Converting Images to Hand Drawn Sketches Using Novel Orthogonal Gaussian Lattice Method

Anish Sachdeva
Department of Applied Mathematics
Delhi Technological University
Delhi, India
anish_bt2k16@dtu.ac.in

Rajiv Kapoor
Department of Electronics and Communication
Delhi Technological University
Delhi, India
rajivkapoor@dce.ac.in

***Abstract*—Converting Images into styles that can be artistically defined by humans and perceived by humans to have artistic value is a formidable task for a machine to perform. As there is no standard metric for measuring art, hence creating a function or constraint for a computer to optimize becomes difficult. In this paper we introduce a method of extracting information from images in a manner which can be then used as the backbone for many different Image Processing and Video Processing Tasks. In this paper we represent the power of our method by making the computer draw sketches of images without any constraint function to optimize.**

***Index Terms*—Computer Graphics, Computer Vision, Image Processing, Data Extraction**

I. Introduction

The problem that selected here is not something that is highly critical in nature or something that concerns our well being very much. It is also not something whose advancement will be very beneficial to us in terms of observable metrics. That is because the problem selected is that of creating a sketch composite from any given image I, such that the composite looks as if it was drawn by a sketch artist and not one created by a computer.

This problem is artistic in nature and hence can't be evaluated using a standard metric and the results given below have been produced using Hyperparameters tuned by the authors (so that the results look akin to something of our liking) and these hyperparameters can be changed and the results can be made to look to something other people's liking easily.

The key thing here are not the hyperparameters or results, but the method. This method gives excellent results not only for this application but is a completely new method of feature extraction from Images and can be used in several Applications. Some other examples are given in *Future Scope*. Here a simple example of sketching has been selected to display the power of the Orthogonal Feature Extractor.

There are several ways of making a sketch from an Image. Below are examples of 3 different methods to produce a Sketch Composite from a given Image I. We study the different methods below and introduce the Orthogonal Gaussian Lattice Method and discuss trade-offs and benefits.

We observe in Figure 1 that the basic Canny edge Detection Algorithm although correctly identifies edges, doesn't identify different gradients in different regions of the image and the results are too basic to be declared sufficient for a sketch composite.

We observe from Figure 2 given below that the texture based method looks very much like the image itself and not at all like a hand drawn sketch. Furthermore the texture based method has a disadvantage that it changes the original image as it needs to clip away the image to convert it into the same aspect ratio as the texture. The Gaussian Blur Blend method in (c) gives much better results and doesn't need to change the aspect ratio of the method, but the Gaussian Lattice Method in (d) gives considerable better results overall and as this is art, to the authors looks the best of the bunch.

II. Basic Canny Edge Detection

Canny Edge Detection as the name suggests is an algorithm to detect edges and is a very popular and robust algorithm used heavily in Computer Vision applications. It was developed by John F. Canny in 1986. [2]. Canny Edge Detection gives us the edges by computing the Gradient.

Since all edge detection results are easily affected by the noise in the image, it is essential to filter out the noise to prevent false detection caused by it. To smooth the image, a Gaussian filter kernel is convolved with the image.

A. Gaussian Filtering

This step will slightly smooth the image to reduce the effects of obvious noise on the edge detector. The equation for a Gaussian filter kernel of size (2k+1)×(2k+1) is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2} \right)$$

Fig. 1. Results after applying the different techniques

Fig. 2. Results after applying the novel method

Here is an example of a 5×5 Gaussian filter, used to create the adjacent image, with $\sigma = 1$. (The asterisk denotes a convolution operation).

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A$$

It is important to understand that the selection of the size of the Gaussian kernel will affect the performance of the detector. The larger the size is, the lower the detector's sensitivity to noise. Additionally, the localization error to detect the edge will slightly increase with the increase of the Gaussian filter kernel size. A 5×5 is a good size for most cases, but this will also vary depending on specific situations.

B. Finding The Intensity Gradient of The Image

An edge in an image may point in a variety of directions, so the Canny algorithm uses four filters to detect horizontal, vertical and diagonal edges in the blurred image. The edge detection operator (such as Roberts, Prewitt, or Sobel) returns a value for the first derivative in the horizontal direction (G_x) and the vertical direction (G_y). From this the edge gradient and direction can be determined:

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\Theta = \arctan\frac{G_y}{G_x}$$

C. The Algorithm (Canny Edge Detection Algorithm for a Given Image I)

The algorithm for performing canny edge Detection is given below. On my machine Intel Core i7-9750H CPU @ 2.6 GHz with 16GB RAM and a Nvidia GeForce GTX 1650 GPU with 4GB of VRAM the following operation performed in under 400 ms.

