Semi-automated binary segmentation with 3D MRFs

Submission for Assignment 1 EE55C1

1st Haohan Zhu
Student ID: 22308057
Trinity College Dublin

***Abstract*—This academic paper focuses on the application of image matting in video streams to solve the problem of binary segmentation of moving objects. The Markov Random Field (MRF) algorithm, based on maximum likelihood keys, is explored in this study. The research investigates the performance of the MRF algorithm in different dimensions and with or without motion estimation applied. The research aims to provide a comprehensive analysis of the effectiveness of the MRF algorithm in binary segmentation of moving objects in video.**

I. Introduction

Compositing combines different images or videos and uses digital matting to separate foreground and background from the source media by estimating the opacity or transparency of each pixel in an image or video frame. Compositing is used in a wide variety of image and video processing, for example, in film projects where a green screen is used as a background for compositing effects in post-production. In digital matting, the technique is often used in conjunction with compositing to achieve accurate colour segmentation of the foreground elements. It involves estimating the α at each pixel of the foreground region and then using this information to create a matte or mask that accurately separates the foreground from the background.

Matting achieved by automatic segmentation, which contains color keying and compositing, occupies an important place in academic and industrial fields, that analyzes images or videos by an algorithm and segments the target object without any external input or human intervention. The advantage of automatic segmentation is that it allows for fast and efficient unification of large volumes of data. This advantage reduces the cost of human data processing and increases algorithm operation speed.

In this report, we focus on using video streams as a dataset for Bayesian matting rather than a single image, where the synthesis in the video stream requires algorithms to process all frames simultaneously. Suppose each frame is considered as a single image to be matted asynchronously. In that case, the computational cost will be much higher than the synchronous update approach. In the algorithm of this research, the approach is based on the Bayesian probabilistic matte model proposed by Yung et al.(2001) and the Markov Random Field algorithm (MRF) from the Prof. Kokaram materials and deployed in Foundry's Nukex, in addition to which I explore the differences between the 2D-MRF algorithm and 3D-MRF in the paper.

II. Modeling Mathematically

The algorithm for the target involves creating a binary segmentation of a video captured from a green screen background based on fractional opacity (ranging from 0 to 1) to extract the background. The prior probability for this extraction is defined as $p(\alpha = 0)$. Likelihood for the maximum α probability distribution is determined by the pixel at site $\mathbf{x} = [h, k]$ in the observed frames. This process is critical for accurately isolating the green screen background from the video. The focus of this report is on the algorithm and its effectiveness in achieving this task.

$$P_b(\mathbf{I} \mid \alpha = 0) \propto \exp - \left[\frac{\left(\mathbf{I_r} - \bar{\mathbf{B}}_\mathbf{r}\right)^2}{2\sigma_r^2} + \frac{\left(\mathbf{I_g} - \bar{\mathbf{B}}_\mathbf{g}\right)^2}{2\sigma_g^2} + \frac{\left(\mathbf{I_b} - \bar{\mathbf{B}}_\mathbf{b}\right)^2}{2\sigma_b^2} \right] \tag{1}$$

where the $\mathbf{I}$ means the pixels at site $\mathbf{x} = [h, k]$ of the frames.At this step, the algorithm expects to maximize the maximum a posteriori (MAP) distribution as follows [1]:

$$\begin{aligned} &\arg \max_{F,B,\alpha} P(F, B, \alpha \mid C) \\ &= \arg \max_{F,B,\alpha} P(C \mid F, B, \alpha) P(F) P(B) P(\alpha) / P(C) \\ &= \arg \max_{F,B,\alpha} \log P(C \mid F, B, \alpha) + \log P(F) + \log P(B) + \log P(\alpha) \end{aligned} \tag{2}$$

Then the task creates a $3 * 3$ neighborhood around the unknown pixel, which is the central point, to identify the value of the central point named Markov Random Field (MRF). The algorithm calculates the potential energy of Cliques in the field to calculate the energy.

$$p\left(\alpha(\mathbf{x}) \mid \mathcal{N}_\alpha(\mathbf{x})\right) = \frac{1}{Z} \exp -\Lambda \left[\sum_{k=1}^{4} V\left(\alpha(\mathbf{x}), \alpha\left(\mathbf{x} + \mathbf{q}_k\right)\right) \right] \tag{3}$$

where the q_k stands for the nodes around the central point.

As with all the mathematics mentioned in this report, the final energy function obtained is as follows.

$$\begin{aligned} &- \ln\left(p\left(\alpha \mid I, \mathcal{N}\alpha\right)\right) = \ln(Z) + Edata(\alpha) + E_{smooth}(\alpha) \\ &E_{data}(\alpha) = (\mathbf{I} - \bar{\mathbf{I}}\alpha)^T \mathbf{R} \alpha^{-1} (\mathbf{I} - \bar{\mathbf{I}}_\alpha) \\ &E_{smooth}(\alpha) = \sum_{k=1}^{4} V(\alpha(\mathbf{x}), \alpha(\mathbf{x} + \mathbf{q}_k)) \end{aligned} \tag{4}$$

III. Algorithm and Optimisation

In this research, the algorithm consists of 2D MRF and 3D MRF method, where the 2D refer to the image size. In contrast, the 3D MRF refers to the image size dimension and a time series.

A video stream with a green screen as the background is used as input data. Subsequently, the average pixel value in the background is calculated to apply a maximum likelihood keyer to generate a binary segmentation to extract the foreground and background.

