Dynamatte: A dynamic matting method to generate in scene video fusion

Christopher Klammer
cklammer@andrew.cmu.edu

Achleshwar Luthra
achleshl@andrew.cmu.edu

Richa Mishra
richmis@andrew.cmu.edu

April 17, 2023

1 Motivation

Imagine standing on a sandy beach, the wind whipping through your hair as you gaze out at the horizon. The sun, a giant ball of fire, seems to kiss the ocean as it sinks lower and lower, casting a warm glow across everything it touches. You realize that this moment is too perfect not to capture. and you begin to record the frothy waves lapping at the shore, the gulls circling overhead, and the distant hum of laughter and conversation from the people around you.

As you pan the camera to the horizon, you wish there was something even more to accentuate the colors in the sky. Maybe a rainbow kite, soaring high above the waves, and then, just beyond it, you imagine spotting a surfer riding a wave, his silhouette outlined against the fiery sky. You think of the lost potential − the combination of the stunning sunset, the playful kite, and the daring surfer would have been nothing short of magical. Watching the footage back, you feel awe at the world's beauty but dissatisfied with the limitations of current technology to capture vibrant moments.

In video editing, much care and attention must be dedicated towards creating seamless and temporally consistent video with synthetically inserted objects or elements. These efforts consume an obscene amount of time to execute and perfect in post production. However, what if that burden could be ameliorated with a method that could add dynamic objects with photorealistic qualities into a live scene?

Overall, the impact this project could have on video editing cannot be understated. The benefits include a streamlined VFX process to obtain clean foreground and background mattes. Furthermore, scene complications such as dust, reflections, and shadows are challenging cases that our neural network will be able to handle in seconds instead of hours.

2 Prior Work

Many works in the past have looked at the task of matte creation. Lu et al.[**lu2021**] propose Omnimatte, a method to associate pixels in the screen with masked objects in a given scene. This method allows complex effects such as shadows, reflections, smoke, and water ripples to appear in the Omnimatte. This method takes videos as input to train a 2D U-Net that processes the video frame by frame. Each object is processed through a different layer in the model with dense optical flow fields calculated to maintain temporal consistency in the RGBA output.

Sengupta et al.[**BMSengupta20**] capture image and background pairs to estimate an alpha matte and foreground in order to place humans in scenes with novel backgrounds. It uses context switching to combine different cues and a self supervised adversarial loss to take a blended image with the novel background, generating realistic images. This method requires an additional photo of the background without the subject at the time of capture which limits us from applying this method to a random image where there is unavailability of subject-free background and it also demands for a certain amount of foresight during the process of data collection.

Our project idea aims to overcome this limitation[**BMSengupta20**] by taking inspiration from[**lu2021**] that needs a video, segmentation masks, and dense optical flow for objects of interest. This method is capable of estimating omnimattes − an alpha matte and color image that includes the subject along with all its time-varying scene elements.

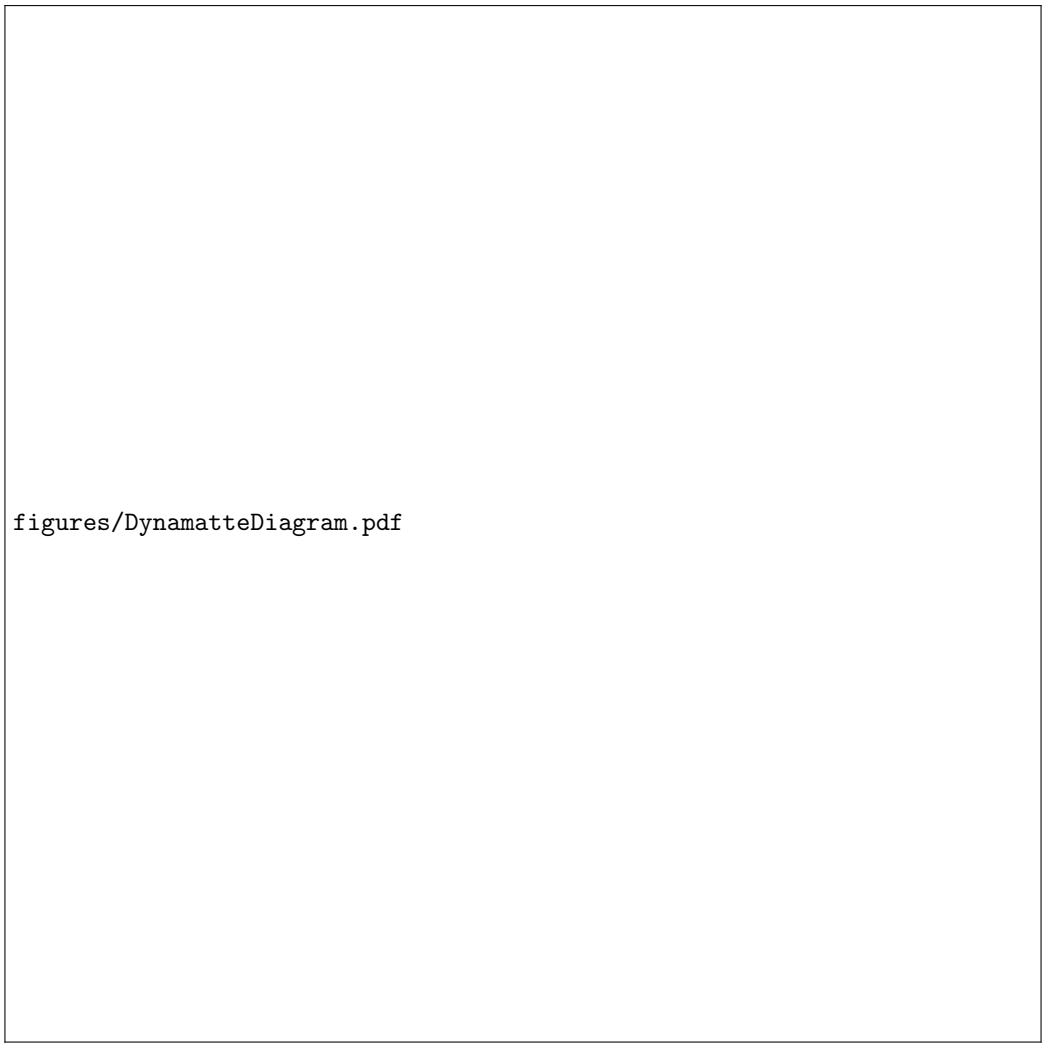

Figure 1: Network Architecture for Dynamatte

3 The Idea

3.1 Using Omnimattes for Video-in-Video Insertion

As stated before, Omnimatte ingests an input mask for a given object, optical flow, homographies, and returns an omnimatte which contains color and opacity along with an optical flow field. Our solution aims to use omnimattes extracted from a video sequence and inserts the omnimatte seamlessly into a different video sequence. More formally, for a given video sequence $\mathcal{V}$, we process the video sequence and extract out omnimattes for object i at each time t, homography H, mask M and flow F as $O(I_t, H_t, M_t^i, F_t^i) = \{\alpha_t^i, C_t^i, \hat{F_t^i}\}$ where α_t^i represents the opacity for the object at a given frame, C_t^i represents the color, and $\hat{F_t^i}$ is the object's flow.

3.2 Method

We plan to take the result of omnimatte and use back-to-front compositing to insert them into another existing video. By doing this, we plan to use existing videos and allow dynamic objects to be added to the videos. The main challenge of this work will be to perform domain transfer and conform to noise and background objects.

Our contribution will look to have two stages. Stage one will include extracting the omnimatte(s) from a video sequence. Stage two will include taking the omnimatte and inserting it into sequence with our network. Our network will have to make sure the omnimatte is temporally consistent, is blended with the background, is inserted into a logical location, and accounts for style transfer for the target sequence.

3.2.1 Insertion Gradient Loss

Laplacian image compositing is a popular method for blending two images. This method enforces gradient smoothness at the border of the insertion region. In order to enforce this, we first plan to take the omnimatte mask as an input and enforce this gradient loss at the border of the inserted object.

3.2.2 Content Loss

We also plan to enforce a content loss to ensure that the inserted object is not being overly distorted or masked after inserting into the image. In order to do this, we find the LPIPS [**zhang2018unreasonable**] loss between the masked inserted object in the blended image and the masked omnimatte.

3.2.3 Style Loss

The style of an image usually refers to the texture and other low level features of an image. We plan to use the intermediate layer outputs from a VGG-19 encoder for both the target image and the composite image and compute the similarity between the low level features.

We compute the **gram matrix** for all C feature vectors of the feature maps for the equence with the inserted object and the original target sequence. We can then calculate the style loss as a mean squared error of the object and target's gram matricies.

Most of the time, the size of the object will not equal the size of the target scene. However, we instead choose to look at the gram matrix of the image before and after insertion of the target in order to measure the information gain (or loss) of our synthetic insertion. To clarify notation, we refer to G^{obj} as the gram matrix after the object is inserted and G^{tgt} to signify the gram matrix of the target image before the object is inserted.

$$G_{i,j} = F_i \cdot F_j$$

$$\mathcal{L}_{style} = \frac{1}{N_{obj} N_{tgt}} \sum_i \sum_j (G^{tgt}_{i,j} - G^{obj}_{i,j})^2$$

Where F is a list of flattened feature vectors for all C feature maps of the backbone output. N_{obj} and N_{tgt} are the number of entries in the gram matrix of the object and target respectively.

3.2.4 Temporal Consistency Loss

A novel contribution and main challenge of this work is to maintain temporal consistency. Intuitively, when inserting an object into a video, we want to see the following:

1. Cohesive and smooth motion frame to frame
2. Possible occlusion if passing by another object in the scene
3. Disappearance of the object when at the edge of the frame

4 Experiments and Timeline

4.1 Data Preprocessing

Setting up the baseline for extracting omnimatte involved multi-step preprocessing of the input videos. Along with RGB frames, the method [**lu2021**] takes segmentation masks (corresponding to the subject of interest), optical flow, and homography between frames as input. We have setup a small piece of software that takes in a normal RGB video and yeilds all the necessary inputs. **??** shows all the modalities that we need to pass in to our network.

We have used Segment Anything Model (SAM) [**kirillov2023segany**] to generate binary masks. SAM takes input points as prompts and gives corresponding masks. Currently, there is no model for videos so we have assumed our foreground to have a constant velocity for 50 frames. We chose this approach over fine-tuning as we wanted to segment any foreground of our choice and not just a particular category. For optical flow extraction, we have used RAFT [**teed2020raft**] provided by

