# Q1 Code

import numpy as np   
import matplotlib.pyplot as plt   
from mpl\_toolkits.mplot3d import Axes3D # noqa: F401   
import matplotlib.animation as animation   
from matplotlib.widgets import Button, Slider   
from typing import Optional, Tuple   
import matplotlib   
   
# 设置色系主题和中文字体   
plt.style.use('dark\_background')   
matplotlib.rcParams['font.sans-serif'] = ['Microsoft YaHei', 'SimHei', 'Arial Unicode MS', 'DejaVu Sans']   
matplotlib.rcParams['font.monospace'] = ['Microsoft YaHei', 'SimHei', 'Fira Code', 'DejaVu Sans Mono']   
matplotlib.rcParams['axes.unicode\_minus'] = False   
   
# 色系配色方案   
matplotlib.rcParams['axes.facecolor'] = '#2F2F2F' # 深灰背景   
matplotlib.rcParams['figure.facecolor'] = '#3A3A3A' # 图形背景   
matplotlib.rcParams['grid.color'] = '#5A5A5A' # 网格颜色   
matplotlib.rcParams['grid.alpha'] = 0.5   
matplotlib.rcParams['text.color'] = '#E5E5E5' # 文字颜色   
   
   
class RealTimeProjection:   
 """   
 Q1 场景：   
 - M1 以 300 m/s 朝 FO 直线运动。   
 - RO 抽象为球：中心(0,200,5)，半径√74。   
 - FY1 投放烟雾干扰：半径固定 10 m，t\_det=5.1 s 起爆，起爆后下沉 3 m/s 持续 20 s。   
 - 实时可视化切锥、烟团、以及"完全遮蔽"时间段，并给出关键几何数据。   
 """   
   
 def \_\_init\_\_(self) -> None:   
 # 色系配色定义   
 self.colors = {   
 'morandi\_sage': '#9CAF88', # 鼠尾草绿 - 用于RO目标   
 'morandi\_dusty\_rose': '#D4A5A5', # 玫瑰粉 - 用于FO假目标   
 'morandi\_warm\_gray': '#A8A5A0', # 暖灰 - 用于烟团   
 'morandi\_soft\_blue': '#8FA5C7', # 柔和蓝 - 用于FY1   
 'morandi\_mauve': '#B09FAC', # 淡紫 - 用于M1导弹   
 'morandi\_beige': '#C7B299', # 米色 - 用于轨迹   
 'morandi\_lavender': '#A5A2C7', # 薰衣草 - 用于切锥   
 'morandi\_cream': '#E5D5C8', # 奶油色 - 用于文字背景   
 'background\_dark': '#3A3A3A', # 深色背景   
 'text\_light': '#E5E5E5', # 浅色文字   
 }   
   
 # 场景定义   
 self.M1\_start = np.array([20000.0, 0.0, 2000.0])   
 self.FO = np.array([0.0, 0.0, 0.0])   
 self.RO\_center = np.array([0.0, 200.0, 5.0])   
 self.RO\_radius = float(np.sqrt(74.0))   
 self.FY1\_start = np.array([17800.0, 0.0, 1800.0])   
   
 # 基础物理参数   
 self.v\_missile = 300.0   
 self.fy\_speed = 120.0   
 self.g = 9.8   
   
 # 时间节点   
 self.t\_drop = 1.5   
 self.t\_det = 5.1   
 self.smoke\_duration = 20.0   
 self.total\_time = float(np.linalg.norm(self.FO - self.M1\_start) / self.v\_missile)   
 self.dt = 0.01   
   
 # 运动参数   
 self.smoke\_v\_down = 3.0   
 self.smoke\_radius = 10.0   
 self.R\_smoke = 10.0   
   
 # FY1飞行方向：朝向假目标FO   
 self.fy\_dir = self.FO - self.FY1\_start   
 self.fy\_dir = self.fy\_dir / np.linalg.norm(self.fy\_dir)   
   
 # 关键位置预计算   
 self.pos\_drop = self.FY1\_start + self.fy\_dir \* (self.fy\_speed \* self.t\_drop)   
 det\_dt = self.t\_det - self.t\_drop   
 v0 = self.fy\_speed \* self.fy\_dir   
 self.pos\_det = self.pos\_drop + v0 \* det\_dt + np.array([0.0, 0.0, -0.5 \* self.g \* det\_dt \* det\_dt])   
   
