K-Planes

> K-Planes: Explicit Radiance Fields in Space, Time, and Appearance(CVPR 2023)

简介

[主页：https://sarafridov.github.io/K-Planes/](https://sarafridov.github.io/K-Planes/)

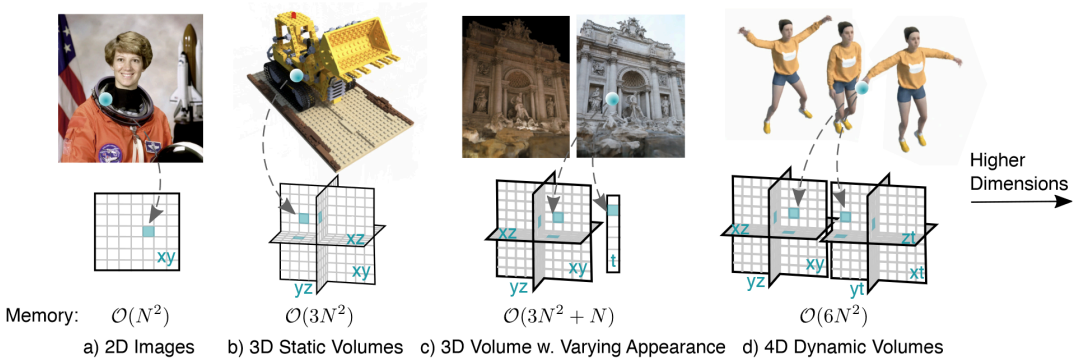

Figure 1. **Planar factorization of d-dimensional spaces.** We propose a simple planar factorization for volumetric rendering that naturally extends to arbitrary-dimensional spaces, and that scales gracefully with dimension in both optimization time and model size. We show the advantages of our approach on 3D static volumes, 3D photo collections with varying appearances, and 4D dynamic videos.

图像使用一个平面表示，静态三维场景用三个平面表示，后续动态场景用三个平面加一维时间 t 表示，论文提出使用六个平面表示动静态场景，即静态场景占三个平面，动态场景占三个平面，动态场景可以理解为三维坐标x，y，z分别与时间 t 构成平面。可以看出，平面数量与维度d之间构成关系$\binom{d}{2}$

这种平面分解使得添加特定维度的先验(例如时间平滑性和多分辨率空间结构)变得容易，并可实现场景静态和动态成分的自然分解

在一系列合成和真实，静态和动态，固定和变化的外观场景中，k-planes产生了具有竞争力的，通常是最先进的重建保真度，低内存使用，在完整的4D网格上实现1000倍压缩，并使用纯PyTorch实现快速优化，无需使用任何定制的CUDA内核

