基于虚幻引擎5的沉浸式桌面环境架构与上下文感知渲染技术研究报告

1. 绪论：桌面应用的高保真化与资源共存悖论

在当今计算机图形学与人机交互（HCI）的交叉领域中，一个显著的趋势是将游戏引擎的高保真渲染能力引入操作系统的基础层级，即桌面环境。传统的桌面壁纸应用（如Wallpaper Engine）通常依赖于视频播放或轻量级的2D渲染，以确保对系统资源的占用维持在最低水平。然而，随着虚幻引擎5（Unreal Engine 5, UE5）的普及，开发者获得了一个前所未有的机会：构建一个全3D、光线追踪、具备物理交互能力的“数字透视窗”（Digital Diorama）。

本报告旨在详细阐述一种基于UE5的高级桌面环境架构。该架构不仅仅是简单地将3D场景置于图标之下，而是通过深度集成Windows操作系统底层视窗管理机制，解决Windows 11 24H2版本带来的架构性挑战[1]；利用非对称视锥体投影（Off-Axis Projection）技术，结合计算机视觉头部追踪，将二维显示器转化为具备物理正确视差的三维窗口[3]；更核心地，本报告将构建一套“上下文感知渲染系统”。该系统响应用户的生产力状态，在用户进行高负载任务（如游戏或渲染）时，通过叙事性的交互（如角色拉上窗帘、关灯）将渲染管线从繁重的3D deferred path无缝切换至零负载的2D Slate界面，从而实现真正的“邻里友好型”软件架构[4]。

1.1 研究背景与技术挑战

传统的游戏开发逻辑是“独占式”的，即假设游戏在前台运行时应当占用尽可能多的GPU和CPU资源以提供最佳体验。然而，作为一款“伴随式”桌面应用，其设计哲学必须发生根本性转变。用户在进行生产力工作（如使用Photoshop、Visual Studio）或运行AAA级游戏时，桌面应用必须能够“退居二线”。

现有的解决方案通常采用简单的帧率限制（t.MaxFPS）[7]，但这并不能释放显存（VRAM）或完全消除GPU的上下文切换开销。本报告提出的“休眠架构”通过截取渲染目标的最后一帧作为静态覆盖层（Imposter），并彻底挂起世界渲染（World Rendering），在逻辑层面上保持“活跃”而在渲染层面上进入“假死”状态，从而解决了这一资源共存悖论[5]。

2. Windows视窗合成层的深度集成与24H2架构适配

桌面嵌入技术的核心在于拦截Windows桌面窗口管理器（DWM）的层级结构，将UE5的视口（Viewport）注入到桌面图标层（SHELLDLL_DefView）之下，但位于系统背景层（WorkerW或Progman）之上。随着Windows 11 24H2版本的发布，微软对底层视窗管理进行了重大重构，导致传统的注入技术失效 [10]。本章将深入分析这一变革及其技术应对方案。

2.1 传统注入机制（Legacy Architecture）回顾

在Windows 7至Windows 11 23H2版本中，开发者广泛采用一种基于未公开消息的“WorkerW生成法”。

系统默认的桌面背景由程序管理器（Progman）绘制。当应用程序向Progman发送十六进制消息0x052C时，Progman会触发一个未公开的内部逻辑：它会在桌面图标窗口（SHELLDLL_DefView）和壁纸之间生成一个新的窗口，类名为WorkerW [10]。这个WorkerW的设计初衷是为了支持Windows Vista时代的DreamScene动态壁纸功能，尽管该功能早已废弃，但其底层机制被保留了下来。

开发者只需通过EnumWindows遍历窗口句柄，找到包含SHELLDLL_DefView的WorkerW的兄弟窗口，调用SetParent将UE5的窗口句柄挂载到该WorkerW上，即可实现完美的桌面嵌入 [12]。这种方法的优势在于，WorkerW是持久存在的，且天然处于Z序（Z-Order）的正确位置。

