CS-512 : Computer Vision
Image Inpainting

Smitesh Lokare (A20380808)
Department of Electrical and Computer Engineering
Illinois Institute of Technology

Sushmitha Sekuboyina (A20365741)
Department of Computer Science
Illinois Institute of Technology

Work Distribution:
Smitesh Lokare: Survey of existing image inpaint methods, analysis and implementation of fast marching method.
Sushmitha Sekuboyina: Survey on the issues with image reconstruction, analysis and implementation of Exemplar based image-inpainting method and comparison of the algorithms with the built-in functions.

I. Problem Statement

Image inpainting is a technique of reconstructing lost or damaged images by filling missing spaces. Image inpainting serves a wide range of applications of removing scratches on barcodes, removing text and logos from still images, red eye correction, compression, image coding and transmission. This technique is not only used for removing unwanted interlopers from images, but is also used to fix small holes in old images by utilizing spatial information of the neighborhood pixel region. The goal of image inpainting is not only to reconstruct the image but also to produce a modified image in which the inpainted region is not detectable by a typical viewer who has no knowledge of the original image.
Due to the wide range of applications of image inpainting. We plan to implement two algorithms and compare their results with the traditional inpainting algorithms.

II. Proposed Solution

Approach:

Image inpainting works as follows. First, the region of the image to be inpainted is selected. Then from the known image information, properties such as texture and pixel values in the missing image spaces are filled. For the image reconstruction to be plausible, the image should continue to be isophotes - a curve on a chart joining points of equal light intensity, in our case joining lines of equal gray value inside the reconstructed image.

Many inpainting methods were proposed with a principal goal of preserving isophotes but never perfectly attained in practice. The main problem is that the isophotes propagated to the inpainting region produced a certain amount of blur. Also, there was a problem with isophote estimation and time complexity in reconstructing the image.

But most of these proposed methods require solving a Partial Differential Equation (PDE) or a system of PDEs to describe color propagation to missing regions which is quite slow and has only partial implementation detail. Other inpainting methods use a 3x3 convolution filter over the mission region to diffuse image information. But this method requires manual selection of image gradients to avoid blurring of an image and it has no provision for isophote direction.

New inpainting algorithms were proposed to overcome the aforementioned problems. We implement two of the proposed algorithms and compare their results with the traditional inpainting algorithms.

1. *An Image Inpainting Technique Based on the Fast Marching Method*

The fast marching method (FMM), an image inpainting method, is fast and very simple to implement. The FMM produces nearly identical results as the slower and more complex methods Algorithm starts from the boundary of the inpaint region and gradually moves up along it by propagating an image smoothness estimator along the gradient of image, where the smoothness estimate is an weighted average over known image region. The fast marching method propagates new pixels along the boundary of the region in the order of their distance until all the pixels in the missing region are filled. We know that the information to fill the missing region is determined by the values of the known image region close to it and hence higher weight is assigned to the pixels lying near the normal of the boundary and boundary contours. FMM also solves the problem of *boundary values* since it maintains a narrow band which is within the inpainting image boundary. This method describes problem of closed surface as a function of time with speed in the direction normal to the propagation surface. The FMM methods takes the advantage of optimal values and generates known information i.e., boundary values.

$$\begin{aligned} &|\nabla u(x)| = 1/f(x) \text{ for } x \in \Omega \\ &u(x) = 0 \text{ for } x \in \partial\Omega \end{aligned}$$

Where f(x) is a speed function in normal direction at the point (x) on the boundary curve. ‘u’ is the time at which the contour crosses the point (x), which is obtained by solving the above equation.

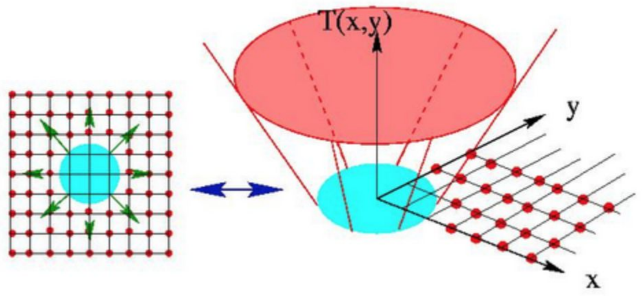


Figure 1: Fast marching method

In this way, the FMM method obtains the boundary values and inpaints the pixel of the missing region near the pixels of the known region first. It works just like a manual heuristic operation by moving from pixel to pixel until the boundary region is inpainted. This algorithm is enable in openCv by using the flag cv2.INPAINT_TELEA.