实现流程

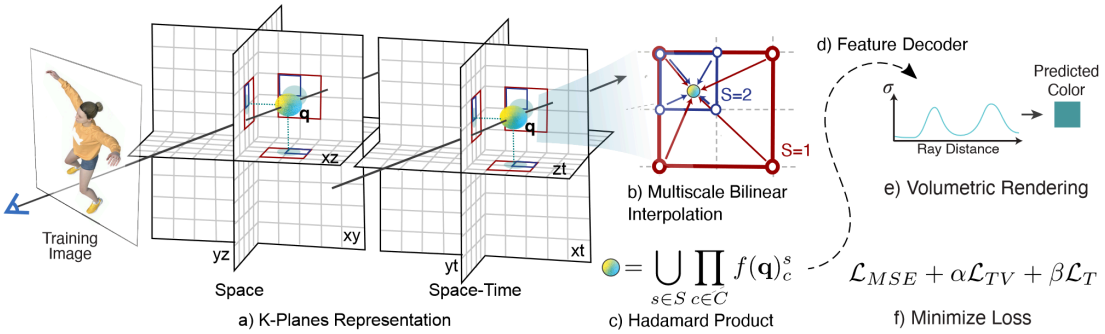

Figure 2. **Method overview.** (a) Our k-planes representation factorizes 4D dynamic volumes into six planes, three for space and three for spatiotemporal variations. To obtain the value of a 4D point $\mathbf{q} = (x, y, z, t)$, we first project the point into each plane, in which we (b) do multiscale bilinear interpolation. (c) The interpolated values are multiplied and then concatenated over the S scales. (d) These features are decoded either with a small MLP or our explicit linear decoder. (e) We follow the standard volumetric rendering formula to predict ray color and density. The model is optimized by (f) minimizing the reconstruction loss with simple regularization in space and time.

1. 将4D动态体积分解为六个平面，三个用于空间，三个用于时空变化。为了获得一个4D点 q = (x, y, z,t)的值，首先将这个点投影到每个平面上
2. 做多尺度双线性插值。
3. 将插值值**相乘**，然后在S尺度上连接。
4. 这些特征可以用一个小的MLP或显式线性解码器解码。

5. 遵循标准的体积渲染公式来预测射线的颜色和密度
6. 在空间和时间上进行简单正则化，使重构损失最小化，对模型进行优化。

Hex-planes

空间平面表示为P_{xy}, P_{xz}, P_{yz}时空平面表示为：P_{xt}, P_{zt}, P_{yt}，空间和时间分辨率为 N，平面的形状 为$N * N * M$，M 是存储的特征的大小

通过将其采样点归一化到 [0,N) 之间并将其投影到这六个平面上来获得4D坐标$q = (i, j, k, t)$的特征

$$f(\mathbf{q})_c = \psi(\mathbf{P}_c, \pi_c(\mathbf{q}))$$

π_c将 q 投影到第 c 个平面上，ψ表示点的双线性插值到规则间隔的二维网格中，重复公式一，得到特征向量$f(q)$，使用Hadamard积(元素乘法)在六个平面上组合这些特征，以产生长度为 M 的最终特征向量

$$f(\mathbf{q}) = \prod_{c \in C} f(\mathbf{q})_c$$

之后特征将使用**线性解码器**或 **MLP** 解码成 **颜色** 和 **密度**

为什么这里用乘法，而不是加法?

- 通过乘法组合平面可以使 k 平面产生空间局部化的信号，这是加法所不可能做到的
- Hadamard积(元素乘法)对**线性解码器**产生了显著的渲染改进，对**MLP解码器**产生了适度的改进
- **MLP解码器**参与了依赖于视图的颜色和确定空间结构，Hadamard积(元素乘法)减轻了**特征解码器**的这一额外任务，并使用单独负责视相关颜色的**线性解码器**达到类似的性能

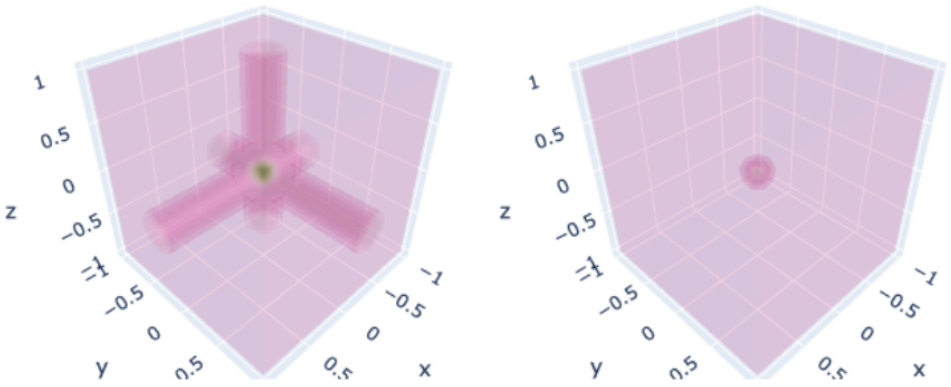

Figure 3. **Addition versus Hadamard product.** Elementwise addition of plane features (left) compared to multiplication (right), in a triplane example. A single entry in each plane is positive and the rest are zero, selecting a single 3D point by multiplication but producing intersecting lines by addition. This selection ability of multiplication improves the expressivity of our explicit model.