Algorithm 1 Calculate Maximum Likelihood Keyer α at pixel $x = [h, k]$ to generate binary segmentation

Require: α is 1 if the x is foreground and 0 if it is background

- Extract a area from the background of a given image to calculate the Background colour mean of I_r, I_g, I_b
- Subtract the background colour mean to obtain the foreground area image
- Apply the Bayesian probability and the model proposed by Yung et al. (2001) to calculate the probability distribution of pixels.
- Delineate with a appropriate energy threshold to generate a binary segmentation

After generating the binary segmented image, the region called "Unknown" in the image interferes with the segmentation of foreground and background, and the pixel values of the background are not strictly excluded completely, so after applying energy threshold to segment the background, this project also applies Markov Random Field to smooth the image.

This paper explores two approaches to implementing the MRF algorithm, one implemented at a single image level is referred to as a 2D MRF, whose dimensions are the dimensions of the image sequence, i.e., length and height. The following pseudo-code describes the 2D MRF implementation in the experiment.

Algorithm 2 2D MRF implementation

Require: The smoothness algorithm in binary segmentation

For N Iteration:

- Convert energy to luminance as background energy input and binary segmentation image as reference α input
- Extract a $3 * 3$ region as the neighborhood around the pixel $x = [h, k]$
- Identify the cliques in the neighborhood
- Calculate the spatial energy for background and foreground
- Re-generate the energy with smoothness weight
- Compare energies of background and foreground
- Output a new α at $x = [h, k]$

Considering that the experimental object of this project is a video stream, this research implement the algorithm of 3D MRF with time series as the third dimension as shown in figure 1.

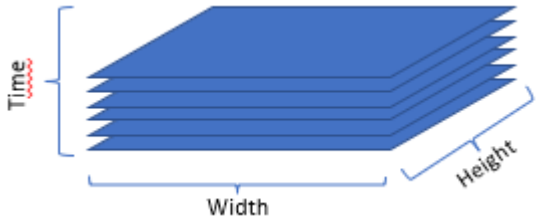


Fig. 1. 3D MRF Size

The 3D MRF model builds on the 2D MRF by treating each frame in an image sequence as a node. The next frame inherits the alpha in each image frame and continues to run. The smoothness prior term in the energy function is extended to incorporate the temporal dependence between adjacent frames.

Algorithm 3 3D MRF implementation

Require: A framework similar to 2D MRF

For N Iteration:

- Build a essential algorithm framework similar to 2D MRF
- Call a time-offset from the last frame
- Calculate the energy for current frame and last frame
- Consider and compare the new energy to generate new α

Since 3D MRF constructs a data structure with time series as the dimension, this paper institutes a motion estimation approach, motion compensation, to direct MRF along the motion trajectory. Motion compensation reduces redundancy in the video data by exploiting temporal redundancy, since objects in the foreground do not differ significantly in consecutive frames. It compensates for any residuals in the binary segmentation by estimating the difference between the previous and current frames, thus compensating for α.

Algorithm 4 Motion Compensation

- Use motion compensation before 3D MRF algorithm to reduce errors in the foreground
- $I_n(\mathbf{x}) = I_{n-1}\left(\mathbf{x} + d_{n,n-1}(\mathbf{x})\right) + \mathrm{e}(\mathbf{x})$
- The formula calculates the new pixel after motion compensation

IV. Nuke Implementation

In the NukeX project file, the project is divided into categories according to the role of the modules so that they can be identified. As shown in Figure 2, the program in Nuke extracts the foreground by subtracting the average pixel value of the green screen background and implements a binary segmentation based on a maximum likelihood colour key and an energy threshold.

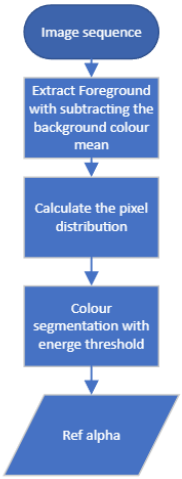


Fig. 2. Basic Color Segmentation with Maximum Likelihood Keyer

In implementing MRF, the module calculates the spatial energy in the image by entering the reference alpha and background energy. Subsequently, the energy of the foreground and background is measured in each pixel value in the image according to the cliques in the neighborhood, and it is determined whether the pixel value belongs to the foreground or the background by comparing the energies. The 2D MRF and 3D MRF implementation diagrams in Nuke are shown below and the code will be plotted in the Appendix in the Figure 9.

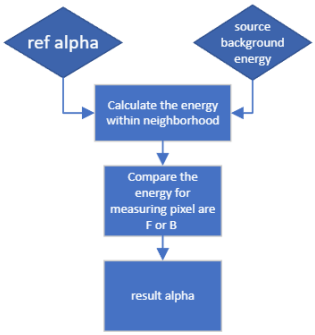


Fig. 3. A general description of MRF

In 3D MRF, the algorithm added an input module called timeoffset to each Blinkscript as a way to read the previous input for the current frame and the structure has been plotted in the Figure 4.

As the motion compensation mentioned in this paper, a VectorGenerator node was implemented to calculate the motion vector between two frames and an IDistort node to warp the image to compensate for the difference in motion in Nuke and the Figure5shows a general diagram of program.