 # 界面控制   
 self.show\_sphere = True   
 self.show\_fy1 = True   
 self.show\_smoke = True   
 self.show\_cone = True   
 self.show\_rim = True   
 self.show\_axis = True   
 self.show\_overlay = True   
 self.cone\_alpha = 0.3   
 self.\_preserve\_view = True   
 self.\_default\_view = (30.0, -60.0)   
   
 # 动画控制   
 self.running = False   
 self.current\_frame = 0   
 self.play\_speed = 1.0   
 self.\_frame\_accum = 0.0   
 self.\_updating\_slider = False   
   
 # 界面组件   
 self.fig = None   
 self.ax3d = None   
 self.ax\_info = None   
 self.ax\_area = None   
 self.ax\_dist = None   
 self.ax\_control = None   
 self.info\_text = None   
 self.status\_text = None   
 self.slider = None   
 self.ani = None   
   
 # 预计算数据缓存   
 self.\_times = None   
 self.\_areas = None   
 self.\_dists = None   
 self.\_occluded\_ts = None   
 self.\_occluded\_flags = None   
 self.\_occluded\_total = None   
   
 def get\_M1\_position(self, t: float) -> np.ndarray:   
 direction = self.FO - self.M1\_start   
 direction = direction / np.linalg.norm(direction)   
 return self.M1\_start + direction \* self.v\_missile \* t   
   
 def \_smoke\_center(self, t: float) -> Optional[np.ndarray]:   
 if t < self.t\_det:   
 return None   
 dt = t - self.t\_det   
 if dt > self.smoke\_duration:   
 return None   
 z\_offset = -self.smoke\_v\_down \* dt   
 return self.pos\_det + np.array([0.0, 0.0, z\_offset])   
   
 def is\_fully\_occluded(self, t: float) -> bool:   
 M1 = self.get\_M1\_position(t)   
 smoke\_center = self.\_smoke\_center(t)   
 if smoke\_center is None:   
 return False   
   
 to\_center = self.RO\_center - M1   
 dist = float(np.linalg.norm(to\_center))   
 if dist <= self.RO\_radius:   
 return True   
   
 view\_dir = to\_center / dist   
 apex\_to\_smoke = smoke\_center - M1   
 proj\_length = float(np.dot(apex\_to\_smoke, view\_dir))   
   
 if proj\_length <= 0:   
 return False   
   
 half\_angle = float(np.arcsin(self.RO\_radius / dist))   
 cone\_radius\_at\_smoke = proj\_length \* float(np.tan(half\_angle))   
 lateral\_distance = float(np.linalg.norm(apex\_to\_smoke - proj\_length \* view\_dir))   
   
 return lateral\_distance + self.R\_smoke <= cone\_radius\_at\_smoke   
   
 def analyze\_full\_occlusion(self, ts: np.ndarray):   
 flags = np.array([self.is\_fully\_occluded(float(t)) for t in ts])   
 occluded\_ts = ts[flags]   
 total\_time = float(np.sum(np.diff(ts)[:-1][flags[1:]])) if len(occluded\_ts) > 1 else 0.0   
 return occluded\_ts, flags, total\_time   
   
 @staticmethod   
 def \_shade\_occlusion(ax, ts: np.ndarray, flags: np.ndarray) -> None:   
 """为遮蔽时间段添加风格的阴影显示"""   
 if ts is None or flags is None:   
 return   
 on = False   
 t\_start = None   
 for i in range(len(ts)):   
 if flags[i] and not on:   
 on = True   
 t\_start = ts[i]   
 if (not flags[i] and on) or (on and i == len(ts) - 1):   
 t\_end = ts[i]   
 # 使用鼠尾草绿阴影表示遮蔽区域   
 ax.axvspan(t\_start, t\_end, color='#9CAF88', alpha=0.3,   
 label='遮蔽时段' if t\_start == ts[flags].min() else '')   
 on = False   
   
 def \_draw\_scene(self, ax, t: float) -> None:   
 # 在清空前保存用户当前视角   
 try:   
 elev, azim = float(getattr(ax, 'elev', 30.0)), float(getattr(ax, 'azim', -60.0))   
 except Exception:   
 elev, azim = 30.0, -60.0   
   
 ax.clear()   
   