$kernel \leftarrow$ 5x5 Gaussian Kernel
$I \leftarrow I * kernel$
Create Sobel Kernel to calculate Image Derivatives I_x and I_y
$k_x \leftarrow \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
$k_y \leftarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$
Compute the Derivatives I_x and I_y using k_x and k_y
$I_x \leftarrow I * k_x$
$I_y \leftarrow I * k_y$
Compute Magnitude G and angle Θ as
$G = \sqrt{I_x^2 + I_y^2}$
$\Theta = \arctan\frac{I_y}{I_x}$
We now perform Non Maximum Suppression
for all pixels p in I **do**
  for all permitted angles θ in $0°$, $45°$, $90°$, $135°$, $180°$, $225°$, $270°$, $315°$, $360°$ **do**
    Threshold all pixels with this gradient angle to 255
  end for
end for
$result \leftarrow I$
return $result$

Fig. 3. Sketch Composite Using Canny Edge Detection

D. Results

III. Fixed Texture Application For Creating Sketch Composite

A. Method

In this method, we simply take a pre-defined texture file which is basically a mask that contains the pixel intensity values that should exist given an image I. Here the result produced will depend on the mask value of this texture file rather than the image I.

B. Algorithm (Applying Texture Mask on Image I)

Require: The Image I
Require: The mask M
Reshape the Image I to match the ratio of M,
crop the image if you must
$I \leftarrow$ Grayscale of I
$result \leftarrow I \cdot M$
return $result$

Fig. 4. Sketch Composite Using Simple Texture Application

C. Conclusion

We can clearly see from the results that the result doesn't depend on any give Image but on the texture that we have selected. This is a computationally good method as it requires negligible time in applying a single texture filter, but the results don't seem very convincing and it doesn't seem as if it were drawn by a human sketch artist.

Another disadvantage of using this method is that the texture may not be the same aspect ratio as the given image and hence we either need to stretch out the texture to fit the image I or we need to crop the image I to fit to the texture. Changing the aspect ratio of the texture doesn't yield good results, hence we always need to change the aspect ratio of the Image as can be seen above in the results. More examples of this can be seen at [3].

The results for some photos such as the Landscape Photograph in Figure 8 seem very good, but when we compare the same method for a portrait image such as Figure 7, we do not get good results and the result seems like a Sepia Photograph rather than a hand drawn sketch, so this method has a hit and miss approach to this problem.

IV. Gaussian Blur Blend Technique For Creating Sketch Composite

A. Method

This is a relatively straightforward method that borrows some techniques from the Canny Edge Detection Method, but rather than applying a Non Maximum Suppression in the Canny edge Detection, we will use a Dodge and Burn Blend Technique with our Gaussian Blur to obtain a sketch effect. [1]

The steps are as follows,

1) Take the Image I.
2) Take the Grayscale [6] of this Image I.
3) Take the Negative of the grayscale obtained in step 2.
4) Apply a Gaussian Blur [4] to the Negative Obtained in Step 3.
5) Blend the grayscale image from step 2 with the blurred negative from step 4 using a color dodge [5].

B. Gaussian Blur

The Gaussian blur is a type of image-blurring filters that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The formula of a Gaussian function in one dimension is

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

In two dimensions, it is the product of two such Gaussian functions, one in each dimension:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution. When applied in two dimensions, this formula produces a surface whose contours are concentric circles with a Gaussian distribution from the center point.

The time complexity of the Gaussian Blur Operation is $\mathcal{O}(w_{kernel}w_{image}h_{image}) + \mathcal{O}(h_{kernel}w_{image}h_{image})$. Gaussian Blur is a low pass filter attenuating high frequency signals.

C. Color Dodge and Burn

Dodging and burning are terms used in photography for a technique used during the printing process to manipulate the exposure of a selected area(s) on a photographic print, deviating from the rest of the image's exposure. In a darkroom print from a film negative, dodging decreases the exposure for areas of the print that the photographer wishes to be lighter, while burning increases the exposure to areas of the print that should be darker.

Any material with varying degrees of opacity may be used, as preferred, to cover and/or obscure the desired area for burning or dodging. One may use a transparency with text, designs, patterns, a stencil, or a completely opaque material shaped according to the desired area of burning/dodging.

In both color doge and color burn we receive an Image I and a mask M and we use division operators in images to obtain the dodge or burn using the mask we have. In the case of color doge the image is brightened at the regions specified by the Mask.

