Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes

Related Work

view synthesis nerf (static) { kernels btw two images; optical flow, warping features; ? layered

novel time synthesis

space-time view synthesis scene change with time
{ illumination changes: previous relighting work only assume static
{ 3D scene motion: previous require multi-view, time synchronized as input

Approach

Neural scene flow fields for dynamic scenes (NSFF, the dynamic version of nerf)

recall: $(c, \sigma) = F_\Theta(x, d)$ # nerf takes a position, a direction, outputs a rgb, a density.

now: $(c_i, \sigma_i, F_i, W_i) = F_\Theta^{dy}(x, d, i)$ # nsff*dyn takes additionally a time i, outputs additionally:

$$\begin{cases} F_i = (f_{i\to i+1}, f_{i\to i-1}) & \text{\# 3D scene flow, offset of location } x \text{ @} i-1, i+1 \\ W_i = (w_{i\to i+1}, w_{i\to i-1}) & \text{\# 1D disocclusion weights } W_i\text{, details below} \end{cases}$$

Optimization

core of nsff. to understand this paper from THE NEW LOSS

temporal photometric consistency scene @i should be consistent with scene @$j \in \mathcal{N}(i)$, if neighbor scene got from nsff.

$\hat{C}_{j\to i}(r_i) = \int_{t_n}^{t_f} T_j(t)\sigma_j(r_{i\to j}(t))c_j(r_{i\to j}(t), d_i)dt$ where $r_{i\to j}(t) = r_i(t) + f_{i\to j}(r_i(t))$ noticing t is distance (consider as a position is good) # color for r_i is from time neighbors

roughly: $L_{pho} = \sum_{r_i}\sum_{j\in\mathcal{N}(i)} \|\hat{C}_{j\to i}(r_i) - C_i(r_i)\|_2^2$ # ray-wise, all neighbors should render same as ground truth for a single pixel

However, this is not always correct for neighboring. # motion causes 3D disocclusion regions, this is ambiguity. Or to say:

> you cannot ask neighbors render exact same! what if there is occlusion/disocclusion ?

So, results from neighbors should be weighted (not always average)

Weights for each neighbor: $\hat{W}_{j\to i}(r_i) = \int_{t_n}^{t_f} T_i(t)\sigma_j(r_{i\to j}(t))\underbrace{w_{i\to j}(r_i(t))}_{[0,1]\text{. to } 1}dt$ # you should be noticing rgb is function of direction. but this weight is not

for all pixels, your render results should be weighted neighbor-consistent.

for all rays, your render points should have everywhere close-to-1 weights

the final: $\boxed{L_{pho} = \underbrace{\sum_{r_i}\sum_{j\in\mathcal{N}}}_{\text{for all pixels each ray}} \hat{W}_{j\to i}(r_i)\|\hat{C}_{j\to i}(r_i) - C_i(r_i)\|_2^2 + \beta_w \underbrace{\sum_{x_i}}_{\text{for all rays each position}} \|w_{i\to j}(x_i) - 1\|_1}$

β_w: reg weight (=0.1). $\mathcal{N}(i) = 5$. ($i\pm 2$ chained?)

$j = i$. self case, $\hat{W}_{i\to i}(r_i) = 1$, $f_{i\to i} = 0$. $\hat{C}_{i\to i}(r_i) = \hat{C}_i(r_i)$. original case.

Scene flow priors points back & forth with nsff should be consistent with each other

x_i have forward scene flow $f_{i\to j}$. gives point $x_{i\to j} = x_i + f_{i\to j}$

$x_i \underset{f_{j\to i}}{\overset{f_{i\to j}}{\rightleftarrows}} x_j \, (= x_{i\to j})$ this has to be true $f_{i\to j}(x_i) + \underline{f_{j\to i}(x_j)} = f_{i\to j}(x_i) + \underline{f_{j\to i}(x_{i\to j})} = 0$

no need to really bother j now

> you are predicting next x position with forward flow, then take the new next position. Naturally, should the new position take one step back at exactly x.

$\boxed{L_{cyc} = \sum_{x_i}\sum_{j\in i\pm 1} w_{i\to j}\|f_{i\to j}(x_i) + f_{j\to i}(x_{i\to j})\|_1}$ # for each position, this should be true always. only closest neighbors 2

data-driven priors results better consistent with other methods w.r.t. positions

geometric consistency 2D optical flow guides 3D scene flow fields

> optical flow methods give a next-time 2D position, nsff gives a 3D position but we do projection. They should be same.

