最大限降低虚拟现实系统中的运动到光子延迟（MPD）

Akanksha Dixit印度理工学院电气工程印度新德里
Akanksha.Dixit@ee.iitd.ac.in

Smruti R. Sarangi印度理工学院电气工程印度新德里
srsarangi@cse.iitd.ac.in

arXiv:2301.10408v1 [cs.AR] 25 Jan 2023

摘要—**随着低功耗超高速硬件和图形处理器（GPU）的出现，虚拟现实（VR）在过去几年中日益受到重视，并被应用于教育、娱乐、科学可视化和计算机辅助设计等多个领域。基于虚拟现实的应用具有很强的交互性，而这类应用最重要的性能指标之一就是运动到光子延迟（MPD）。MPD 是指从用户头部移动到虚拟现实屏幕上的图像更新之间的延迟。由于人类视觉系统甚至能够察觉几个像素的误差（对空间变化非常敏感），因此 MPD 应尽可能小。**

主流虚拟现实供应商采用 GPU 加速异步时间扭曲（ATW）算法来降低 MPD。只有在扭曲操作恰好于显示器刷新前完成时，ATW 才能降低 MPD。然而，由于多个应用程序争用共享 GPU，GPU 加速的 ATW 算法会面临难以预测的 ATW 延迟，因此很难找到启动时间扭曲的理想时机，并确保其相对于屏幕刷新以尽可能小的延迟完成。因此，当前最先进的方案是使用独立硬件单元执行时间扭曲操作。我们提出的 *PredATW* 使用基于机器学习的预测器来预测 VR 应用的 ATW 延迟，并据此尽可能延迟调度。这是首项采用这种方法的工作。我们的预测器在多个主流 VR 应用中预测 ATW 延迟的误差为 0.77 毫秒。与基线架构相比，我们将截止期限错失减少了 73.1%。

一、引言

过去几年间，随着众多颠覆性技术相继问世，计算机图形学应用（例如虚拟现实 [1], [2]应用程序）变得极为流行。虚拟现实通过让用户沉浸于高度互动的环境中，彻底改变了游戏和娱乐行业 [3], [4]。它也在医学领域占据了重要地位 [5], [6]。全球虚拟现实市场在2021年估值为218.3亿美元，预计从2022年至 2030 [7]将以15.0%的复合年增长率（CAGR）扩张。基于虚拟现实的应用程序具有高度互动性，其中最重要的性能指标之一是运动到光子延迟（MPD） [8]，[9]。MPD指的是从用户头部开始运动，到虚拟现实屏幕上的图像更新所需的时间延迟。

为了获得良好的虚拟现实体验，MPD 应尽可能小（详见第 III-E 节）。否则，用户会感知到异常视图，这可能会导致不适和视觉定向障碍。最严重的情况下，还可能引发更严重的健康问题，例如晕动症及后续的恶心 [10], [11], [12]。较大且变化不定的 MPD 所造成的影响，往往要过很久才会显现（我们自己使用 Oculus Quest 2 时就有过这样的体验）。此外，在人类面临风险的应用场景中，例如使用虚拟现实进行机器人手术，MPD 就显得更加关键。多种虚拟现实设备，如 Oculus Quest 2 [13]和 HTC Vive [14]，都采用了一种通常称为 ATW（异步时间扭曲）的技术来降低 MPD。使用此技术的应用程序通过图形处理器加速；其总体目标是实现较短的 MPD。

时间扭曲是一种图像重投影技术，它根据最新的头部方向信息，将先前渲染的帧映射到正确位置，并与渲染过程并行运行 [15]。具体而言，它会根据渲染开始至启动时间扭曲 [16]操作期间头部位置的变化，将图像平移若干像素。这样做的优势在于能够显著改善整体用户体验。目前，虚拟现实头显的刷新率为90 Hz，这意味着每隔11 ms，屏幕上就会显示一帧新画面。如果渲染耗时不到11 ms，且还有充足时间，我们便可以在显示刷新截止时间前对渲染图像进行“时间扭曲”，使其更加贴近当前状态。若渲染耗时更长（超过11 ms），则可以对上一帧进行“时间扭曲”并将其显示出来。

这需要一个能够并发运行的异步时间扭曲（ATW）操作 [16]。鉴于我们无法确定渲染任务能否在期限内完成，至少完成 ATW 任务至关重要，这样上一渲染帧才能根据当前姿态进行变换。这意味着 ATW 可能需要抢占渲染任务，并将上下文从渲染切换到时间扭曲。因此，抢占和上下文切换的延迟也会增加到 ATW 的延迟中。由于 ATW 使用与渲染过程相同的图形资源，其延迟因此难以预测。此外，同一图形处理器上可能还有其他应用程序在运行，这会进一步增加噪声。这就是当前 GPU 加速的 ATW 解决方案无法实现理想 MPD 的原因：ATW 操作的延迟难以预测 [17]。

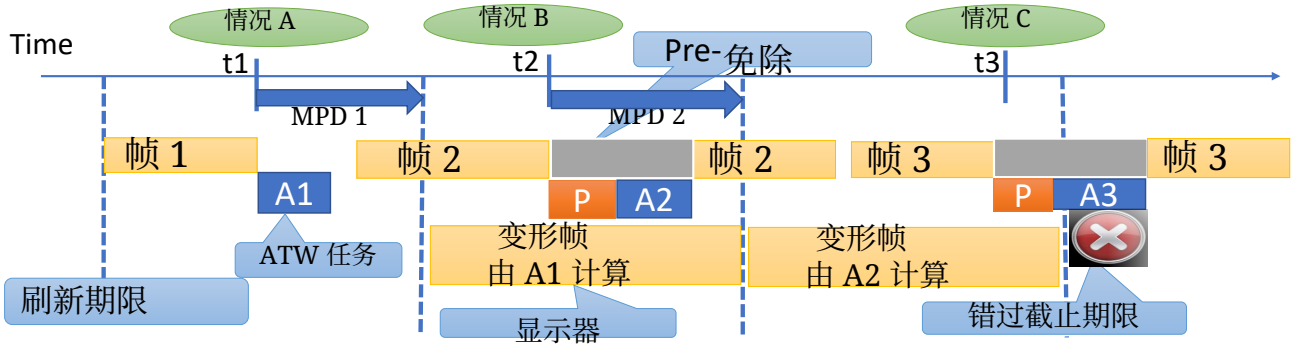

图 1：GPU 加速的 ATW 算法中的可能场景。 $A1$、 $A2$和 $A3$ 是 ATW 任务。 P 是抢占延迟

显著降低运动到光子延迟（MPD）的一种可能方法是加快时间扭曲操作。然而，Xie 等人 [17] 表明，即使时间扭曲的实现速度很快，如果未能在正确的时机启动，应用程序仍可能出现较长的 MPD。他们展示了两种情况下 MPD 都会增加：时间扭曲在显示刷新前完成，或者错过刷新期限。图 1 对此作了解释。图中展示了三种情况 A、B 和 C。在 A 中，时间扭曲任务（$A1$）早在刷新期限之前就已完成，而显示器刷新后显示的变形帧对应的是在时间 $t1$ 采集的头部姿态；这会导致较长的 MPD。因此，我们应尽量减少延迟。在 B 和 C 这两种情况下，渲染操作都会中断以调用 ATW；但在后一种情况下，由于抢占延迟较长 P，错过了刷新期限，用户体验也因此不佳。因此，这里的主要问题是找到启动时间扭曲操作的正确时刻。这并非易事，因为 ATW 延迟是可变的。

虚拟现实应用程序属于硬实时系统，截止期限在大多数情况下都很重要（正如我们刚才所看到的）。为了降低 MPD，已有大量研究尝试将 ATW 从图形处理器卸载到独立的硬件组件上 [17], [18]。一些研究聚焦于预测跟踪 [19], [20] 和运动预测 + ，以及可编程显示层（PDL） [21], [22]。[23], [24]则试图优化渲染过程本身，以降低 MPD。然而，当 ATW 和虚拟现实应用程序共享同一图形处理器时，如何降低 MPD 仍然是一个尚未解决的问题。

这构成了我们开展*PredATW*工作的动机。我们的工作旨在预测 ATW 操作应开始的时间点，从而在不对帧率和画质造成负面影响的情况下，将 MPD 降至最低。据我们所知，我们是首个预测 ATW 延迟并在同一图形处理器上运行所有操作的团队。我们提出了一种简单的基于决策树的预测器，它使用帧特定特征（当前帧或上一渲染帧），例如绘制调用的数量、亮度和图形处理器执行时间，以及硬件性能计数器的值，作为预测器的特征。基于决策树的预测器的优势在于，我们的结果具有可解释性。