Plane Combination	Explicit	Hybrid	# params ↓
Multiplication	35.29	35.75	33M
Addition	28.78	34.80	33M

Table 2. **Ablation study over Hadamard product.** Multiplication of plane features yields a large improvement in PSNR ↑ for our explicit model, whereas our hybrid model can use its MLP decoder to partially compensate for the less expressive addition of planes. This experiment uses the static *Lego* scene [24] with 3 scales: 128, 256, and 512, and 32 features per scale.

Interpretability

空间平面和时空平面的分离使模型具有可解释性，能够纳入特定维度的先验，比如，如果场景的一个区域从未移动，它的时间分量将始终为 1 (乘性单位)，从而仅使用来自空间平面的特征

这提供了压缩的好处，因为静态区域可以很容易地识别和紧凑地表示，时空分离提高了可解释性，即可以通过可视化时空平面中非 1 的元素来跟踪时间的变化，这种简单性、分离性和可解释性使得添加先验很简单

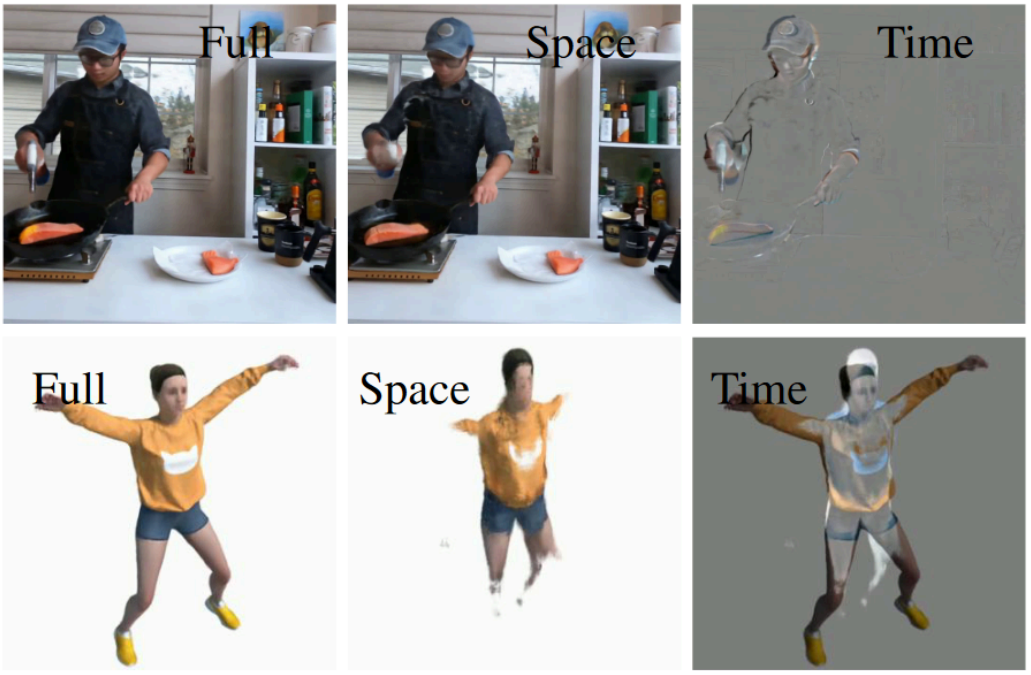

Figure 9. **Decomposition of space and time.** K-planes (left) naturally decomposes a 3D video into static and dynamic components. We render the static part (middle) by setting the time planes to the identity, and the remainder (right) is the dynamic part. Top shows the *flame salmon* multiview video [16] and bottom shows the *jumping jacks* monocular video [30].