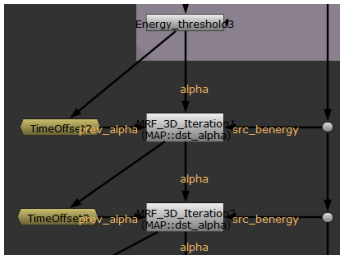


Fig. 4. The 3D MRF structure in Nuke

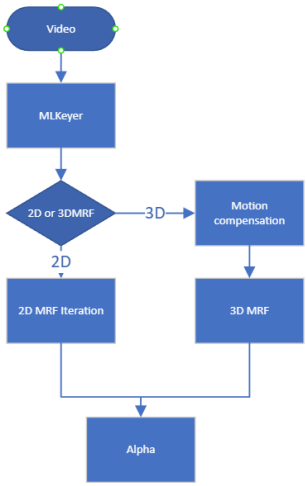


Fig. 5. A general view of program

V. Experiments and Performance Measures

This research used ten iterations of each module to check the effectiveness and output of the various algorithms at each iteration. Subsequently, to verify the accuracy of the algorithms, the report compared the generated binary images with GroundTruth images and obtained the associated confusion matrix to calculate MSE, F1 Score, Precision, Recall, and Accuracy, which has been shown in the Figure 13 to determine the performance of the algorithms. In the appendix, this paper have also included the confusion matrix.

VI. Results and Discussion

In this paper, by comparing the MSE and performance parameters of 2D MRF and 3D MRF, the study found that the MSE of the 2D MRF was optimized more efficiently than the 3D MRF in the figure 6. The additional dimensionality in the 3D MRF means that the number of neighboring voxels that can influence the central pixel value is more significant than in the 2D MRF. This can lead to more complex models with more influential errors than simpler 2D models.

According to the data in the figure 13, the accuracy and fidelity of the 3D MRF are always lower than that of the 2D MRF for the same number of iterations for a limited number of iterations. However, when the number of cliques selected by MRF in the domain is increased to 8 pairs in the figure 7, the fidelity and accuracy of the 3D MRF exceed that of the

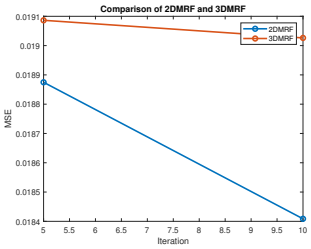


Fig. 6. Comparison of 2D and 3D MRF Mean Squared Error

2D MRF, indicating that the accuracy curve of the 3D MRF converges later than that of the 2D MRF with a guaranteed number of iterations and training volume, and therefore takes more time to obtain more accurate results. Regarding visual perception, the MRF deployed in the time dimension looks closer to ground truth.

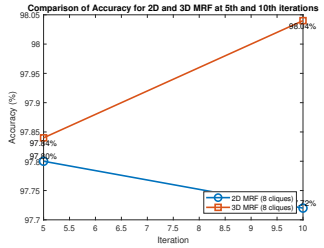


Fig. 7. Comparison of Accuracy for 2D and 3D MRF at 5th and 10th iterations

Comparing likelihood keying applied to a single frame, it is visually intuitive that the algorithm with maximum likelihood keying applied to each frame individually has a far less accurate outlook than the algorithm with MRF applied, and it also has more noise in the entire image. This is because the maximum likelihood algorithm is based on removing the average pixel value from the background. However, in an actual shooting situation, despite using a green screen as the background, the background cannot be eliminated very cleanly in light conditions and under various harsh conditions. This was evident in this experiment when the probability distribution of the background pixel values was extracted and the image was converted to the luminance channel as shown in the figure 8.

Fig. 8. The result of Maximum likelihood keying in luminance image

These resulted in an MSE of 0.0203 for MLK compared to an average MSE of 0.02539967 for a 2D MRF of the same size. Additionally, the algorithm applying the maximum likelihood algorithm to each frame individually is not suitable for large volumes of data. In the environment of this research, the video stream is only five frames. However, in the actual case of processing data there is the potential for substantial amounts of video data. Using separate frames would result in the algorithm having to be performed asynchronously, significantly reducing the efficiency of the program's operation.

As mentioned earlier in the paper, the 3D MRF algorithm with motion estimation compensates for image distortion due to motion changes by creating a trajectory vector. In the 3D MRF algorithm with motion estimation, I also iterated ten times and recorded the performance parameters for the fifth and tenth iterations in figure 13.

From the data, the 3D MRF with motion compensation is not superior to the normal 3D MRF algorithm, due to the additional computational effort involved in the motion estimation vector in the motion estimation module, which consists of compensating for blurring and distortion due to motion, thus making motion artefacts in the foreground more visible and its complexity leading to more bias being introduced into the model.

However, in terms of visual perception, the 3D MRF algorithm with motion compensation has a significantly clearer outlook than the normal algorithm, with less noise than the normal algorithm.

In future work, this project expects to save project memory by not using duplicate kernels for redundant iterations but a single kernel to realise the iterative process.

VII. Conclusions

Regarding the data results from this experiment, the matting effect is only sometimes more accurate for video data than for image data. This is because in two-dimensional data, the positions of the pixels are relative to each other and there are no motion artifacts such as these that could potentially interfere with the algorithm. In contrast, in video streaming data, the original pixel values may deviate from the attempted image at rest due to motion artifacts and uncertainty of movement. This experimental study demonstrates that image matting with time outperforms single-frame images in terms of both performance and convergence interval. The algorithm's robustness with motion estimation applied in the time-series image matte is greatly improved, and the problem of image noise and motion blur can be solved in the face of various adverse shooting conditions.