2.2 Windows 11 24H2的架构性断裂

在24H2版本中，微软为了优化DWM的内存占用并引入新的视频壁纸支持 [14]，改变了WorkerW的生命周期管理策略。根据技术社区的反馈和调试分析，新的行为模式如下：

1. 瞬态生命周期（**Ephemeral Lifecycle**）：WorkerW不再是一个常驻窗口。它仅在壁纸切换动画（Fade-in/Fade-out）期间被创建，动画结束后立即销毁 [10]。
2. 父窗口失效（**Parental Invalidity**）：由于WorkerW被销毁，任何通过SetParent挂载在其下的子窗口（即UE5游戏窗口）会面临“孤儿”状态。通常表现为窗口突然消失、黑屏，或者被操作系统强制重置为顶级窗口，从而遮挡住桌面图标 [15]。
3. **Progman**层级变更：桌面合成的责任更多地回归到了Progman自身或其直接管理的层级中，

不再依赖分离的WorkerW作为渲染容器[15]。

这一变更对Lively Wallpaper和Wallpaper Engine等软件造成了广泛的兼容性崩溃，迫使开发者寻找新的注入锚点[15]。

2.3 24H2兼容的注入策略：分层直接注入法

针对24H2及其后续版本，本报告提出一种基于Progman直接子类化与显式Z序管理的注入策略。该策略不再依赖不稳定的WorkerW，而是直接介入Progman的渲染层级。

2.3.1 窗口层级重构

在新的架构中，我们需要构建如下的窗口层级关系：

- **Root**: Progman (桌面根容器)
  - **Layer 1 (Top)**: SHELLDLL_DefView (包含桌面图标的ListView，背景透明)
  - **Layer 2 (Injected)**: **UE5 Game Window** (用户自定义渲染层)
  - **Layer 3 (Bottom)**: System Background (系统壁纸层)

2.3.2 C++实现核心逻辑

为了实现这一层级，必须编写一个UE5的C++插件或在GameInstance的初始化阶段执行WinAPI操作。具体的实施步骤如下：

步骤一：环境初始化与消息发送
尽管WorkerW变得不稳定，但在初始化阶段向Progman发送0x052C消息仍然是必要的步骤 15。这一操作不仅是为了生成潜在的WorkerW，更重要的是它会强制DWM将桌面进入“分层渲染模式”（Split Layer Rendering）。如果未发送此消息，SHELLDLL_DefView可能会与背景层合并，导致无法在其间插入窗口。
步骤二：句柄发现（Handle Discovery）
我们需要获取关键的锚点窗口句柄。在C++中，应结合FindWindow和FindWindowEx，甚至EnumChildWindows来确保鲁棒性，因为不同版本的Windows可能会微调类名层级 15。

C++

```
// 伪代码逻辑：获取关键句柄
HWND hProgman = FindWindow(L"Progman", nullptr);
HWND hShellView = nullptr;
HWND hWorkerW = nullptr;

// 递归查找 SHELLDLL_DefView
EnumWindows((HWND hwnd, LPARAM lParam) -> BOOL {
    HWND hShell = FindWindowEx(hwnd, nullptr, L"SHELLDLL_DefView", nullptr);
    if (hShell) {
        *(HWND*)lParam = hShell;
        return FALSE; // 停止遍历
    }
    return TRUE;
}, (LPARAM)&hShellView);
```

步骤三：父子关系重置（Re-parenting）
获取到UE5的主窗口句柄（GGameWindow）后，调用SetParent将其父窗口设置为Progman（在旧版中是设置为WorkerW）。
SetParent(hUE5Window, hProgman);
步骤四：显式Z序注入（Explicit Z-Order Injection）
这是解决24H2黑屏问题的核心。由于我们直接挂载在Progman下，必须明确告诉DWM我们的窗口应位于图标层之下。使用SetWindowPos函数，并将hWndInsertAfter参数设置为SHELLDLL_DefView的句柄 15。