我们的主要贡献如下：

- 我们表明，可以利用按帧收集的简单指标和性能计数器的数值来预测图形处理器上的 ATW 延迟。此外，我们还表明，最重要的两个指标是上一渲染帧的渲染时间和上一次 ATW 操作的延迟。 我们比较了各种基于机器学习的预测模型的性能，并表明与其他基于机器学习的模型相比，简单的决策树能够提供准确的结果。
- 与 GPU 加速的 ATW 架构相比，我们将 ATW 刷新截止期限错失率降低了 73%。在 11.1 毫秒的帧时长内，我们的预测误差为 0.77 毫秒，表现尚可。

本文的组织结构如下：第二节介绍虚拟现实架构和基于机器学习的模型背景。第三节讨论本研究的动机。第四节给出实现细节。第五节展示实验结果。第六节讨论相关工作，最后在第七节总结全文。

II. 背景

A. VR 架构循环

图 2 简要展示了典型的 VR 架构循环。在虚拟现实系统中，头戴式显示器追踪器首先收集用户的位置或运动数据，然后将这些信息传输给应用程序。应用程序将全部数据复制到图形处理器内存中，并设置渲染过程。最后，渲染系统为双眼生成两帧，并在头戴式显示器（HMD）上显示这些帧。在此过程中，用户头部运动与图像在虚拟现实屏幕上更新之间存在延迟；这种延迟称为运动到光子延迟（MPD）。为了提供良好的用户体验，MPD 应尽可能短。时间扭曲是一种用于减少这一延迟的图像重投影技术。

如前所述，时间扭曲会根据最新的头部朝向信息，将前一帧渲染帧映射到正确的位置。图 2 展示了这种扭曲架构的高层概览。请注意，ATW 仅适用于

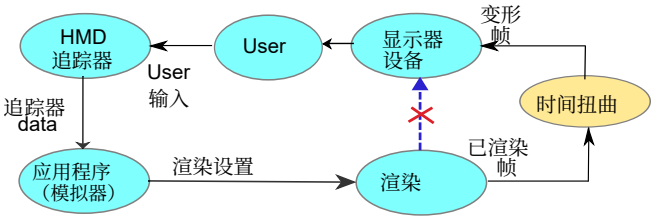

图 2：VR 架构循环概览

头部旋转追踪，这意味着修改前一帧显示帧时，仅考虑头部的旋转变化。

B. 帧间相似度

Zhao 等人表明， [25] VR 应用中存在帧间相似度。我们将重点关注 VR 应用的这一方面，以预测 ATW 延迟。本研究采用两种方法来证明 VR 应用中存在帧间相似度。第一种方法是直接比较帧内容（经过预处理）。第二种方法则是比较少量帧特定特征（基于图像和运行时性能计数器）。这里先讨论第一种方法，因为它依赖于先前的工作；第二种方法将在第 III 节中介绍，因为它需要我们开发的一些定制机制。

1）方法 *1*：宏块相似性：为了计算帧内容的相似性，我们采用 [26], [27] 中提到的基于 DCT（离散余弦变换）的方法来计算帧之间的相似度。在该方法中，帧相似性的计算完全基于 DCT 的 DC（零频率）系数，因为这些系数表示块摘要信息。由于人眼对亮度比分量更敏感，因此只检查亮度分量的 DC 系数，而忽略色度分量。当我们获得了要计算相似度的两帧的 DC 系数后，便在宏块层面分析相似度，以高效捕获帧差异。A宏块 是一个 2×2 矩阵，传统上用于基于线性块变换的图像和视频压缩[28]。设两帧分别为 A 和 B。我们为每一帧构造宏块矩阵 M_1..... M_n ，其中 n 表示宏块（DC 系数）的数量。如果至少有最小数量的宏块相似，则认为两帧相似。

C. 基于机器学习的预测模型

在机器学习中，预测问题有两种类型：分类和回归。在分类中，我们要寻找一个能够帮助我们为数据项分配类别的模型。而回归则是一种为数据分配连续实数值，而非类别或标签的方法。由于我们要寻找 ATW 延迟，因此我们的问题属于回归类别。回归模型使用均方误差（MSE）作为损失函数。

1) 线性回归： 线性回归是用于回归的最简单模型之一。它试图找出因变量 Y 与一组自变量 X之间的线性关系。换句话说，它拟合一条直线或一个曲面，使实际值与预测值之间的误差最小化。

2) 决策树回归：：决策树既可用于分类任务，也可用于回归任务。该模型最重要的特点之一是可解释性强，而且其性能不受数据非线性的影响。决策树以树状结构构建模型，并将整个数据集划分为更小的子集，以降低误差。树中有两类节点：决策（内部）节点和叶节点。根节点是位于树顶端的决策节点，覆盖整个样本。每个决策节点都有一个条件，用于决定选择哪条路径；而叶节点则包含最终的预测结果。

构建模型时，我们从根节点开始。要确定根节点的特征，我们遍历所有特征，并找出数据集的最优划分。数据集会持续划分，直到误差最小化或达到特定阈值，例如树深度或叶节点数量的上限。用于决定如何划分数据集的指标是平方误差和。此过程以递归方式持续进行。要预测给定数据点的值，我们沿着决策节点中的条件遍历整棵树，直至到达叶节点，由此得到最终输出。

3）随机森林回归：：顾名思义，随机森林由一组决策树组成。它将这些决策树的结果组合起来，得出最终结果。将多个模型或弱学习器的结果合并以得出最终结果，这称为集成学习。首先，我们将包含 n 个点的数据集随机划分为 k 个部分。然后，为所有部分分别独立构建决策树。在预测过程中，将所有这些决策树的输出取平均值，得出最终结果。这种模型的缺点是需要严格训练，因此速度很慢且结构复杂。

4）梯度提升回归：：梯度提升回归器也是集成方法的一种变体。它会构建多棵回归树，然后将它们的结果组合起来，得出最终结果。它与随机森林回归器的不同之处在于，构建回归树时会依据残差；残差是实际输出与预测输出之间的差值。利用这些残差和特征来训练回归树，并将这些模型预测出的残差纳入输入模型。这个过程会重复多次，使输入模型逐步趋向正确的预测结果。

5) 卷积神经网络（*CNN*）：：CNN 是一种用于预测的神经网络。由于以下原因，我们在本例中不使用 CNN。

- CNN 的结构非常复杂，每一层都使用多个乘法器和加法器。因此，它运行速度较慢，而且需要大量存储空间。由于我们的目标是预测 ATW 延迟以降低 MPD，因此不适合使用延迟较大的预测器。
- 第二个原因是 CNN 不具备可解释性。我们无法判断每个特征对最终预测的贡献。由于我们希望分析应用程序和系统的行为，因此需要一个可解释的模型。

III. 动机

本节首先列出我们在实验中使用的基准测试。然后，我们评估共享 GPU 环境中时间扭曲的延迟。接着，我们将展示 VR 应用中连续帧之间存在显著相似性，而我们的预测器需要利用这一点。最后，我们指出，由于人眼中央凹区域的分辨率非常高，因此迫切需要尽可能降低 MPD，这也正是我们采用此方法的动机。

A. 基准测试概述

表 I 展示了本研究使用的基准测试。Orange Room 是 VRMark 基准测试套件中的一项基准测试 [29]。其余基准测试均为知名 VR 应用，取自 Steam，与以往研究类似 [17], [18]。为更准确地评估工作负载大小对我们方案的影响，我们以不同分辨率渲染这些应用程序（见表中所列）。

缩写	Name	平台	引擎	API	分辨率
IM *IM2*	InMind VR [30] InMind VR 2	Steam	Unity	DX11	1920×1080 1290×800
SH *SH2*	Shooter 射击游戏 2	自定义	Unity	DX11	1920×1080 800×600
Sanc	Sanctuary VR [31]	Steam	Unity	DX11	1280×800
AS	Altspace VR [32]	Steam	Unity	DX11	1920×1080
OR	VRMark [29] Orange Room	Steam	Unity	DX11	1728×972

表 I：VR基准测试

B. 实验设置

为了展示 VR 应用中存在的帧间相似度（见第 II 节），我们在一台配置如表II所示的台式机上运行这些应用程序。我们的台式机配备了英特尔GPU。我们使用英特尔提供的性能分析工具图形帧分析器（GFA） [33], 对这些应用程序进行了检测。该工具可帮助我们分析帧流中的每一帧。在图形应用程序中，为了渲染一帧，应用程序会发出若干绘制调用，它们实际上是对图形处理器的函数调用。图形处理器会为每个绘制调用执行一组预定义指令。我们可以将其视为一个着色器函数。GFA 会以绘制调用为粒度，对应用程序的每一帧进行性能分析。借助该性能分析器，我们收集以下信息：渲染帧的内容、渲染该帧所需的绘制调用数量、所执行的着色器/内核