 # 设置风格的3D场景外观   
 ax.xaxis.pane.fill = False   
 ax.yaxis.pane.fill = False   
 ax.zaxis.pane.fill = False   
 ax.xaxis.pane.set\_edgecolor('#5A5A5A')   
 ax.yaxis.pane.set\_edgecolor('#5A5A5A')   
 ax.zaxis.pane.set\_edgecolor('#5A5A5A')   
 ax.grid(True, alpha=0.4, color='#5A5A5A')   
   
 M1 = self.get\_M1\_position(t)   
 occluded = self.is\_fully\_occluded(t)   
   
 # 固定对象：使用配色，增大尺寸避免重叠   
 ax.scatter(\*self.FO, color=self.colors['morandi\_dusty\_rose'], s=180,   
 label='FO (假目标)', marker='\*', edgecolors=self.colors['text\_light'], linewidth=2)   
 ax.scatter(\*self.RO\_center, color=self.colors['morandi\_sage'], s=180,   
 label='RO (真目标)', marker='o', edgecolors=self.colors['text\_light'], linewidth=2)   
   
 # 球体线框：使用配色，减少密度避免视觉混乱   
 if self.show\_sphere:   
 u = np.linspace(0, 2 \* np.pi, 24)   
 v = np.linspace(0, np.pi, 16)   
 x = self.RO\_center[0] + self.RO\_radius \* np.outer(np.cos(u), np.sin(v))   
 y = self.RO\_center[1] + self.RO\_radius \* np.outer(np.sin(u), np.sin(v))   
 z = self.RO\_center[2] + self.RO\_radius \* np.outer(np.ones\_like(u), np.cos(v))   
 ax.plot\_wireframe(x, y, z, color=self.colors['morandi\_sage'], alpha=0.4, linewidth=1)   
   
 # M1 位置：根据遮蔽状态动态改变颜色   
 m1\_color = self.colors['morandi\_mauve'] if not occluded else self.colors['morandi\_dusty\_rose']   
 m1\_marker = 'D' if not occluded else '^'   
 m1\_size = 200 if not occluded else 220   
 ax.scatter(\*M1, color=m1\_color, s=m1\_size, label=f'M1 t={t:.1f}s', marker=m1\_marker,   
 edgecolors=self.colors['text\_light'], linewidth=2)   
   
 # FY1 无人机运动与投弹/弹体轨迹：使用配色   
 if self.show\_fy1:   
 fy\_t = float(max(0.0, t))   
 fy\_pos = self.FY1\_start + self.fy\_dir \* (self.fy\_speed \* fy\_t)   
 tt\_fy = np.linspace(0.0, fy\_t, 50)   
 traj\_fy = self.FY1\_start + self.fy\_dir[None, :] \* (self.fy\_speed \* tt\_fy[:, None])   
 ax.plot(traj\_fy[:, 0], traj\_fy[:, 1], traj\_fy[:, 2], color=self.colors['morandi\_soft\_blue'],   
 alpha=0.8, linewidth=3, label='FY1 航迹')   
 ax.scatter(\*fy\_pos, color=self.colors['morandi\_soft\_blue'], s=120, label='FY1', marker='s',   
 edgecolors=self.colors['text\_light'], linewidth=2)   
 ax.scatter(\*self.pos\_drop, color=self.colors['morandi\_lavender'], s=100,   
 label='投弹点 t=1.5s', marker='v', edgecolors=self.colors['text\_light'], linewidth=2)   
   
 if t >= self.t\_drop:   
 t0 = self.t\_drop   
 t1 = min(t, self.t\_det)   
 ts\_seg = np.linspace(t0, t1, 50)   
 dt\_seg = ts\_seg - t0   
 pos\_seg = self.pos\_drop[None, :] + (self.fy\_speed \* self.fy\_dir)[None, :] \* dt\_seg[:, None] \   
 + np.array([0.0, 0.0, -0.5 \* self.g])[None, :] \* (dt\_seg[:, None] \*\* 2)   
 ax.plot(pos\_seg[:, 0], pos\_seg[:, 1], pos\_seg[:, 2], color=self.colors['morandi\_beige'],   
 linestyle='--', linewidth=2.5, alpha=0.9, label='弹体轨迹')   
   