Additionally, this paper has found that the MRF prior is inversely proportional to the number of iterations. The effect of the prior decreases as the number of iterations increases, and conversely the importance of the alpha generated by the previous frame in the sequence increases. This is evident in the comparison between 3D MRF and 2D MRF, where it is found that the convergence limit of 3D MRF is significantly higher than that of 2D MRF, indicating that the amount of data and the accuracy of the algorithm have a more significant impact on the final convergence accuracy of the data.

The effect of smoothing hyperparameters on image fading is investigated in this paper. The smoothness hyperparameter is

used to control the smoothness of the output image or signal. In MRF algorithms, the smoothness hyperparameter is often used as a regularisation parameter to balance the trade-off between fitting the data accurately and maintaining the spatial smoothness of the output.

Suppose the smoothness hyperparameter is set too high. In that case, the output image or signal will be too smooth and important details and features in the data may be lost. On the other hand, if the smoothness hyperparameter is set too low, the output image or signal may be too noisy and may not accurately represent the underlying structure in the data.

In this study, a relatively smoothing hyperparameter was compared to a more appropriate balancing hyperparameter and it is evident from the output result graph that some of the information in the image is missing with the larger smoothing hyperparameter applied.

VIII. Declaration

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar. I certify that this submission is my own work.

References

[1] Y.-Y. Chuang, B. Curless, D. Salesin, and R. Szeliski, "A bayesian approach to digital matting," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 2, 2001, pp. II–II.

Appendix

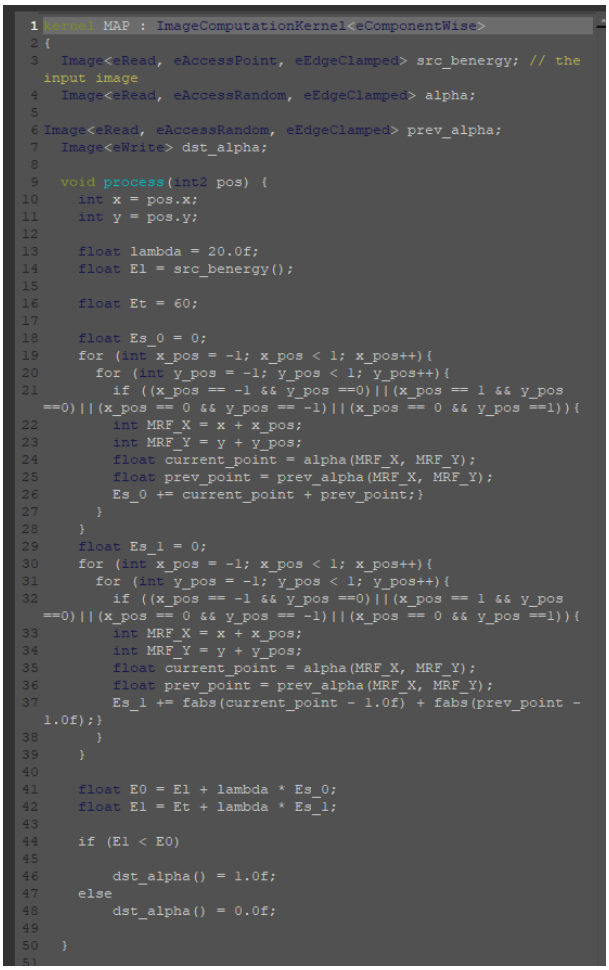
The Figure 9 shows the code of the 2D MRF implementation while the implementation of 3D MRF has been plotted in Figure 10.

A. Code plot

B. Coufusion result

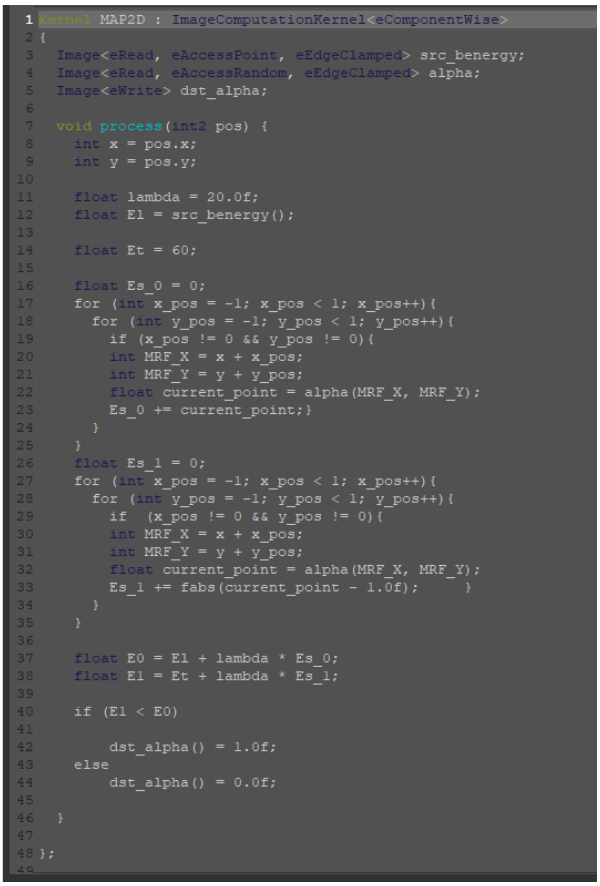
```
kernel MAP2D : ImageComputationKernel<eComponentWise>
{
  Image<eRead, eAccessPoint, eEdgeClamped> src_benergy;
  Image<eRead, eAccessRandom, eEdgeClampec> alpha;
  Image<eWrite> dst_alpha;

  void process(int2 pos) {
    int x = pos.x;
    int y = pos.y;

    float lambda = 20.0f;
    float El = src_benergy();

    float Et = 60;

    float Es_0 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if ((x_pos == -1 && y_pos ==0)||(x_pos == 1 && y_pos
==0)||(x_pos == 0 && y_pos == -1)||(x_pos == 0 && y_pos ==1)){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        Es_0 += current_point;}
      }
    }
    float Es_1 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if ((x_pos == -1 && y_pos ==0)||(x_pos == 1 && y_pos
==0)||(x_pos == 0 && y_pos == -1)||(x_pos == 0 && y_pos ==1)){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        Es_1 += fabs(current_point - 1.0f);     }
      }
    }

    float E0 = El + lambda * Es_0;
    float E1 = Et + lambda * Es_1;

    if (E1 < E0)

        dst_alpha() = 1.0f;
    else
        dst_alpha() = 0.0f;

  }

};
```