参数	类型/值
桌面配置	
CPU	10th 第10代英特尔®酷睿 i7-10700
GPU	英特尔® UHD Graphics 630
VR头显	Oculus Quest 2
模拟器配置	
GPU架构	Kepler
时钟频率	1 GHz
#纹理处理集群	8
#每个TPC的流式多处理器	2
#每个SM的流处理器	8
线程束大小	32
每个SM的共享内存大小	16 KB

表 II：基线系统详情

图形处理器（指令轨迹），以及每个绘制调用的实例数量，即软件线程。问题就在这里。VR 应用采用 DirectX 11 编写，因此我们必须使用基于英特尔核心的台式机，并配合 GFA 获取所有渲染帧的内容及其他信息。问题在于，我们的模拟器框架或其他常用框架 [34], [35] 是为英伟达图形处理器设计的，并且可以运行 CUDA 程序。因此，我们尽可能在模拟环境中对英特尔图形处理器架构进行建模（核心数量相同等），并编写了一个工具，将英特尔指令转换为 PTX指令。只有少数指令在语义方面存在问题；我们在其他英特尔手册中找到了更详尽的说明。就架构而言，英伟达开普勒架构最为接近（参见表II）。

我们的模拟器是基于 Java 的周期近似通用图形处理器模拟器，*GPUTejas* [36], 该模拟器已通过原生硬件进行了严格验证。为了获得 ATW 的延迟，我们用 CUDA 实现了 ATW算法，并在 GPUTejas 上对其进行模拟。模拟场景的详情见第 III-C2 节。

C. 时间扭曲

1）时间扭曲操作的延迟：我们为时间扭曲函数编写了一个 CUDA 程序 [17]。该程序的输入是需要进行扭曲的帧，以及渲染时刻和显示时刻的头戴式显示器（HMD）矩阵。这些 HMD 矩阵是 4×4 齐次矩阵，其中包含特定时刻的头部位置。然后，我们在模拟器上运行该内核，处理由 GFA性能分析器收集的帧和各种头部姿态。在这种情况下，我们不使用连续帧，而是选择图像复杂度较高的帧，以便让输入具有更大的差异；正如第 II 节所述，连续帧通常具有相似的性质。图 3 展示了将所有资源分配给时间扭曲内核时的执行时间。

我们从图中得出以下观察结果：

- 对于某个特定的基准测试，最大值和平均值大致相同，这意味着无论帧的复杂度和头部姿态如何，时间扭曲内核所需的时间都几乎相同。延迟平均值与最大值之间的最大差值

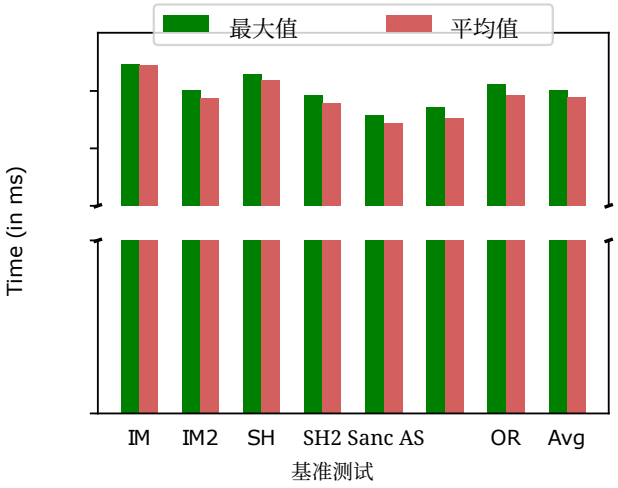

图 3：时间扭曲操作的延迟 on

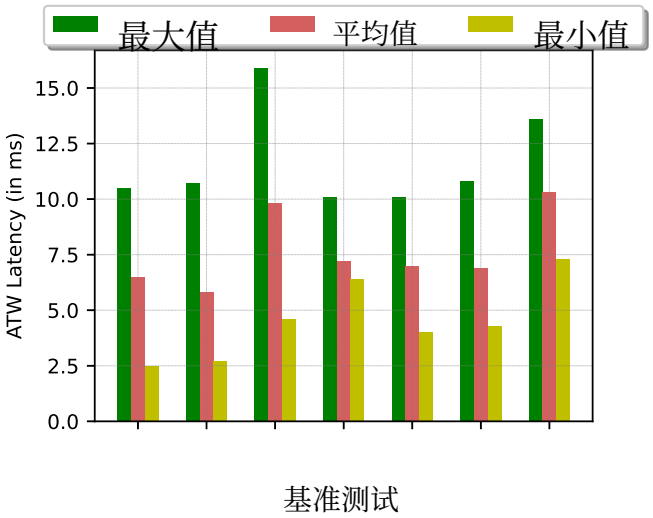

图 4：ATW 延迟的变化

在所有基准测试中均为 0.035 ms。

- 所有基准测试的平均延迟介于 2.32 毫秒和 2.55 毫秒之间；这表明延迟几乎不受帧属性的影响。

2) GPU 加速异步时间扭曲（*ATW*）的延迟：为了评估 ATW 的延迟，我们在 GPUTejas 中模拟了两种环境：所有核心均由一个使用 GPU 加速 ATW 的 VR 应用占用。该 VR 应用与另一个应用共享图形资源。我们不考虑同一 GPU 上运行两个以上应用的情况，因为这种情况并不常见。图 4 展示了所有基准测试的 ATW 延迟。某项基准测试内部以及不同基准测试之间的 ATW 延迟差异很大，这表明 ATW 延迟取决于渲染被抢占的帧以及图形资源的可用性。

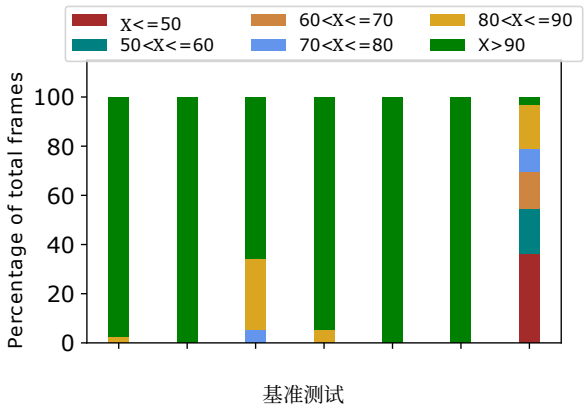

图 5：连续两帧之间的宏块相似百分比 (X)

我们从图中得出以下观察结果：

- 最大延迟接近 15 ms，远高于图 3 中所示的时间扭曲内核延迟（≈ 2.5 ms）。

- 对于某个基准测试，延迟的最小值、最大值和平均值之间差异很大，这意味着 ATW 延迟在不同帧之间波动很大，且高度不可预测、不确定。不同基准测试中的 ATW 延迟范围从 2 ms 到接近 16 ms。

D. 帧间相似度

为了展示帧与帧之间的相似度，我们从每个应用程序中提取了连续的 100 帧。

1）帧级相似度：如第II-B1节所述，我们使用宏块相似度度量来计算帧级相似度。图5展示了两个连续帧之间的宏块相似百分比。从图中可以明显看出，除一个基准测试（*OR*）外，所有基准测试中至少有70%的帧与其相邻帧的相似度至少为70%。即使对于*OR*，也有60%的帧之间相似度至少达到50%。

2) 特征级相似度：为了判断两帧是否相似，我们会收集一些帧特定的特征/属性，并比较相应的数值。这些属性如表 III 所示。这些特征并非都来自同一来源。对于亮度，我们使用来自 GFA 的渲染帧，并根据参考文献 [37] 中给出的公式计算数值；它是 RGB数值的函数。线程数和像素数直接从 GFA 收集。$GPUTime$、 $L2acc$和 $ATWLat$则取自仿真结果。图6展示了某个应用程序相邻帧之间的相似性。比较各子图中的趋势后，我们得出以下观察结果： 对于 80% 的帧，所有特征值相较于前一帧的变化幅度最多为 25%。 在各帧中表现出近乎恒定状态的特征是亮度、线程数和像素数。