In the following example we apply color dodge to standard Lenna Image using different Masks. We take same pixel value masks, where each pixel p in the mask has the same value and the mask has dimensions same as the image.

$$M = \begin{bmatrix} p & p & p & \cdots & p \\ p & p & p & \cdots & p \\ \cdots & & & & \\ \cdots & & & & \\ p & p & p & \cdots & p \end{bmatrix}$$

D. Algorithm (Creating the Sketch Composite from I using Gaussian Blur and Blend Method)

Require: The Image I

Require: *Sketch Density*: A Hyperparameter which will be used in The Gaussian Blurring Kernel Size and will decide the number of sketch lines to appear in the final result.

We create an Kernel of odd size, for convolution

Kernel Size $\leftarrow 2 * (\textit{Sketch Density}, \textit{Sketch Density}) + 1$

$J \leftarrow$ Grayscale of Image I

$B \leftarrow$ Gaussian Blur of J with kernel of size $k =$ Kernel Size

We divide the Grayscale image with the Gaussian Blur of the Grayscale Image. The following is a pixel by pixel level operation which can be parallelized using a standard numerical matrix package.

$result \leftarrow J/B$

return $result$

V. Novel Method Using Orthogonal Gaussian Lattice

There are several regions in an Image, initial observation gives us the major features such as cars, people, the road, streets, bicycles, the footpath etc. If we observe closely we can identify softer features such as the lines on a person's face, or the strands of gray hair on the hair of a person. There are many different such features present in an Image and the following method is proposed.

Fig. 5. Sketch Composite Using Gaussian Blend Method

1) We Take 3 different Gaussian with different mean μ and std. deviation σ.
2) We compute the Grayscale of the Image I, therefore reducing the Image pixel value from 3-dimensional to 1 dimensional.
3) We normalize the pixel values in our grayscale Image.
4) We compute 3 Gaussian Inverses using the 3 different Gaussian we took initially from the grayscale Image.
5) Using the Gaussian Inverses we computed, we now take a sliding window of size w and compute deviation spread Vectors (explained later) between a central and surrounding pixel.
6) We take 3 different bounds, denoted in this project by α for the 3 different Gaussian.
7) We compute 3 Simple Graphs from the 3 Gaussian Inverse using the deviation spread vectors and connectivity parameters α we took in step 6.
8) We compute the Different Components in the 3 different Simple Graphs that we calculated in Step 7 and the separate components in a single Frame (Simple Graph) is called a Lattice.
9) We can vary the type of lattices we create, the density of lattices and change the different features we discover by changing our 3 initial Gaussian that we select and also changing the connectivity bound parameter α.

A. Grayscale of Image I

An Image I is defined as a function of 2 space coordinates x and y if we assume the image to be a discrete function with 2 axes. This can be denoted as $I = f(x, y)$ and each pixel is denoted by 3 distinct values denoting the amount of red, green and blue in the pixel. So each pixel can be denoted by a 3 dimensional vector (r, g, b). When we convert a RGB Image into grayscale, for every pixel we assign a single gray value between 0 and 255.

We compute Gray Value using the following method [6]:

$$\text{Grayscale Image}(x,y) = 0.2126 \cdot R_{linear} + 0.7152 \cdot G_{linear} + 0.0722 \cdot B_{linear}$$

B. Gaussian Inverse

A standard Gaussian is defined as

$$G(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

where μ is the mean and σ the standard deviation of the curve. The Gaussian Inverse which computes x given y is hence defined as (we are taking the positive x branch in the formula below).

$$G^{-1}(y;\mu,\sigma) = \sigma * \sqrt{-2 * \log(y\sigma\sqrt{2\pi})} + \mu$$

If we are given an image which is represented by a 2-dimensional Matrix I, we can compute the Gaussian Inverse of the image, which implies we calculate Gaussian Inverse of all pixel values separately. This can be parallelized using numerical Matrix library. The maximum value of the Gaussian Curve is at $x = \mu$, where $G(x = \mu;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}$.

If we take a Gaussian with high standard deviation than that would imply that not all values would have an inverse and any value larger than $\frac{1}{\sigma\sqrt{2\pi}}$ would not have an inverse, so one of our 3 Gaussian should have $\sigma = \frac{1}{\sqrt{2\pi}}$ so that for this Gaussian inverse exists for all $y \in [0,1]$.