C++

```
// 将UE5窗口置于图标窗口之后
SetWindowPos(hUE5Window, hShellView, 0, 0, 0, 0,
             SWP_NOMOVE | SWP_NOSIZE | SWP_NOACTIVATE | SWP_SHOWWINDOW);
```

此操作确保了即便WorkerW被销毁，UE5窗口依然作为Progman的直接子窗口，且被严格限制在图标层下方。

表 1：**Windows**版本架构差异与注入策略对比

特性维度	**Windows 10 / 11 (23H2及**	**Windows 11 (24H2及以后)**

	以前)	
渲染容器	WorkerW (通过0x052C生成)	Progman (直接挂载)
生命周期	持久存在 (Persistent)	瞬态/易变 (Ephemeral)
Z序管理	隐式 (挂载即生效)	显式 (需调用 SetWindowPos 指定 InsertAfter)
故障表现	极少	窗口消失、黑屏、遮挡图标
推荐策略	查找并挂载至 WorkerW	挂载至 Progman 并强制 Z-Order < SHELLDLL_DefView

---

3. 自适应渲染管线：基于上下文的“深度休眠”架构

用户需求的独特之处在于其对“优化”的定义超越了单纯的技术指标，融合了叙事性与逻辑性。用户希望当全屏游戏或生产力软件运行时，桌面角色不仅仅是“暂停”，而是通过“关灯”、“拉窗帘”等行为进入逻辑上的休息状态，从而彻底释放GPU算力。这要求我们构建一个**渲染与逻辑解耦（Decoupled Rendering and Logic）**的架构。

3.1 渲染挂起（Rendering Suspension）的底层机制

在虚幻引擎中，简单的t.MaxFPS限制虽然能减少帧数，但每一帧的渲染开销（G-Buffer生成、光照计算、后处理）依然存在，且显存占用不会减少 [4]。为了实现“零负载”，必须切断渲染线程（Render Thread）的工作流。

3.1.1 叙事驱动的管线切换

当系统检测到需要进入休眠状态时，状态机应执行以下序列：

1. 叙事过渡（**Narrative Transition**）：角色执行Montage_Sleep，例如走到窗前拉上窗帘。此时，场景内的动态光源（Stationary/Movable Lights）强度逐渐通过Timeline插值降至0，模拟“关灯”效果。
2. 帧捕获（**Frame Capture**）：在窗帘完全闭合或画面全黑的那一帧，引擎触发一次SceneCapture2D或者直接复制当前的BackBuffer到一个UTexture2D中。这一步至关重要，它生成了一张“替身纹理”（Imposter Texture），用于在真正的3D渲染停止后欺骗用户的视觉[5]。
3. **UI覆盖（UI Overlay）**：创建一个全屏的UMG Widget，将上述“替身纹理”作为Image控件显示在最上层。
4. 渲染截断（**Render Cutoff**）：调用UGameViewportClient::SetEnableWorldRendering(false)。
    - 技术解析：该函数直接作用于引擎的主循环（Main Loop）。当设置为false时，FEngineLoop::Tick会跳过Draw阶段的大部分内容，不再向GPU发送绘制指令（Draw Calls）。此时，GPU的使用率将从30%-50%骤降至接近0%（仅剩Windows DWM的合成开销）[4]。

3.2 逻辑层的持续性与时间同步

在渲染停止后，UE5的**游戏线程（Game Thread）**并不会自动停止[18]。这正是实现用户需求的关键：角色需要在“后台”计算“回来”的动画。

- 逻辑更新（**Logic Tick**）：即使没有画面输出，角色的蓝图逻辑（Blueprint Logic）、状态机（State Machine）和AI控制器依然在运行。为了进一步节省CPU资源，此时应配合t.MaxFPS 1或t.MaxFPS 5，将逻辑更新频率降至最低[9]。
- 虚拟时间累积：在休眠期间，系统记录DeltaTime。如果用户离开电脑4小时，游戏逻辑会累积这段时间，增加角色的“饥饿值”或“疲劳恢复值”。
- 唤醒预测（**Wake-up Prediction**）：当IPC模块（见第5章）检测到前台窗口关闭的信号时：
    1. 恢复t.MaxFPS 60。
    2. 恢复SetEnableWorldRendering(true)。
    3. 在渲染恢复的第一帧，UMG覆盖层依然存在，掩盖了3D场景重新加载或初始化的潜在抖动。
    4. 播放Montage_WakeUp（拉开窗帘），UMG层淡出，无缝过渡回3D场景。