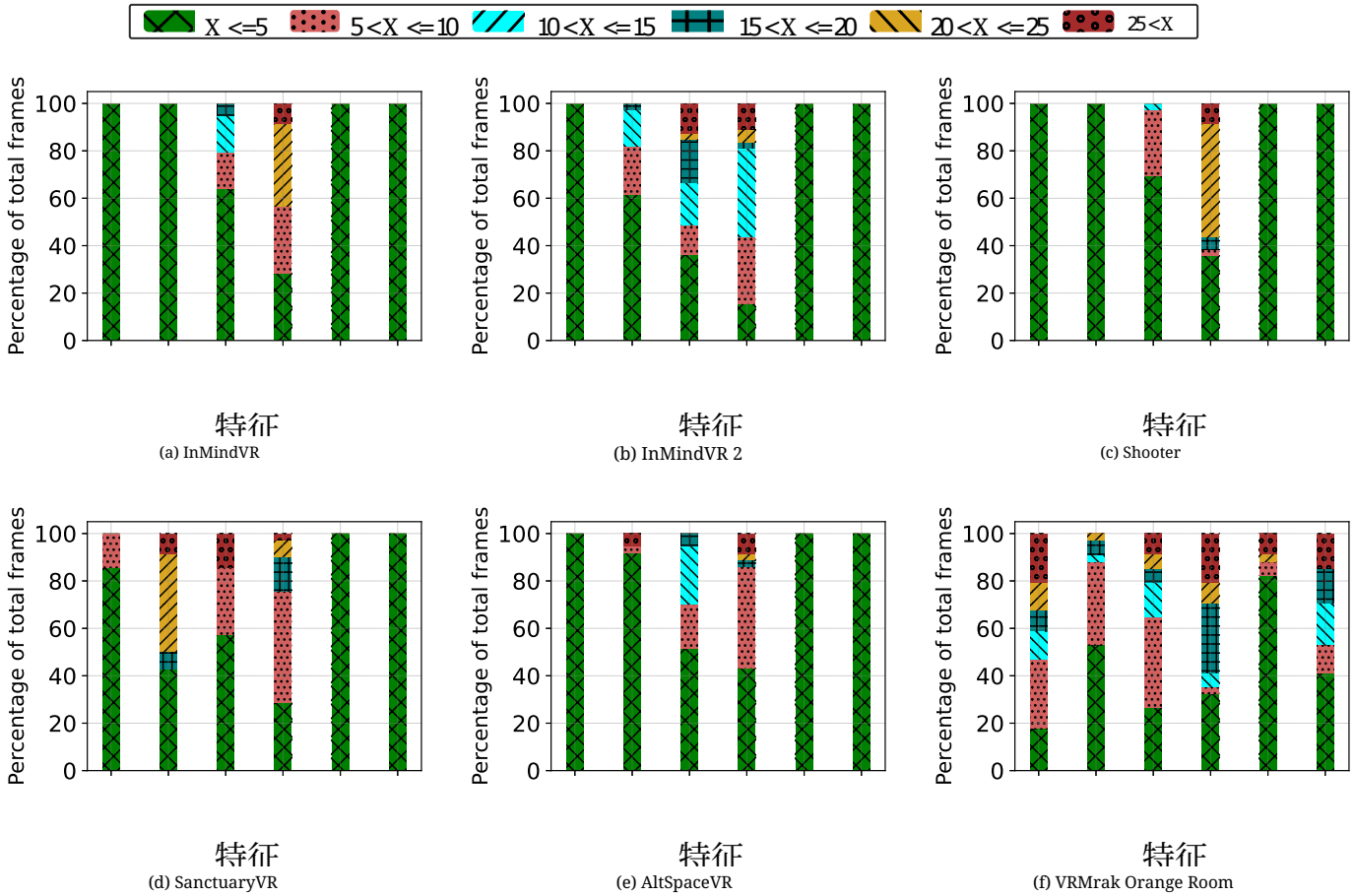

图 6：相对于上一渲染帧的值（记为 X）的百分比变化

参数	描述	采集来源
亮度	感知亮度 帧 [37]	GFA
GPU 时间	GPU 执行时间或 渲染时间（毫秒）	GPUTejas
L2 缓存访问	期间的 L2 缓存访问次数 渲染该帧	GPUTejas
ATW延迟	ATW 延迟	GPUTejas
线程数	生成的软件线程数 用于绘制帧	GFA
像素数	一帧中渲染的像素数 帧	GFA

表 III：帧特定属性

E. 人眼分辨率与低 *MPD*

视力为 6/6 的人，每线对的分辨率为 0.59 角分 [38]（分辨率： $\approx$ 0.3 角分）。现在假设头部以每秒 200 度的速度旋转 [39]（在一款节奏很快的 VR 游戏中）。因此，眼睛的最小分辨率对应于 0.02 毫秒。如果 MPD 超过这个值，我们就会感知到延迟，这可能导致不适和视觉方向感紊乱。遗憾的是，我们在 Oculus Quest 2 上玩一款哈利·波特风格、包含扫帚和宝珠的游戏时，确实体验到了这种情况。不过，这一计算只是理论上的，视觉系统可以进行补偿，以应对一些

度。因此，人们提出了 JND（最小可觉差）这一概念，用来表示实际极限（在实验室环境中测得）。对于常规的基于偏航的头部运动，当存在持续时间超过 60 ms 的其他刺激时，JND 最低可测得为 1 ms [40]；没有其他刺激时，约为 3.2 ms [41]。对于机械臂的触觉 VR 应用，JND 极限为 2 ms [42]。因此，我们不能错过帧截止期限（每 11 ms 显示一帧），也不能让 ATW 延迟超过几毫秒。至少部分视力良好的用户会察觉到这些差异。**这解释了为什么在先前的研究中 [17]，作者试图将 MPD 降低 4–5 ms，也解释了为什么游戏产业正转向采用 144 至 360 Hz 的显示器**。

见解：

- 在我们的基准测试中，ATW 延迟从 2.5 毫秒到 16 毫秒不等。
- VR 应用中存在较高的帧间相似度（$\approx$ 80% 的时间）。
- 人类视觉系统对截止期限错失以及超过 3-5 毫秒的 ATW 延迟非常敏感。

IV. 实现

如第 II 节所述，我们的目标是预测 VR 应用情况下的 ATW 延迟。我们将实现大致划分为以下多个子组件：我们首先创建一个模拟 GPU 加速的 ATW 算法的环境。然后，我们确定回归模型的输入特征。 我们运行模拟以收集样本数据，并将收集的数据划分为训练集和测试集。随后，我们训练多个预测器，并比较它们在测试数据上的性能。 选出性能最佳的预测器后，我们在运行时根据刷新截止期限错失率，以及时间扭曲完成时刻与显示刷新时刻的接近程度，检查其性能。

A. 仿真环境

如第 III-D 节所述，我们在实验中使用*GPUTejas* [36] 模拟器。表 II 展示了模拟器配置。我们实现了两种场景：模拟器上仅运行一个基准测试，并启用 GPU 加速的 ATW。由于显示器刷新率为 90 Hz，因此每隔 11.11 ms，我们都需要抢占帧的模拟并启动 ATW 操作。 主要的虚拟现实基准测试与另一个应用程序共享核心。为此，我们将核心平均分配给这两个应用程序；如果其中一个应用程序运行完毕，核心便会分配给仍在运行的应用程序。

B. 特征识别

设计任何预测模型时，重要的任务是定义一组输入特征，这些特征应具有模型能够学习并用于准确预测输出的模式。良好的特征选择对模型的成功至关重要。在我们的案例中，所选特征必须与帧特征和资源争用程度相关。这些特征还应与我们希望预测的 ATW 延迟相关。

在第 III-C2 节中，我们观察到，由于与当前帧渲染任务抢占相关的活动，也就是说，由于当前资源使用情况，ATW 延迟会随帧而变化。此外，如图 4 所示，在所有基准测试中，80% 的帧的连续两帧之间的 ATW 延迟变化幅度小于 20%。因此，我们可以认为，上一渲染帧的 ATW 延迟与当前帧的 ATW 延迟相关。它也反映了资源争用的影响。因此，我们选择 $PrevATWLat$ 作为输入特征之一。

同样，帧的图形处理器执行时间或渲染时间取决于资源可用性和帧复杂度（参见第 III-C2 节）。因此，我们将图形处理器执行时间作为一项输入特征。由于时间扭曲抢占的帧并未完整渲染，因此在进行预测时，我们无法获得该帧的执行时间。所以，我们采用上一渲染帧的执行时间。此方法有效，因为如图 4 所示，在约 90% 的情况下，连续两帧的执行时间变化不到 10%。

特征	描述	来源	帧
GPU 时间	帧渲染时间或	GPUTejas	prev
L2 缓存访问	L2 缓存访问次数	GPUTejas	prev
前一帧 *ATW* 延迟	ATW 延迟	GPUTejas	prev
线程数	# 软件线程	GPUTejas	curr
亮度	感知亮度帧[37]	GFA	prev
像素数	# 渲染像素	GFA	prev
顶点数	# 帧中的顶点数	GFA	curr
绘制调用次数	绘制调用次数	GFA	curr