This formula is defined as follows for the computer, as we need to deal with imaginary quantities.

$$G^{-1}(y;\mu,\sigma) = \mu + \begin{cases} 0 & y > \frac{1}{\sigma\sqrt{2\pi}} \\ \sigma * \sqrt{-2 * \log(y\sigma\sqrt{2\pi})} & \text{otherwise} \end{cases}$$

Let us take 3 Gaussian as

$$G_1 \leftarrow (\epsilon, \frac{4}{\sqrt{2\pi}})$$
$$G_1 \leftarrow (\epsilon, \frac{2}{\sqrt{2\pi}})$$
$$G_1 \leftarrow (\epsilon, \frac{1}{\sqrt{2\pi}})$$

where ϵ is a very small quantity, 10^{-5} in my program. The purpose of such a small quantity is for deviation spread ratio smoothing, which is explained later on.

We now compute the Gaussian Inverse of Grayscale Images, for the first 2 Gaussian G_1 and G_2, not all values will have an inverse and for such values the pixel values will be represented as 0, for others we will see valid values. So, the Gaussian Inverse will act as a mask for weeding out undesired high pixel values and will also give us a non-polynomial smooth curve for feature extraction.

Pixel values exist in the range $p \in (0,1,2,\cdots,255)$ (8-bit) where $p \in \mathbb{N}$. If we take a very high mean μ, then our inverse function will give many values over 1, resulting in them being thresholded down to 255 and hence having a mean larger than $\mu > 0$ serves no purpose.

C. Deviation Spread Ratio Vector

We will now take a window of size w ($w = 3$ in our program implementation) and will convolute the Image I with this window. In our window we will have a central pixel p_x and surrounding pixels (8 in our case), which we denote by p_s.

In every such window we will compute the ratio of surrounding pixels and the central pixel for the 8 possible surrounding pixel and central pixel pair. For 2 pixels in the window we will get 3 such ratios for the different Gaussian. Let us assume we have the following values for a 3×3 window of the 3^d Gaussian.

$$W = \begin{bmatrix} 0.43 & 0.76 & 0.12 \\ 0.18 & 0.90 & 0.44 \\ 0.28 & 0.91 & 0.93 \end{bmatrix}$$

Taking the central pixel value as 0.90, we get the 8 ratios as $\frac{p_s}{p_c}$ as:

$$\begin{bmatrix} 0.47 & 0.84 & 0.13 \\ 0.19 & - & 0.48 \\ 0.31 & 1.01 & 1.03 \end{bmatrix}$$

for every pixel pair (p_s, p_c) we will get 3 such ratios for the 3 different Gaussian, which is what we will call the deviation ratio vector. These deviation ratios tell us different things. They are in fact the ratio of the z-score of the given pixel inverse values. In the Gaussian with smaller standard deviation σ will have a lower ratio for the same amount of change in pixel values as compared to the Gaussian with larger σ, where even a small change in values will trigger a large change in the deviation spread Ratio. So, the Gaussian with a larger σ is being used to extract subtle features whereas the Gaussian with a smaller σ is being used to extract regions of the image where large changes will trigger a change in Ratio.

The 3 Different Gaussian are hence acting as an orthogonal Feature extraction mechanism where from the same pixel pair we are getting different ratios hence using the siding window technique we will get a deviation spread ratio vector for every adjacent pixel pair in the image.

We will now use these deviation spreads to create 3 different Simple Graphs for the 3 Gaussian using the connectivity Parameters $< \alpha_i > i \in \{1,2,3\}$. Here $\alpha_i \in (0,1]$.

D. Lattices in Images

We will consider every pixel in our image as a vertex, we will then compare the deviation spread ratio for very adjacent pixel with the connectivity parameter α for that particular Gaussian and if our ratio lies in the range $(\alpha, 1/\alpha)$ then we will add an edge between these vertices. We will hence create 3 Simple Graphs for the 3 Gaussian and by varying the connectivity parameters $< \alpha_i >$, we can make our graph more or less connected and change the density of connected components. [16] In the first 2 Gaussian where not every pixel will have an inverse, if you recall we had chosen a small value ϵ for such pixels and this ϵ value will ensure that when we compute the ratio between any such pixels we do not encounter a Null value [17], but rather a number the computer can store. This step is what we call **Epsilon Smoothing** and prevents underflow, overflow and Null values in actual program.

In our application and we will be proceeding by calling a single component in our Graph as a Lattice. [18] Mathematically a Lattice is a connected, unweighted simple graph. [19] Vertices inside a lattice actually represent a single pixel and have maximum degree 8 and minimum degree 0. The largest component possible can be the size of the image and the smallest possible will be a trivial Lattice of size 1 pixel.