Fig. 9. Code of 2D MRF with 4 cliques



```
kernel MAP : ImageComputationKernel<eComponentWise>
{
  Image<eRead, eAccessPoint, eEdgeClamped> src_benergy; // the
 input image
  Image<eRead, eAccessRandom, eEdgeClamped> alpha;

Image<eRead, eAccessRandom, eEdgeClamped> prev_alpha;
  Image<eWrite> dst_alpha;

  void process(int2 pos) {
    int x = pos.x;
    int y = pos.y;

    float lambda = 20.0f;
    float El = src_benergy();

    float Et = 60;

    float Es_0 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if ((x_pos == -1 && y_pos ==0)||(x_pos == 1 && y_pos
 ==0)||(x_pos == 0 && y_pos == -1)||(x_pos == 0 && y_pos ==1)){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        float prev_point = prev_alpha(MRF_X, MRF_Y);
        Es_0 += current_point + prev_point;}
      }
    }
    float Es_1 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if ((x_pos == -1 && y_pos ==0)||(x_pos == 1 && y_pos
 ==0)||(x_pos == 0 && y_pos == -1)||(x_pos == 0 && y_pos ==1)){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        float prev_point = prev_alpha(MRF_X, MRF_Y);
        Es_1 += fabs(current_point - 1.0f) + fabs(prev_point -
 1.0f);}
      }
    }

    float E0 = El + lambda * Es_0;
    float El = Et + lambda * Es_1;

    if (El < E0)

        dst_alpha() = 1.0f;
    else
        dst_alpha() = 0.0f;

  }
```

Fig. 10. Code of 3D MRF with 4 cliques



```
kernel MAP2D : ImageComputationKernel<eComponentWise>
{
  Image<eRead, eAccessPoint, eEdgeClamped> src_benergy;
  Image<eRead, eAccessRandom, eEdgeClamped> alpha;
  Image<eWrite> dst_alpha;

  void process(int2 pos) {
    int x = pos.x;
    int y = pos.y;

    float lambda = 20.0f;
    float El = src_benergy();

    float Et = 60;

    float Es_0 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if (x_pos != 0 && y_pos != 0){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        Es_0 += current_point;}
      }
    }
    float Es_1 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if  (x_pos != 0 && y_pos != 0){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        Es_1 += fabs(current_point - 1.0f);     }
      }
    }

    float E0 = El + lambda * Es_0;
    float El = Et + lambda * Es_1;

    if (El < E0)

        dst_alpha() = 1.0f;
    else
        dst_alpha() = 0.0f;

  }
};
```

Fig. 11. Code of 2D MRF with 8 cliques

```
kernel MAP : ImageComputationKernel<eComponentWise>
{
  Image<eRead, eAccessPoint, eEdgeClamped> src_benergy; // the input image
  Image<eRead, eAccessRandom, eEdgeClamped> alpha;

Image<eRead, eAccessRandom, eEdgeClamped> prev_alpha;
  Image<eWrite> dst_alpha;

  void process(int2 pos) {
    int x = pos.x;
    int y = pos.y;

    float lambda = 20.0f;
    float El = src_benergy();

    float Et = 60;

    float Es_0 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if (x_pos != 0 && y_pos != 0){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        float prev_point = prev_alpha(MRF_X, MRF_Y);
        Es_0 += current_point + prev_point;}
      }
    }
    float Es_1 = 0;
    for (int x_pos = -1; x_pos < 1; x_pos++){
      for (int y_pos = -1; y_pos < 1; y_pos++){
        if (x_pos != 0 && y_pos != 0){
        int MRF_X = x + x_pos;
        int MRF_Y = y + y_pos;
        float current_point = alpha(MRF_X, MRF_Y);
        float prev_point = prev_alpha(MRF_X, MRF_Y);
        Es_1 += fabs(current_point - 1.0f) + fabs(prev_point - 1.0f);}
      }
    }

    float E0 = El + lambda * Es_0;
    float E1 = Et + lambda * Es_1;

    if (E1 < E0)

        dst_alpha() = 1.0f;
    else
        dst_alpha() = 0.0f;

  }

};
```