场景的静态部分可以通过将三个时间平面设置为1(乘法恒等式)来获得。从完整渲染中减去仅静态渲染的图像(即时间平面参数未设置为1)，可以显示场景的动态部分。

Multiscale planes

为了促进空间平滑和一致性，我们的模型包含不同空间分辨率的多个副本，例如64、128、256和512

每个尺度的模型被单独处理，不同尺度的M维特征向量被连接在一起，然后传递给解码器

这种表示有效地编码了不同尺度的空间特征，能够减少存储在最高分辨率下的特征数量，从而进一步压缩模型

但是没有必要在多个尺度上表示**时间维度**

Scales (32 Feat. Each)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
64, 128, 256, 512	35.26	35.79	34M
128, 256, 512	35.29	35.75	33M
256, 512	34.52	35.37	32M
512	32.93	33.60	25M
64, 128, 256	34.26	35.07	8M

Scales (96 Feat. Total)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
64, 128, 256, 512	35.16	35.67	25M
128, 256, 512	35.29	35.75	33M
256, 512	34.50	35.16	47M
512	33.12	34.09	76M
64, 128, 256	34.26	35.07	8M

Table 4. **Ablation study over scales.** Including even a single lower scale improves performance, for both our explicit and hybrid models, even when holding the total feature dimension constant. Using lower scales only (excluding resolution 512^3) substantially reduces model size and yields quality much better than using high resolution alone, though slightly worse than including both low and high resolutions. This experiment uses the static *Lego* scene; in the top table each scale is allocated 32 features and in the bottom

Total variation in space

空间总变分正则化鼓励稀疏梯度(使用L1范数)或平滑梯度(使用L2范数)，编码空间中稀疏或平滑的边缘的先验

在每个时空平面的空间维度上的一维和在只有空间的平面上的二维中这样做

$$\mathcal{L}_{TV}(\mathbf{P}) = \frac{1}{|C|n^2} \sum_{c,i,j} \left(\|\mathbf{P}_c^{i,j} - \mathbf{P}_c^{i-1,j}\|_2^2 + \|\mathbf{P}_c^{i,j} - \mathbf{P}_c^{i,j-1}\|_2^2 \right),$$

i, j 是平面分辨率上的指标，全变分是逆问题中常用的正则化子，如 Plenoxels，TensoRF

实验发现L2或L1都能产生相似的质量，在结果中使用 L2 版本

Smoothness in time

使用一维拉普拉斯(二阶导数)滤波器来平滑运动

$$\mathcal{L}_{smooth}(\mathbf{P}) = \frac{1}{|C|n^2} \sum_{c,i,t} \|\mathbf{P}_c^{i,t-1} - 2\mathbf{P}_c^{i,t} + \mathbf{P}_c^{i,t+1}\|_2^2,$$

随着时间的推移，惩罚急剧的"加速"。只在时空平面的时间维度上应用这个正则化项

Time Smoothness Weight (λ)	Explicit PSNR ↑	Hybrid PSNR ↑
0.0	30.45	30.86
0.001	31.61	32.23
0.01	32.00	32.64
0.1	31.96	32.58
1.0	31.36	32.22
10.0	30.45	31.63

Table 6. **Ablation study over temporal smoothness regularization.** For both models, a temporal smoothness weight of 0.01 is best, with PSNR degrading gradually with over- or under-regularization. This experiment uses the *Jumping Jacks* scene with 4 scales: 64, 128, 256, and 512, and 32 features per scale.

时间平滑度权重为0.01是最好的，PSNR会随着过正则化或欠正则化而逐渐下降。本实验使用4个尺度的开合跳场景:64、128、256和512，每个尺度有32个特征。

Sparse transients

通过将时空平面中的特征初始化为 1(可乘性恒等式)，并在训练期间在这些平面上使用L_1正则化项，使得场景的静态部分由空间平面建模

$$\mathcal{L}_{sep}(\mathbf{P}) = \sum_{c} \|\mathbf{1} - \mathbf{P}_c\|_1, \qquad c \in \{xt, yt, zt\}.$$

如果对应的空间内容不随时间变化，k 平面分解的时空平面特征将固定为 1

Feature decoders

有两种方法将等式(2)中的 M 维时间和空间局部特征向量$f(q)$解码为密度 σ 和视相关颜色 c

Learned color basis: a linear decoder and explicit model

使用空间局部化特征作为球调和(SH)基的系数的模型，以描述视点依赖的颜色

与MLP解码器相比，这种SH解码器可以提供高保真重建和增强的可解释性，但是，SH系数很难优化，而且它们的表达能力受到所使用的SH基函数数量的限制(通常局限于产生模糊镜面反射的二次谐波

这里使用一个小的 MLP 来表示基，它将每个视图方向 d 映射到红色$b_R(d) \in R^M$、绿色$b_G(d) \in R^M$和蓝色$b_B(d) \in R^M$的基向量

MLP作为自适应的插入式替换，用于在三个颜色通道上重复的球面谐波基函数

$$\boldsymbol{c(q,d)} = \bigcup_{i \in \{R,G,B\}} \boldsymbol{f(q) \cdot b_i(d)},$$

·为点积,∪为拼接

与视图方向无关的学习基$b_\sigma \in R^M$作为密度的线性解码器

$$\boldsymbol{\sigma(q) = f(q) \cdot b_\sigma}.$$

对 c(q, d) 应用 sigmoid函数，对 σ(q) 应用指数函数(梯度截断)，最终将颜色和密度的预测值强制在其有效范围内

MLP decoder: a hybrid model

类似 Instant-ngp 特征由两个小mlp解码

g_σ将空间局部化特征映射为密度σ和附加特征$\hat{f}$

g_{RGB}将$\hat{f}$和嵌入的视图方向 γ(d) 映射为RGB颜色

$$\sigma(\boldsymbol{q}), \hat{\boldsymbol{f}}(\boldsymbol{q}) = g_\sigma(f(\boldsymbol{q}))$$

$$\boldsymbol{c(q,d)} = g_{RGB}(\hat{\boldsymbol{f}}(\boldsymbol{q}), \gamma(\boldsymbol{d})).$$

与线性解码器的情况一样，预测的密度值和颜色值最终分别通过指数和sigmoid进行归一化

Global appearance

为了使其能够在不同的光照或外观条件下以一致的静态几何形状表示场景(NeRF-W)，为每个训练图像 1...T 增加了一个 M 维向量的 k 平面

优化了这个每张图像的特征向量，并将其作为额外的输入传递给MLP学习的颜色基b_R、b_G、b_B，或传递给MLP颜色解码器g_{RGB}，以便它可以影响颜色而不是几何形状

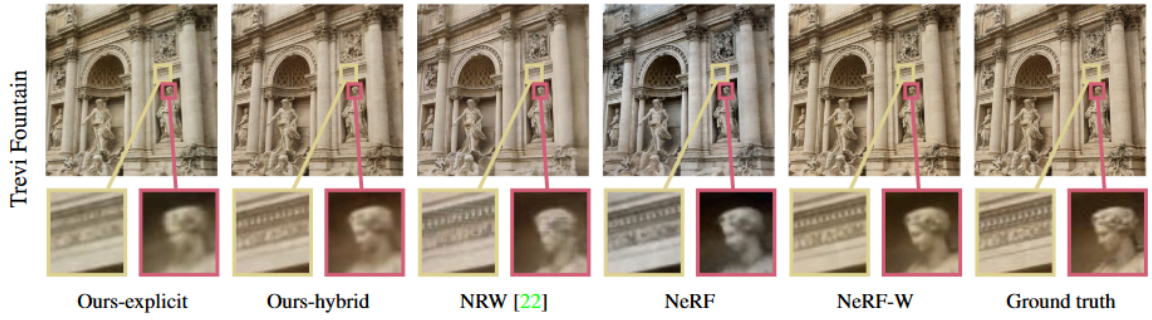

Figure 11. **Qualitative results from Phototourism dataset.** We compare our model with strong baselines. Our method captures the geometry and appearance of the scene, but produces slightly lower resolution results than NeRF-W. Note that our model optimizes in just 35 minutes on a single GPU compared to NeRF-W, which takes 2 days on 8 GPUs.