We can visualize different Lattices in Images by coloring each Lattice [20] a Different Color and hence visualizing our Lattices. A tighter bound i.e $\alpha \to 1$ will produce many disconnected Lattices whereas a lenient bound $\alpha << 1$ will create a small number of large connected Lattices. Given below are a few examples: ***(The colors are chosen at random for a lattice and have no significance)*** [**?**]

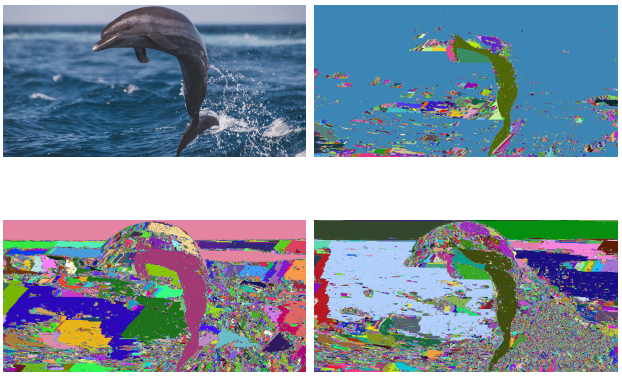

Fig. 6. Lattices in Different Simple Graphs with the connectivity parameter $< \alpha >= (0.86, 0.94, 0.98)$

E. Lattice Vertex Coloring

The above method to visualize the Lattices in the Images is a good method, but we have another method which will be the foundation for our sketching composite. The major problem we were facing with previous methods was there was no way to identify strong edges from lighter edges, and hence we could not make light and strong strokes as a real Human Sketch Artist will make. This issue can be resolved using the degree of pixels (vertices) in our Simple Graph. [21]

We know that every pixel must have at most a degree of 8 and in the trivial Lattice case, a degree of 0. This gives us 9 different degree values a pixel can have. Pixels that are heavily connected will lie deep inside a Lattice and Pixels that are at the border of the Lattice can at maximum be connected to 7 other pixels and would normally be connected to even less, normally 3-4. And completely isolated lattices will have a degree of 0.

We can assign different colors to pixels on the basis of their degree, where strongly connected pixels (degree 8) will have a white color and as the degree decreases, colors move towards Gray in the color spectrum. This will automatically give stronger edges a bolder color.

Fig. 7. Vertex Shading with Gaussian Parameters $G_1 = (\epsilon, \frac{4}{\sqrt{2\pi}}), G_1 = (\epsilon, \frac{1.3}{\sqrt{2\pi}}), G_1 = (\epsilon, \frac{1}{\sqrt{2\pi}})$ and $\alpha = (0.86, 0.9, 0.94)$

F. Linear Fusion of Orthogonal Gaussian Lattice Results with Standard Gaussian Blend

We have seen in the results above that we are obtaining excellent shadings of different regions of the Image using our Vertex Shading Method of the different Gaussian Lattices, but we want only one image where all these different features our prominent. The simplest way to achieve this is taking a simple weighted average of our 3 Gaussian Vertex Shaded results and the Gaussian Blend Image Created in the above section (which is current State of The Art Method). To take a weighted average we assume a Hyperparameter $weight, W = (w_1, w_2, w_3, w_{gb})$ where $< w_i > \forall i \in \{1, 2, 3\}$ are for the 3 Gaussian and w_{gb} is for the Gaussian Blend Method. Also,

$\sum_{w^{(i)} \in W} w^{(i)} = 1$.
[22] [23]

The weight w_3 is most important as the 3^d Gaussian captures inverses of all Pixel values and acts as the base for the entire image. This weight w_3 is hence being treated as a hyper-parameter **Hand Drawn** which implies to the user how much hand drawn feel [24] the user would like. High values like $[0.7, 1)$ would make it feel very sketchy and completely negate the Blend part, whereas lower values between $[0, 0.2]$ will make it feel more like the Gaussian-Blend result. The parameters *Hand Drawn* and *Sketch Density* are the 2 parameters the user will control to vary results.

$$result = w_1 \cdot VS_1 + w_2 \cdot VS_2 + w_3 \cdot VS_3 + w_{gb} \cdot R_{gb}$$

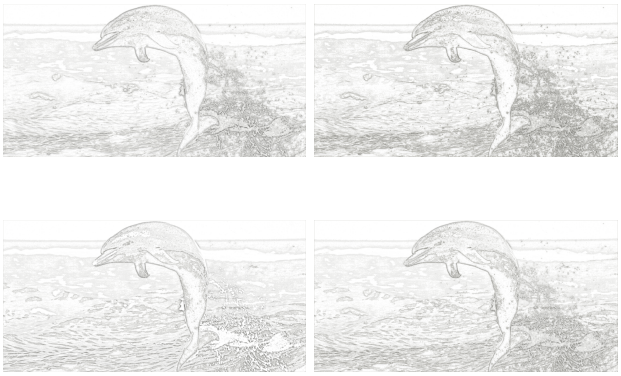

Fig. 8. Weighted Mean Results with Gaussian Parameters $G_1 = (\epsilon, \frac{4}{\sqrt{2\pi}}), G_1 = (\epsilon, \frac{2}{\sqrt{2\pi}}), G_1 = (\epsilon, \frac{1}{\sqrt{2\pi}})$ with $\alpha = (0.86, 0.94, 0.94)$ and **Sketch Density** = 17