 # 烟团球：使用配色   
 S = self.\_smoke\_center(t)   
 if self.show\_smoke and S is not None:   
 u\_s = np.linspace(0, 2 \* np.pi, 20)   
 v\_s = np.linspace(0, np.pi, 15)   
 xs = S[0] + self.smoke\_radius \* np.outer(np.cos(u\_s), np.sin(v\_s))   
 ys = S[1] + self.smoke\_radius \* np.outer(np.sin(u\_s), np.sin(v\_s))   
 zs = S[2] + self.smoke\_radius \* np.outer(np.ones\_like(u\_s), np.cos(v\_s))   
   
 smoke\_color = self.colors['morandi\_warm\_gray'] if not occluded else self.colors['morandi\_dusty\_rose']   
 smoke\_alpha = 0.6 if not occluded else 0.8   
 ax.plot\_wireframe(xs, ys, zs, color=smoke\_color, alpha=smoke\_alpha, linewidth=1.2)   
 ax.scatter(\*self.pos\_det, color=self.colors['morandi\_cream'], s=80, label='烟团起爆点',   
 marker='\*', edgecolors='#3A3A3A', linewidth=1.5)   
   
 # 切锥：使用配色   
 to\_center = self.RO\_center - M1   
 dist = float(np.linalg.norm(to\_center))   
 if dist > self.RO\_radius + 1e-9:   
 view\_dir = to\_center / dist   
 half\_angle = float(np.arcsin(self.RO\_radius / dist))   
 v1 = np.cross(view\_dir, np.array([0.0, 0.0, 1.0]))   
 if np.linalg.norm(v1) < 1e-9:   
 v1 = np.cross(view\_dir, np.array([1.0, 0.0, 0.0]))   
 v1 = v1 / np.linalg.norm(v1)   
 v2 = np.cross(view\_dir, v1)   
 v2 = v2 / np.linalg.norm(v2)   
 h\_max = dist \* float(np.cos(half\_angle))   
 center\_rim = M1 + view\_dir \* h\_max   
 rim\_radius = self.RO\_radius   
 h = np.linspace(0.0, h\_max, 20)   
 uu = np.linspace(0.0, 2.0 \* np.pi, 60)   
 H, U = np.meshgrid(h, uu, indexing='ij')   
 R\_h = H \* float(np.tan(half\_angle))   
 X = M1[0] + view\_dir[0] \* H + R\_h \* (np.cos(U) \* v1[0] + np.sin(U) \* v2[0])   
 Y = M1[1] + view\_dir[1] \* H + R\_h \* (np.cos(U) \* v1[1] + np.sin(U) \* v2[1])   
 Z = M1[2] + view\_dir[2] \* H + R\_h \* (np.cos(U) \* v1[2] + np.sin(U) \* v2[2])   
   
 cone\_color = self.colors['morandi\_lavender'] if not occluded else self.colors['morandi\_dusty\_rose']   
 cone\_alpha = 0.3 if not occluded else 0.5   
   
 if self.show\_cone:   
 ax.plot\_surface(X, Y, Z, color=cone\_color, alpha=cone\_alpha, shade=True, linewidth=0)   
 theta = np.linspace(0.0, 2.0 \* np.pi, 120)   
 rim = center\_rim + rim\_radius \* (np.cos(theta)[:, None] \* v1 + np.sin(theta)[:, None] \* v2)   
 if self.show\_rim:   
 rim\_color = self.colors['morandi\_lavender'] if not occluded else self.colors['morandi\_dusty\_rose']   
 ax.plot(rim[:, 0], rim[:, 1], rim[:, 2], color=rim\_color, linewidth=3, label='切面圆')   
 ax.scatter(\*center\_rim, color=rim\_color, s=80, zorder=5, marker='o',   
 edgecolors=self.colors['text\_light'], linewidth=1.5)   
   
 # 轴线：使用配色   
 if self.show\_axis:   
 axis\_color = self.colors['morandi\_lavender'] if not occluded else self.colors['morandi\_dusty\_rose']   
 ax.plot([M1[0], self.RO\_center[0]], [M1[1], self.RO\_center[1]], [M1[2], self.RO\_center[2]],   
 color=axis\_color, linestyle='-.', linewidth=2.5, alpha=0.9, label='视线轴线')   
   
 # M1轨迹：使用配色   
 t2 = min(float(t) + 1.0, self.total\_time)   
 traj\_t = np.linspace(max(0.0, t2 - 1.0), t2, 50)   
 traj = np.array([self.get\_M1\_position(tt) for tt in traj\_t])   
 ax.plot(traj[:, 0], traj[:, 1], traj[:, 2], '--', color=self.colors['morandi\_beige'],   
 alpha=0.8, linewidth=2.5, label='M1轨迹')   
   
