 4. 渲染循环（后台窗口）

```
for frame in range(total_frames):  
    animator.update(dt)           # 更新骨架姿态  
    deformer.update()            # LBS变形  
    renderer.render_frame()      # OpenGL渲染  
    exporter.save_frame()        # 保存PNG
```

### 5. 视频合成

- └ ffmpeg/opencv 编码
- └ 输出: output/videos/walk\_circle.mp4

## 5.3 GUI模式

```
python ui_main.py
```

### 执行流程示例

用户操作: 选择动画 "head\_nod"

↓

信号: animation\_selected.emit("head\_nod")

↓

槽函数: \_on\_animation\_selected()

- └ 加载JSON动画文件
- └ animator.load\_clip(animation)
- └ 更新时间轴显示

用户操作: 点击"播放"按钮

↓

信号: play\_clicked.emit()

↓

槽函数: \_on\_play()

- └ animator.play()

```
└─ timer.start(33ms)  # 30 FPS
└─ 每次timeout触发:
    └─ animator.update(dt)
    └─ deformer.update()
    └─ gl_widget.update()  # 触发OpenGL重绘
```

## 六、核心算法实现

### 6.1 蒙皮权重计算

#### 算法选择

采用反距离加权法（IDW）+ 区域分割。

尝试使用双线性插值、热传导等多种方法进行测试，发现最近邻算法与反距离加权法的效果最好。当前实现为反距离加权基础上增加了区域划分和约束，以实现更好的绑定效果。

原理：

$$w_i = \frac{1}{(d_i/d_{\min})^p + \varepsilon}$$
$$\hat{w}_i = \frac{w_i}{\sum_{j=1}^k w_j}$$

- $d_i$ ：顶点到骨骼i的距离
- $p$ ：衰减指数（falloff）
- $k$ ：max\_influences（每个顶点的最大影响骨骼数，默认4）

#### 核心思路

##### Step 1：区域分割

根据骨骼命名规则自动识别身体部位：

关键词	区域	示例骨骼
Head , Jaw	头部	RigHead_021
Neck	颈部	RigNeck1_019

关键词	区域	示例骨骼
Spine , Pelvis	躯干	RigSpine1_011
FLeg + L	左前腿	RigLFLeg1_06
BLeg + R	右后腿	RigRBLeg2_028
Ankle	脚踝	RigLFAnkle_010

Step 2：约束候选骨骼

对于每个顶点，只考虑允许的骨骼集合：

```
# 根据最近骨骼的区域决定候选骨骼
nearest_region = bone_regions[nearest_bone]
allowed_bones = self.classifier.get_allowed_bones(
    nearest_region, bone_regions, nearest_bone
)
```

邻接关系表（部分）：

- 头部：可与 颈部 混合
- 颈部：可与 头部 + 躯干 混合
- 肩部：可与 躯干 + 前腿 混合
- 前腿：只允许 前腿 骨骼链

Step 3：反距离加权

计算顶点到每个候选骨骼的距离，按距离分配权重：

```
def _assign_weights(self, vertex_idx, bone_distances, weights, falloff=2.0):
    """
    根据距离分配权重

    Args:
        bone_distances: [(bone_idx, distance), ...]
        falloff: 距离衰减指数（默认 2.0）
    """
    min_dist = max(bone_distances[0][1], 0.001) # 避免除零

    # 计算权重
    bone_weights = []
```

```

total = 0.0
for bone_idx, dist in bone_distances:
    # IDW 公式
    w = 1.0 / ((dist / min_dist) ** falloff + 0.01)
    bone_weights.append((bone_idx, w))
    total += w

# 归一化
for bone_idx, w in bone_weights:
    weights[vertex_idx, bone_idx] = w / total