VI. Future Scope

A. Novel Feature Extraction Method

The Orthogonal Gaussian Lattice Feature extraction can be used for all Applications that require Feature extraction such as Object Detection, Facial Expression Recognition, Face Detection etc. [25] [26]

This method gives us 3 dimensions from a single adjacent pixel pair and these values can be used as data for the Machine Learning and Deep Learning Models, where extra data about the Image will improve performance and reduce training times.

B. Object Tracking In Image Sequences

Videos are nothing but Images being represented with a temporal component as well. We can represent Videos mathematically as $V = f(x, y, t)$. There are several objects that are entering and leaving the frames, such as people, cars, trucks etc. and there are several objects that are stable such as the Road, sidewalk etc. [27] [28]

If we extract the Frame level lattices for each Gaussian in the Video, we will obtain fixed lattices for objects, e.g. we might obtain separate Lattices for the windshield, tire, bonnet etc. for a car. For object tracking we need to apply 2 novel methods of inter-frame lattice association and intra-frame lattice association.

Inter-frame Lattice association is the more important and easier to accomplish. Inter-frame association determines a lattice from one frame is associated with which lattice from the next frame. E.g. given a lattice of a car bonnet, which lattice in the next frame represents this lattice. We calculate this as the probability of a given lattice in frame$_i$ being mapped to another lattice in frame$_{i+1}$ as $P(L_{i+1,j_2}|L_{i,j_1})$ where j_1 is the lattice number in Lattice of frame$_i$ and lattice number j_2 is the lattice in frame$_{i+1}$.

We compute this using intersection probability mapping. We compute the Ring Sum of the lattice in frame$_i$ with all lattices in frame$_{i+1}$ and normalize the [29] results with the size of lattice in frame$_i$. The lattice with the highest probability is associated as the next lattice . We can also use their probabilistic inter-frame mapping to perform MCMC (Markov Chain Monte Carlo) and compute the probabilities of different trajectories, where a trajectory here is the motion of a lattice in 3 spatial and 1 temporal dimension, with 3 Gaussian Dimensions. This creates a higher dimension surface $\mathbb{R}^9$ which can be used to track multiple objects parallely.

C. Anomaly Detection In Image Sequences

We can compute the probability of a lattice trajectory [30] and we can also measure different telemetry from the lattices we have [31]. Such as:

1) **Mass**: Mass is measured as the number of pixels in a Lattice, where each pixel contributes a mass of 1 unit.
2) **Center Of Mass**: Center of Mass is measured as the mean of x and y Coordinates.

$$\text{C.O.M. } (M_\mu) = \frac{1}{\text{Mass}} (\Sigma x_i, \Sigma y_i)$$

3) **Moment Of Inertia**: We are measuring both Moment of Inertia and Mass because 2 different objects with the same size might have similar masses, but very different moment of inertia. E.g. a person with the same mass as a bag will have a higher moment of Inertia than a bag of similar mass and having a person stand at a street corner may be normal behaviour but a bag at a street corner won't be normal and should be considered an anomaly. By recording these separate metrics we can train a model based on both mass and moment of inertia. Moment of inertia is measured as the sum of distance squares of all points with the center of mass.

$$\textit{Moment Of Inertia } (I) = \sum_{r_i \in r} (r_i - r_\mu)^2$$

4) **Displacement**: Displacement is a quantity that is not measured for a lattice per frame independent of the next

frame. Displacement requires the probabilistic Inter-frame mappings between lattices and then displacement is measured as the distance between the center of mass of a lattice between 2 frames as it moves along it's trajectory. If the lattice doesn"t map onto any other lattice in the next frame we say that the displacement was very high as the lattice moved completely out of the frame and this is recorded in our experiment as a displacement of ∞.

$$\textit{Displacement } (\nabla L) = \overrightarrow{M_\mu(L(n_i, frame_i))} - \overrightarrow{M_\mu(L(n_j, frame_{i+1}))}$$

In the above notation M_μ denotes the center of mass and $L(n_i, frame_i)$ denotes the lattice number n_i from $frame_i$ which is mapped onto lattice number n_j in the next frame $frame_{i+1}$.