3.3 显存管理与资源释放

虽然SetEnableWorldRendering(false)停止了计算，但加载到显存中的纹理和模型依然占用VRAM。对于极端的优化（如用户运行显存敏感的3A大作），可以引入**流送暂停（Streaming Pause）**机制 20。
通过FStreamingPauseRenderingModule，可以在休眠期间强制卸载非必要的Mips或将高分辨率纹理标记为不可流送，从而释放数百兆的显存空间供前台游戏使用。这需要精细的C++内存管理策略，但能显著提升“邻里友好度”。

---

4. 光学错觉工程：非对称视锥体与物理视差

将显示器转变为“窗口”的核心在于非对称视锥体投影（**Asymmetric Frustum Projection**），亦称为离轴投影（Off-Axis Projection）。标准的3D摄像机模拟的是单眼且头部固定的观察者，而本系统需要根据用户头部的实时位置（X, Y, Z）动态调整投影矩阵，使得画面产生物理上正确的视差变化 [21]。

4.1 投影矩阵的数学重构

标准透视投影矩阵假设视点位于屏幕平面的中心法线上。当观察者移动时，我们需要构建一个剪切（Sheared）的视锥体。根据线性代数理论，通用的透视投影矩阵定义如下：

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$
其中，n 和 f 分别是近裁剪面和远裁剪面距离。关键变量 l (left), r (right), t (top), b (bottom) 定义了近裁剪面上的视窗边界。在对称投影中，$l = -r$。而在离轴投影中，这些值由头部相对于屏幕中心的位置决定：

- 设屏幕宽度为 W，高度为 H。
- 头部位置为 $P_{head} = (x, y, z)$（以屏幕中心为原点）。
- $l = (-W/2 - x) \cdot \frac{n}{z}$
- $r = (W/2 - x) \cdot \frac{n}{z}$
- $t = (H/2 - y) \cdot \frac{n}{z}$
- $b = (-H/2 - y) \cdot \frac{n}{z}$

这意味着当用户向左移动（$x < 0$）时，r 的绝对值将大于 l 的绝对值，视锥体向右倾斜，从

而让用户看到更多物体右侧的细节，形成“透视”错觉 [3]。

4.2 UE5中的投影矩阵覆写（Matrix Override）

虚幻引擎的UCameraComponent并不直接暴露l, r, t, b参数，也不支持直接设置投影矩阵。为了实现这一功能，必须深入引擎底层，覆写ULocalPlayer类的投影数据获取逻辑 [23]。

实现路径：

1. 自定义**LocalPlayer**：创建一个继承自ULocalPlayer的C++类，例如UOffAxisLocalPlayer。
2. 虚函数覆写：重写GetProjectionData函数。这是渲染线程获取投影矩阵的源头。
   C++
   ```
   bool UOffAxisLocalPlayer::GetProjectionData(FViewport* Viewport,
     FSceneViewProjectionData& ProjectionData, int32 StereoViewIndex) const
   {
     // 首先调用父类方法填充基础数据
     bool bResult = Super::GetProjectionData(Viewport, ProjectionData, StereoViewIndex);

     if (bResult)
     {
       // 从GameInstance或IPC模块获取最新的头部位置
       FVector HeadPos = GetHeadTrackingData();

       // 根据上述公式手动构建 FMatrix
       FMatrix OffAxisMatrix = CalculateAsymmetricProjection(HeadPos,...);

       // 强制覆盖引擎计算的矩阵
       ProjectionData.ProjectionMatrix = OffAxisMatrix;
     }
     return bResult;
   }
   ```
3. 全局配置：在项目的DefaultEngine.ini中，将默认的LocalPlayer类指定为UOffAxisLocalPlayer。