Fig. 12. Code of 3D MRF with 8 cliques

	frames	Cliques	Iterations	MSE	Precision	Recall	F1 Score	Accuracy
2DMRF	47			1.89E-02	0.9334	0.9416	0.9375	0.9811
	48			2.13E-02	0.932	0.9208	0.9264	0.9787
	49			3.65E-02	0.9321	0.8303	0.8782	0.9635
3DMRF	47			0.019086372	0.9328	0.9407	0.9367	0.9809
	48		5	0.021558974	0.9313	0.9201	0.9257	0.9784
	49			0.036631944	0.9316	0.8298	0.8777	0.9634
3DMRF+MC	47			0.018898293	0.9308	0.9445	0.9376	0.9811
	48			0.022214084	0.9296	0.9172	0.9233	0.9778
	49			0.037359393	0.926	0.8306	0.8757	0.9626
		4						
2DMRF	47			0.018408655	0.9316	0.947	0.9392	0.9816
	48			0.021108218	0.9306	0.9242	0.9274	0.9789
	49			0.036682129	0.9301	0.831	0.8777	0.9633
3DMRF	47			0.019026241	0.9285	0.9462	0.9373	0.981
	48		10	0.02181939	0.9268	0.9234	0.9251	0.9782
	49			0.037376121	0.926	0.8305	0.8757	0.9626
3DMRF+MC	47			0.018965205	0.9279	0.9473	0.9375	0.981
	48			0.021615488	0.9289	0.9224	0.9256	0.9784
	49			0.038093623	0.9234	0.8284	0.8733	0.9619
2DMRF	47			0.019533963	0.9285	0.9425	0.9355	0.9805
	48			0.022024649	0.9274	0.9211	0.9242	0.978
	49			0.037361202	0.9269	0.8296	0.8756	0.9626
3DMRF	47			0.019499602	0.9259	0.9459	0.9358	0.9805
	48		5	0.021636737	0.9266	0.925	0.9258	0.9784
	49			0.038012243	0.9237	0.8286	0.8736	0.962
3DMRF+MC	47			0.019094962	0.928	0.9464	0.9371	0.9809
	48			0.021859628	0.9282	0.9214	0.9248	0.9781
	49			0.03804163	0.9258	0.8261	0.8731	0.962
		8						
2DMRF	47			0.020184552	0.9238	0.9434	0.9335	0.9798
	48			0.022758879	0.9221	0.9218	0.922	0.9772
	49			0.038168222	0.9215	0.8298	0.8733	0.9618
3DMRF	47			0.01962981	0.9218	0.9499	0.9356	0.9804
	48		10	0.021822103	0.922	0.929	0.9255	0.9782
	49			0.039130317	0.9185	0.8264	0.87	0.9609
3DMRF+MC	47			0.019298412	0.9231	0.9508	0.9367	0.9807
	48			0.022111455	0.9236	0.9249	0.9243	0.9779
	49			0.038770435	0.9203	0.827	0.8711	0.9612