5) **Birth and Death Rate Factor**: We can measure the amount of new lattices created in a particular portion of the frame. [32] E.g. if cars are entering the street from some corner many new lattices will be created there and if they are leaving the street at some other corner, many lattices will die off at that location. By tracking the lattice births and deaths we can develop a probability distribution of these regions and then measure abnormality when some spurious births or deaths take place.

VII. Conclusion

The new method for feature extraction using orthogonal Gaussian Distributions to create Simple Graphs and Lattices from an Image gives us high dimensional data from an image and this can be used for many different tasks and even with learning models utilizing Machine Learning and Deep Learning.

Implementing this method to create a sketch composite from an image was just the first step in showcasing what it can do and it barely scratches the surface. We have also seen another example of object tracking above with very good results right off the bat.

As for creating a sketch composite, the simple method of using a pre-rendered texture gives the same result for all images with similar pattern on it and looks more like a sepia filter than a hand drawn sketch. The Gaussian Blur Blend method is the one which comes the closest and is the current state-of-the-art method, but even in that Method the results lack gradient and in many cases can look just like an image with deep shadows at strong borders.

The Orthogonal Gaussian Lattice Method introduces a mechanism to identify strong edges from weaker edges and also introduces a mechanism for us to understand which vertices lie at the border and which vertices lie in the interior of a region and it also gives us a way of identifying regions within an image using Lattices. Using the telemetry data from these lattices like mass, Moment Of Inertia, Center Of Mass we can then decide on how to shade and color different lattices differently and even when to combine and associate lattices. This method is more computationally expensive than the Gaussian Blur Blend.

We can then combine the orthogonal Lattice Results we have with the Gaussian blend model using a weight vector W in the Linear Fusion Technique and obtain results which look much better than the Gaussian Blur-Blend and highly convincing as a sketch drawn by a real artist.

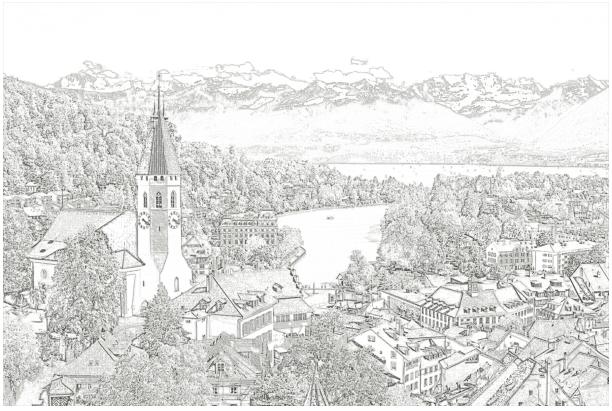

Fig. 9. An Example of Sketch Composite of a landscape where the traditional methods were struggling. $W = (0.1, 0.3, 0, 0.6)$ $\alpha = (0.86, 0.88, 0.92)$ and $G_1 = (\epsilon, \frac{4}{\sqrt{2\pi}})$, $G_1 = (\epsilon, \frac{1.3}{\sqrt{2\pi}})$, $G_1 = (\epsilon, \frac{1}{\sqrt{2\pi}})$

References

[1] Wang, J., Bao, H., Zhou, W. et al. **Automatic image-based pencil sketch rendering.** J. of Comput. Sci. Technol. 17, 347–355 (2002).

[2] John Canny **A Computation Approach to Edge Detection** in IEEE Transactions On Pattern Analysis and Machine Intelligence, Volume PAMI-8, Issue 6 November 1986

[3] Gao, Xingyu et al. **Automatic Generation of Pencil Sketch for 2D Images.** ICASSP (2010).

[4] E. S. Gedraite and M. Hadad, **Investigation on the effect of a Gaussian Blur in image filtering and Segmentation** Proceedings ELMAR-2011, 2011, pp. 393-396.

[5] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, James Hays; **Scribbler: Controlling Deep Image Synthesis With Sketch and Color** Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5400-5409

[6] Pekka J. Toivanen, **New geodosic distance transforms for gray-scale images** Pattern Recognition Letters, Volume 17, Issue 5, 1996, Pages 437-450, ISSN 0167-8655

[7] S. Keerativittayanun, T. Kondo, K. Kotani and T. Phatrapornnant, **An innovative of pyramid-based fusion for generating the HDR images in common display devices** 2015 14th IAPR International Conference on Machine Vision Applications (MVA), 2015, pp. 53-56, doi: 10.1109/MVA.2015.7153131.