 # 坐标轴标签：使用浅色文字   
 ax.set\_xlabel('X (m)', fontsize=12, color=self.colors['text\_light'], weight='bold')   
 ax.set\_ylabel('Y (m)', fontsize=12, color=self.colors['text\_light'], weight='bold')   
 ax.set\_zlabel('Z (m)', fontsize=12, color=self.colors['text\_light'], weight='bold')   
   
 # 动态标题：根据遮蔽状态变化颜色   
 title\_color = self.colors['morandi\_dusty\_rose'] if occluded else self.colors['morandi\_sage']   
 occlusion\_status = "完全遮蔽" if occluded else "无遮蔽"   
 ax.set\_title(f'烟幕干扰三维场景 - {occlusion\_status} (t={t:.1f}s)',   
 fontsize=16, family='Microsoft YaHei', color=title\_color, weight='bold', pad=20)   
   
 # 3D 叠加关键参数   
 if self.show\_overlay:   
 to\_center = self.RO\_center - M1   
 d = float(np.linalg.norm(to\_center))   
 if d > self.RO\_radius:   
 alpha = float(np.arcsin(self.RO\_radius / d))   
 alpha\_deg = float(np.degrees(alpha))   
 apex\_deg = 2.0 \* alpha\_deg   
 overlay = (   
 f"时间: {t:.2f}s 距离: {d:.1f}m 半角: {alpha\_deg:.2f}° "   
 f"顶角: {apex\_deg:.2f}° 遮蔽: {'完全' if occluded else '无'}"   
 )   
 else:   
 overlay = f"时间: {t:.2f}s M1位于球内 遮蔽: {'完全' if occluded else '无'}"   
 try:   
 text\_color = self.colors['morandi\_dusty\_rose'] if occluded else self.colors['morandi\_sage']   
 ax.text2D(0.02, 0.98, overlay, transform=ax.transAxes, va='top', ha='left',   
 fontsize=11, family='Microsoft YaHei', color=text\_color, weight='bold',   
 bbox=dict(facecolor=self.colors['background\_dark'], alpha=0.9,   
 edgecolor=text\_color, linewidth=1.5, pad=8))   
 except Exception:   
 pass   
   
 # 图例：改进样式，避免重叠   
 try:   
 handles, labels = ax.get\_legend\_handles\_labels()   
 seen = set()   
 new\_h, new\_l = [], []   
 for h, lb in zip(handles, labels):   
 if lb not in seen and lb.strip() != '':   
 new\_h.append(h)   
 new\_l.append(lb)   
 seen.add(lb)   
 if new\_h:   
 # 调整图例位置，放在左下角避免与其他元素重叠   
 legend = ax.legend(new\_h, new\_l, loc='lower left', fontsize=8, framealpha=0.95,   
 facecolor=self.colors['background\_dark'],   
 edgecolor=self.colors['text\_light'], linewidth=1,   
 bbox\_to\_anchor=(0.02, 0.02), ncol=2)   
 legend.get\_frame().set\_linewidth(1.5)   
 for text in legend.get\_texts():   
 text.set\_color(self.colors['text\_light'])   
 except Exception:   
 pass   
   
 # 恢复用户视角   
 if self.\_preserve\_view:   
 try:   
 ax.view\_init(elev=elev, azim=azim)   
 except Exception:   
 pass   
   
 # 设置显示范围   
 ax.set\_xlim(-1000, 21000)   
 ax.set\_ylim(-100, 300)   
 ax.set\_zlim(-50, 2500)   
   
 def \_build\_layout(self):   
 """构建色系的界面布局，优化间距避免重叠"""   
 # 增大窗口尺寸以提供更好的视觉体验和避免重叠   
 self.fig = plt.figure(figsize=(20, 16))   
   
 # 调整网格布局比例，给控制区域更多空间   
 gs = self.fig.add\_gridspec(3, 2,   
 height\_ratios=[3.0, 1.8, 1.0],   
 width\_ratios=[2.5, 1.2],   
 hspace=0.35, wspace=0.25)   
   
 # 左上：3D查看器   
 self.ax3d = self.fig.add\_subplot(gs[0, 0], projection='3d')   
   
 # 右上：参数信息面板   
 self.ax\_info = self.fig.add\_subplot(gs[0, 1])   
 self.ax\_info.axis('off')   
   
 # 中下：图1和图2   
 self.ax\_area = self.fig.add\_subplot(gs[1, 0])   
 self.ax\_dist = self.fig.add\_subplot(gs[1, 1])   
   
 # 底部控制区域（跨两列）   
 self.ax\_control = self.fig.add\_subplot(gs[2, :])   
 self.ax\_control.axis('off')   
   
 # 调整边距，优化按钮区域布局   
 self.fig.subplots\_adjust(left=0.06, right=0.96, top=0.92, bottom=0.08, hspace=0.35, wspace=0.25)   
   
 # 分析曲线 + 遮蔽预计算   
 ts, angles\_deg, dists = self.analyze\_projection\_area()   
 self.\_times, self.\_areas, self.\_dists = ts, angles\_deg, dists   
 self.\_occluded\_ts, self.\_occluded\_flags, self.\_occluded\_total = self.analyze\_full\_occlusion(ts)   
   
 # 图1：视线半角变化 - 配色   
 self.ax\_area.plot(ts, angles\_deg, color=self.colors['morandi\_dusty\_rose'], linewidth=3, alpha=0.9)   
 self.ax\_area.set\_title('图1 - 视线半角变化', fontsize=14, family='Microsoft YaHei',   
 color=self.colors['text\_light'], weight='bold', pad=15)   
 self.ax\_area.set\_xlabel('时间 (s)', fontsize=12, color=self.colors['text\_light'])   
 self.ax\_area.set\_ylabel('半角 (°)', fontsize=12, color=self.colors['text\_light'])   
 self.ax\_area.grid(True, alpha=0.4, color='#5A5A5A')   
 self.ax\_area.tick\_params(colors=self.colors['text\_light'], labelsize=10)   
 self.\_shade\_occlusion(self.ax\_area, ts, self.\_occluded\_flags)   
   
 # 图2：距离变化 - 配色   
 self.ax\_dist.plot(ts, dists, color=self.colors['morandi\_soft\_blue'], linewidth=3, alpha=0.9)   
 self.ax\_dist.set\_title('图2 - M1到RO距离', fontsize=14, family='Microsoft YaHei',   
 color=self.colors['text\_light'], weight='bold', pad=15)   
 self.ax\_dist.set\_xlabel('时间 (s)', fontsize=12, color=self.colors['text\_light'])   
 self.ax\_dist.set\_ylabel('距离 (m)', fontsize=12, color=self.colors['text\_light'])   
 self.ax\_dist.grid(True, alpha=0.4, color='#5A5A5A')   
 self.ax\_dist.tick\_params(colors=self.colors['text\_light'], labelsize=10)   
 self.\_shade\_occlusion(self.ax\_dist, ts, self.\_occluded\_flags)   
   
 # 初始化右上角参数面板 - 样式，调整文字大小避免重叠   
 self.info\_text = self.ax\_info.text(0.05, 0.95, self.\_compose\_info\_text(0.0),   
 va='top', ha='left', fontsize=8,   
 family='Microsoft YaHei', color=self.colors['text\_light'],   
 transform=self.ax\_info.transAxes,   
 bbox=dict(boxstyle="round,pad=0.8",   
 facecolor=self.colors['background\_dark'],   
 edgecolor=self.colors['morandi\_sage'],   
 linewidth=2, alpha=0.95))   
   
 # 初始 3D 视角   
 try:   
 if hasattr(self.ax3d, 'view\_init'):   
 self.ax3d.view\_init(elev=self.\_default\_view[0], azim=self.\_default\_view[1])   
 except Exception:   
 pass   
   
 # 初始化一帧   
 try:   
 self.\_draw\_scene(self.ax3d, 0.0)   
 if self.info\_text is not None:   
 self.info\_text.set\_text(self.\_compose\_info\_text(0.0))   
 except Exception as e:   
 print(f"初始化绘制失败: {e}")   
   
 # 控制按钮：配色，优化尺寸和布局   
 try:   
 # 优化按钮尺寸参数   
 btn\_height = 0.04 # 适中的按钮高度   
 btn\_width = 0.10 # 稍宽的按钮便于点击   
 btn\_spacing = 0.13 # 合适的按钮间距   
 btn\_y = 0.02 # 底部合适位置   
   