```

## 特殊处理

算法对四个特殊区域做了定制化处理，优先级从高到低：

### 1. 脚踝区域（Ankle Region）

脚踝在动画时需要刚性绑定，如果与小腿骨骼混合会导致脚部“软塌”。

解决方案：强制单骨骼绑定（权重 = 1.0）

### 2. 肩部区域（Shoulder Region）

肩部是躯干和前腿的连接处，需要平滑过渡。

解决方案：允许躯干、前腿、颈部骨骼混合，使用柔和衰减。

### 3. 头部区域（Head Region）

头部在四足动物模型中容易受腿部和躯干骨骼影响。

解决方案：严格排除所有腿部和躯干骨骼。

### 4. 普通区域（Normal Region）

按照上述算法进行计算即可。

## 6.2 线性混合蒙皮（LBS）

线性混合蒙皮（Linear Blend Skinning）是最常用的蒙皮算法，公式为：

$$\mathbf{v}' = \sum_{i=1}^n w_i \cdot \mathbf{M}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{v}$$

符号说明：

- $\mathbf{v}$ : 绑定姿态下的顶点坐标（世界空间）
- $\mathbf{v}'$ : 动画姿态下的顶点坐标（世界空间）
- $w_i$ : 顶点对骨骼  $i$  的权重 ( $\sum w_i = 1$ )
- $\mathbf{B}_i^{-1}$ : 骨骼  $i$  的**绑定逆矩阵** (Bind Inverse Matrix)
  - 作用: 将顶点从世界空间变换到骨骼的局部空间
  - $\mathbf{B}_i = \mathbf{M}_i^{\text{bind}}$  (绑定姿态下的全局变换矩阵)
- $\mathbf{M}_i$ : 骨骼  $i$  的**当前全局变换矩阵** (Current Global Transform)
  - 作用: 将顶点从骨骼局部空间变换到世界空间

**变换流程:** 世界空间  $\rightarrow$  骨骼局部空间  $\rightarrow$  世界空间

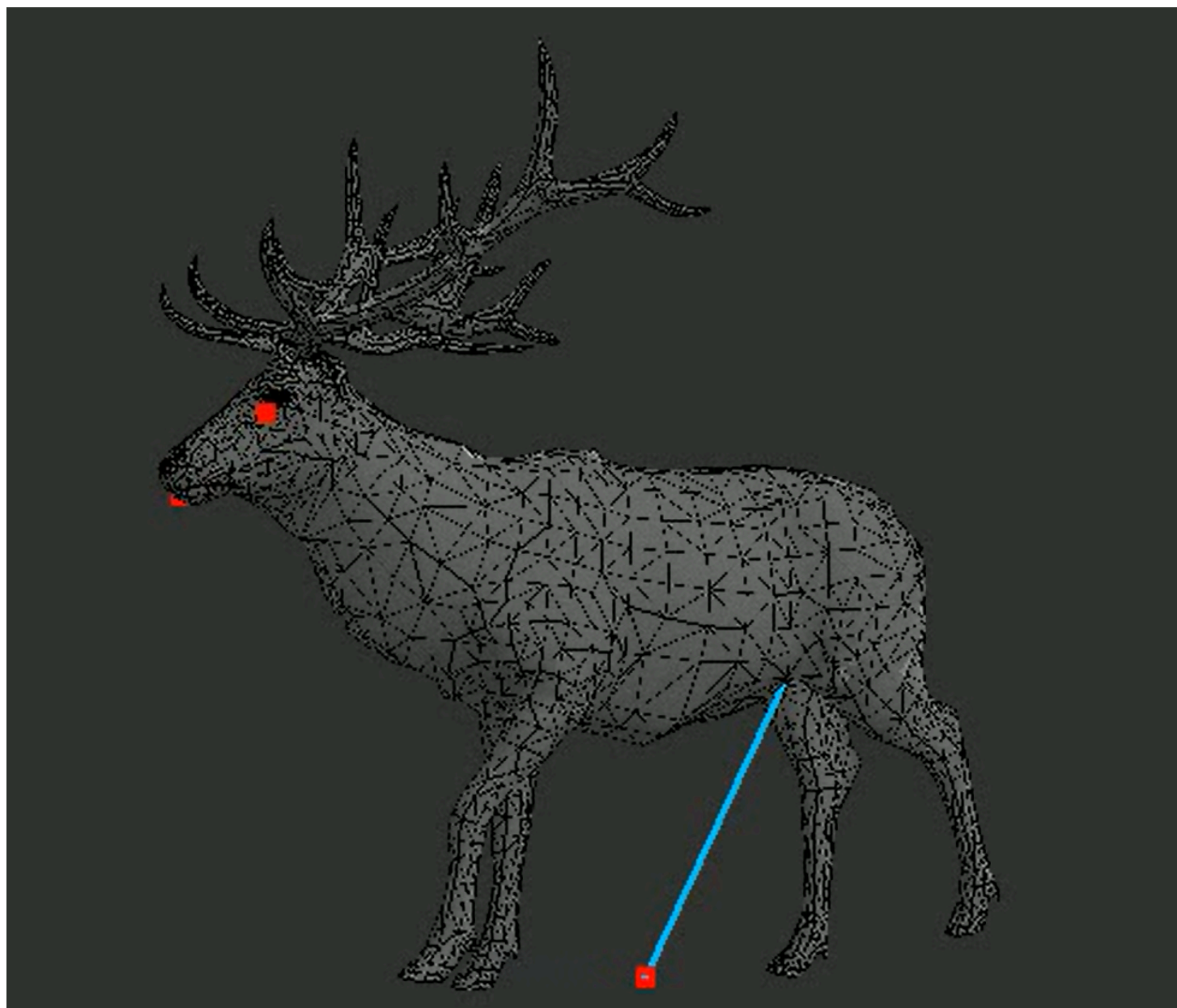
## 七、效果展示

目前在 data/animations/ 目录下有4个动画json文件，分别为：小幅度点头、行走、奔跑1、奔跑2。

由于模型为真实生物，骨架相对复杂，难以生成比较自然的关键帧动画（真的调了很久但是效果还是很诡异），但依然可以看出模型加载、权重计算、绑定、渲染和导出功能都是基本正确的。