这一方法的优势在于它不仅影响几何体的渲染，还会自动修正阴影贴图（Shadow Maps）的投影裁剪、遮挡剔除（Frustum Culling）以及后处理体积的计算，确保整个渲染管线的一致性 [26]。如果仅在Shader层面进行顶点偏移，会导致阴影计算错误和严重的剔除伪影。

---

5. 感知系统与进程间通信（IPC）架构

为了实现“上下文感知”和“头部追踪”，UE5需要一只“天眼”。由于UE5本身的沙盒限制，它无法高效地监控外部进程（如检测Photoshop是否在前台）或直接访问底层摄像头硬件进行复杂的计算机视觉计算。因此，引入一个外部的Python守护进程（Watchdog）是最佳实践。

5.1 基于MediaPipe的头部追踪系统

头部追踪的精度和延迟直接决定了视窗体验的成败。传统的OpenCV人脸检测（Haar Cascades）抖动大且缺乏深度信息。本方案推荐使用**Google MediaPipe Face Mesh** [27]。

- 高精度拓扑：MediaPipe Face Mesh能够实时检测人脸上的468个3D关键点（Landmarks）。
- 姿态解算（**Pose Estimation**）：通过选取其中的关键锚点（如鼻尖、眼角、下巴），利用OpenCV的solvePnP（Perspective-n-Point）算法，结合通用的3D人脸模型，可以解算出头部在空间中的旋转向量（Rotation Vector）和平移向量（Translation Vector）[29]。
- 平滑处理：原始数据通常含有高频噪声。必须在Python端应用**卡尔曼滤波（Kalman Filter）或一欧元滤波器（1€ Filter）**对(X, Y, Z)坐标进行平滑处理，以防止画面抖动引发用户晕眩[31]。

5.2 Python看门狗与进程监控

该Python脚本同时承担着环境感知的任务。它利用psutil和win32gui库周期性地（如每1秒）查询当前的前台窗口句柄 [32]。

- 全屏独占检测：通过对比前台窗口的尺寸与屏幕分辨率，判断用户是否在运行全屏游戏 [32]。
- 白名单/黑名单机制：维护一个进程名列表（如blender.exe, cod.exe）。一旦检测到这些进程获得焦点，立即触发“休眠”信号。

5.3 UDP异步通信管线

Python进程与UE5之间的数据传输必须具备极低的延迟。**UDP（User Datagram Protocol）**协议是此类实时数据的首选，优于TCP和Named Pipes [35]。

- 协议设计：定义一个紧凑的字节流数据包结构。
  - Header: 标识符（防止端口串扰）
  - Payload:
    - State (byte): 0=Active, 1=Sleep
    - HeadX (float), HeadY (float), HeadZ (float)
- **UE5**接收端设计：
  - 直接在主线程（Game Thread）接收网络数据会导致卡顿。必须使用FRunnable接口创建一个专用的接收线程 [38]。
  - 该线程在一个while循环中阻塞式地监听Socket，一旦收到数据，将其存入一个线程安全的队列（Thread-Safe Queue）或直接更新GameInstance中的原子变量（Atomic Variables）。
  - 游戏线程在Tick中读取这些变量，驱动摄像机矩阵或触发休眠逻辑。

表 **2**：通信协议与延迟分析

通信方式	典型延迟	丢包容忍度	实现复杂度	适用场景
TCP Socket	10-50ms	低 (重传导致阻塞)	中	状态同步 (如休眠信号)
UDP Socket	1-5ms	高 (丢弃旧帧)	低	头部追踪数据 (推荐)
Named Pipes	<1ms	无 (本地内存复制)	高 (UE5支持有限)	极低延迟的本地数据交换