Optimization details

对于面向前方的场景，应用归一化设备坐标(NDC)来更好地分配分辨率，同时实现无限深度

实现了Mip-NeRF 360中提出的场景收缩的 ℓ∞ 版本(而不是 ℓ2)，将其用于无界摄影旅游场景

Proposal sampling

使用Mip-NeRF 360建议采样策略的变体，以 k 平面的小实例作为密度模型。建议采样通过沿一条射线迭代优化密度估计，在密度较高的区域分配更多的点。使用两级采样器，导致必须在整个模型中评估的样本更少，并通过将这些样本放置在更接近物体表面的地方，使细节更清晰。用于建议采样的密度模型使用直方图损失进行训练。

Importance sampling

针对多视角动态场景，从DyNeRF中实现了一种基于时间差分(IST)的重要性采样策略。在优化的最后一部分，根据前后25帧内颜色的最大变化比例对训练光线进行采样。这导致动态区域的采样概率更高。在静态场景与均匀采样的光线收敛后应用此策略。在实验中，IST对全帧指标只有适度的影响，但提高了小动态区域的视觉质量。重要性采样不能用于单目视频或具有移动摄像机的数据集。

实验

	PSNR ↑	SSIM ↑	Train Time ↓	# Params ↓
NeRF [24] (static, synthetic)				
Ours-explicit	32.21	0.960	38 min	33M
Ours-hybrid	32.36	0.962	38 min	33M
Plenoxels [10]	31.71	0.958	11 min	∼500M
TensoRF [6]	33.14	0.963	17 min	18M
I-NGP [25]	33.18	-	5 min	∼ 16M
LLFF [23] (static, real)				
Ours-explicit	26.78	0.841	33 min	19M
Ours-hybrid	26.92	0.847	33 min	19M
Plenoxels	26.29	0.839	24 min	∼500M
TensoRF	26.73	0.839	25 min	45M
D-NeRF [30] (dynamic, synthetic)				
Ours-explicit	31.05	0.97	52 min	37M
Ours-hybrid	31.61	0.97	52 min	37M
D-NeRF	29.67	0.95	48 hrs	1-3M
TiNeuVox [9]	32.67	0.97	30 min	∼12M
V4D [11]	33.72	0.98	4.9 hrs	275M
DyNeRF [16] (dynamic, real)				
Ours-explicit	30.88	0.960	3.7 hrs	51M
Ours-hybrid	31.63	0.964	1.8 hrs	27M
DyNeRF [16]	[1]29.58	-	1344 hrs	7M
LLFF [23]	[1]23.24	-	-	-
MixVoxels-L [37]	30.80	0.960	1.3 hrs	125M
Phototourism [15] (variable appearance)				
Ours-explicit	22.25	0.859	35 min	36M
Ours-hybrid	22.92	0.877	35 min	36M
NeRF-W [20]	27.00	0.962	384 hrs	∼2M
NeRF-W (public)[2]	19.70	0.764	164 hrs	∼2M
LearnIt [35]	19.26	-	-	-

[1] DyNeRF and LLFF only report metrics on the *flame salmon* video (the first 10 seconds); average performance may be higher as this is one of the more challenging videos. [2] Open-source version https://github.com/kwea123/nerf_pl/tree/nerfw where we re-implemented test-time optimization as for k-planes.