 # 播放按钮   
 ax\_btn\_play = plt.axes((0.12, btn\_y, btn\_width, btn\_height), facecolor=self.colors['morandi\_sage'])   
 self.btn\_play = Button(ax\_btn\_play, '播放', color=self.colors['morandi\_sage'],   
 hovercolor=self.colors['morandi\_dusty\_rose'])   
 ax\_btn\_play.tick\_params(labelsize=10)   
   
 # 暂停按钮   
 ax\_btn\_pause = plt.axes((0.12 + btn\_spacing, btn\_y, btn\_width, btn\_height),   
 facecolor=self.colors['morandi\_mauve'])   
 self.btn\_pause = Button(ax\_btn\_pause, '暂停', color=self.colors['morandi\_mauve'],   
 hovercolor=self.colors['morandi\_dusty\_rose'])   
 ax\_btn\_pause.tick\_params(labelsize=10)   
   
 # 重置按钮   
 ax\_btn\_reset = plt.axes((0.12 + 2\*btn\_spacing, btn\_y, btn\_width, btn\_height),   
 facecolor=self.colors['morandi\_beige'])   
 self.btn\_reset = Button(ax\_btn\_reset, '重置', color=self.colors['morandi\_beige'],   
 hovercolor=self.colors['morandi\_dusty\_rose'])   
 ax\_btn\_reset.tick\_params(labelsize=10)   
   
 # 时间滑块：优化位置和尺寸   
 slider\_y = btn\_y + 0.01 # 紧贴按钮上方   
 slider\_width = 0.35 # 更宽的滑块便于操作   
 slider\_height = 0.025 # 合适的滑块高度   
 ax\_slider = plt.axes((0.55, slider\_y, slider\_width, slider\_height), facecolor=self.colors['background\_dark'])   
 self.slider = Slider(ax\_slider, '时间进度', 0.0, self.total\_time, valinit=0.0,   
 color=self.colors['morandi\_soft\_blue'],   
 facecolor=self.colors['background\_dark'])   
   
 # 状态文本：移到顶部显示   
 status\_y = 0.95   
 self.status\_text = self.fig.text(0.5, status\_y, "时间: 0.0s | 遮蔽状态: NO",   
 ha='center', va='center', fontsize=12,   
 color=self.colors['text\_light'], weight='bold',   
 bbox=dict(boxstyle="round,pad=0.5",   
 facecolor=self.colors['background\_dark'],   
 edgecolor=self.colors['morandi\_sage'],   
 linewidth=1.5, alpha=0.9))   
   
 # 绑定事件   
 self.btn\_play.on\_clicked(self.\_on\_play)   
 self.btn\_pause.on\_clicked(self.\_on\_pause)   
 self.btn\_reset.on\_clicked(self.\_on\_reset)   
 self.slider.on\_changed(self.\_on\_slider)   
 except Exception as e:   
 print(f"控件创建失败: {e}")   
   
 def \_compose\_info\_text(self, t: float) -> str:   
 """生成右上角参数面板的详细信息显示"""   
 M1 = self.get\_M1\_position(t)   
 to\_center = self.RO\_center - M1   
 d = float(np.linalg.norm(to\_center))   
 R = self.RO\_radius   
 occluded = self.is\_fully\_occluded(t)   
 total\_val = self.\_occluded\_total if self.\_occluded\_total is not None else 0.0   
 S = self.\_smoke\_center(t)   
   
 # 切锥几何参数   
 if d > R:   
 alpha = float(np.arcsin(R / d))   
 alpha\_deg = float(np.degrees(alpha))   
 apex\_deg = 2.0 \* alpha\_deg   
 h\_max = d \* float(np.cos(alpha))   
 rim\_radius = R   
 d\_tangent = float(np.sqrt(max(0.0, d\*d - R\*R)))   
   
 geom\_info = (   
 f"切锥几何参数\n"   
 f"{'─' \* 14}\n"   
 f"距离: {d:.1f}m\n"   
 f"半角α: {alpha\_deg:.2f}°\n"   
 f"顶角: {apex\_deg:.2f}°\n"   
 f"切面高: {h\_max:.1f}m\n"   
 f"切面半径: {rim\_radius:.1f}m\n"   
 )   
 else:   
 geom\_info = (   
 f"切锥几何参数\n"   
 f"{'─' \* 14}\n"   
 f"⚠️ M1位于球体内部\n"   
 f"距离: {d:.1f}m\n"   
 )   
   
 # 运动状态参数   
 motion\_info = (   
 f"\n运动状态参数\n"   
 f"{'─' \* 14}\n"   
 f"时间: {t:.2f}s\n"   
 f"M1位置: ({M1[0]:.0f},{M1[1]:.0f},{M1[2]:.0f})\n"   
 f"速度: {self.v\_missile:.0f}m/s\n"   
 )   
   
 # 烟团状态   
 if S is not None:   
 smoke\_info = (   
 f"\n烟团状态\n"   
 f"{'─' \* 14}\n"   
 f"中心: ({S[0]:.0f},{S[1]:.0f},{S[2]:.0f})\n"   
 f"半径: {self.R\_smoke:.0f}m\n"   
 f"起爆: {self.t\_det:.1f}s\n"   
 f"下沉: {self.smoke\_v\_down:.1f}m/s\n"   
 )   
 else:   
 smoke\_info = (   
 f"\n烟团状态\n"   
 f"{'─' \* 14}\n"   
 f"状态: 未起爆\n"   
 f"起爆: {self.t\_det:.1f}s\n"   
 )   
   
 # 遮蔽分析   
 occlusion\_info = (   
 f"\n遮蔽分析\n"   
 f"{'─' \* 14}\n"   
 f"当前: {'完全遮蔽' if occluded else '无遮蔽'}\n"   
 f"总时长: {total\_val:.2f}s\n"   
 )   
   
 return geom\_info + motion\_info + smoke\_info + occlusion\_info   
   
 def analyze\_projection\_area(self):   
 ts = np.arange(0.0, self.total\_time + 1e-9, self.dt)   
 half\_angles\_deg = []   
 dists = []   
 for t in ts:   
 M1 = self.get\_M1\_position(float(t))   
 d = float(np.linalg.norm(self.RO\_center - M1))   
 dists.append(d)   
 if d > self.RO\_radius:   
 half\_angles\_deg.append(np.degrees(np.arcsin(self.RO\_radius / d)))   
 else:   
 half\_angles\_deg.append(np.nan)   
 return ts, np.array(half\_angles\_deg), np.array(dists)   
   
 def \_on\_play(self, event):   
 self.running = True   
   
 def \_on\_pause(self, event):   
 self.running = False   
   
 def \_on\_reset(self, event):   
 self.running = False   
 self.current\_frame = 0   
 self.\_frame\_accum = 0.0   
   
 def \_on\_slider(self, val):   
 if not self.\_updating\_slider:   
 frame = int(val / self.dt)   
 self.current\_frame = min(frame, int(self.total\_time / self.dt))   
   
 def \_update\_frame(self, frame\_idx: int):   
 t = frame\_idx \* self.dt   
 try:   
 self.\_draw\_scene(self.ax3d, t)   
 if self.info\_text is not None:   
 self.info\_text.set\_text(self.\_compose\_info\_text(t))   
 if self.status\_text is not None:   
 occluded = self.is\_fully\_occluded(t)   
 status\_color = self.colors['morandi\_dusty\_rose'] if occluded else self.colors['morandi\_sage']   
 self.status\_text.set\_text(f"时间: {t:.1f}s | 遮蔽状态: {'YES' if occluded else 'NO'}")   
 self.status\_text.set\_color(status\_color)   
 except Exception:   
 pass   
   
 if self.slider is not None and not self.\_updating\_slider:   
 try:   
 cur = float(self.slider.val)   
 except Exception:   
 cur = None   
 if cur is None or abs(cur - t) > 1e-9:   
 self.\_updating\_slider = True   
 try:   
 self.slider.set\_val(t)   
 finally:   
 self.\_updating\_slider = False   
   
 def run\_interactive(self):   
 self.\_build\_layout()   
 assert self.fig is not None, "Figure not initialized"   
 total\_frames = int(self.total\_time / self.dt) + 1   
   
 def \_animate(\_i):   
 if self.running:   
 self.\_frame\_accum += float(self.play\_speed)   
 step = int(self.\_frame\_accum)   
 if step >= 1:   
 self.current\_frame = min(self.current\_frame + step, total\_frames - 1)   
 self.\_frame\_accum -= step   
 self.\_update\_frame(self.current\_frame)   
 return []   
   
 self.ani = animation.FuncAnimation(self.fig, \_animate, frames=total\_frames, interval=int(self.dt \* 1000), blit=False)   
 plt.show()   
   
   
if \_\_name\_\_ == "\_\_main\_\_":   
 proj = RealTimeProjection()   
 proj.run\_interactive()