---

6. 资产优化与技术美术策略：单一房间的极限性能

用户明确指出只需“一个房间”，并提到了“远景用法线贴图”的优化思路。这涉及到了技术美术（Tech Art）中的**伪造深度（Fake Depth）**技术。

6.1 立方体贴图与视差遮挡映射（POM）

对于窗外的景色（如城市、森林），构建真实的3D模型是极大的浪费。即便使用了LOD（细节层次），Draw Calls依然不可忽视。

推荐使用**视差遮挡映射（Parallax Occlusion Mapping, POM）或内部立方体贴图（Interior Cubemap）**技术 39。

- 实现原理：在窗户外部放置一个简单的平面（Plane）。在材质中，利用视线向量（Camera Vector）对UV坐标进行偏移采样。配合法线贴图和高度图，可以在一个平面上模拟出具有深度的建筑结构或街道。
- 性能收益：整个窗外景色仅由2个三角形构成，Draw Call为1。无论头部如何移动，视差效果都能保持正确的透视感，完全满足“更远景的用法线贴图”这一需求。

6.2 静态光照与动态角色的混合

为了节省GPU算力，应避免使用Lumen全动态全局光照。

- **GPU Lightmass**：利用UE5的GPU Lightmass烘焙静态光照。房间的墙壁、家具阴影被永久写入光照贴图（Lightmaps），运行时的开销几乎为零。
- 胶囊体阴影（**Capsule Shadows**）：对于动态的角色，使用物理资产（Physics Asset）生成的胶囊体阴影。这种阴影基于屏幕空间或简化的距离场计算，比全动态的虚拟阴影贴图（Virtual Shadow Maps, VSM）极其廉价，却能提供柔和且令人信服的接触阴影（Contact Shadows）[41]。

---

7. 结论与未来展望

本报告详细论证了一套基于UE5的高性能、沉浸式桌面环境架构。该架构通过底层的WinAPI集成解决了Windows 11 24H2的兼容性危机；通过数学层面的投影矩阵重构实现了裸眼伪3D的视觉体验；更通过创造性的“叙事性休眠”机制，将渲染管线的管理与用户的使用场景深度绑定，实现了真正意义上的“零干扰”。

这种设计范式不仅仅适用于桌面壁纸，它代表了未来**环境计算（Ambient Computing）**应用的一种方向：软件应当像家具一样，平时静默存在，仅在被注视时展现其丰富性。随着硬件算力的提升和操作系统的演进，未来的桌面可能会完全演变为这种3D空间，而本报告所述的技术——层级注入、非对称投影、上下文感知渲染——将是构建这一未来的基石。

7.1 实施建议汇总

1. 首要任务：优先开发并测试C++的Progman注入模块，确保在Windows 11 24H2上窗口不黑屏、不置顶。

2. 核心优化：尽早实现SetEnableWorldRendering(false)与UMG覆盖层的切换逻辑，这是达成性能指标的关键。
3. 原型验证：使用Python脚本快速验证MediaPipe到UE5 UDP的数据通路，确保头部追踪的延迟在可接受范围内（<20ms）。
4. 美术规范：严格限制窗外景物为POM材质，室内场景采用全烘焙流程，确保在唤醒状态下的GPU占用也控制在合理范围（如RTX 3060上<20%）。