Table 3. **Results.** Averaged metrics over all scenes in the respective datasets. Note that Phototourism scenes use MS-SSIM (multiscale

structural similarity) instead of SSIM. K-planes timings are based on a single NVIDIA A30 GPU. Please see the appendix for per-scene results and the website for video reconstructions.

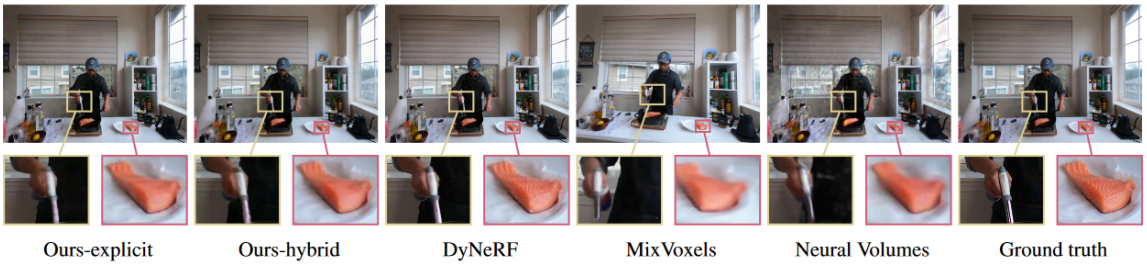

Figure 6. **Qualitative video results.** Our hexplane model rivals the rendering quality of state-of-the-art neural rendering methods. Our renderings were obtained after at most 4 hours of optimization on a single GPU whereas DyNeRF trained for a week on 8 GPUs. MixVoxels frame comes from a slightly different video rendering, and is thus slightly shifted.

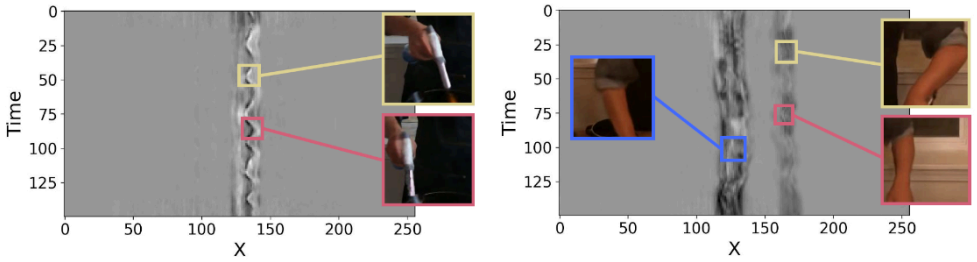

Figure 8. **Visualization of a time plane.** The xt plane highlights the dynamic regions in the scene. The wiggly patterns across time correspond to the motion of the person's hands and cooking tools, in the *flame salmon* scene (left) where only one hand moves and the *cut beef* scene (right) where both hands move.

时间平面的可视化。xt 平面突出了场景中的动态区域。在火焰鲑鱼场景(左)中，只有一只手在移动，而在切牛肉场景(右)中，两只手都在移动，随着时间的推移，摆动的模式对应着人的手和烹饪工具的运动。

Feature Length (M)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
2	30.66	32.05	2M
4	32.27	34.18	4M
8	33.80	35.12	8M
16	34.80	35.44	17M
32	35.29	35.75	33M
64	35.38	35.88	66M
128	35.45	35.99	132M

Table 5. **Ablation study over feature length M.** Increasing the feature length M learned at each scale consistently improves quality for both our models, with a corresponding linear increase in model size and optimization time. Our experiments in the main text use a mixture of $M = 16$ and $M = 32$; for specific applications it may be beneficial to vary M along this tradeoff between quality and model size. This experiment uses the static *Lego* scene with 3 scales: 128, 256, and 512.

增加在每个尺度上学习到的特征长度 M，持续提高了模型的质量，相应的模型大小和优化时间线性增加。在实验使用了M = 16和M = 32的混合;对于特定的应用，可以根据质量和模型大小之间的权衡来改变M。