2. ***Exemplar based image-inpainting method:***

Inpainting algorithms are used to reconstruct images by removing scratches, objects or filling holes in the damaged region to produce a 'reasonable' image to the human eye. But these algorithms introduce some blur when reconstructing large portions of an image. In order to solve this, the exemplar based approach combines the inpainting techniques with '*texture synthesis*', which generates large portions of image from the sample textures of the image. The performance of this approach is efficient and qualitative since it combines the strengths of the both inpainting and texture synthesis, i.e., it pays special attention to the linear structures of image, but only along the target region that influence the fill order of the pixels.

The exemplar based approach decomposes the original image into two components - inpainting process and texture synthesis process. The output of the approach is the sum of the two processes. In the first step, patch priority is computed and in the second step, best matching patch is selected which is sampled from the known region by using similarity metric. This selected is patch is pasted on the target patch in the missing region of image. These aforementioned steps are repeated iteratively until the missing region i.e., the target region is inpainted.

Exemplar based approach has the following three steps:

1. *Computing the patch priorities*: The filling order of the pixels in important in this approach. Higher priority is given to those regions of the target region that lie on the continuation of the image edge, because it is most likely that the best match will also lie along the same edge. Priority is defined as the product of the confidence C(p) term and the data D(p) term.

$$P(\mathbf{p}) = C(\mathbf{p})D(\mathbf{p}).$$

And they are defined as follows: C(p) gives the concentric fill order. D(p) is the strength function of the isophote.

$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\Psi_{\mathbf{p}}\cap(\mathcal{I}-\Omega)} C(\mathbf{q})}{|\Psi_{\mathbf{p}}|}, \qquad D(\mathbf{p}) = \frac{|\nabla I_{\mathbf{p}}^{\perp}\cdot \mathbf{n}_{\mathbf{p}}|}{\alpha}$$

Where Ψ_p is the target patch, α is a normalization factor, n_p is a unit vector.

2. *Propagating texture and structure information*: Once the patch with highest priority is found by computing the patch priority, we then fill the target region with the data

extracted from the source region. We search the source region for a patch that is similar to the target patch and then propagate the pixels.

3. *Updating confidence values*: We iteratively update the confidence values after filling each patch with the best matching patch. The confidence value is frozen once a pixel is filled i.e., this value decays after every iteration, indicating that we are less sure of the center pixel in the target region.

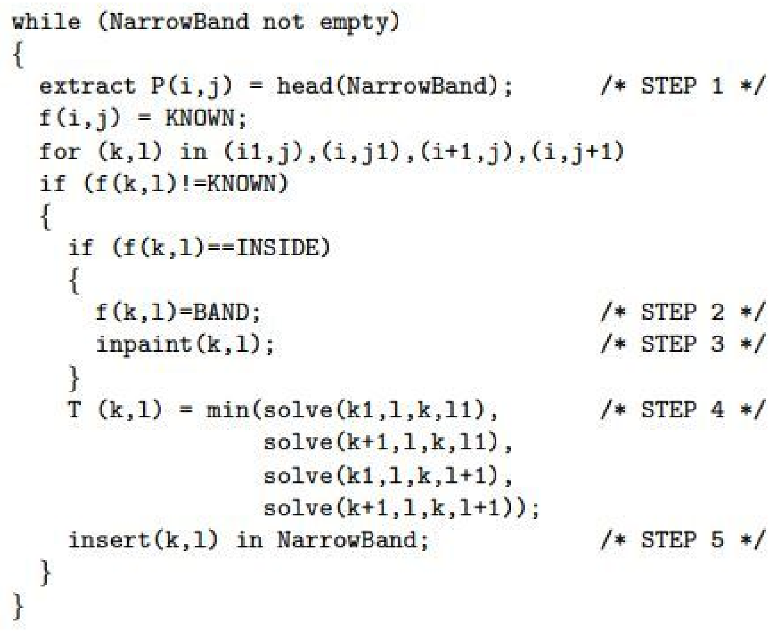
III. IMPLEMENTATION DETAILS

1. *An Image Inpainting Technique Based on the Fast Marching Method:*

The image to be inpainted is split into 3 parts, KNOWN part - where image information is known, BAND part- part of masked area currently under consideration to be updated and UNKNOWN part- area about which we have no information about This information is known as flags.