表 IV：特征列表

如前所述，ATW 延迟取决于当前正在渲染的帧，因此我们的模型应能够学习帧的行为。因此，我们还选取了一些帧特定的度量作为输入特征。这些特征包括顶点数、渲染像素数、绘制调用数，以及这些绘制调用所使用的线程数。我们还在第 III 节中使用这些特征计算帧间相似度。为了捕捉帧渲染期间的内存访问模式，我们将 L2 缓存访问次数选作一项特征。同样，亮度也是帧的一项特征，因此我们也将其选作输入特征。选择亮度作为特征还有一个原因：要设置帧的亮度，需要使用像素着色器将每个像素与预定义颜色值进行混合。帧的亮度不同，着色器的复杂度也会有所变化，因此帧的抢占延迟也会改变。表 IV 列出了所有输入特征。请注意，其中有一列标明数据对应的是当前帧还是上一帧。许多特征（例如渲染时间或亮度）只能针对上一帧采集；当前线程尚未计算出这些特征。

C. 预测模型

如第 II-C 节所述，我们实现了四种回归模型。我们使用 scikit-learn [43],这一开源机器学习库来实现这些回归模型。整个数据集首先被划分为训练集和测试集。这两个集合互不重叠。数据集被随机划分为训练数据和测试数据。在训练阶段，模型无法访问测试数据。训练完成后，将测试数据输入模型以检验其准确性。结果表明，决策树回归器的预测误差最小。此外，如第 III 节所述，决策树能够提供可解释的模型，而我们的应用需要可解释性。我们检查了预测过程中所有特征的重要性，以及模型对某些特征的敏感性。最后，我们为学习得到的决策树结构编写了一个 CUDA 内核。

D. 空间复用

为降低 MPD，Smit 等人 [21], [22] 提出了一种多进程架构，使渲染过程与 ATW 进程在同一图形处理器硬件上并行运行。资源采用静态分区（空间复用）。这样就无需抢占渲染过程。

他们利用英伟达内置的空间复用特征 MPS，在单个图形处理器上实现了这一架构：将图形处理器的计算核心划分为渲染核心和 ATW 核心。

我们想要批评这种方法。现代图形处理器已经被渲染工作负载充分利用；因此，为 ATW 专门分配核心会产生性能开销，并最终减少可供 VR 应用使用的渲染带宽 [17]。此外，ATW 核心大多处于闲置状态；这会造成资源浪费。目前，大多数基于 GPU 的应用程序和运行时都不够灵活，无法根据可用核心数量动态调整自身规模。

尽管如此，为了展示空间复用对 ATW 延迟的影响，我们在架构中对其进行了模拟。

我们考虑三种场景：

- 总核心数的八分之一，即 12.5%，专用于 r ATW（*SM1*）。 四分之一的核心分配给 ATW
- （*SM2*）。 一半的核心分配给 ATW（*SM3*）。

最后，我们将这些架构的 ATW 延迟与 GPU 加速的 ATW 进行比较。

V. 结果与分析

我们提出的方案由预测器和分析代码组成，并使用 CUDA 库以软件方式实现。使用模拟器的原因是我们没有可供进行此类侵入性修改的虚拟现实平台（与先前的工作类似 [18]）。

A. 各种预测器的性能

如上一节所述，我们实现了四种预测模型，分别是随机森林、线性回归、梯度提升和决策树。所有模型使用同一组特征。我们根据平均绝对误差（MAE）比较这些模型的性能。MAE 的计算公式如公式 1 所示。图 7 根据预测的平均绝对误差，对这四种模型进行了比较。

$$MAE = \frac{\sum_{i=1}^{N} |Y_i - \hat{Y}_i|}{N} \quad (1)$$

结果如图 7 所示。我们观察到，基于决策树的回归器在所有模型中表现最佳。对于所有基准测试，其平均误差为 0.77 ms。除一个基准测试（即 或）外，决策树的预测误差在所有基准测试中都小于 1 ms。这是因为它的帧间相似度得分较低。

以随机森林预测器为例，其平均误差为 1.4 毫秒，比决策树的预测误差高 81%。尽管我们随机生成了 100 棵决策树，仍然

无法展现更高的准确率。这是由于帧间相似度较高，而我们的特征也反映了这一点。随机性并未带来任何收益。在另一方面，每棵决策树所使用的训练样本更少，这从根本上降低了所有组成树的准确率。

以线性回归为例，它在七个基准测试中有六个表现最差。线性回归器的平均误差为 1.65 ms，比决策树高 107%。由此可推断，这种关系远非线性。因此，这类简单模型无法奏效。

与随机森林类似，复杂的梯度提升模型的预测误差比决策树高 88%。原因也类似。

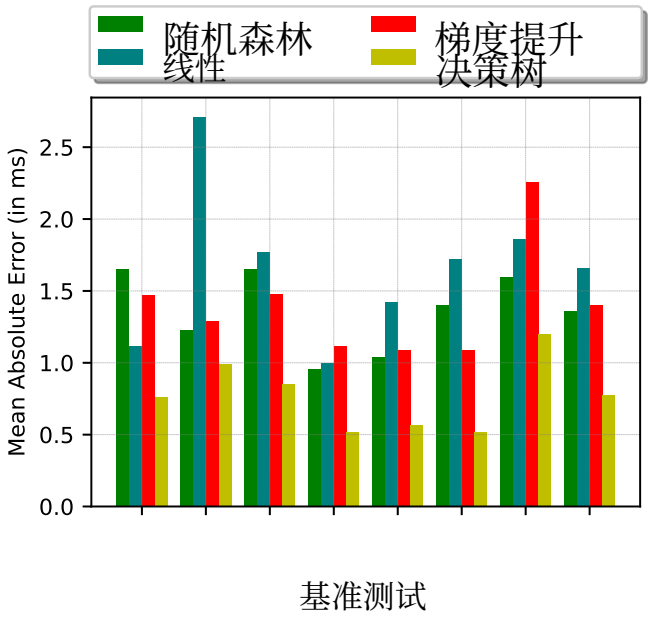

图 7：不同预测模型的平均绝对误差

B. 预测器对 *MPD* 的影响

为了了解我们提出的方法对 MPD 的影响，我们记录了使用和不使用预测器时的截止期限错失率。我们还考察了时间扭曲操作的完成时间与显示刷新时刻的接近程度。如第四节所述，显示刷新率为 90 Hz，这意味着显示器每隔 11.11 ms 刷新一次。我们分析以下三种情况下的刷新截止期限错失率：

- 图形处理器加速的 ATW，即基线架构。
- ATW 会在帧渲染完成后立即调用，或者在屏幕刷新操作开始前 2.55 毫秒调用，以先发生者为准。请注意，2.55 毫秒是时间扭曲内核的平均延迟。（参见图 3）（我们将其称为即时执行方法）
- ATW 根据预测的 ATW 延迟进行调用，这是我们提出的方法。我们称之为*PredATW*。

1) 刷新截止期限错失率： 如第 I 节所述，如果 ATW 未能在显示器刷新前完成，MPD 就会增加，用户看到的世界也会出现异常或模糊的视图。图 8 展示了所述三种情况下的刷新截止期限错失率

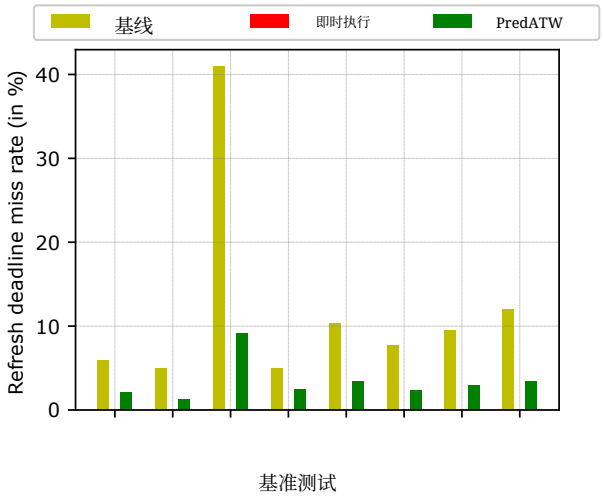

图 8：预测器对刷新期限错失率的影响。使用 即时执行 时，我们从未错失任何截止期限。因此，错失率始终为 0%。

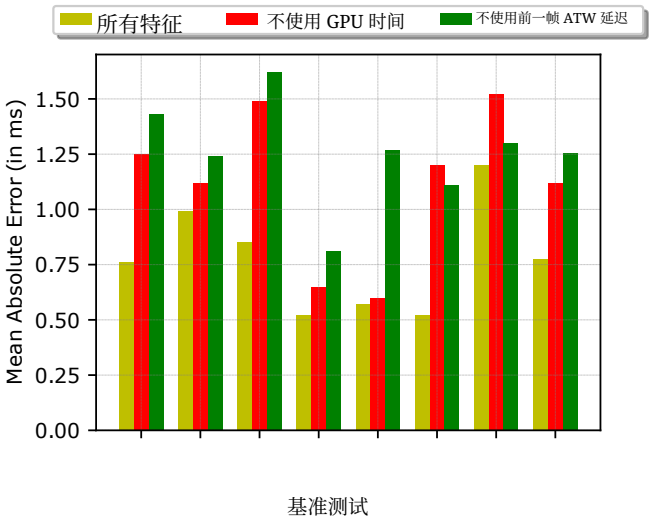

图 9：*GPU* 时间和前一帧 *ATW* 延迟对预测误差的影响

如上所述。图中显示，对于所有基准测试，我们的预测器都降低了截止期限错失率。对于 GPU 加速的 ATW（基线）在所有基准测试中，截止期限错失率从 5% 上升至近 41 %。平均错失率约为 12%，相当显著。

对于 ATW 在帧渲染完成后立即调用的 即时执行 情况，不会错失任何截止期限。然而，正如第 I 节所述，这会导致较大的 MPD，原因显而易见：ATW 完成的时间与显示刷新时刻之间存在较大的间隔。结果表明，*PredATW*（我们的方案）大幅减少了截止期限错失次数：从 12% 降至 3.4%。在这种情况下，除一个基准测试 *Shooter* 外，所有基准测试的错失率都在 2% 到 4% 之间。不过，如果比较相对降幅，改善相当显著。基线下的错失率约为 40%，我们将其降至 9.1 %。*PredATW* 的错失率总体降低了 73.1%。

2）*ATW* 完成 → 显示刷新时刻：现在来看 ATW 完成到显示刷新时刻之间的平均时长：越短越好（但需避免错失截止期限）。

基准测试	基线	即时执行	PredATW
IM	4.10	4.51	2.39
IM2	5.05	7.79	2.84
SH	1.45	2.93	2.53
SH2	2.53	5.53	2.51
Sanc	3.40	7.49	1.55
AS	3.45	8.12	3.88
OR	2.58	4.68	3.15
Mean	3.22	5.86	2.69

我们观察到，与即时执行相比，*PredATW* 将这一时长缩短了 55%。它也比基线低 17%。在某些情况下，后者表现更好；但这通常发生在不安全区域，此时会错失截止期限。需要注意的重要一点是，对于 $PredATW$，这一时长平均而言始终低于 3.88 毫秒，因此很可能不会被感知到（参见第 III-E 节）。对于另外两种方案，我们在结果中看到许多较高的数值（7–8 毫秒），这会导致错失截止期限，而且即使没有错失，也会被感知到。

C. 敏感性分析

如第 III-B 节所述，我们确定了两个特征*GPU* 时间和前一帧 *ATW* 延迟，它们会对 ATW 延迟产生争用效应。为展示这些特征如何影响预测误差，我们使用包含和不包含这些特征的数据分别训练并测试模型。

1) GPU 时间特征的影响：图 9 展示了 GPU 时间特征对预测误差的影响（以平均绝对误差衡量）。从图中可以看出，对于所有基准测试，引入 GPU 时间特征后，预测误差大约降低了 33%。不同基准测试的趋势大致相同。

2) 前一帧 *ATW* 延迟特征的影响：图 9 中的第三个柱形图展示了前一帧 ATW 延迟特征对预测误差的影响。与 GPU 时间类似，加入前一帧 ATW 延迟也会改善预测误差。我们发现，从输入特征列表中移除前一帧 ATW 延迟后，预测误差会增加 62%。

3）各特征的重要性：为了展示预测过程中各特征的重要性，我们统计了 75 个具有代表性的测试样本的决策路径中每个特征出现的频率，并用图 10 中的雷达图绘制出来。图中的雷达图显示，三个特征——亮度 前一帧 *ATW* 延迟 和 绘制调用次数 —— 在决策路径中的使用次数最多。

这里有一些有趣的趋势。在前 50% 的基准测试（0-37）中，只使用了 2-3 个特征：亮度、绘制调用次数和顶点数。而且，每个特征只使用 1-3 次。这基本上意味着 ATW 很容易预测，因此可以根据这些特征快速做出决策。对于

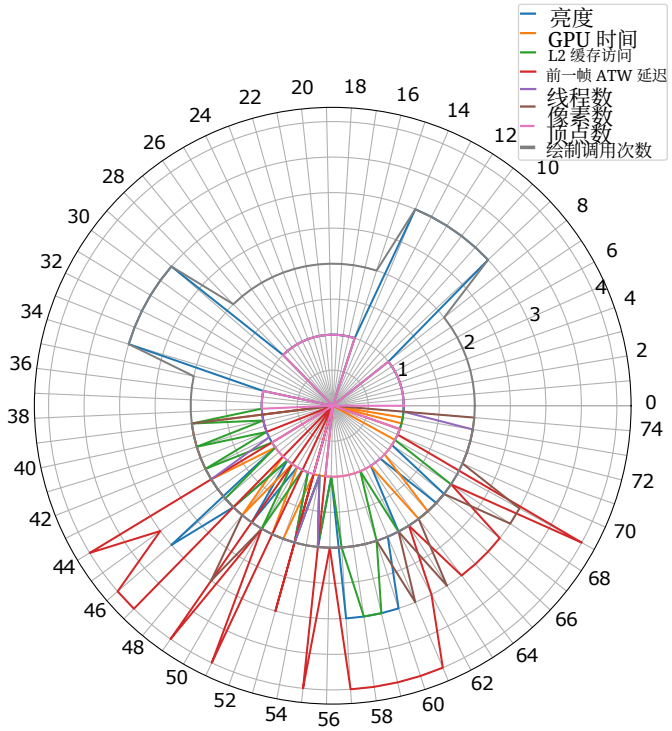

图 10：不同测试点的决策路径中各特征出现的频率

其余样本（≈ 50%），几乎使用了所有特征。在这里，GPU 时间发挥着重要作用，因为它靠近根节点，并用于做出最初的几项决策。前一帧 ATW 延迟则处于另一个极端——它被使用了 2-4 次。这基本上意味着预测值对这个参数非常敏感。其余特征的使用频率适中，因此敏感性也适中。

D. GPU 加速的 *ATW* 与空间复用

我们在第 IV-D 节中提到了三种空间复用方式。图 11 展示了这三种情况及基线配置下的 ATW 延迟。从图中可以看出，基线系统的 ATW 延迟在 2.5 ms 到接近 16 ms 之间变化，这一点在第 III-C2 节中也有提及。由于采用了专用资源，三种空间复用情况（*SM1*、*SM2* 和 *SM3*）下的 ATW 延迟变化较小。

对于 *SM1* （为 ATW 分配 12.5% 的核心），ATW 延迟为 18.71 ms，远高于 11.1 ms 的帧时间。改用 *SM2* （预留 25%）后，延迟降至 9.39 ms，但仍然很高。唯一勉强可以接受的数据是*SM3*，其 ATW 延迟为 4 ms。

E. 预测器的延迟

如前所述，我们在 CUDA 中实现了学习得到的决策树结构，并在模拟器上运行，以评估预测器的延迟。图 12 展示了我们的基准测试中预测过程的最大延迟。从图中可以

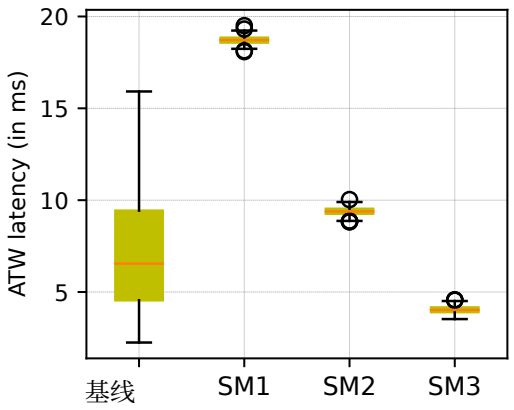

图 11：基线系统以及 *SM1* （为 ATW 分配 12.5% 的核心）、*SM2* （为 ATW 分配 25% 的核心）和 *SM3* （为 ATW 分配 50% 的核心）的 ATW 延迟