[8] Bradski, G. **The OpenCV Library** Dr. Dobb's Journal of Software Tools 2000

[9] Harris, Charles R. and Millman, K. Jarrod and van der Walt, Stéfan J and Gommers, Ralf and Virtanen, Pauli and Cournapeau, David and Wieser, Eric and Taylor, Julian and Berg, Sebastian and Smith, Nathaniel J. and Kern, Robert and Picus, Matti and Hoyer, Stephan and van Kerkwijk, Marten H. and Brett, Matthew and Haldane, Allan and Fernández del

Río, Jaime and Wiebe, Mark and Peterson, Pearu and Gérard-Marchant, Pierre and Sheppard, Kevin and Reddy, Tyler and Weckesser, Warren and Abbasi, Hameer and Gohlke, Christoph and Oliphant, Travis E. **Array programming with NumPy** Nature 2020

[10] **Topic Models for Scene Analysis and Abnormality Detection** J. Varadarajan and J.M. Odobez, in Proc. ICCV Visual Surveillance workshop (ICCV-VS), Kyoto, October 2009.

[11] Jin Wang, Hujun Bao, Weihua Zhou, Quensheng Peng Xu Yingqing **Automatic image-based pencil sketch rendering** J. of Comput. Sci. Technol. 17, 347–355 (2002).

[12] Veryorka O, Buchanan J. **Comprehensive Halftoning of 3D scenes**. Computer Graphics Forum, EUROGRAPHICS'99, 1999, 18(3): 13–22.

[13] Deussen O, Hiller S. **Floating points: A method for computing stipple drawings**. Computer Graphics Forum, EUROGRAPHICS'2000, 2000, 19(3): 455–466.

[14] J. Shi, **Good Feature to Track**, in: IEEE Conference on Computer Vision and Pattern Recognition, 1994.

[15] "G. van Rossum, **Python tutorial** Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995."

[16] Luis Garrido, Philippe Salembier, David Garcia, **Extensive operators in partition lattices for image sequence analysis** Signal Processing, Volume 66, Issue 2, 1998

[17] T. Shima, S. Sugimoto and M. Okutomi, **Comparison of image alignment on hexagonal and square lattices** 2010 IEEE International Conference on Image Processing, 2010, pp. 141-144

[18] RevModPhys.26.311, **Group Theory and Crystal Lattices** Bell, Dorothy G. 1954, American Physical Society

[19] Birkhoff, G. **Lattices and their applications**Bulletin of the American Mathematical Society 44 (1938): 793-800.

[20] Li, H.; Chong, E.K.P. **On a Connection between Information and Group Lattices. Entropy** 2011, 13, 683-708.
bibitem22 Martinet, Jacques. **Perfect Lattices in Euclidean Spaces** Germany: Springer Berlin Heidelberg, 2013.

[21] **Applied Graph Theory in Computer Vision and Pattern Recognition** Germany: Springer, 2007.

[22] Multisensor Fusion for Computer Vision. Germany: Springer Berlin Heidelberg, 2013.

[23] A. Yilmaz, **Sensor Fusion in Computer Vision** 2007 Urban Remote Sensing Joint Event, 2007, pp. 1-5

[24] Raghu Krishnapuram, Joonwhoan Lee, **Fuzzy-set-based hierarchical networks for information fusion in computer vision** Neural Networks, Volume 5, Issue 2, 1992, Pages 335-350, ISSN 0893-6080

[25] Xiaofei He, Shuicheng Yan, Yuxiao Hu, P. Niyogi and Hong-Jiang Zhang, **Face recognition using Laplacianfaces** in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 3, pp. 328-340, March 2005

[26] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. 2003. **Face recognition: A literature survey** ACM Comput. Surv. 35, 4 (December 2003), 399–458

[27] Alper Yilmaz, Omar Javed, and Mubarak Shah. 2006. **Object tracking: A survey. ACM Comput. Survey** 38, 4 (2006), 13–es.

[28] D. Comaniciu, V. Ramesh and P. Meer, **Kernel-based object tracking** in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 5, pp. 564-577, May 2003

[29] Chong, Edwin K. P.., Zak, Stanislaw H.. **An Introduction to Optimization** Germany: Wiley, 2004.

[30] V. Mahadevan, W. Li, V. Bhalodia and N. Vasconcelos, **Anomaly detection in crowded scenes** 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, pp. 1975-1981

[31] **Anomaly Detection in Video Sequence With Appearance-Motion Correspondence** Trong-Nguyen Nguyen, Jean Meunier; Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 1273-1283

[32] Robert B. Cooper. 1981. **Queueing theory** In Proceedings of the ACM '81 conference (ACM '81). Association for Computing Machinery, New York, NY, USA, 119–122. DOI: