arXiv:2611.32732v1 [cs.NI] 24 Nov 2026

FugaStream：面向充分利用多视图的流水线实时体积流传输

EdwardJusuk Ki
mjusukkim@snu.ac.kr 首尔国立大学 首尔，大韩民国

Huijeong C
hoehuijeongchoe@snu.ac.kr 首尔国立大学 首尔，大韩民国

Minjae Lee
minjaelee@snu.ac.kr 首尔国立大学 首尔，大韩民国

Chanjeong Park
cjpark99@snu.ac.kr 首尔国立大学 首尔，大韩民国

Kyunghan Lee
kyunghanlee@snu.ac.kr 首尔国立大学 大韩民国首尔

摘要

实时体积视频流已成为下一代沉浸式应用的基石，使用户能够体验六自由度（6-DoF）。为支持交互性，实时体积视频流必须在满足严格延迟预算的同时，将多视图RGB-D帧重建为统一三维表示，而现有系统不得不在重建质量与延迟之间进行权衡。本文提出 FugaStream，这是首个采用流水线架构的实时体积视频流系统，通过提前提交基础表示，然后逐步合并新颖的延迟视图，缓解了这一权衡。实现此类功能带来了两项根本性挑战：（i）帧可用性的不确定性；（ii）如何以轻量级方式处理延迟视图更新。针对这些挑战，我们提出了两种关键算法：（i）自适应基准提交控制，根据每个缺失视图预期带来的重建增益与不断增加的延迟成本，确定基础提交时间；（ii）基于基准的后到视图补全，仅传输延迟视图相较于早期已提交基础视图中的新颖区域，从而支持客户端的轻量级融合。我们在广泛使用的体积数据集和真实世界测试平台上验证了 FugaStream 的有效性，结果表明，在相同的 150 ms 预算下，与基线方法相比，FugaStream 将结构相似性指数（SSIM）提高了 10.9 %，将交并比（IoU）提高了 13.7%。

关键词

实时体积视频流、体积融合

1 引言

随着体积计算平台的商业化，实时体积流媒体已成为新一代交互式应用的关键推动力，包括多方临场通信、沉浸式课堂讲座和协作式混合现实游戏。在这类服务中，用户可以自由移动并改变

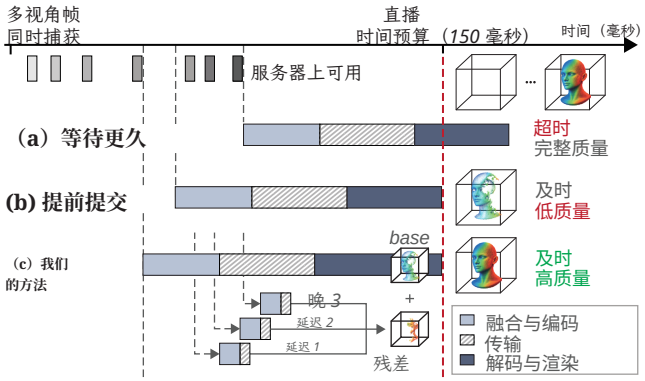


图1：实时体积视频流中的延迟-质量权衡：(a) 等待所有视图可保留质量，但会超出延迟预算；(b) 提前提交融合可满足延迟预算，但要以牺牲质量为代价；(c) 流水线流传输可尽可能利用更多视图，在满足延迟预算的同时不牺牲质量。

他们的视点，获得六自由度（6-DoF），仿佛与远程参与者身处同一地点。

为了实现这种沉浸式体验，实时体积视频流处理管线的核心在于汇集多视图帧，并在服务器端将其重建为统一的三维表示（例如，点云、体素网格或网格）[3, 11, 13, 22, 26, 27]。在这种多视图设置中，多个 RGB-D 相机从不同角度观测同一物体，其视野在物体表面上部分重叠。这些重叠观测使各视图能够相互补足、抑制噪声，而每个不同的视点则能捕获其他视图中被遮挡的区域 [4, 13]。因此，为了实现高保真实时体积视频流，服务器必须将所有视图纳入三维重建。

然而，重建前必须汇集所有视图这一要求，与沉浸式应用严格的延迟预算相冲突。为保持交互性，整个采集到显示流水线必须满足约 150 ms 的时延约束 [17]。在这一预算内，每一帧都必须经过端到端流水线中的多个处理和通信阶段，因此留给服务器端视图汇集和三维重建的时间十分有限。更棘手的是，不同摄像机同步捕获的帧，准备好进行重建的时间却并不同步，其就绪时间往往相差数十毫秒 [7, 16]。这种差异源于 RGB-D 编码 [20, 36], 上行链路争用 [11, 16], 服务器端预处理（如背景和噪声滤波） [15], 以及其他开销。就绪度差异使服务器陷入两难：等待所有视图可能会超出延迟预算（图1(a)），而过早开始重建则可能导致重建出的三维模型严重不完整（图1(b)）。

本文的目标是通过在每个视图准备就绪之前开始重建，同时继续利用延迟视图，在这两个极端之间实现更优的运行点。如图1(c)所示，一旦可用视图足够，服务器便提交重建。生成的重建结果构成基础表示，并在无需等待延迟视图的情况下传输给客户端。在基础提交之后准备就绪的视图将作为轻量级更新被纳入，以细化基础表示。

实现这一系统面临两个关键挑战。首先，确定可用视图何时足以进行基础提交，这本质上极具挑战性。每个视图的重建就绪时间及其身份都无法预先得知。等待更长时间可能只能带来有限的质量提升，而过早提交则可能舍弃那些本可显著改善重建的视图。其次，识别每个延迟视图中能够在基础表示之外提供新信息的区域并非易事。延迟视图可能观测到许多已在基础表示中呈现的表面区域，同时也可能揭示此前被遮挡、能够改善重建的区域。因此，将整个延迟视图视为新信息，会为基础表示中已有的区域引入冗余传输和重建开销。

为此，我们提出了 FugaStream，据我们所知，这是首个流水线式实时体积流系统，能够缓解重建时序与视图利用之间僵化的依赖关系。FugaStream 包含以下两个关键组件：

自适应基准提交控制。 每当某个视图变得可用时，FugaStream 都会评估是等待额外视图，还是使用当前视图集提交基础重建。它会根据近期的就绪历史，估计每个缺失视图在近期变得可用的概率。通过权衡预期重建增益与迅速增加的等待成本，FugaStream 仅在预期增益足以证明延迟合理时继续等待，否则便将当前可用的视图作为基础重建提交。

基于基准的后到视图补全。为降低客户端更新的高昂成本，该模块利用中间融合状态实现轻量级增量式细化。FugaStream 在传输基础表示的同时，也传输其底层状态，包括表面距离和观测置信度。当延迟视图在基础提交后准备就绪时，这些状态可用于筛除基础视图中已表示的区域，并仅识别新颖几何。随后，提取出的新颖区域会以流水线方式无缝融入客户端的基础视图。通过随基础视图传输重建状态，并且仅发送延迟视图中的新颖部分，FugaStream 降低了更新开销，同时保留了延迟视图带来的重建质量提升。

我们将贡献总结如下 .

- 我们提出 FugaStream，这是首个流水线式实时体积视频流系统，它通过在额外视图就绪时增量优化结果，而非阻塞直到收到所有视图，从而消除了重建前必须等待所有视图的需求。
- 我们设计了自适应基准提交控制与基于基础视图的延迟视图补全，以自适应地决定何时提交基础重建，以及如何利用延迟到达视图对其进行优化，从而使 FugaStream 能够在延迟预算允许的范围内尽可能利用更多视图。
- 我们在广泛使用的体积数据集以及真实世界测试平台上评估了 FugaStream。与现有方法相比，在 150 ms 的延迟预算下，FugaStream 将结构相似性指数和交并比分别提升了 10.9% 和 13.7%。

2 背景

体积流管线通常由三个阶段组成：（i）多路 RGB-D 相机采集，（ii）服务器端三维重建，以及（iii）客户端渲染，如图 2 所示。为确保各视图之间的一致性，位于不同视点的 RGB-D 相机同时采集场景。我们将与同一采集时间戳相关联的同步多视角帧称为 帧集合。服务器在收到帧集合后，去除背景和噪声，并将处理后的视图合并为统一三维表示。随后，重建得到的三维数据被传输至客户端，以便从用户当前视点渲染该对象。

为了支持从任意视点进行渲染，三维重建必须融合来自所有视图的测量数据

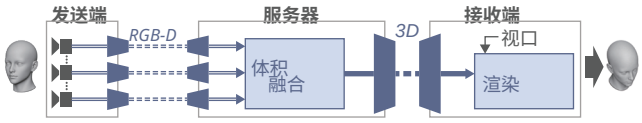


图 2：传统实时体积流传输流水线。多个 RGB-D 视图被传输至服务器，融合为三维表示，经过编码后发送至接收端进行视口渲染。

摄像机	8 台 Orbbec Femto Mega RGB-D 相机 [32] NVIDIA Jetson Nano
服务器	AMD Ryzen 9 7950X 32 GB DDR5 NVIDIA GeForce RTX 4090
客户端	NVIDIA Jetson Orin NX

表 1：真实世界测试平台的规格。所有组件均通过 1 Gbps 以太网连接。

将其转化为连贯的表面表示，且视图之间没有孔洞或不连续之处。为此，截断符号距离函数（TSDF）融合被广泛用于体积重建 [6, 31]。TSDF 将三维空间表示为体素网格，其中每个体素存储一个 TSDF 值及其对应的置信度权重。TSDF 值编码体素沿相机射线到观测表面的有符号距离，而置信度权重则决定该体素在融合过程中的贡献。对于每个摄像机视图，将体素投影到相应的深度图上，并将其深度与观测表面的深度进行比较，以获取有符号距离。位于表面前方的体素取正值，位于表面后方的体素取负值，距离则截断在表面周围的有限范围内。当多个视图观测到同一区域时，通过加权平均逐体素合并其 TSDF 值。随后，从融合后的 TSDF 场的零交叉处提取重建表面。

TSDF融合的一个重要特性是，每个视图并非表示为独立的三维数据，而是累积到共享的体素网格中。我们利用这种共享表示来支持基于基准的后到视图补全——首先可以根据提交时刻可用的视图构建基准模型，之后延迟视图到达时可更新同一体素网格，而无需从头重建。

3 动机

本节中，为了严格研究实时体积视频流的特性，我们使用真实世界测试平台进行实证分析。测试平台的配置总结见表1。每台摄像机以 30 fps 捕获 RGB-D帧，即帧周期为 33 ms，而同步集线器将各摄像机的捕获时刻对齐至 1 ms 以内。每个相机节点都配备了一台 NVIDIA Jetson Nano，用于本地帧处理和传输。如 § 1 所述，当帧集合中的视图未能同时就绪时，服务器将面临两种极端选择：等待更久，或提前提交。我们首先量化测试平台中观测到的帧可用性差异，然后评估每种选择对延迟和重建质量的影响。

视图可用性差异较大。 图 3 展示了八个视图的融合就绪时间累积分布函数，这些视图按照各自在每个帧集合中的就绪顺序排列。我们在包含 5,180 个帧集合的视频序列中，测量了每个视图从捕获到在服务器端融合就绪所需的时间。在 p99 处，最先就绪的视图约在 30 ms 后可用，而最后就绪的视图则需要超过 65 ms。考虑到帧周期为 33 ms，在当前帧集合尚未完整时，下一帧集合中的视图可能已经可用；如果重建需要等待所有视图，就可能导致流水线停顿或丢帧。在最坏情况下，最后就绪时间可达 138 ms，在重建尚未开始前就已耗用 150 ms 端到端延迟预算的近 92%。

缺失视图导致的质量下降。体积流传输的显著特征在于支持 6 自由度，这要求为从任意视点进行渲染提供足够的表面覆盖。因此，缺失视图可能会留下未被观测的区域，并在用户移动时暴露出来。图 4 显示，即使省略单个视图，也会显著降低重建质量。在此示例中，仅缺失一个视图就造成了严重的几何失真，并使大部分表面处于不完整状态。对于体积流传输而言，这种质量下降尤其成问题，因为用户可以自由改变视点，直接暴露受影响的区域。

这些观测表明，对于实时体积视频流而言，无论是等待所有视图，还是在延迟视图到达之前过早提交，都不是理想选择。受这两项要求的启发，我们提出了一种流水线流媒体系统：服务器利用已可用的视图重建并传输基础表示，随后，延迟视图准备就绪后，在客户端将其逐步并入基础表示。这样既无需等待完整的帧集合即可进行重建，又能在后续细化过程中利用延迟视图。

4 挑战

我们构建了一种流水线式体积流系统，提前提交基础视图，并在延迟视图就绪后将其纳入。然而，实现这一目标带来了根本性的

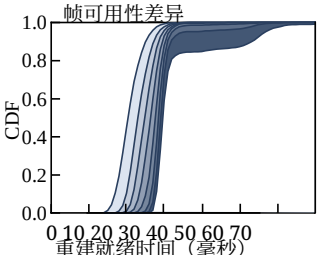


图 3：帧中的 Spread



（a）缺少一个视图　（b）由所有视角融合

图 4：缺失视图可用性对视图的影响示例。

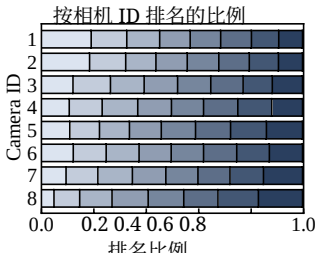


图 5：各摄像机的融合就绪排名分布。

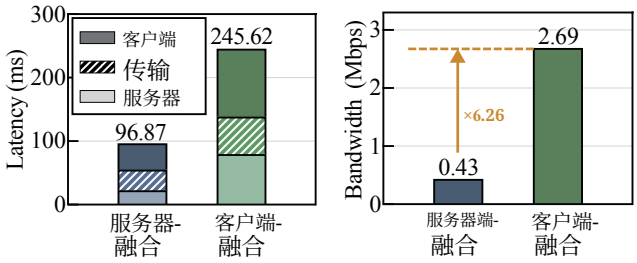


图 6：在客户端独立传输并融合多视图观测与在服务器上进行融合的成本。

挑战，这些挑战在传统实时体积流式传输中并不存在。本节将对这些挑战进行分析。**视图可用性的不确定性。**固定的提交决策并不可行，因为不同帧集之间的视图就绪状态并不一致。为考察这种变化，图 5 展示了对于每个视图，其帧在帧集中的帧集合中分别以第 1 至第 8 名排名达到融合就绪状态的比例。如图所示，没有任何单一视图始终占据最后就绪排名。每个视图在相当一部分帧集中的就绪顺序都是最后一个，因此不存在能够通过离线分析识别出的持续延迟视图。 consequently，固定提交控制会在相邻帧集之间舍弃不同的视图。由于每个视图覆盖表面的不同区域，缺失区域也会随帧变化，使用户看到时间不稳定的空洞或闪烁。简单地延长等待时间并不能解决这一问题：由于可用性分布范围 (§ 3) 的变化，这可能导致某些帧集中出现空闲等待，同时在其他帧集中仍无法纳入长尾视图。因此，提交决策必须在运行时适应每个帧集中观察到的这种不确定性。

纳入延迟视图的成本。以细粒度纳入延迟视图，可能会削弱提前提交带来的延迟和带宽优势。一种直接的设计是，在每个延迟视图准备就绪后立即对其进行编码并传输，使客户端能够逐步融合收到的视图。然而，多视图观测之间存在大量重叠，因此，若将每个视图视为独立更新，就会反复传输并融合已被其他视图捕获的信息。图 6 通过比较两种设计量化了这种开销。服务器端融合在服务器上融合全部八个视图，并将融合表示发送给客户端；而客户端融合则独立传输每个视图，并在客户端执行融合。延迟测量不包括服务器端等待时间，计时范围从融合开始直至帧准备好进行渲染；带宽则衡量服务器到客户端流量。如图所示，细粒度的客户端融合所产生的延迟和带宽开销显著高于服务器端融合。因此，系统必须纳入延迟视图的有效贡献，同时避免反复传输和融合基础视图中已有的信息。

综合来看，这些观测表明，仅仅将体积流传输分解为多个阶段是不够的。由于视图可用性在不同帧集合之间有所变化，系统必须根据运行时观测到的就绪状态调整其提交决策。同时，独立处理每个延迟视图会产生冗余传输和融合，从而增加延迟和带宽消耗。因此，系统必须纳入延迟视图，同时限制由此产生的额外成本。下一节将介绍我们为应对这些挑战而设计的系统。

5 系统设计

5.1 系统概述

FugaStream 的总体系统架构如图 7 所示。在该系统中，多帧 RGB-D帧以不同的延迟传输至服务器。为提交基础视图，服务器会估计每个视图何时可用，并自适应地调整其等待时间（§ 5.2）。随后，服务器提交基础重建，并仅传输其稀疏表面带体素。这样既减少了传输带宽，也降低了客户端计算。对于基础传输之后的延迟

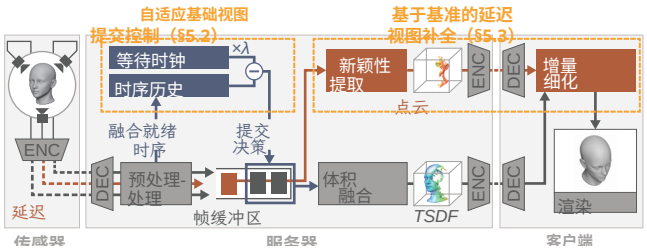


图 7：FugaStream 系统概览。该流水线式体积流传输系统会先提交，再合并延迟视图，以改善延迟-质量权衡。

视图，服务器仅提取基础视图未能充分覆盖的新颖区域（§ 5.3）。客户端将这些延迟视图合并到基础视图中，并在直播延迟要求内增量更新三维表示。

为实现这一目标，我们定义了以下设计目标：

- **端到端交互延迟**。我们的目标是在应用的端到端延迟预算内交付基础表示。为此，服务器会决定在融合前最多等待多长时间。
- **延迟视图利用**。与此同时，延迟视图中有用的几何贡献应在可用时得到保留。因此，我们的系统会让基础表示保持可细化，并以最小开销仅整合延迟视图中的新颖区域。

5.2 自适应基准提交控制

为构建基础表示，我们引入一种自适应基准提交控制（ABCC）算法，用于确定何时提交基础重建。算法 1 总结了 ABCC 的工作流程。每当视图$i \in \{1、\ldots\ldots N\}$变为融合就绪时，ABCC 会评估：对于尚未就绪的视图 M_g，额外等待时间 Δ^* 是否可接受。在 Δ^* 内变为可用的每个视图都会被添加到 R_g，即将纳入基础视图集的视图集合。ABCC 根据两个因素计算 Δ^*：(i) 等待 M_g中的视图所获得的覆盖增益；(ii) 随着时间流逝呈指数增长的等待成本。

覆盖增益。如第 3 节所述，每个视图在各帧集中的可用时序和顺序存在显著差异。为适应此类变化，ABCC 在每个摄像机最近的融合就绪时间历史 $\{H_i\}$ 上维护一个滑动窗口，覆盖最近 150 个帧集。随后，根据该窗口推导出每个帧集的融合就绪时间经验累积分布函数 $\{F_i\}$。在提交之后到达的视图仍会被记录，而从未到达的视图则记录为 $+\infty$。给定当前时间 t，视图 $i \in M_g$ 在额外等待 Δ 后变为就绪的条件概率 $p_i(t,\Delta)$ 计算如下：

算法 1 自适应基准提交控制

输入： 每个视图的融合就绪历史记录 $\{H_i\}_{i=1}^{N}$
输出： 已提交的基础视图集 $R_g \subseteq \{1、\ldots\ldots N\}$

```
对于 每个帧集合 g 执行
    {F_i} ← 估计累积分布函数({H_i})
    R_g ← ∅                                  ▷ 重置候选基集合
    当 g 尚未提交时
        等待 ViewReady(i, t) 或超时
        如果超时则
            提交 R_g；中断
        R_g ← R_g ∪ {i}
        如果 |R_g| = N 则
            提交 R_g;中断
        M_g ← {1, ..., N} \ R_g ▷ 未提交视图集合
        Δ* ← 选择等待(t, {F_i}_{i∈M_g}, g)
        如果 Δ* = 0 则
            提交 R_g;中断
        else
            在 t + Δ* 设置超时
    更新 {H_i}                               ▷ 记录融合就绪时间
```

$$p_i(t,\Delta) = \frac{F_i(t+\Delta) - F_i(t)}{\max(1 - F_i(t), \epsilon)}, \tag{1}$$

其中，一个较小的常数 ϵ 可避免数值不稳定性。等待 Δ 所获得的覆盖增益 $G_t(\Delta)$ 为

$$G_t(\Delta) = \frac{1}{N} \sum_{i \in \mathcal{M}_g} p_i(t,\Delta). \tag{2}$$

由于每个视图都会贡献其他视图无法提供的表面观测，因此，更大的 G_t 表示在提交时拥有更完整的基础视图。

等待成本。由于 G_t 随 Δ单调增加，覆盖增益本身总是倾向于等待更长时间。然而，等待并非没有代价：每增加一次等待，都会延迟基础视图提交，并直接增加端到端延迟。此外，随着当前时间 t 的增加，相同的等待间隔 Δ 会带来更大的惩罚，因此，等待多久以及何时等待都很重要。因此，我们将从 t 到 $t+\Delta$ 的等待成本表述为

$$C_t(\Delta) = e^{(t+\Delta)/\tau} - e^{t/\tau}, \tag{3}$$

其中 τ 控制成本随时间增长的速度。随着基础提交延迟，成本不断增加，这会抑制进一步等待，除非覆盖增益足够大，从而有助于保障延迟预算。

提交决策。结合公式 2 和公式 3，ABCC 定义了再等待 Δ 的目标函数：

$$J_t(\Delta) = G_t(\Delta) - \lambda C_t(\Delta), \tag{4}$$

其中，$\lambda > 0$ 是一个超参数，用于权衡覆盖增益与等待成本。随后，ABCC 将等待时间 Δ^* 确定为：

$$\Delta^* = \arg\max_{\Delta} J_t(\Delta). \tag{5}$$

如果 $\Delta^* = 0$，即预期覆盖增益不再超过等待成本，系统便停止等待并提交基础视图重建。否则，它会将额外等待时间延长 Δ^*。我们在 § 6 中验证 ABCC 的有效性。

5.3 基于基准的后到视图补全

由于基础视图会在所有视图准备就绪之前传输给客户端，我们引入一种基于基准的后到视图补全算法，将延迟视图融入基础视图。如前所述，视图间冗余会消耗带宽和客户端计算，却不会增加新的表面信息。因此，我们将说明服务器如何仅提取每个延迟视图中的新颖区域，以及客户端如何将接收到的区域合并到基础视图中。

服务器端新颖性提取。基础视图重建完成后，冗余会出现在两个方面：基础视图与每个延迟视图之间，以及各延迟视图之间。因此，服务器会针对每个延迟视图，仅提取相对于这两者均为新颖的区域。

首先，服务器会移除相对于基础视图的冗余。对于映射到基础网格体素 $v(\mathbf{x})$ 的延迟视图观测 $\mathbf{x}$，我们将其新颖性分数定义为：

$$P_g(\mathbf{x}) = S(\mathbf{x})\left(1 - C_g(v(\mathbf{x}))\right), \tag{6}$$

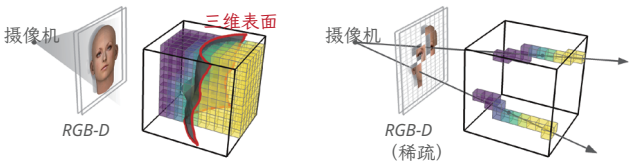
其中，$S(\mathbf{x})$ 表示 $\mathbf{x}$ 的局部深度支持，$C_g(v)$ 表示冻结基础中体素 v 的归一化覆盖度。$S(\mathbf{x})$ 根据相邻深度样本之间的一致性计算得出，用于抑制不一致测量。$C_g(v)$ 根据累积融合权重 $W_g(v)$ 和已提交视图的总数 $|\mathbb{R}_g|$ 得出：

$$C_g(v) = \min\left(\frac{W_g(v)}{|\mathcal{R}_g|}, 1\right). \tag{7}$$

当体素已被基础视图充分覆盖时，$C_g(v)$ 会趋近于 1。分数低于阈值θ_p的观测将被视为相对于基础视图存在冗余，并被丢弃。

随后，服务器会移除延迟视图之间的冗余。为此，它会为每个帧集合维护一个临时重叠图。每当某个体素被延迟视图选中时，重叠图就会记录该体素；后续延迟视图中映射到同一体素的任何观测都会被丢弃。因此，每个体素恰好由一个延迟视图承载，并且帧集合处理完成后将丢弃该重叠图。由此得到的稀疏点集合既减少了下游带宽，也减少了冗余的客户端融合计算。

客户端增量式基础细化。客户端会在每个视图到达时逐步将其整合进来。正如



(a) 逐体素 TSDF **(b) 逐射线 TSDF**

图 8：TSDF 融合路径对比。(a) 逐体素融合将体素网格投影到深度图上，而 (b) 逐射线融合则沿射线进行遍历。

第 2 节所述，TSDF融合通常会将体素网格投影到深度图上。在这种逐体素融合中，如图 8(a) 所示，无论视图实际观测到多少内容，每个体素都会更新。每当一个视图到达时都重复这一过程，会给客户端带来显著的计算开销。

为了解决这一问题，我们采用了逐射线融合，如图 8(b) 所示。由于服务器只提取新颖区域，每个延迟视图仅包含少量点，因此没有必要更新每个体素。每当延迟视图到达时，客户端根据摄像机标定重建视线射线，并仅更新射线穿过的体素。具体而言，客户端沿射线遍历截断符号距离场截断带，更新体素权重；当延迟视图观测到表面的角度比已存储的视图更接近正面时，则替换已存储的颜色。

总体而言，这些机制使体积融合能够在所有视图到齐之前开始，同时不牺牲这些视图原本能够带来的质量。ABCC 限制等待基础视图集的时间，使融合能在延迟预算内启动；而基于基准的后到视图补全则弥补基础表示中缺失的覆盖范围。FugaStream 无需丢弃提交后才可用的视图，即可满足端到端延迟预算。我们在 § 6 中验证 FugaStream 的有效性。

6 评估

6.1 评估设置

网络设置。我们使用 Linux 流量控制（tc [18]）和 NetEm [14],模拟摄像机到服务器的网络，将每条链路的带宽配置为 600 Mbps、往返时间配置为 40 ms，且不产生丢包。摄像机流使用 H.264 [35] 编码，并通过 WebRTC [8] 传输至服务器。对于服务器到客户端的传输，服务器与客户端通过 1 Gbps 以太网连接，并在该连接上模拟 40 ms 的往返时间且不产生丢包。我们实现了一种基于 UDP 的传输方式，发送端速率控制为 250 Mbps，并采用基于 NACK 的重传。客户端不使用抖动缓冲区。

数据集。 我们在 CMU Panoptic-Studio 数据集 [19],上评估 FugaStream。该数据集是一个公开的多视图采集数据集，已广泛用于体积流传输。在其多个序列中，我们选用了*Mafia*，该序列包含约 18,000 个帧集合。序列*Mafia*包含多个人物，并呈现视角相关遮挡，构成几何形状变化显著的动态场景。

指标。 我们测量端到端延迟，以评估各系统是否支持实时交互式流传输。我们还从两个层面评估质量。在图像层面，我们报告客户端渲染帧的结构相似性指数（SSIM） [34] 。在几何层面，我们报告渲染重建结果与参考视图之间轮廓的交并比（IoU），以衡量在所有视图到达前就提交一组帧时，表面有多少部分缺失或被错误重建。我们使用 SAM3 [2] 从每张参考图像中提取人物轮廓。

基线。 为评估 FugaStream 的端到端性能，我们将其与静态系统和自适应系统进行比较。

• *Fixed-δ* 会等待固定时间间隔，然后提交重建结果。我们评估的时间间隔为 $\delta \in \{40,\ 65,\ 100\}$ 毫秒。40 毫秒的设置代表一种激进策略，其等待时间接近 30 fps 下单帧的间隔；而 100 毫秒的设置会等待足够长的时间，以纳入几乎所有视图，从而最大限度地减少质量损失。为实现最佳的延迟与质量权衡，我们将提交阈值设为 65 毫秒，这恰好对应于系统中观测到的融合就绪扩散范围的 p99。

• *Event-δ* 以一个等待间隔开始，并在每个新增视图可用时重置计时器。具体而言，在每个新可用的视图之后，它会从该融合就绪时间起再等待$\delta \in \{15,\ 25,\ 40\}$毫秒——该扩展不会跨帧累积。此策略体现了一种简单的自适应策略：当视图接连快速就绪时，持续延长等待时间。

• *MetaStream* [11] 根据观测到的视图到达 Spread 与当前 U 之间的中位数差值，自适应地调整每 U 个 g帧集合的等待时间，从而利用网络状况的时间相关性。因此，当观测到的 Spread 通常超过当前 U时，它会增加U，反之则会减少。遵循原始

在实现中，我们设置 $g = 30$。

据我们所知，现有系统中没有一个会在帧集提交后传输迟到的视图。MetaStream 会丢弃在 U 之后到达的视图，而我们实现其余基线方法时，也使其具有相同的行为。

6.2 FugaStream 的总体性能

在本节中，我们评估 FugaStream 的端到端性能，重点考察其在有效利用流水线设计的同时，是否满足实时体积流传输严格的延迟要求

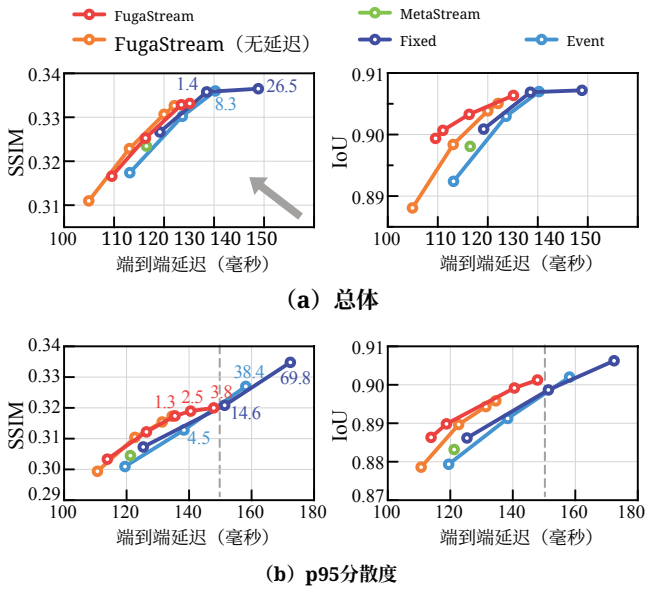


图 9：FugaStream 与基线方法的延迟-质量权衡。(a) 所有帧集上的平均性能；(b) 在 p95融合就绪分散度下的性能，此时延迟到达视图会带来更大的延迟惩罚。

。具体而言，我们首先在平均视图可用性分布和尾部视图可用性分布下，研究端到端延迟-质量权衡，并在第 6.1 节将其与基线方法进行比较。随后，我们分解端到端延迟，展示早期基础提交和延迟视图整合如何与传输、客户端处理及渲染重叠，并量化这种流水线设计如何使显示结果保持在目标延迟预算以内。**延迟-质量权衡**。首先，我们评估 FugaStream 和基线方法的端到端延迟-质量权衡。图 9(a) 报告了各帧集的平均端到端延迟和渲染质量，而图 9(b) 则关注视图可用性条件的 p95 分布，其中延迟视图会带来显著更高的延迟成本。我们同时报告结构相似性指数（SSIM）和交并比（IoU），以反映渲染图像质量。每个点旁的数字表示落在连续三个或更多帧组违反延迟预算的运行中的帧组百分比；我们仅标注超过 1% 的数值。FugaStream 不含延迟视图表示在整合任何延迟视图之前的基础表示。

为在广泛的延迟范围内评估所有方法，我们调整各自的延迟控制参数。对于 Fixed 和 Event 基线方法，我们扫描其服务器端等待间隔，以覆盖不同的延迟

预算。与此同时，MetaStream 仅在单一运行点进行评估，因为它依赖中位到达时间，导致其延迟无法调节。相比之下，对于 FugaStream，改变延迟惩罚权重——λ 对应 $\{0.1, 0.2\}$，τ对应 $\{50, 100\}$——可以在延迟–质量权衡空间中连续调节，从而在等效延迟上限下实现公平比较。

如图 9(a) 所示，与各基线方法相比，FugaStream 总体上实现了更优的延迟-质量权衡。通过提前提交基础视图，并随后纳入延迟视图更新，FugaStream 在保留提前提交带来的延迟优势的同时，还能从较晚才可用的视图中恢复质量。具体而言，未进行延迟视图细化的 FugaStream 表明，在质量水平相当时，ABCC 的触发时间明显早于所有基线方法。这证实 ABCC 能够自适应地终止对不太可能及时到达的延迟帧的等待，从而有效避免严重的等待时间惩罚。相比之下，Fixed 只有通过等待更长时间以获取额外视图才能提升质量，这会直接增加延迟，却无法带来显著的质量提升；在 Fixed-100 下，延迟有 26.5% 超出预算。同时，尽管 MetaStream 和 Event 会调整其提交时序，但由于未考虑视图可用性的分散情况，它们无法反映在线动态，因而产生额外的延迟惩罚。最后，由于 FugaStream 在客户端等待期间纳入延迟视图，重建质量有所提升，延迟也略有增加，但仍满足 150 ms 的要求。关键的是，由于延迟更新优先填补新颖几何信息，而非细化基础视图中已有的颜色，因此总体结构相似性指数仅略有提升，而交并比则显著提高。这直接证实，我们的新颖性提取机制能够成功筛选并传输基础提交中缺失的有意义数据。

在图 9(b) 所示的 p95 视图可用性分布范围下，FugaStream 的这一优势更加明显。虽然不采用延迟视图的 FugaStream 通过抑制过长的等待时间，仍处于低延迟状态，但启用客户端渲染时序控制后，延迟视图可以持续整合，直至达到应用目标延迟。因此，在严格满足

150 毫秒尾延迟预算。相比之下，Fixed 和 Event 无法满足延迟约束，而 MetaStream 则通过丢弃延迟视图来满足约束，直接损害了最终重建质量。

流水线执行。接下来，我们考察 FugaStream 如何降低流水线的端到端延迟。图10分解了执行时间线，并将 Fixed-40 和 Fixed-100 与 FugaStream 进行比较。对于 FugaStream，延迟视图 #1–#3 表示在基础视图提交后才可用、并作为独立更新处理的延迟视图。Fixed-100 基线包含更多视图，但会使整个流水线

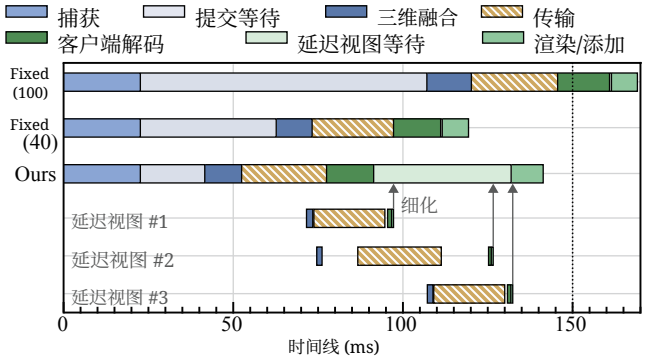
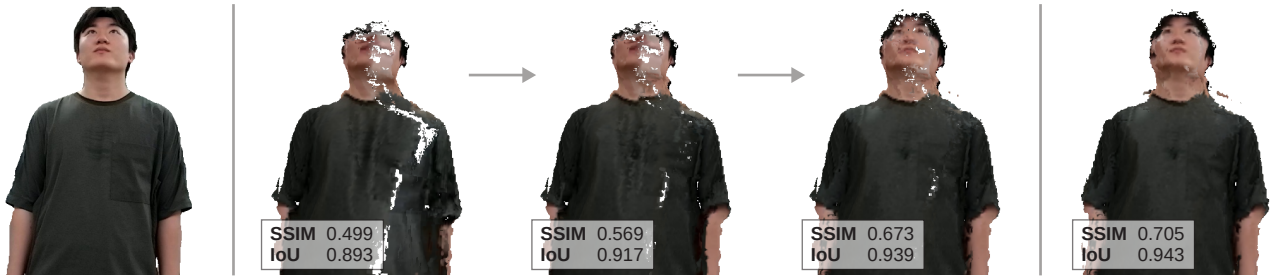


图10：端到端时间线分解。延迟视图 #1–#3 在基础提交后变为可用，并以流水线方式进行处理。

超出 150 毫秒的延迟预算，而短等待方案 Fixed-40 则通过排除尚未就绪的视图来提前提交。

FugaStream 会提前提交基础视图，并立即使其依次经过服务器三维融合、传输和客户端解码。在剩余的延迟预算内，延迟视图会如图10所示（延迟视图#1–#3）独立传输和处理。它们的处理过程与基础视图的下游推进重叠，而不是等待基础视图流水线完成。这种并行处理使延迟视图的融合就绪延迟能够与计算密集型基础三维融合形成流水线。例如，当基础视图完成解码时，延迟视图#1 已经排队等待立即集成，从而使其初始延迟对客户端完全透明。由于合并这些排队更新只需要轻量级的逐点操作，因此多个延迟视图可以在最终渲染之前无缝参与其中。这种流水线执行利用剩余的延迟预算从延迟视图中恢复信息，同时确保最终显示保持在150毫秒目标以内。因此，FugaStream 在继续改善渲染前表示的同时，保留了提前提交带来的延迟优势。

真实世界评估。我们使用§ 3和表1中所述的硬件，在真实世界测试平台上进一步评估FugaStream。请注意，奥比中光 Femto Mega使用了与Azure Kinect DK [30], 相同的深度相机模块，该模块已被体积视频系统广泛采用 [7, 15, 16, 21]。每台RGB-D相机使用Orbbec SDK传输H.264编码的彩色流和16位Y16深度流。服务器到客户端的网络采用与§ 6.1相同的配置。我们评估了三个捕获序列——*Stretching*、*Roundwalk-ing*和*Talking*——每个序列包含1,800个帧集。这些序列分别代表单人动态场景、四人动态场景和四人相对静态场景。



(a) 真实值　(b) 包含6个视角的基线　(c) 第+ 1个缺失视图　(d) 第+ 2个缺失视图　(e) 包含8个视角的基线

图11：基于基准的延迟视图补全定性示例：两个延迟视图逐步细化六视图基准，最终形成八视图重建。

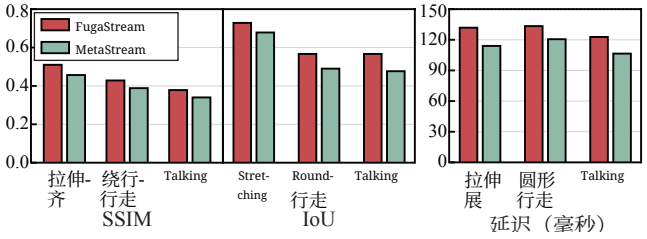
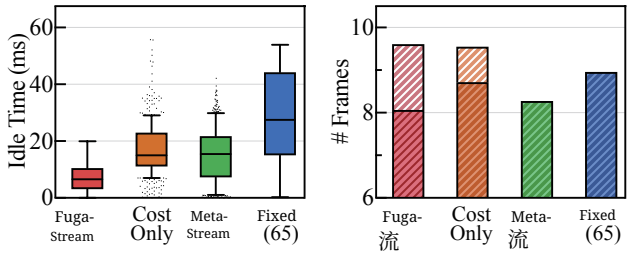


图12：在三个捕获序列——*Stretching*、*Round-walking* 和*Talking*——上的真实世界测试平台评估。



(a) 提交前空闲　(b) 有效帧

图13：在第95百分位视图就绪度差异下 ABCC 的有效性。有效帧是指合并到渲染图像中的帧。

我们首先在图11中展示真实世界轨迹之一（Stretching）的定性示例，直观呈现随着延迟视图逐渐可用，FugaStream如何将早期已提交基底逐步演化为细化后的重建结果。(a)展示真实值，(b)展示由最初可用的六个视图形成的基底重建。六视图基准的有限覆盖范围使初始重建缺失了若干表面区域，这种不完整性也反映在(b)中较低的结构相似性指数和交并比上。(c)和(d)展示了在接收到剩余两个延迟视图并将其逐步合并后，我们基于基准的后到视图补全结果。每次细化都会填补初始基底中缺失的区域，这一点也体现在结构相似性指数和交并比的逐步提升上。在纳入两个延迟视图后，(d)中的重建结果接近于(e)中基于全部八个视图联合构建的模型。

为了进行定量比较，图12从结构相似性指数（SSIM）、交并比（IoU）和端到端延迟（E2E latency）三个方面比较了FugaStream与MetaStream。在全部三个序列中，FugaStream在SSIM和IoU方面始终实现了更高的渲染质量。MetaStream通过更早提交而获得了更低的延迟，但错过其提交节点的延迟视图将无法再为重建作出贡献，从而降低感知质量和几何精度。相比之下，FugaStream在早期提交基础视图后仍保留了这些视图的贡献，以适度增加的延迟换取了更高的质量。重要的是，在所有序列中，其端到端延迟均保持在150毫秒目标以内。这些结果表明，FugaStream在真实采集、编码和传输动态下仍能保持延迟与质量方面的优势，而不仅限于我们早期实验所采用的受控环境。

6.3 组件级评估

空闲时间与帧利用率。我们考察 ABCC 是否能在严重的视图就绪度差异下减少不必要的等待和空闲时间，同时不牺牲帧利用率。图13(a)报告了基础提交前的空闲时间，而图13(b)展示了客户端接收到的帧数及由此产生的交并比。我们将 ABCC 与仅成本方法、MetaStream 和 *Fixed-65*进行比较。此处，仅成本方法将 $G_t(\Delta)$ 替换为其上界 $|\mathrm{M}_g|/N$，即假设每个缺失视图都会在 Δ 内就绪。

ABCC 显著降低了 p95 分布下空闲时间的大小及其波动性。具体而言，其空闲时间中位数比仅成本变体低约 8.5 ms。这一差异表明，已流逝的等待时间

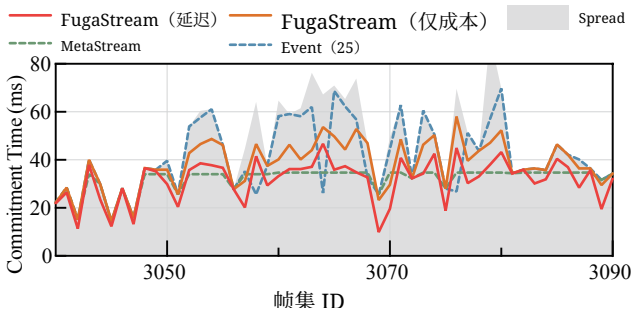


图 14：视图可用性 Spread 变化时的帧集级提交。

本身不足以做出有效的提交决策。还必须考虑剩余视图的预期覆盖增益，才能决定是否继续等待。空闲时间缩短并未降低帧利用率，因为 ABCC 和仅成本变体交付给客户端的帧总数几乎相同。请注意，仅成本变体与 MetaStream 的空闲时间相近，而 MetaStream 会丢弃提交后才准备就绪的视图，导致帧利用率降低。相比之下，FugaStream 通过基于基准的后到视图补全恢复这些视图，从而减少服务器端等待时间，同时不会永久牺牲它们的贡献。

运行时提交行为。 我们进一步考察 ABCC 如何在连续帧集之间调整其提交时序。图 14 展示了各方案在一个*Mafia* 序列中的提交时刻，以及观测到的视图可用性 Spread。我们将 FugaStream 与仅成本方法 变体、MetaStream 和 Event-25 进行比较。

FugaStream 按帧集合调整其提交时刻，而不是简单地随就绪时间离散度的波动而变化。随着等待时间增加，ABCC 会持续重新评估剩余视图的预期贡献，以及它们在额外等待期间就绪的条件概率。因此，即使视图可用性的离散度很大，只要剩余视图短时间内就绪的可能性很低，或其预期覆盖增益已不足以抵偿等待成本，FugaStream 就会停止等待。相比之下，仅成本方法通常等待更久，因为它无法判断剩余视图是否值得额外延迟；而 Event 每当有新视图就绪时都会强烈响应，导致提交时刻波动更大。MetaStream 则表现相反：其基于中位数的自适应抑制了尾部行为，使形成较大离散度的最后就绪视图对其等待时间影响甚微，因此这些视图往往会被丢弃。FugaStream 根据当前状态做出每项提交决策，能够响应短暂变化，同时既不会激进地追踪单个延迟视图，也不会忽略就绪时间的尾部行为。

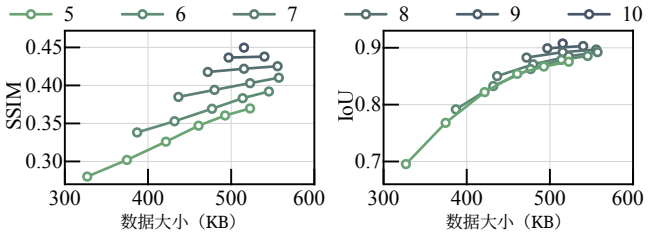


图 15：不同基础视图规模下延迟视图细化的效果。

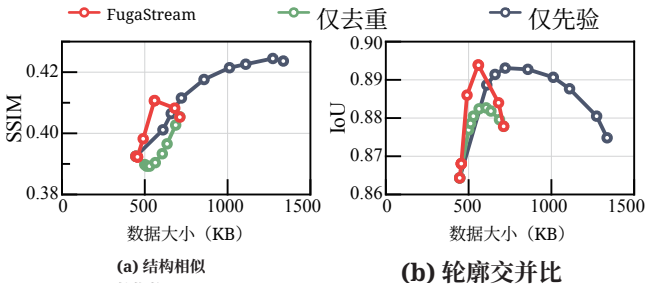
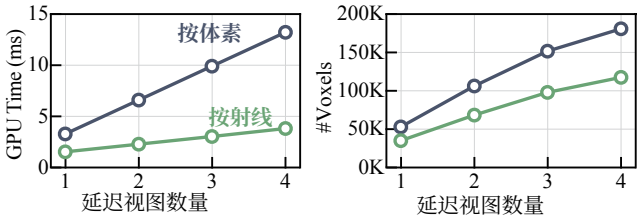


图 16：基于基准的后到视图补全的消融研究。仅先验仅考虑新颖性分数（公式 6），而仅去重 仅为每个帧集合维护临时重叠图。

延迟视图带来的重建增益。我们考察通过 FugaStream 的延迟视图细化路径纳入延迟视图后，是否能带来有意义的重建增益。在总共十个视图中，我们将基础表示中包含的视图数量B 设为五至九个，并逐步将其余 $M = 10 - B$ 个视图作为延迟视图更新应用。每条曲线固定包含 B 个视图的基础表示，并逐次增加一个延迟视图，因此最左侧的数据点表示仅基视图质量，而曲线上的增幅则是图 15 中单由延迟视图带来的增益。

结果展示了两点。首先，在所有基础视图规模下，延迟视图细化都有益处，能够持续提升重建质量。延迟视图补充了基础表示未能充分捕获的表面信息，使结构相似性指数最多提升 0.09，交并比最多提升 0.18。其次，延迟视图无法完全弥补基础视图覆盖范围不足的问题。即使总视图数达到十个，从较小基础视图初始化的重建效果仍不如从较大基础视图初始化的重建。这表明，过早提交会限制最终重建质量的上限，因此服务器应等待形成足够大的基础视图后再进行提交。

新颖性提取的效果。FugaStream 新颖性提取的核心目标是在最大限度保留三维重建



(a) 细化延迟 **(b) 更新体素数量**

图 17：在 NVIDIA Jetson Orin NX 上测得的逐射线和逐体素 TSDF 融合的客户端细化开销。

质量的同时，大幅降低传输开销。为剖析各机制的贡献，我们将集成的新颖性提取方法与两种消融变体进行比较：(i) 仅先验，仅根据相对于基础表示的新颖性分数（公式 6）筛选点；以及 (ii)仅去重，仅利用延迟视图之间的重叠图（图 16）选择点。图中的每个标记对应一个新颖性阈值 θ_p；由于仅去重不使用该分数，因此我们通过在每个阈值下随机抽取相同数量的点，使其速率与 FugaStream 匹配。

为隔离延迟视图之间视图间去重的影响，我们首先将 FugaStream 与仅先验基线进行比较。去重是我们节省带宽的主要因素。对于每个帧集合，它将更新严格限制为每个体素仅保留一个点，从而过滤冗余点并减小总体 3D帧大小。具体而言，在不进行去重的情况下，仅先验方法在其最佳运行点上每个 3D帧消耗 723 KB，比 560 KB 大 29%。如图 15 所示，这种严格的空间去重确保总带宽消耗主要由所纳入的视图数量决定，而仅在较小程度上取决于这些视图如何在基础视图和延迟更新之间划分。

然而，单纯依赖去重（仅去重）会直接以重建质量为代价限制带宽。由于去重每个体素仅允许一次更新，因此保留的点必须分别在几何保真度（交并比）和视觉保真度（结构相似性指数）方面带来有意义的改进。在没有基于效用的优先级排序时，仅去重无法过滤噪声；虽然它最初能够构建主体核心，但随后会将宝贵的带宽分配给嘈杂的外部区域，导致交并比下降。相比之下，FugaStream 的优先级过滤专门提取相对于基础的新颖性。FugaStrea m 仅转发必要的更新，并防止噪声抢占关键数据，从而推动两项指标均大幅提升，避免了基线方法的性能下降。

总体而言，FugaStream 的新颖性提取有效地平衡了几何恢复与传输效率。通过准确选择几何新颖性，该系统实现了与未过滤基线相当的交并比，并成功重建了 3D形状。其固有的权衡体现在颜色细化上：由于去重明确阻止对已提交基底进行冗余更新，颜色增强仅限于新纳入的新颖区域，因此绝对峰值结构相似性指数略有降低。尽管如此，正是这种精确的折中，使 FugaStream 能够整合延迟到达视图，并以相近的带宽开销保持几何保真度。

基础细化开销。 我们的系统要求基础细化保持轻量，以便纳入延迟视图不会损害提前提交所带来的延迟优势。我们评估了客户端基础细化的计算开销。图17(a)比较了随着视图数量增加，我们的逐射线细化与逐体素替代方案的GPU计算时间，而图17(b)报告了每种方法更新的体素数量。

随着更多延迟视图到达，两种方法之间的延迟差距逐渐扩大：逐射线细化仍然轻量，而逐体素细化带来的延迟高出2.14–3.47×倍。逐射线细化仅跟踪与每个延迟视图相关的射线，并直接更新新观测区域所需的体素。相比之下，逐体素细化不仅必须应用所需的更新，还必须访问已提交基底中的每个体素，以识别这些体素，其中包括与延迟视图无关的体素。随着延迟视图不断累积，这种不必要的遍历会变得越来越昂贵。通过将计算限制在确实需要细化的体素上，逐射线细化使客户端基底更新能够以极小的延迟开销实现。

7 讨论

应用感知运行时优化。我们当前的设计表明，利用轻量级运行时信息、融合就绪时间的历史记录以及额外等待的成本，可以有效实现流水线式体积流传输。不过，进一步优化运行时仍有望改善延迟-质量权衡。例如，能够容纳更丰富逐帧推理的应用可以采用基于深度神经网络的预测器，提取复杂的时间和场景相关模式，从而做出更复杂的承诺决策。网络带宽、服务器负载和渲染反馈等其他运行时信号也可为这些决策提供依据。此类信息和机制可以根据应用特点加以整合，并依据延迟要求以及可用的推理与控制预算，限制决策的复杂度。

异构下游条件。我们当前的评估主要是在受控条件下验证 FugaStream

的性能。实际部署中，系统可能需要同时服务多位用户，而下行带宽可能随时间变化，也可能因用户而异。不同客户端可用的计算资源和应用层延迟要求也可能存在显著差异。这些异构下游条件可能导致用户偏好在延迟、带宽消耗和重建质量之间选择不同的运行点。尽管我们的评估未明确分析这些场景，但我们预计，所提出的流水线能够自然地为此类异构运行条件提供调整空间。具体而言，FugaStream 的运行时优先级调度可调整基础视图的提交时机，以及随后传输的延迟视图信息量，从而在不根本改变底层流水线的情况下实现不同的运行点。

带宽感知交付。 FugaStream 目前使用 Draco 压缩重建后的几何数据，但 Draco 对于根据带宽状况变化调整传输表示的支持十分有限。借助包括学习型或可扩展编解码器在内的更先进体积视频编解码器，我们认为系统能够进一步减小基础表示及后续延迟视图更新的大小，并适应服务器到客户端网络带宽。具体而言，提前交付的基础视图与后续延迟视图更新之间的分离，为带宽自适应提供了天然机会：在带宽受限时，系统可以优先传输基础视图，仅选择性地传输必要更新；随着可用带宽增加，再交付额外更新。我们将这种带宽感知自适应留作未来工作中一个有价值的研究方向。

8 相关工作

实时体积视频流。 实时体积视频流的端到端系统，通常可根据其在网络上传输内容的维度自然分为两类：重建后的三维内容，或客户端稍后据此重建三维表示的二维传感器数据。在三维方面，MetaStream [11]假设摄像机配备了计算资源，其服务器根据前一帧集合中二维帧的中位到达时间，自适应地为每个帧集合设置等待时间。PointPresence [3] 让服务器对传入的RGB-D帧执行目标检测，并仅将边界框内的点云转发到对其进行解码的边缘侧。二维方面则受到编解码器成熟度存在显著不对称这一事实的推动。专注于移动实时远程呈现的FarfetchFusion [25], 让发送端提取二维和三维地标并传输变换后的RGB-D数据，由接收端执行融合并恢复三维网格。在用于沉浸式体积视频会议的LiVo [7] 中，发送端从接收端获取观察者视锥体，仅传输相应的RGB和深度流，并将深度打包到Y 通道中，同时为UV 通道设置固定值，以便由视频编解码器承载。两条路径都保留了一个未改变的假设：对一个帧作出贡献的视图集合在部署时即已固定，而错过截止时间的视图不会在之后得到恢复，而是完全不作贡献。

多视图压缩。 除流传输系统架构之外，另一类研究关注这样一个限制：体积流传输中的带宽消耗会随摄像机数量增加而增长。DeltaStream [24] 将二维 RGB 和深度中的帧间变化转换为三维运动向量，然后针对前一帧压缩并传输增量点云以及这些向量。BlenDR [21]将深度打包到 Y 通道中，将颜色打包到 UV 通道中，并将低分辨率 UV 流与 Y 深度流复用后进行传输。ImmerScope [4], 专注于视图间依赖，其中每台摄像机通过基于深度神经网络的编码器传输其帧，服务器将多个视图聚合成单个帧，再由基于深度神经网络的解码器进行重建。同时，Draco [9] 以及 MPEG V-PCC/G-PCC 系列 [10],等标准点云和体素编解码器，会生成能够由通用三维压缩解码的比特流。我们系统在这类工作中的定位是：利用相同的冗余，但并非在每个视图发送前缩小其规模，而是从同一帧集合中一个提前提交的基础视图中恢复错过截止时间的视图。

9 结论

本文介绍了 FugaStream，这是一种流水线式体积流系统，可缓解视图可用性分散造成的延迟，同时不会永久舍弃延迟视图中的信息。FugaStream 不会停下来等待所有视图就绪，而是自适应地提交三维基础表示，并在延迟视图变为可用时逐步对其进行细化。其自适应基础提交控制器会权衡额外视图预期带来的几何增益与等待所付出的延迟成本；与此同时，基于基础视图的延迟视图补全机制会识别并传输基础视图中尚未表示的新颖几何信息。我们使用标准数据集和真实世界测试平台评估 FugaStream，以展示其在各种视图可用性模式下的有效性。结果表明，FugaStream 在保持高渲染质量的同时，大幅降低了端到端延迟；轻量级的延迟视图更新则保留了提前提交带来的优势。FugaStream 表明，在所有视图就绪之前先提交并传输有用的基础视图，随后再纳入延迟视图，为视图可用性不确定情况下实现低延迟体积流传输提供了一条切实可行的路径。