After initializing flags, and boundary distance, we smoothen the BAND with filter that is dependent on given radius. Higher values of radius blur the sharp details to be reconstructed, although they are useful when inpainting thicker regions. After smoothing, the pixel with smallest distance from boundary is extracted. We change its flag to known and then march this separation boundary inward using FMM from this pixel. Compute distance of new pixels from boundary. If the new pixels are unknown, add them to the BAND and inpaint the pixel. This procedure is repeated for every pixel in BAND.

This loop finally terminates when all the pixels are iterated and there aren't any new pixels to inpaint.



```
while (NarrowBand not empty)
{
  extract P(i,j) = head(NarrowBand);      /* STEP 1 */
  f(i,j) = KNOWN;
  for (k,l) in (i1,j),(i,j1),(i+1,j),(i,j+1)
  if (f(k,l)!=KNOWN)
  {
    if (f(k,l)==INSIDE)
    {
      f(k,l)=BAND;                        /* STEP 2 */
      inpaint(k,l);                       /* STEP 3 */
    }
    T (k,l) = min(solve(k1,l,k,l1),       /* STEP 4 */
                  solve(k+1,l,k,l1),
                  solve(k1,l,k,l+1),
                  solve(k+1,l,k,l+1));
    insert(k,l) in NarrowBand;            /* STEP 5 */
  }
}
```

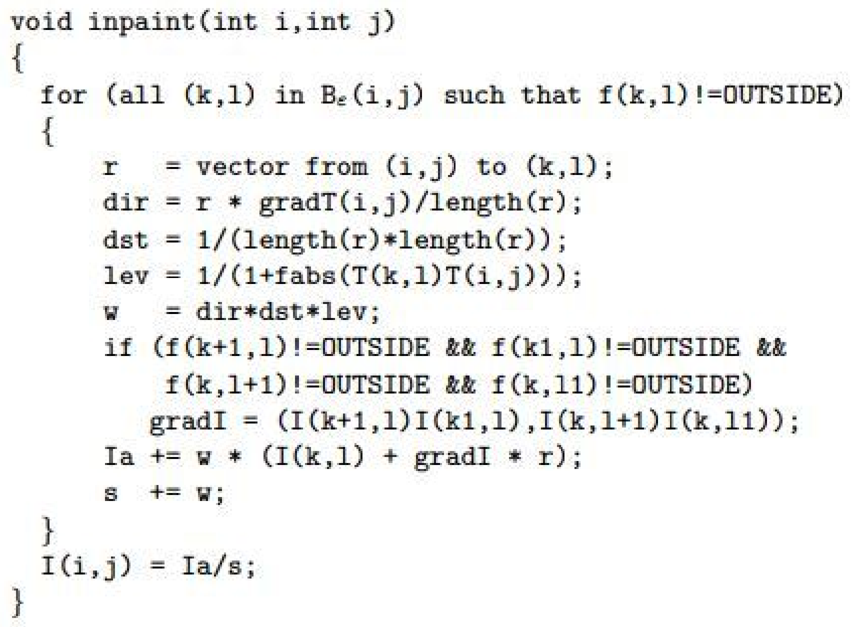
Figure 1:

To compute distance of new pixel from boundary, boundary distance of known pixels closest to this pixel in each of the 4 directions is considered.

```
float solve(int i1,int j1,int i2,int j2)
{
  float sol = 1.0e6;
  if (f(i1,j1)==KNOWN)
    if (f(i2,j2)==KNOWN)
    {
      float r = sqrt(2(T(i1,j1)T(i2,j2))*(T(i1,j1)T(i2,j2)));
      float s = (T(i1,j1)+T(i2,j2)r)/2;
      if (s>=T(i1,j1) && s>=T(i2,j2)) sol = s;
      else
      { s += r; if (s>=T(i1,j1) && s>=T(i2,j2)) sol = s; }
    }
    else sol = 1+T(i1,j1));
  else if (f(i2,j2)==KNOWN) sol = 1+T(i1,j2));
  return sol;
}
```

Figure 2:

There are multiple factors considered for inpainting a single point. Image gradient is estimated using central differences. The directional component ensures that the contribution of the pixels close to the normal direction i.e., close to the FMM's information propagation direction, is higher than for those farther from N. The geometric distance component decreases the contribution of the pixels geometrically farther from pixel. The level set distance component ensures that pixels close to the contour through pixel contribute more than farther pixels.



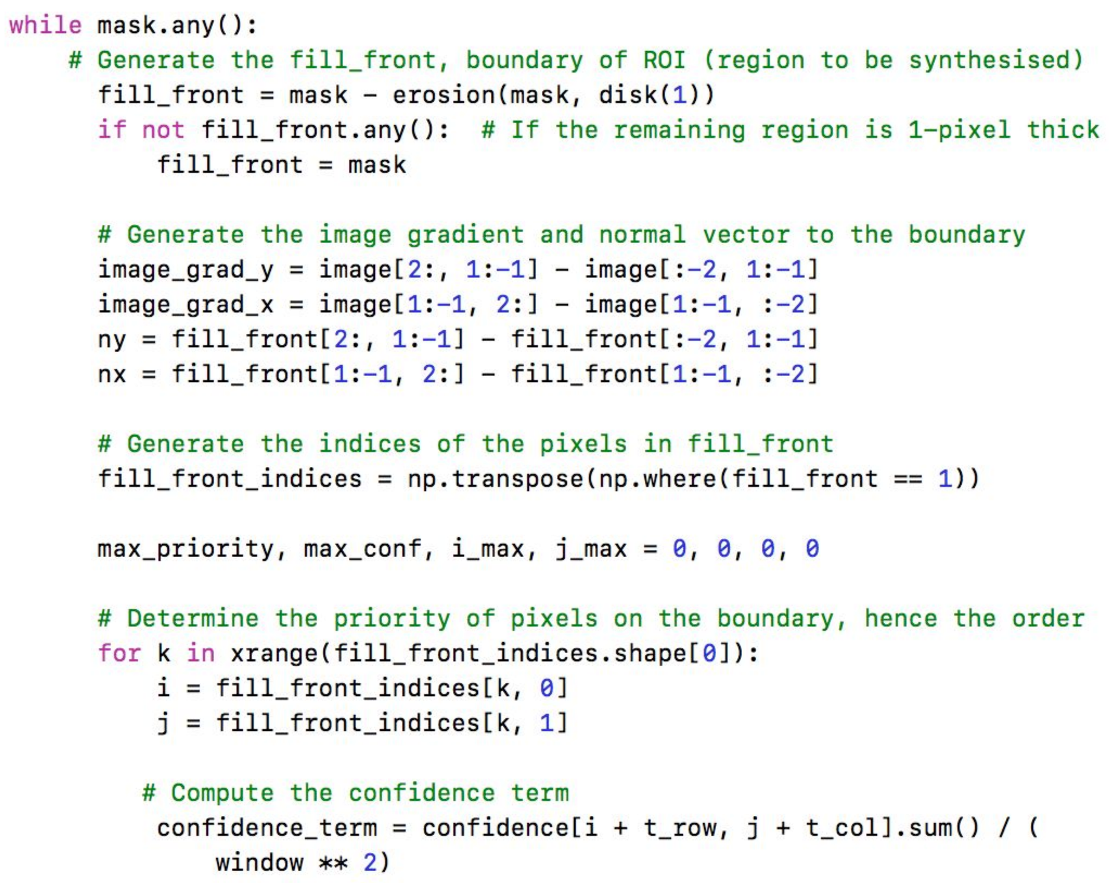
```
void inpaint(int i,int j)
{
  for (all (k,l) in Bε(i,j) such that f(k,l)!=OUTSIDE)
  {
      r   = vector from (i,j) to (k,l);
      dir = r * gradT(i,j)/length(r);
      dst = 1/(length(r)*length(r));
      lev = 1/(1+fabs(T(k,l)T(i,j)));
      w   = dir*dst*lev;
      if (f(k+1,l)!=OUTSIDE && f(k1,l)!=OUTSIDE &&
          f(k,l+1)!=OUTSIDE && f(k,l1)!=OUTSIDE)
         gradI = (I(k+1,l)I(k1,l),I(k,l+1)I(k,l1));
      Ia += w * (I(k,l) + gradI * r);
      s  += w;
  }
  I(i,j) = Ia/s;
}
```