清楚地看出，预测器的延迟仅为几分之一微秒。所有基准测试的延迟几乎保持不变。延迟的最大值为 0.46 微秒。因此，我们发现预测器不会给 ATW 增加显著的延迟。

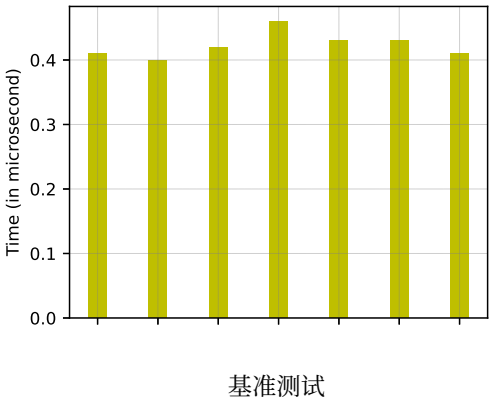

图 12：预测器的最大延迟

VI. 相关工作

由于虚拟现实系统中的 MPD 会对用户体验产生不利影响，过去几年间，人们探索了多种方法来降低交互式虚拟现实系统的这一延迟。近期的研究主要聚焦于 预测跟踪 [19], [20]; 运动预测 +可编程显示层 (PDL) [21], [22]; 将时间扭曲操作卸载到独立硬件单元以加速该操作 [17], [18]; 以及 优化渲染过程 [23], [24]。我们在表 V 中简要比较了相关工作。

近期研究 [19], [20] 聚焦于基于机器学习的方法，以预测观看方向和位置，从而满足虚拟现实系统的超低延迟要求。Xueshi等人 [20] 提出在边缘设备上进行预测性预渲染

Work	基于机器学习预测	预测数据	异步时间扭曲	外包时间扭曲	优化时间扭曲	优化渲染
预测性跟踪 [19], [20]	✓	头部和身体运动	×	-	-	×
FlashBack[44]	×	All 可能观看位置	×	-	-	×
运动预测 + PDL [21], [22]	✓	头部和身体运动	✓	✓	✓	×
PIM-VR [17]	×	×	✓	✓	✓	✓
Q-VR [18]	×	×	✓	✓	×	✓
[23], [24]中的方案	×	×	×	-	-	✓
PredATW	✓	**ATW 延迟**	✓	×	×	×

表 V：相关工作的比较

。Kunda等人 [19]研究了几种常见的预测算法——航位推算和卡尔曼滤波——根据过去的行为预测未来的观看方向和位置。FlashBack [44] 预测所有可能的位置以及方向姿态，并对与这些预测相对应的所有视图进行预渲染。但是，预先计算所有可能的视图并将其存储在内存中，会大幅增加计算和存储开销。

在这项工作中，我们使用从跟踪器设备获取的最新姿态来扭曲帧，而不是预测头部运动，因为头部运动预测可能存在相当大的误差。我们唯一预测的是 ATW 延迟，以便及时调用时间扭曲内核，使其恰好在显示刷新时刻之前完成。

有些研究 [21], [22] 采用双 GPU 架构来避免抢占。在这种架构中，第一块 GPU（客户端）负责渲染应用程序帧，并生成运动信息和像素属性：颜色、深度和强度。数据通过 PCIe 总线发送到共享系统内存，第二块 GPU（服务器）利用这些数据，以新帧更新显示设备。第二块 GPU（服务器 ）运行自定义程序，获取最新的应用程序帧和保存在共享内存中的数据，并根据这些数据和当前视点对帧进行变形。然而，这些解决方案[21], [22] 的硬件开销和功耗开销都很高，因为它们会额外进行内存访问，并向 PCI 总线写入数据。

显著降低 MPD 的一种方法是加快时间扭曲操作。Waveren 等人 [15] 研究了用于实现低延迟时间扭曲算法的各种硬件平台。Xie 等人 [17]提出了 PIM-VR，这是一种基于存内处理的 ATW 设计，可在不中断主机 GPU 上渲染任务的情况下，在 3D 堆叠内存中异步执行 ATW。他们还提出了一种冗余削减机制，以进一步简化并加速 ATW 操作。一些研究 [23], [24] 通过优化渲染过程本身来降低 MPD。Toth 等人 [45]表明，在不影响视觉质量的前提下，通过渲染多个经过优化的子投影，渲染像素数量最多可减少 36%。注视点渐变 是指图像质量会随着远离中央凹（视网膜中心）的距离增加而下降的现象 [46]。注视点渲染算法利用这一现象来提升性能 [23], [24]。这些算法会降低周边区域的渲染复杂度和图像质量，同时保持中央凹区域较高的图像质量。Q-VR [18] 将注视点渲染与时间扭曲操作相结合，以进一步提升渲染性能。

VII. 结论

鉴于该领域已有的大量研究，我们采用了一种截然不同的方法。我们不专注于新的渲染算法、不预测头部运动，也不使用独立的 GPU 硬件，而是提出了一种成本相对较低、只需软件支持的解决方案。我们表明，即使存在噪声，且有另一个 GPU 应用程序正在运行时，也能相当准确地预测 ATW 延迟。预测误差仅为 0.77 ms，截止期限错失率也极低（≈ 3%）。因此，我们认为，无需采用需要额外硬件支持的复杂解决方案。我们推测，ATW 延迟预测算法还能进一步改进；随着数据点增多，或采用集成学习，平均绝对误差可降至 0.5 ms 以下。这是未来工作的一个方向。

参考文献

[1] L. Rondeau, 《虚拟现实与另类事实：数字社群的主观现实》, 2020年, 第 41–49 页。[2] J. D. Bolter、M. Engberg 和 B. MacIntyre, 《现实媒体史》第 2 章, 2021年, 第 23–41 页。[3] J. Gupta, 《虚拟现实如何改变游戏行业》, https://yourstory.com/mystory/how-virtual-reality-is-transforming-the-gaming-ind/amp, 4 2021, [在线；访问日期：2022-10-12]。[4] S. THOMPSON, 《虚拟现实应用：23 个使用虚拟现实的行业》, https://virtualspeech.com/blog/vr-applications, 3 2022, [在线；访问日期：2022-10-14]。[5] 《医疗保健中的虚拟现实》, https://visualise.com/virtual-reality/virtual-reality-healthcare, [在线；访问日期：2022-10-12]。[6] D. L. Thomas, 《虚拟现实在医学中的应用》, https://www.news-medical.net/health/Applications-of-Virtual-Reality-in-Medicine.aspx#:~:text=虚拟现实10 2022, [在线；访问日期：2022-10-14]。[7] 《虚拟现实市场规模与份额报告, 2022-2030年》, https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market, [在线；访问日期：2022-10-12]。[8] D. Wagner, 《移动增强现实和虚拟现实中的运动到光子延迟》, https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926, 8 2018, [在线；访问日期：2022-10-14]。[9] 《运动到光子延迟》, https://xinreality.com/wiki/Motion-to-photon latency, 4 2015, [在线；访问日期：2022-10-14]。

–

[10] J. Coles, 《虚拟现实中的晕动症由什么引起？如何避免？》 https://www.space.com/motion-sickness-in-vr， 9 2021, [在线；访问日期：2022-10-14]。[11] C. Whelan, 《什么会引起虚拟现实（VR）晕动症？》 https://www.healthline.com/health/vr-motion-sickness， 6 2021, [在线；访问日期：2022-10-14]。[12] S. Thompson, 《虚拟现实中的晕动症：为何发生以及如何尽量减轻》, https://virtualspeech.com/blog/motion-sickness-vr， 4 2020, [2022 10 14] 在线；访问日期：--。