Fig. 13. MSE and some Performance parameters

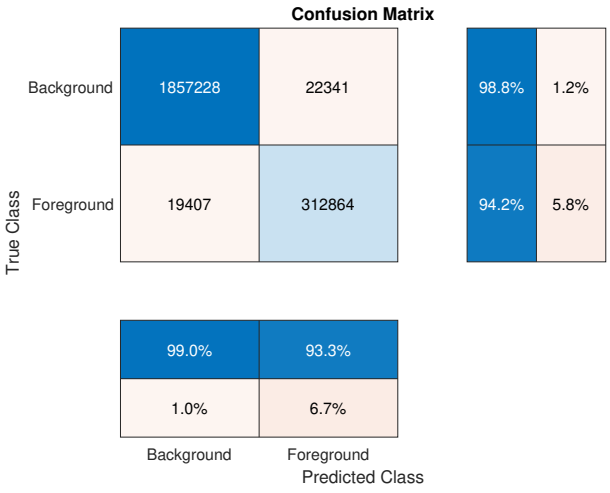


Fig. 14. 2D MRF with 4 cliques at 5th Iteration Confusion Matrix

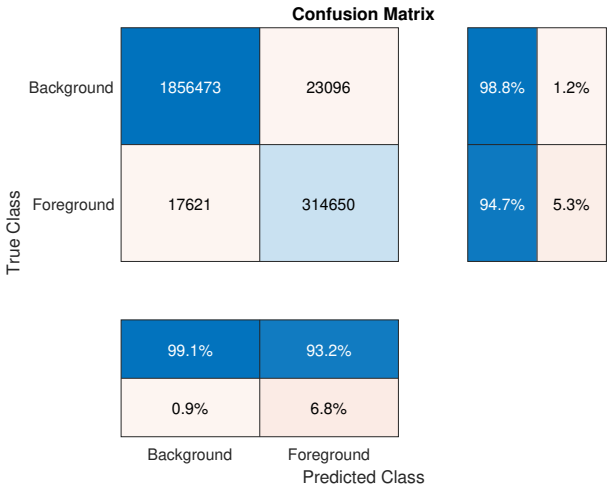


Fig. 15. 2D MRF with 4 cliques at 10th Iteration Confusion Matrix

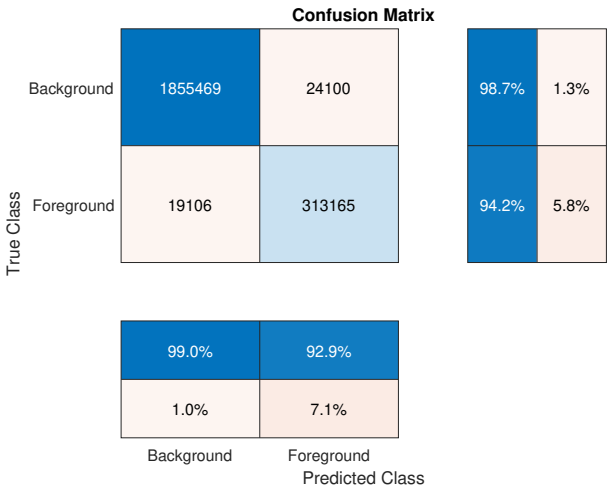


Fig. 16. 2D MRF with 8 cliques at 5th Iteration Confusion Matrix

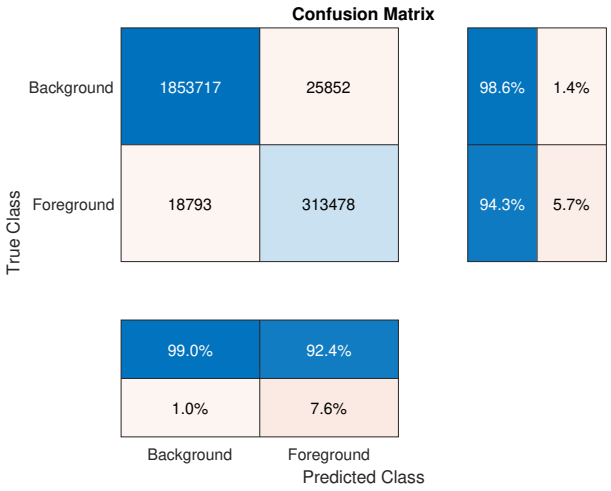


Fig. 17. 2D MRF with 8 cliques at 10th Iteration Confusion Matrix

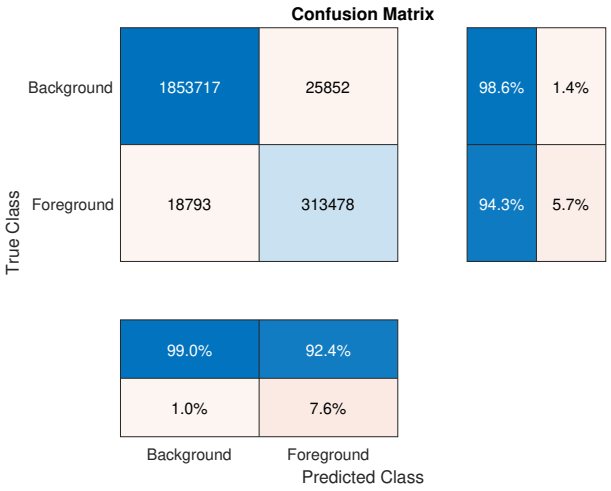


Fig. 18. 3D MRF with 4 cliques at 5th Iteration Confusion Matrix

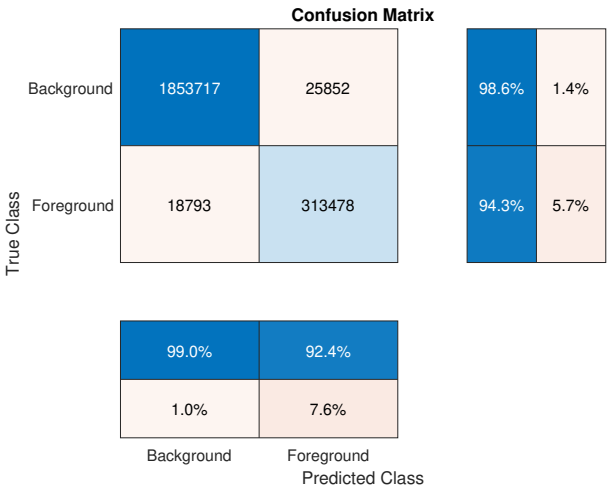


Fig. 19. 3D MRF with 4 cliques at 10th Iteration Confusion Matrix

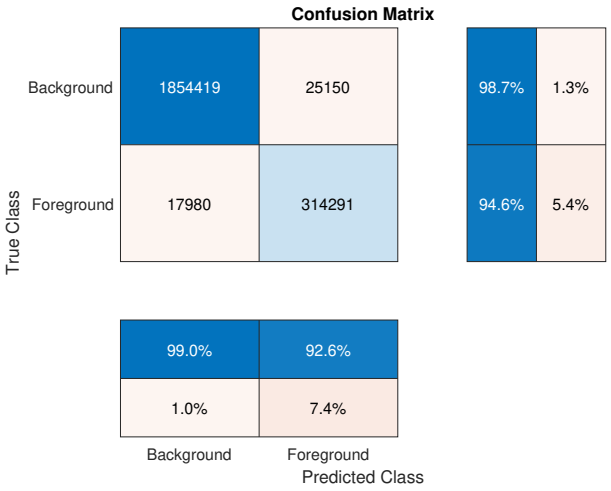


Fig. 20. 3D MRF with 8 cliques at 5th Iteration Confusion Matrix

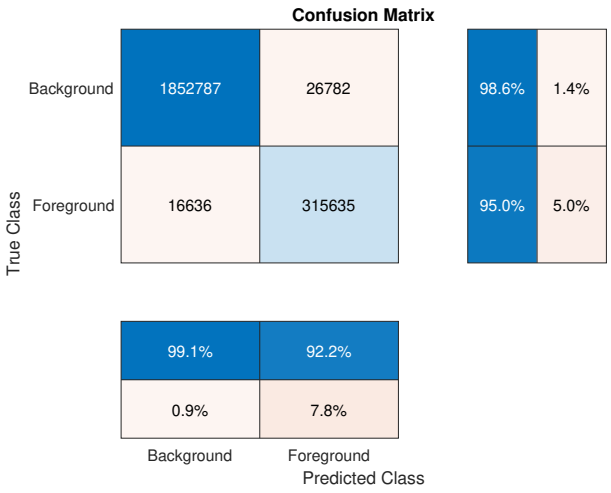


Fig. 21. 3D MRF with 8 cliques at 10th Iteration Confusion Matrix

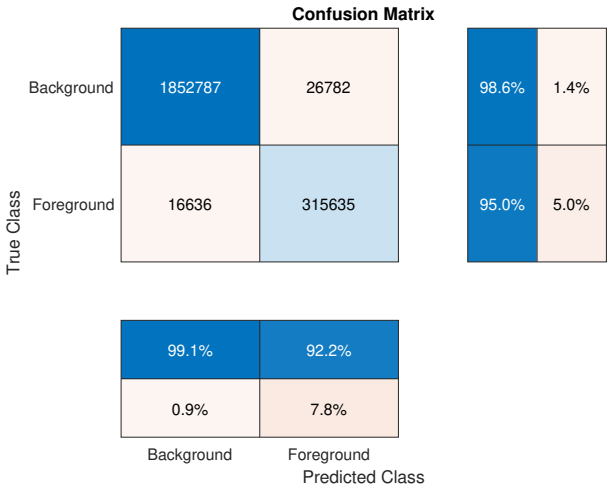


Fig. 22. 3D MRF with 4 cliques and implement Motion Compensation at 5th Iteration Confusion Matrix

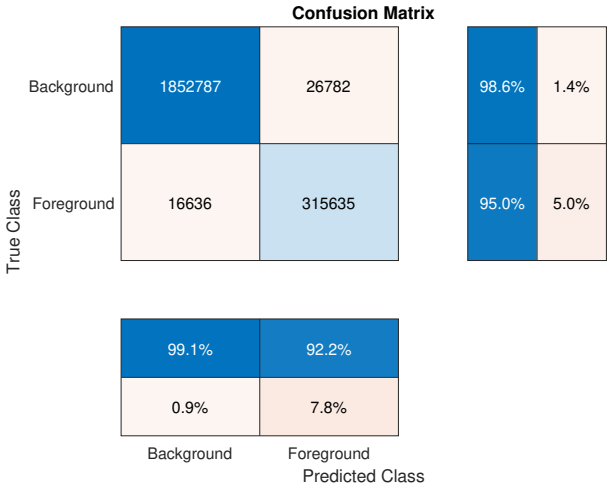


Fig. 23. 3D MRF with 4 cliques and implement Motion Compensation at 10th Iteration Confusion Matrix

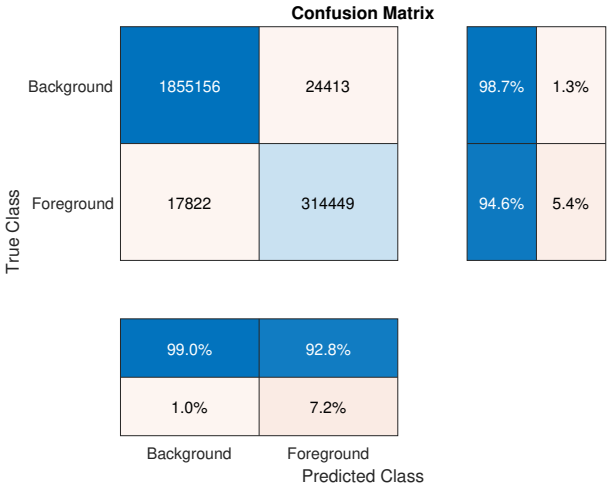


Fig. 24. 3D MRF with 8 cliques and implement Motion Compensation at 5th Iteration Confusion Matrix

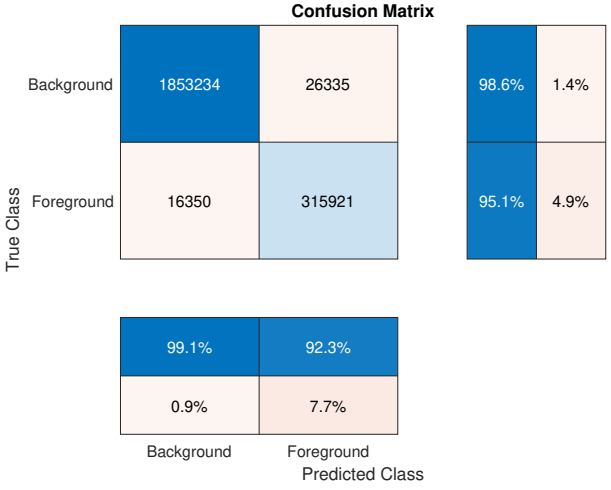


Fig. 25. 3D MRF with 8 cliques and implement Motion Compensation at 10th Iteration Confusion Matrix