Figure 3:

This algorithm is designed for one channel only. For color images with rgb channels, we split the image and inpaint each channel separately.

2. ***Exemplar based image inpainting***:

The image patch centered about the x and y points is computed. This patch is copied and pasted onto the target image. Patch with the highest priority is computed by multiplying the confidence term and the data term and dividing their product by a coefficient alpha, usually alpha = 255.0. The algorithm iterates through all the patches centered at a pixel on the boundary of the unfilled region to find the patch with the highest priority. The image pixels are split in its corresponding RGB color values by flattening the pixels. We then compute the patch distance which is given as the sum of the squared difference between the color values. The best exemplar patch is selected to inpaint the image based on the patch distance. The patch which has a minimum distance from the boundary of the image is selected. Finally, the confidence values are updated.



```
while mask.any():
  # Generate the fill_front, boundary of ROI (region to be synthesised)
   fill_front = mask - erosion(mask, disk(1))
   if not fill_front.any():  # If the remaining region is 1-pixel thick
       fill_front = mask

   # Generate the image gradient and normal vector to the boundary
   image_grad_y = image[2:, 1:-1] - image[:-2, 1:-1]
   image_grad_x = image[1:-1, 2:] - image[1:-1, :-2]
   ny = fill_front[2:, 1:-1] - fill_front[:-2, 1:-1]
   nx = fill_front[1:-1, 2:] - fill_front[1:-1, :-2]

   # Generate the indices of the pixels in fill_front
   fill_front_indices = np.transpose(np.where(fill_front == 1))

   max_priority, max_conf, i_max, j_max = 0, 0, 0, 0

   # Determine the priority of pixels on the boundary, hence the order
   for k in xrange(fill_front_indices.shape[0]):
       i = fill_front_indices[k, 0]
       j = fill_front_indices[k, 1]

      # Compute the confidence term
       confidence_term = confidence[i + t_row, j + t_col].sum() / (
           window ** 2)
```

Figure: Pseudo code for exemplar based image inpainting.

A. Design Issues:

1. ***An Image Inpainting Technique Based on the Fast Marching Method:***

Speed: Main issue with this implementation was speed of execution. In the beginning stages of development, even synthetic grayscale images were taking considerable time for completion. This was not desirable for code development and testing. As this is a computation heavy implementation, we had to develop code in Cython which drastically improved the performance.

Padding: While implementing inpainting for corner pixels, the algorithm tries to access pixels that are out of the boundary. This results in an error. To avoid this, we pad our original image with zero values on all sides and inpaint this modified image. We remove this padding while displaying results

2. ***Exemplar based image inpainting***:

For the scope of the project, mouse selection and mask generation were not implemented. Since this algorithm involves heavy computation, the code is written in cython for faster computation. The boundary of the image could not be computed when holes became small which greatly damaged the output image. Also, many a times, the exemplar algorithm selected a wrong patch to be copied to the target region. This issue was resolved by using variance calculation and

measuring the distance as SSD. For computing the patch distance, the unfilled pixels are ignored to reduce the error rate.
Initially, the gradient of the image is set to zero and maximum gradient magnitude in the patch is computed.