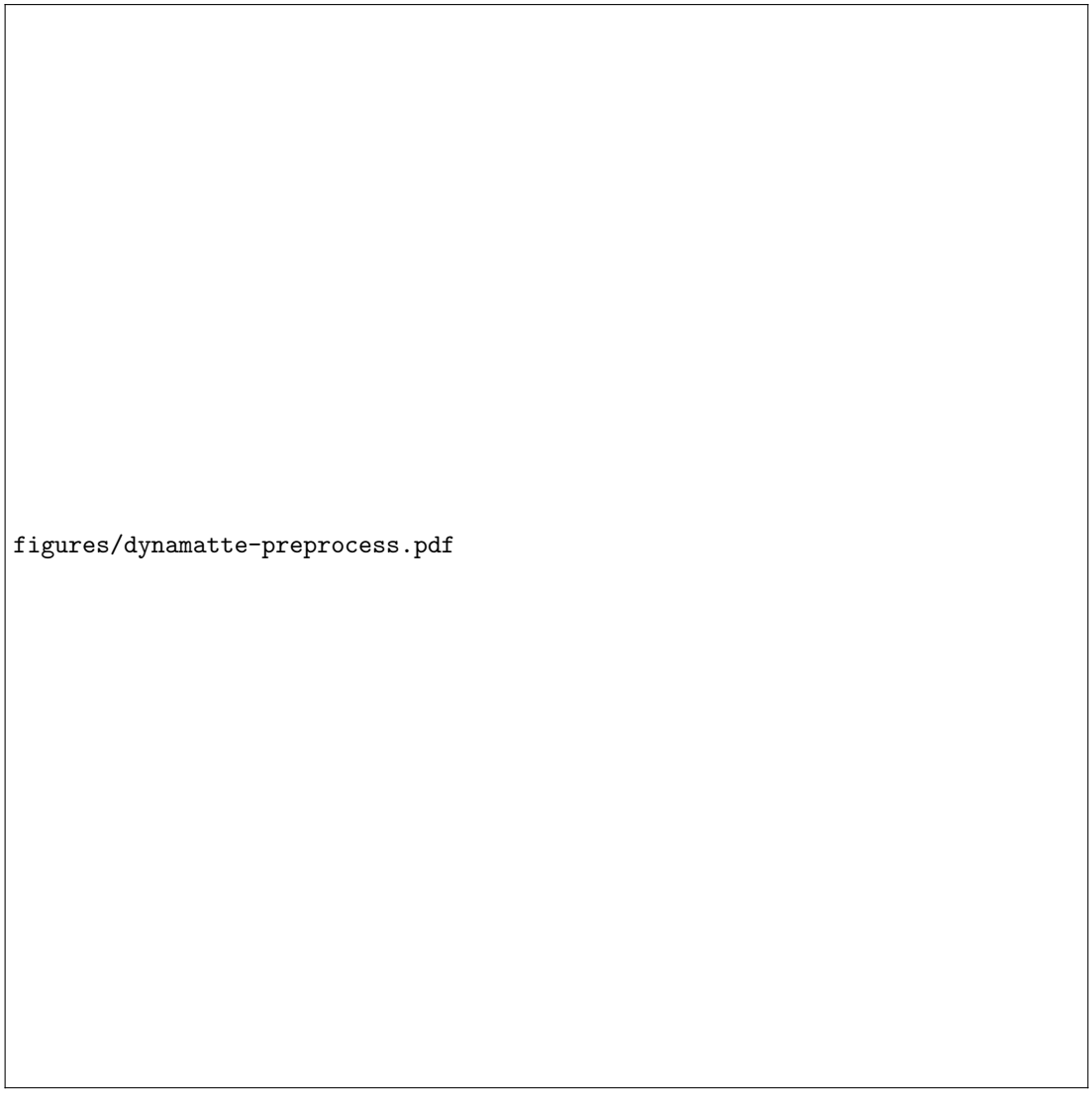

Figure 2: **Different Preprocessing Stages**

torchvision pretrained on Flying Chairs [**fischer2015flownet**] and FlyingThings3D [**Mayer˙2016**]. The confidence maps and homography matrices are estimated using the Omnimatte source code [1].

Status	Date	Milestone
✓	March	Start development of baselines
🔧	Early April	Baseline results finished, mid-term report write-up
✗	Mid April	Start implementation of proposed solution, progress on proposed solution, some initial results with issues documented
✗	End April	Finished attempt of proposed solution with qualitative results, lessons learned, and limitations
✗	Early May	Written final report

5 Baselines

As a baseline, we plan to manually extract omnimattes and attempt to use an extremely simple "lazy" cut and paste as our first baseline. This baseline will not be temporally consistent but will place the omnimattes in a "logical" part in the image. We then will perform frame-by-frame poisson image blending to create a blended video. We then will either use the famous GP-GAN[**10.1145/3343031.3350944**] or Background Matting [**BMSengupta20**] to create the blended video that will be more temporally consistent.

Now that the data preprocessing and infrastructure tasks have been finished, we plan to now move our focus towards running our proposed method for custom videos.

[1] https://github.com/erikalu/omnimatte