参考文献

[1] 苹果公司。2024年。Apple Vision Pro。访问日期：2026年8月30日。https://www. apple.com/apple-vision-pro/ [2] Nicolas Carion、Laura Gustafson、Yuan-Ting Hu、Shoubhik Debnath、Ronghang Hu、Didac Suris Coll-Vinent、Chaitanya Ryali、Kalyan Va- sudev Alwala、Haitham Khedr、Andrew Huang 等。2026年。SAM 3：利用概念进行万物分割。载于国际学习表征会议。138846–138923。[3] Eugene Chai、Kittipat Apicharttrisorn、Limin Wang、Hyunseok Chang 和 Sarit Mukherjee。2025年。PointPresence：面向多用户混合现实临场通信的在线虚拟空间。载于第*23*届移动系统、应用与服务年度国际会议论文集。277–290。[4] Bo Chen、Hongpeng Guo、Mingyuan Wu、Zhe Yang、Zhisheng Yan 和 Klara Nahrstedt。2024年。ImmerScope：面向沉浸式内容服务的边缘侧多视图视频聚合。载于第*22*届 *ACM* 嵌入式网络传感器系统会议论文集。82–96。[5] Hang Chu、Shugao Ma、Fernando De la Torre、Sanja Fidler 和 Yaser Sheikh。2020年。通过模块化编解码头像实现表现力临场通信。载于欧洲计算机视觉会议。330–345。[6] Brian Curless 和 Marc Levoy。1996年。一种利用距离图像构建复杂模型的体积方法。载于第*23*届计算机图形学与交互技术年度会议论文集。303–312。[7] Rajrup Ghosh、Christina Suyong Shin、Lei Zhang、Muyang Ye、Tao Jin、Harsha V Madhyastha、Ravi Netravali、Antonio Ortega、Sanjay Rao、Anthony Rowe 等。2025年。LiVo：面向带宽自适应全沉浸式体积视频会议。*ACM* 网络学报 3，CoNEXT4（2025），1–25。[8] 谷歌。2011年。WebRTC。访问日期：2026年8月30日。https://webrtc.org [9] 谷歌。2017。Draco：用于压缩和解压缩三维几何网格与点云的库。访问日期：2026年8月30日。https: // github.com/google/draco[10] Danillo Graziosi、Ohji Nakagami、Satoru Kuma、Alexandre Zaghetto、Teruhiko Suzuki 和 Ali Tabatabai。2020年。点云压缩标准化工作的概述：基于视频的方法（V-PCC）和基于几何的方法（基于几何的点云压缩）。*APSIPA* 信号与信息处理汇刊 9，1（2020），1–17。[11] Yongjie Guan、Xueyu Hou、Nan Wu、Bo Han 和 Tao Han。2023年。MetaStream：实时体积内容的捕获、创作、传输与渲染。载于第*29*届移动计算与网络年度国际会议论文集。425–439。[12] Anhong Guo、Ilter Canberk、Hannah Murphy、Andrés Monroy-Hernández 和 Rajan Vaish。2019年。Blocks：协作式、持久化的增强现实体验。*ACM* 交互式、移动、可穿戴与泛在技术学报 3，3（2019），1–24。[13] Bo Han、Yu Liu 和 Feng Qian。2020年。ViVo：感知可见性的移动体积视频流。载于第*26*届移动计算与网络年度国际会议论文集。1–13。[14] Stephen Hemminger。2005年。使用 NetEm 进行网络仿真。载于澳大利亚 *Linux* 大会论文集。18–23。[15] Jinhan Hu、Aashiq Shaikh、Alireza Bahremand 和 Robert LiKamWa。2021年。移动设备上实时稠密点云采集与流传输的特性分析。载于*ACM* 视频分析与智能边缘热点主题研讨会论文集。1–6。[16] Kaiyuan Hu、Yongting Chen、Kaiying Han、Boyan Li、Haowen Yang、Yili Jin、Junhua Liu 和 Fangxin Wang。2025 年。LiveVV：以人为中心的实时体积视频流系统。*IEEE* 物联网期刊 12，11（2025），17233–17243。[17] 国际电信联盟。2001年。终端用户多媒体 *QoS* 类别。ITU-T G.1010建议。国际电信联盟。[18] iproute2 项目。2001年。tc(8)：显示/操作流量控制设置。访问日期：2026年8月31日。https://linux.die.net/man/8/tc[19] Hanbyul Joo、Hao Liu、Lei Tan、Lin Gui、Bart Nabbe、Iain Matthews、Takeo Kanade、Shohei Nobuhara 和 Yaser Sheikh。2015年。Panoptic studio：用于社交动作捕捉的大规模多视图系统。载于 *IEEE* 国际计算机视觉会议论文集。3334–3342。[20] Hanseul Jun 和 Jeremy Bailenson。2020年。Temporal RVL：一种深度流压缩方法。载于 *IEEE* 虚拟现实与三维用户界面会议摘要与研讨会。664–665。[21] Jaehong Kim、Joon Ha Kim、Yunheon Lee 和 Dongsu Han。2026年。BlenDR：面向实时三维视频流的高带宽效率 RGB-D 表示与传输。载于第*24*届移动系统、应用与服务年度国际会议论文集。96–110。[22] Marek Kowalski、Jacek Naruniec 和 Michal Daniluk。2015年。LiveScan3D：用于多个 Kinect v2 传感器的快速、低成本三维数据采集系统。载于 *2015*年国际三维视觉会议。318–325。[23] Jason Lawrence、Ryan Overbeck、Todd Prives、Tommy Fortes、Nikki Roth 和 Brett Newman。2024年。Project starline：一种高保真远程临场系统。载于 *ACM SIGGRAPH 2024* 新兴技术。1–2。[24] Hojeong Lee、Yu Hong Kim、Sangwoo Ryu、James Won-Ki Hong、Sangtae Ha 和 Seyeon Kim。2025年。DeltaStream：面向实时体积视频流的二维推断增量编码。载于 第*23*届移动系统、应用与服务年度国际会议论文集。361–373。[25] Kyungjin Lee、Juheon Yi 和 Youngki Lee。2023年。FarfetchFusion：迈向全移动实时三维远程临场平台。载于第*29*届移动计算与网络年度国际会议论文集。1–15。[26] Kyungjin Lee、Juheon Yi、Youngki Lee、Sunghyun Choi 和 Young Min Kim。2020年。GROOT：一种高保真体积视频的实时流系统。载于 第*26*届移动计算与网络年度国际会议论文集。1–14。[27] Zhicheng Liang、Junhua Liu、Mallesham Dasari 和 Fangxin Wang。2024年。Fumos：连续点云视频流的神经压缩与渐进式细化。*IEEE* 计算机图形学与应用汇刊 30, 5（2024），2849–2859。[28] Yu Liu、Qiao Jin、Zejun Zhang、Bo Han、Svetlana Yarosh 和 Feng Qian。2024年。HoloClass：通过实时体积视频流增强虚拟现实课堂。载于 第*37*届 *ACM* 用户界面软件与技术年度研讨会附属论文集。1–3。[29] Meta。2023年。Meta Quest 3。访问日期：2026年8月30日。https://www.meta.com/quest/quest-3/ [30] Microsoft。2019年。Azure Kinect DK。访问日期：2026年9月3日。https://azure.microsoft.com/en-us/products/kinect-dk/[31]Richard A Newcombe、Shahram Izadi、Otmar Hilliges、David Molyneaux、David Kim、Andrew J Davison、Pushmeet Kohli、Jamie Shotton、Steve Hodges 和 Andrew Fitzgibbon。2011年。Kinectfusion：实时稠密表面映射与跟踪。载于 第*10*届 *IEEE* 混合与增强现实国际研讨会。127–136。[32] 奥比中光。2023年。Orbbec SDK 和 K4A 封装（Femto Mega）。访问日期：2026年8月30日。https://www.orbbec.com/products/tof-camera/femto/ [33] Sergio Orts-Escolano、Christoph Rhemann、Sean Fanello、Wayne Chang、Adarsh Kowdle、Yury Degtyarev、David Kim、Philip L Davidson、Sameh Khamis、Mingsong Dou 等。2016年。全息传送：实时虚拟三维传送。载于 第*29*届用户界面软件与技术年度研讨会论文集。741–754。[34] Zhou Wang、Alan C. Bovik、Hamid R. Sheikh 和 Eero P. Simoncelli。2004年。图像质量评估：从误差可见性到结构相似性。*IEEE* 图像处理汇刊 13, 4（2004），600–612。

[35] Thomas Wiegand、Gary J Sullivan、Gisle Bjontegaard 和 Ajay Luthra。2003。H.264/AVC 视频编码标准概述。*IEEE* 视频技术电路与系统汇刊 13, 7（2003），560–576。

[36] Andrew D Wilson。2017。快速无损深度图像压缩。载于 *ACM* 交互表面与空间国际会议论文集。100–105。