引用的著作

1. Setting a window as wallpaper [closed] - c++ - Stack Overflow, 访问时间为 十一月 22, 2025, https://stackoverflow.com/questions/79763352/setting-a-window-as-wallpaper
2. A general notice to those Updating to Windows 11 24H2 and using Wallpaper Engine, 访问时间为 十一月 22, 2025, https://steamcommunity.com/app/431960/discussions/1/4846526727955382962/?l=hungarian&ctp=2
3. Achieving Off-axis projection - c++ - Stack Overflow, 访问时间为 十一月 22, 2025, https://stackoverflow.com/questions/16416552/achieving-off-axis-projection
4. How to limit GPU usage in a project that does not require 3d graphics? (UE4.27), 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/how-to-limit-gpu-usage-in-a-project-that-does-not-require-3d-graphics-ue4-27/492545
5. Pause world rendering during game pause - Unreal Engine Forum, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/pause-world-rendering-during-game-pause/2660567
6. Disable/Enable 3D Rendering at Runtime - Blueprint - Unreal Engine Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/disable-enable-3d-rendering-at-runtime/2286247
7. Ue5 using 100% gpu in an empty/default scene - Unreal Engine Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/ue5-using-100-gpu-in-an-empty-default-scene/492580
8. How do I make UE5 not use so much of my GPU? : r/unrealengine - Reddit, 访问时间为 十一月 22, 2025, https://www.reddit.com/r/unrealengine/comments/12a49vv/how_do_i_make_ue5_not_use_so_much_of_my_gpu/
9. Pause rendering and continually display the last-rendered frame - Unreal Engine Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/pause-rendering-and-continually-display-the-last-rendered-frame/125429
10. Draw on Windows 10 wallpaper in C++ - Stack Overflow, 访问时间为 十一月 22, 2025, https://stackoverflow.com/questions/56132584/draw-on-windows-10-wallpaper-i

n-c
11. 实现桌面动态壁纸（一）原创 - CSDN博客, 访问时间为 十一月 22, 2025, https://blog.csdn.net/qq_59075481/article/details/125361650
12. Attach window to desktop (like a desktop widget) - AutoHotkey - Reddit, 访问时间为 十一月 22, 2025, https://www.reddit.com/r/AutoHotkey/comments/pkk76n/attach_window_to_desktop_like_a_desktop_widget/
13. Doubts about the window of "program manager" - Microsoft Q&A, 访问时间为 十一月 22, 2025, https://learn.microsoft.com/en-us/answers/questions/1386630/doubts-about-the-window-of-program-manager
14. Hands on with Windows 11's new "video wallpaper" feature, supports .mp4 as desktop background, 访问时间为 十一月 22, 2025, https://www.windowslatest.com/2025/09/22/hands-on-with-windows-11s-new-video-wallpaper-feature-supports-mp4-as-desktop-background/
15. Win11 preview version 26XX dynamic wallpaper invalid · Issue #2074 · rocksdanister/lively, 访问时间为 十一月 22, 2025, https://github.com/rocksdanister/lively/issues/2074
16. WARNING!! Wallpapers are UNABLED to be displayed on Windows 24H2 Update : r/wallpaperengine - Reddit, 访问时间为 十一月 22, 2025, https://www.reddit.com/r/wallpaperengine/comments/1fxtrgt/warning_wallpapers_are_unabled_to_be_displayed_on/
17. Prevent drawing over a child window under WorkerW when wallpaper is changed, 访问时间为 十一月 22, 2025, https://stackoverflow.com/questions/78169263/prevent-drawing-over-a-child-window-under-workerw-when-wallpaper-is-changed
18. [UE5] Understanding Render Thread and Animation Thread in Unreal Engine - YouTube, 访问时间为 十一月 22, 2025, https://www.youtube.com/watch?v=RRwNlntV10I
19. Unreal Engine Multithreading Techniques - Epic Games Developers, 访问时间为 十一月 22, 2025, https://dev.epicgames.com/community/learning/tutorials/BdmJ/unreal-engine-multithreading-techniques
20. FStreamingPauseRenderingMod, 访问时间为 十一月 22, 2025, https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Runtime/StreamingPauseRendering/FStreamingPauseRenderingModule
21. Implementation of Headtracking and 3D Stereo with Unity and VRPN for Computer Simulations - NASA Technical Reports Server (NTRS), 访问时间为 十一月 22, 2025, https://ntrs.nasa.gov/api/citations/20130014602/downloads/20130014602.pdf
22. Calibration Methods for Head-Tracked 3D Displays - CORE, 访问时间为 十一月 22, 2025, https://core.ac.uk/download/226118531.pdf
23. ULocalPlayer | Unreal Engine 5.7 Documentation - Epic Games Developers, 访问时间为 十一月 22, 2025, https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Runtime/Engi

ne/ULocalPlayer
24. How to change the projection matrix at the scene hit proxies calculation, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/how-to-change-the-projection-matrix-at-the-scene-hit-proxies-calculation/422781
25. Howto modify the projection matrix - Programming & Scripting - Unreal Engine Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/howto-modify-the-projection-matrix/287457
26. GetDynamicMeshElements and shadow passes - C++ - Unreal Engine Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/getdynamicmeshelements-and-shadow-passes/18322
27. layout: forward target: https://developers.google.com/mediapipe/solutions/vision/face_landmarker/ title: Face Mesh parent: MediaPipe Legacy Solutions nav_order: 2 — MediaPipe v0.7.5 documentation - Read the Docs, 访问时间为 十一月 22, 2025, https://mediapipe.readthedocs.io/en/latest/solutions/face_mesh.html
28. Real-Time Head Pose Estimation FaceMesh with MediaPipe and OpenCV: A Comprehensive Guide | by Jaykumaran R | Medium, 访问时间为 十一月 22, 2025, https://medium.com/@jaykumaran2217/real-time-head-pose-estimation-facemesh-with-mediapipe-and-opencv-a-comprehensive-guide-b63a2f40b7c6
29. Perspective-n-Point (PnP) pose computation - OpenCV Documentation, 访问时间为 十一月 22, 2025, https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html
30. Head Pose Estimation using OpenCV and Dlib | LearnOpenCV #, 访问时间为 十一月 22, 2025, https://learnopencv.com/head-pose-estimation-using-opencv-and-dlib/
31. Simple Head Tracking with OpenCV - RobotShop Community, 访问时间为 十一月 22, 2025, https://community.robotshop.com/tutorials/show/simple-head-tracking-with-opencv
32. Python check if application is in fullscreen - Stack Overflow, 访问时间为 十一月 22, 2025, https://stackoverflow.com/questions/26241540/python-check-if-application-is-in-fullscreen
33. psutil documentation — psutil 7.2.0 documentation, 访问时间为 十一月 22, 2025, https://psutil.readthedocs.io/
34. Is there a way to detect when another app enters fullscreen? - Microsoft Learn, 访问时间为 十一月 22, 2025, https://learn.microsoft.com/en-us/answers/questions/1516325/is-there-a-way-to-detect-when-another-app-enters-f
35. Interprocess Communication Through Pipes - C++ Forum, 访问时间为 十一月 22, 2025, https://cplusplus.com/forum/general/165198/
36. Controlling a Pictorus app over UDP from Python, 访问时间为 十一月 22, 2025, https://blog.pictor.us/interfacing-with-a-pictorus-app-over-udp-with-python/
37. Send image over UDP to Unreal Engine and use it as a Texture - Programming &

Scripting, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/send-image-over-udp-to-unreal-engine-and-use-it-as-a-texture/1745440
38. Multithreading and Performance in Unreal - Programming & Scripting - Epic Developer Community Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/multithreading-and-performance-in-unreal/1216417
39. Improve GPU Performance in Unreal Engine 5 by Allocating Maximum VRAM for Texture Streaming - YouTube, 访问时间为 十一月 22, 2025, https://www.youtube.com/watch?v=yMHXwh_q8TA
40. I've disabled most of UE5 render features to make this Game. Low poly, 256 textures, all scalability = 0, tested on Potato. - Reddit, 访问时间为 十一月 22, 2025, https://www.reddit.com/r/UnrealEngine5/comments/1nz78ot/ive_disabled_most_of_ue5_render_features_to_make/
41. Disable rendering of object, but keep shadows - Unreal Engine Forums, 访问时间为 十一月 22, 2025, https://forums.unrealengine.com/t/disable-rendering-of-object-but-keep-shadows/338285