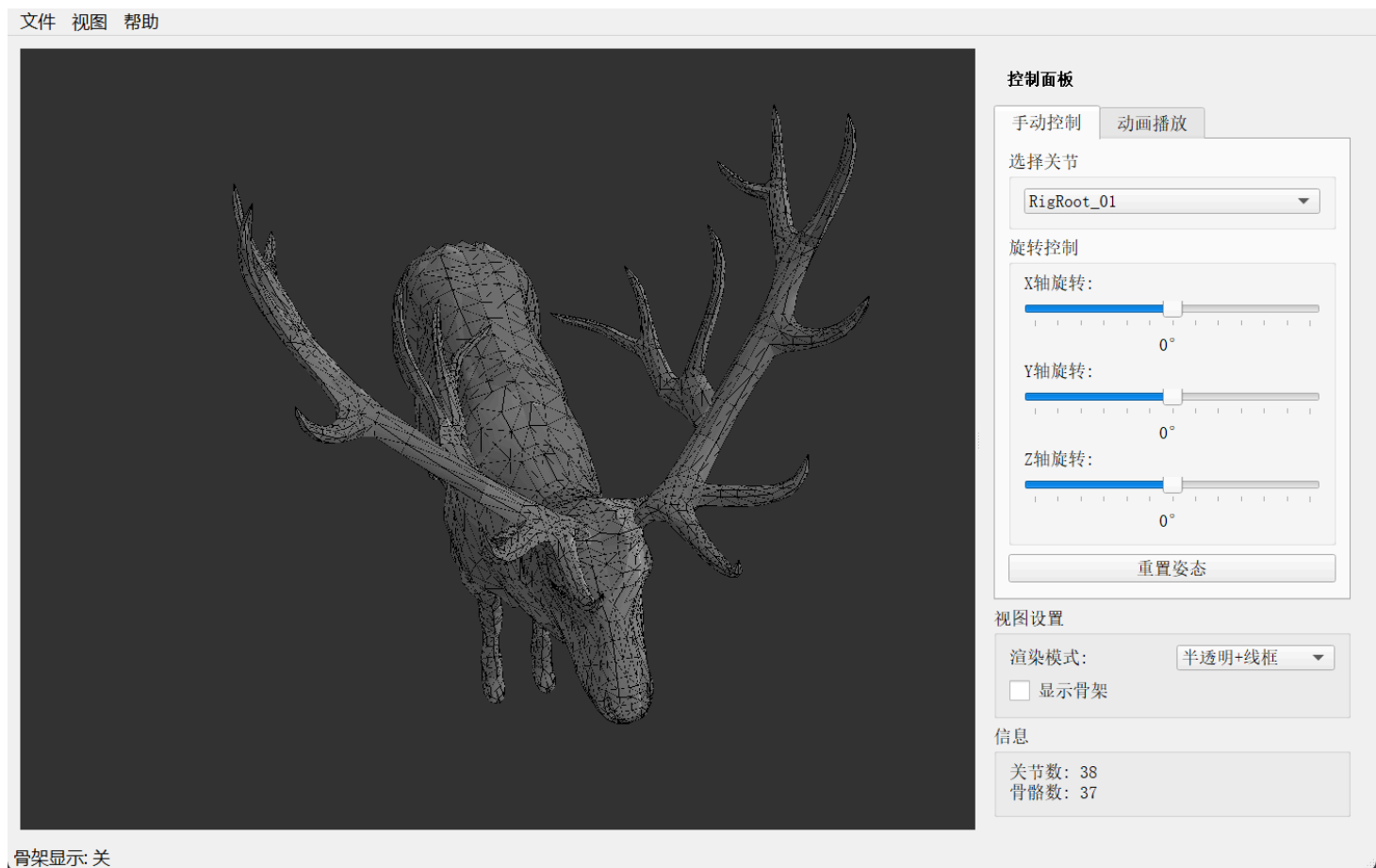
## 命令行模式

应用 main.py 可以直接导出指定参数的视频。



## GUI模式

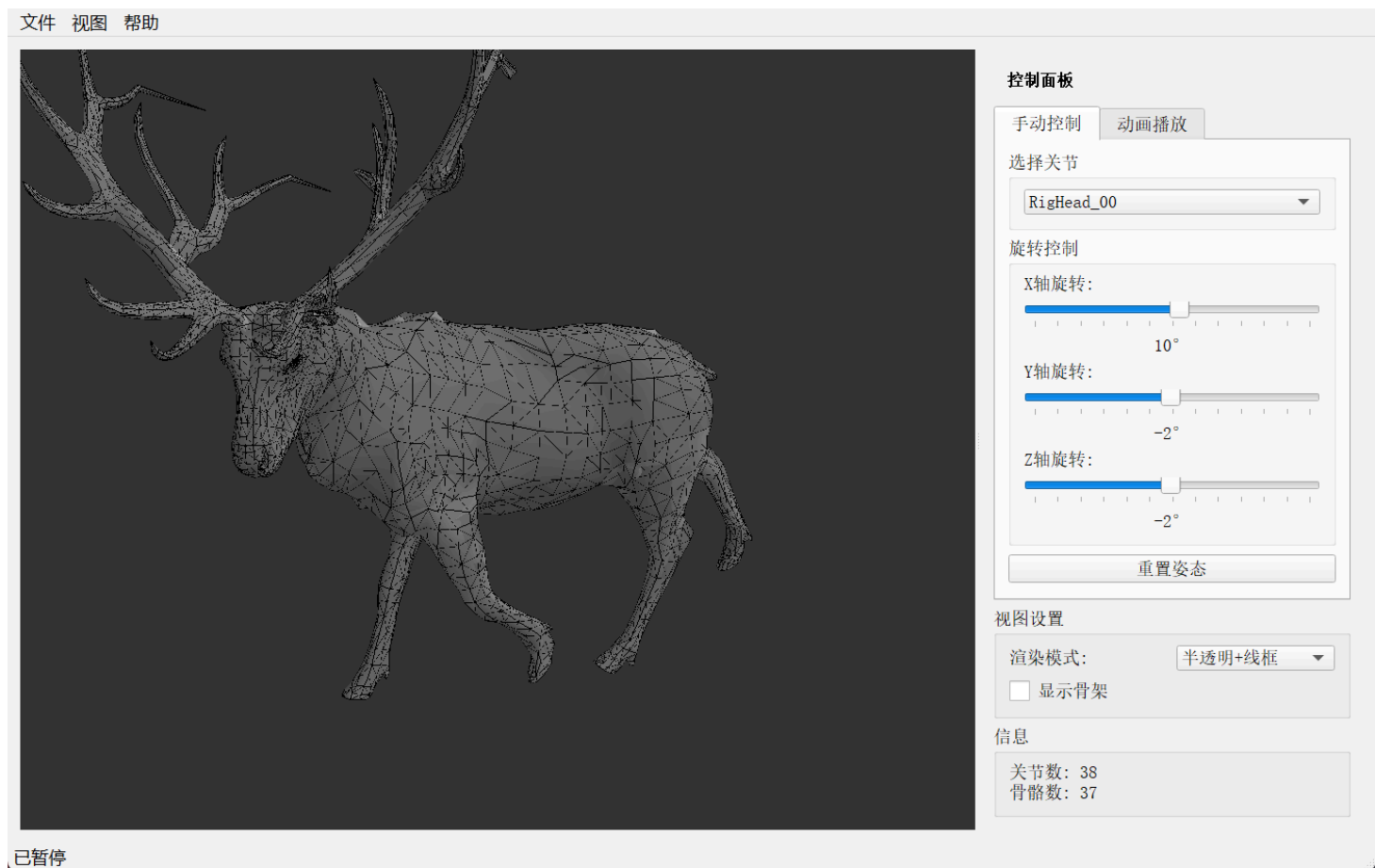