[13] 《Oculus Quest 2》, https://www.meta.com/quest/products/quest-2/。[14] 《HTC Vive》, https://www.vive.com/us/, [在线；访问日期：2022-10-08]。[15] J. M. P. van Waveren, 《面向消费级硬件的虚拟现实异步时间扭曲》, 载于第 22 届 ACM虚拟现实软件与技术会议论文集，VRST 2016，德国慕尼黑，2016年11月2日至4日，D. Kranzlm¨uller 和 G. Klinker 编。ACM，2016年，第 37–46 页。 [在线]。可访问：https://doi.org/ 10 1145 2993369 2993375 。/。S.，《时间扭曲详解》，[16] https://medium.com/insights-on-virtual-reality/time-warp-explained-febea194ca87，4 2016, [在线；访问日期：2022-10-09]。[17] C. Xie、X. Zhang、A. Li、X. Fu 和 S. Song, 《PIM-VR：利用定制内存立方体消除高交互性虚拟现实世界中的运动异常》，载于第 25 届 IEEE高性能计算机体系结构国际研讨会，HPCA 2019，美国华盛顿哥伦比亚特区，2019年2月16日至20日。IEEE，2019年，第 609–622 页。 [在线]。可访问：https://doi.org/ 10 1109 2019 00013 。/HPCA。。C. [18] Xie、X. Li、Y. Hu、H. Peng、M. B. Taylor 和 S. L. Song, 《Q-VR：面向未来移动协作式虚拟现实的系统级设计》，载于 ASPLOS '21：第 26 届 ACM编程语言与操作系统架构支持国际会议，线上活动，美国，2021年4月19日至23日，T. Sherwood、E. D. Berger 和 C. Kozyrakis 编。ACM，2021年，第 587–599 页。 [在线]。可访问：https://doi.org/ 10 1145 3445814 3446715 。/。R. K. Kundu、A. Rahman 和 S. Paul, [19] 《虚拟现实晕动症中的传感器系统延迟研究》, 《传感器与执行器网络》，第 10 卷，第 3 期，第 53, 2021 [ ] 10 3390 10030053 页。在线。[20] 可访问：https://doi.org/。/jsan。X. Hou、J. Zhang、M. Budagavi 和 S. Dey, 《头部与身体运动预测：实现低延迟移动虚拟现实体验》，载于 2019年IEEE全球通信会议，GLOBECOM 2019，美国夏威夷州威可洛亚，2019年12月9日至13日。IEEE，2019年，第 1–7 页。[在线]。可访问：https://doi.org/10.1109/GLOBECOM38437.2019.9014097[21] F. A. Smit、R. van Liere、S. Beck 和 B. Fr¨ohlich, 《虚拟现实图像扭曲架构：低延迟与图像质量》，载于 2009年IEEE虚拟现实会议（VR 2009），2009年3月14日至18日，美国路易斯安那州拉斐特，会议论文集。IEEE计算机学会，2009年，第 27 34 [ ] 10 1109 2009 4810995 – 页。在线。可访问：[22] https://doi.org/。/VR。。F. A. Smit、R. van Liere 和 B. Froehlich, 《面向虚拟现实系统架构的可编程显示层》，《IEEE可视化与计算机图形学汇刊》，第 16 卷，第 1 期，第 28–42 页，2010年。 [在线]。可访问：https://doi.org/ 10 1109 2009 75 。/TVCG。。A. Patney、J. Kim、M. Salvi、A. [23] Kaplanyan、C. Wyman、N. Benty、A. E. Lefohn 和 D. Luebke, 《基于感知的注视点虚拟现实》，载于计算机图形学与交互技术特别兴趣小组会议，SIGGRAPH '16，美国加利福尼亚州阿纳海姆，2016年7月24日至28日，新兴技术。ACM，2016年，第 17:1–17:2 页。 [ ] 10 1145 2929464 2929472 在线。可访问：[24] https://doi.org/。/。A. Patney、M. Salvi、J. Kim、A. Kaplanyan、C. Wyman、N. Benty、D. Luebke 和 A. E. Lefohn, 《迈向眼动追踪虚拟现实的注视点渲染》，《ACM图形学汇刊》，第 35 卷，第 6 期，第 179 1 179 12, 2016 [ ] 10 1145 2980 :– : 页。在线。可访问：179 2980246 https://doi.org/。/。[25] S. Zhao、H. Zhang、S. Bhuyan、C. S. Mishra、Z. Ying、M. T. Kandemir、A. Sivasubramaniam 和 C. R. Das, 《D´ej`a view：用于节能型360度虚拟现实视频流传输的时空计算复用》，载于第 47 届 ACM/IEEE计算机体系结构年度国际研讨会，ISCA 2020，西班牙瓦伦西亚，2020年5月30日至6月3日。IEEE，2020年，第 241–253 页。 [在线]。可访问：https://doi.org/ 10 1109 45697 2020 00030 。/ISCA。。R. Balsree、A. Thawani、[26] S. Gopalan 和 V. Sridhar, 《基于帧间相似度的视频转码》，载于第七届 IEEE多媒体国际研讨会（ISM 2005），2005年12月12日至14日，美国加利福尼亚州尔湾。IEEE计算机学会，2005年，第 618–623 页。 [在线]。可访问：https://doi.org/10.1109/ISM.2005.71

[27] B. Furht 和 P. Saksobhavivat, 《利用压缩数据进行快速基于内容的多媒体检索技术》，载于《多媒体存储与归档系统 III》，C.-C. J. Kuo、S.-F. Chang 和 S. Panchanathan 编，第 3527 卷，国际光学与光子学学会。SPIE，1998年，第 561 571 [ ] 10 1117 12 325851 – 页。在线。[28] [网址：https://doi.org/ 。/ 。“宏块”，https://en.wikipedia.org/wiki/Macroblock，在线；访问日期：2022-10-08]。[29] “VRMark”，https://store.steampowered.com/app/464170/VRMark/，[在线；访问日期：2022-10-08]。[30] “InMind VR”，https://store.steampowered.com/app/343740/InMind VR/，[在线；访问日期：2022-10-20]。[31] “Sanctuary VR”， [在线；访问日期：2022-10-20]。 [在线]。网址：https://store.steampowered.com/app/565730/Sanctuary VR 另包含非虚拟现实版本/[32] “Altspace VR”，https://store.steampowered.com/app/407060/AltspaceVR/，[在线；访问日期：2022-10-20]。[33] “图形帧分析器”，https://www.intel.com/content/www/us/en/developer/tools/graphics-performance-analyzers/graphics-frame-analyzer.html， [在线；访问日期：2022-10-08]。[34] A. Bakhoda、G. L. Yuan、W. W. Fung、H. Wong 和 T. M. Aamodt, 《使用详细 GPU 模拟器分析 CUDA 工作负载》，载于《2009年 IEEE 系统与软件性能分析国际研讨会》。IEEE，2009年，第 163–174 页。[35] J. Power、J. Hestness、M. S. Orr、M. D. Hill 和 D. A. Wood, 《gem5-gpu：一种异构 CPU-GPU 模拟器》，《IEEE 计算机体系结构快报》，第 14 卷，第 1 期，第 34–36 页，2014年。[36] G. Malhotra、S. Goel 和 S. R. Sarangi, 《GPUTejas：一种 GPU 架构并行模拟器》，载于《2014年第21届高性能计算国际会议（HiPC）》，2014年，第 1–10 页。[37] “Python——有哪些方法可以使用 Python 分析图像亮度”，https://itecnote.com/tecnote/python-what-are-some-methods-to-analyze-image-brightness-using-python/， [在线；访问日期：2022-10-14]。[38] “关于人眼分辨率及其他细节的说明”，https://clarkvision.com/imagedetail/eye-resolution.html， 12 2018, [在线；访问日期：2022-10-14]。[39] S. Thompson, 《虚拟现实与人眼分辨率》，https://medium.com/@FreneticPony/virtual-reality-and-the-resolution-of-the-human-eye-5e601b0ef030, 12 2016, [在线；访问日期：2022-10-14]。[40] J. J. Jerald, 《头戴式显示器的场景运动与延迟感知阈值》，博士论文，北卡罗来纳大学教堂山分校，2009年。[41] J. Jerald 和 M. Whitton, 《头戴式显示器场景运动阈值与延迟阈值的关系》，载于《2009年 IEEE 虚拟现实会议》。IEEE，2009年，第 211–218 页。[42] M. Alja'Afreh, 《触觉互联网时代数字孪生系统的体验质量模型》，博士论文，渥太华大学，2021年。[43] “scikit-learn”，https://scikit-learn.org/stable/， [在线；访问日期：2022-10-20]。[44] K. Boos、D. Chu 和 E. Cuervo, 《FlashBack：通过渲染记忆化在移动设备上实现沉浸式虚拟现实》，《GetMobile 移动计算与通信》，第 20 卷，第 4 期，第 23–27 页，2016年。 [在线]。网址：https://doi.org/ 10 1145 3081016 3081026 。/ 。R. Toth、J. Nilsson 和 T. [45] Akenine-Moller, 《用于渲染虚拟现实的投影方法比较》，载于《高性能图形学会议论文集》，HPG 2016，都柏林，爱尔兰，2016年6月20日至22日，D. Luebke 和 S. Molnar 编。欧洲图形学协会，2016年，第 163 171 [ ] 10 2312 20161202 – 页。在线。[46] 网址：https://doi.org/ 。/hpg。B. K. Guenter、M. Finch、S. M. Drucker、D. S. Tan 和 J. M. Snyder, 《注视点渲染三维图形》，《ACM 图形学汇刊》，第 31 卷，第 6 期，第 164 1 164 10, 2012 [ ] 10 1145 23 : –: 页。在线。网址：https://doi.org/ 。/。 66145 2366183