2D optical flow $p_{i\to j} = p_i + u_{i\to j}$ (Given By OF methods)

3D nsff for one ray r_i. we volume render the $f_{i\to j}$ and x_i # recall how we compute depth? recall volume render seems apply everything

$$\left.\begin{aligned} \hat{F}_{i\to j}(r_i) &= \int_{t_n}^{t_f} T_i(t)\sigma_i(r_i(t))f_{i\to j}(r_i(t))dt \\ \hat{X}_i(r_i) &= \int_{t_n}^{t_f} T_i(t)\sigma_i(r_i(t))x_i(r_i(t))dt \end{aligned}\right\}$$

next world position: $\hat{X}_i(r_i) + \hat{F}_{i\to j}(r_i)$, then, do projection to frame at time j.

$\hat{p}_{i\to j}(r_i) = \pi\left(K\left(R^j(\hat{X}_i(r_i) + \hat{F}_{i\to j}(r_i)) + t^j\right)\right)$ $\begin{cases} [R^j, t^j]. & \text{extrinsic at time } j \\ K & \text{. intrinsic shared} \end{cases}$ # world position (trained from nerf/nsff) to camera position.

$\boxed{L_{geo} = \sum_{r_i}\sum_{j\in\mathcal{N}_2} \|\hat{p}_{i\to j}(r_i) - p_{i\to j}(r_i)\|_1}$ # for all pixels/each ray result, it should be same.

single-view depth monodepth prediction guides

$\boxed{L_z = \sum_{r_i} \|\hat{Z}_i^*(r_i) - Z_i^*(r_i)\|_1}$ # check the supp for real implementation here.

Integrating a static scene representation

differentiate background static and foreground dynamic then combine

Static background scene: $(c, \sigma, v) = F_\theta^{st}(x, d)$ # static/dyn balance : v (fyi, this is not a necessarily trained param)

dynamic foreground scene: $(c_i, \sigma_i, F_i, W_i) = F_\theta^{dy}(x, d, i)$ # blending weight v is actually $v = f(c, \sigma_i)$

Combined rendering equation: $\hat{C}_i^{cb}(r_i) = \int_{t_n}^{t_f} T_i^{cb}(t)\,\sigma_i^{cb}(t)\,c_i^{cb}(t)\,dt.$ a weighted render

$$\boxed{L_{cb} = \sum_{r_i} \| \hat{C}_i^{cb}(r_i) - C_i(r_i) \|_2^2}$$

this is a PURE color render, basically has a PURE nerf for background. and a nsff for foreground.
we don't render with neighbor information!

Regularisation

Spatial smoothness

points along ray should have neighboring similar scene flow

$$\boxed{L_{sp} = \sum_{x_i} \sum_{y_i \in N(x_i)} \sum_{j \in \pm 1} \exp(-\lambda \overbrace{\|x_i, y_i\|_2}^{dist}) \| f_{i\to j}(x_i) - f_{i \to j}(y_i) \|_1}$$

for each ray, considering each point

temporal smoothness

backward and forward should equal # for some reason

$$\boxed{L_{temp} = \frac{1}{2} \sum_{x_i} \| f_{i\to i+1}(x_i) + f_{i \to i-1}(x_i) \|_2^2}$$

for each point

sf minimal

encourage sf to be low # cause frames are continuous

$$\boxed{L_{min} = \sum_{x_i} \sum_{j \in \pm 1} \| f_{i \to j}(x_i) \|_1}$$

FINALLY

$$\boxed{L = L_{cb} + L_{pho} + \beta' L_{cyc} + \beta'' L_{data} + \beta''' L_{reg}}$$

where $\begin{cases} L_{data} = L_{geo} + \beta_z L_z \\ L_{reg} = L_{sp} + L_{temp} + L_{min} \end{cases}$

rendering

the splatting-based plane-sweep volume rendering approach # render with changing time. not view

to render intermediate time $i + \delta_i$, $\delta_i \in (0,1)$, sweep over every step t (a location along the ray)

for one point $\begin{cases} \text{query } i \text{ and get: } (c_i, \sigma_i)(F_i) \\ \text{query } i{+}1 \text{ and get: } (c_{i+1}, \sigma_{i+1})(F_{i+1}) \end{cases}$

at time $i + \delta_i$, points will be NEW and BUFFERED.

(c, σ) given by queryed results already. $\begin{cases} x_i + \delta_i f_{i \to i+1}(x_i) \\ x_{i+1} + (1-\delta_i) f_{i+1 \to i} \end{cases}$
position given by new as,

two parts will be linearly blended. # for a buffer

How to understand this algo?
basically we don't know the color @ $i+\delta_i$ and how color flows, so we query @ i and @ $i+1$. the result includes (c, σ) and (f, w). so now, f is indicating where these points in nsff are going to. for time $i + \delta_i$, the querying results come from a BUFFER. this BUFFER has the points warped by f from time i and $i+1$, along with their original queryed (c, σ). By this way, we don't query directly from nsff for (c, σ) at intermediate time. we only query a correct 'NERF' result and correct 'NSFF' flow to get what we want finally

Implementation Details

refer to supp.