主界面：



骨架显示: 关

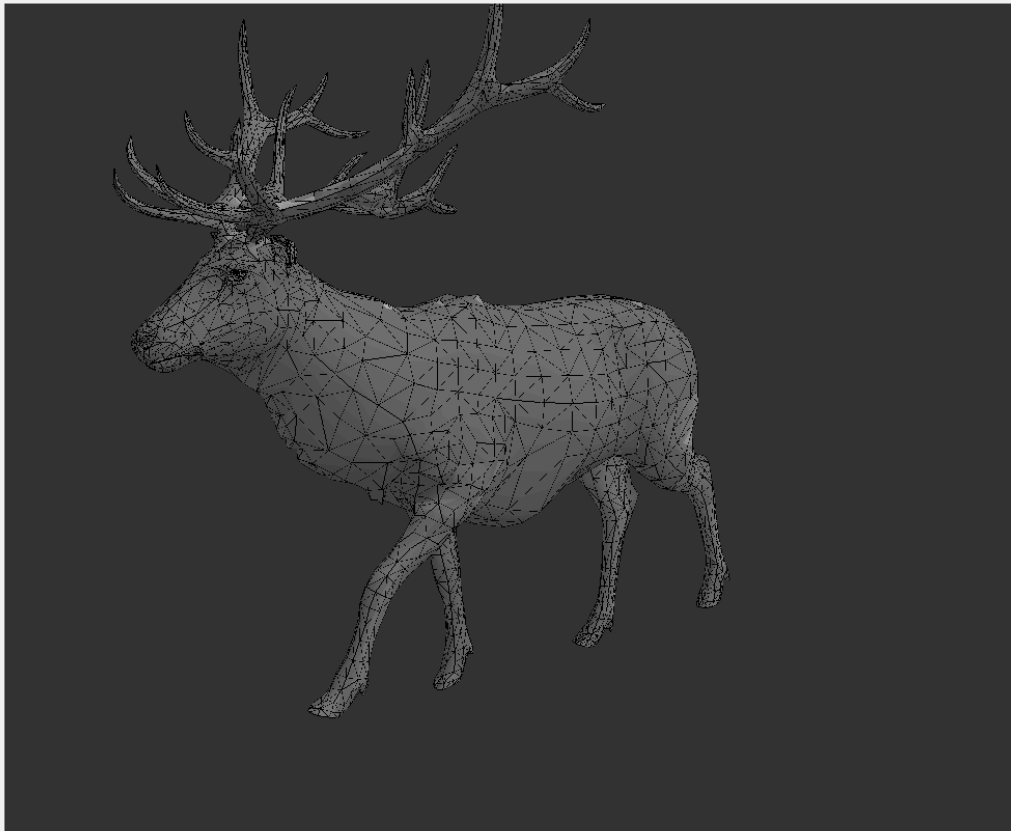
左上角文件菜单支持导出骨架、导出姿态、加载模型/权重。左侧视图支持鼠标旋转、缩放。

**手动控制：**



自己摆了一个还算比较正常的姿势，可选择关节任意旋转调整姿态。

**动画播放：**



### 控制面板

手动控制 动画播放

#### 选择动画

gallop  
head\_nod  
run  
walk\_circle

#### 播放控制

播放

暂停

停止

1.80s / 2.40s

☒ 循环播放

导出视频...

#### 视图设置

渲染模式:

半透明+线框

☐ 显示骨架

#### 信息

关节数: 38

骨骼数: 37

骨架显示: 关

## 导出视频



动画: walk\_circle

动画时长: 2.40秒

导出时长

时长: 4.50 秒

视频设置

帧率: 30 FPS

提示: 将录制左侧OpenGL视图的画面, 如果时长超过动画时长, 动画将循环播放

输出路径

D:  
\vscode\CA\CA\_project\output\videos\walk\_circle.mp4

浏览...

开始录制

取消

可以选择一个已有动画并播放，并可以导出指定长度的视频。视频导出逻辑为录制左侧画面，因此可以手动控制导出视频的相机视角。

支持半透明/线框模式，可选是否渲染骨架。

## 八、遇到的问题

其实核心算法并没有花费太多时间。对我来说最耗费时间的部分是获得一个看起来自然的关键帧动画，但效果依然不理想，最后勉强得到了一个行走动画。

在实现过程中，我遇到了下列问题：

1. 导出的骨架和模型的YZ坐标不一致。解决方法：把其中之一旋转90度就可以对齐了。其他坐标也做一些手动的修正。
2. 模型看起来比较扭曲。后来发现是渲染的光线没调好导致的鹿角投影看起来像模型扭曲了一样。
3. 骨架从模型里穿出来。是理论上只应该受到单骨骼影响的皮肤被附近的其他骨骼影响了，导致一种牵拉效果。
4. 两只脚脚底会互相影响，在走路的时候会有一种类似穿高跟鞋的效果。
5. 鹿角末端有一部分绑定不到头上，要么绑定到前腿上，要么绑定到躯干上了。这个问题最后也没有完全解决，我认为它的处理是比较困难的，因为这个模型的鹿角部分没有骨骼，但鹿角又非常大，导致某些位置离头部非常远。在这种情况下，依靠距离的权重计算方法难免会出现问题。
6. 上述3-5问题都是通过不断修改权重计算类，进行更细致的区域划分解决或改善的。

做完这个作业，我感受到一个动画的制作除了最核心的部分，从头到尾还有超级多的细节工作，而且要使得最终出来的效果自然需要更多技术之外的努力。不过最终能和自己写出来的东西交互看到效果的时候还是很奇妙的。

## 九、参考文献

### 学术文献

[1] Baran, I., & Popović, J. (2007). Automatic rigging and animation of 3d characters. *ACM Transactions on Graphics*, 26(3), 72.

[2] Jacobson, A., et al. (2011). Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics*, 30(4), 78.

## 技术文档

[3] OpenGL Programming Guide. <https://www.opengl.org/documentation/>

[4] PyQt5 Documentation. <https://doc.qt.io/qtforpython-5/>

[5] LearnOpenGL. <https://learnopengl.com/>