B. Running the code:

1. ***An Image Inpainting Technique Based on the Fast Marching Method:***

- Navigate to the target folder .../fmm_code in the command prompt and run $python setup.py build_ext --inplace
- Enter "python inpaintfmm.py [image name]" to run program with image input
- press '**m**' to start detecting mouse actions. After this, click anywhere on 'masked image' window and start drawing mask on it
- "Press '**o**' to display output"
- "Press '**h**' to display output"

2. ***Exemplar based image inpainting***:

- Navigate to the target folder .../exemplar_code in the command prompt and run $python setup.py build_ext --inplace
- And then run $python main.py

IV. Results and discussion

1. **General Application**

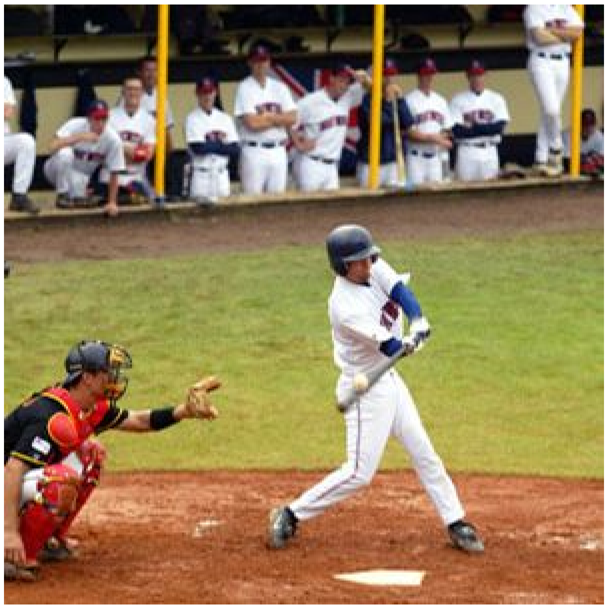


Figure: input image

1. **Object removal**

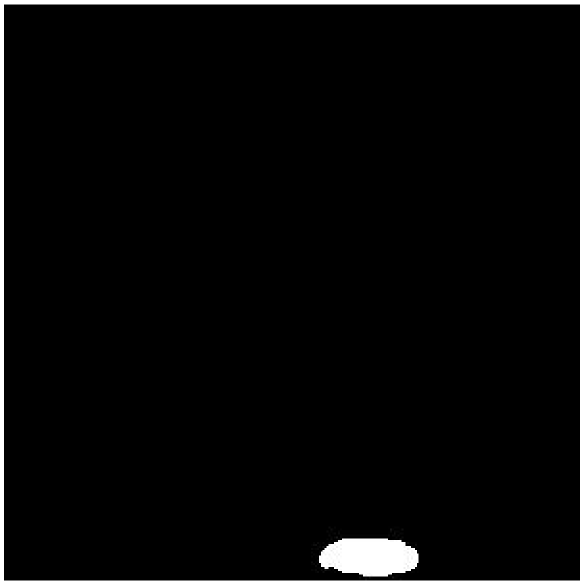


Figure: Mask

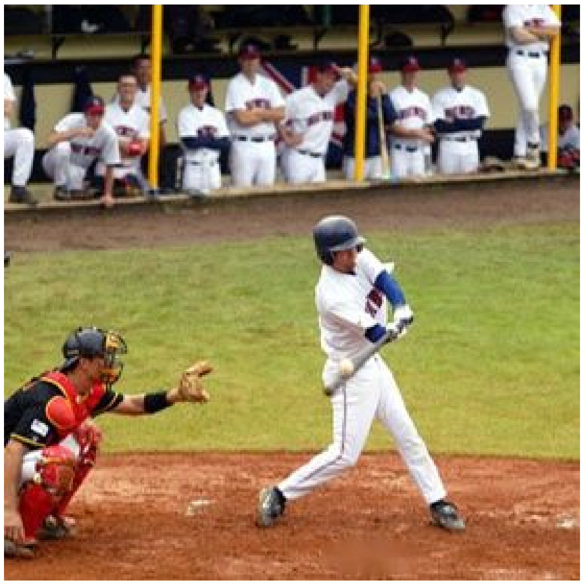


Figure: inpainted image

As we can see, the base is removed from the resultant image. Information around mask area is used to fill inside area.

2. **Image Reconstruction**

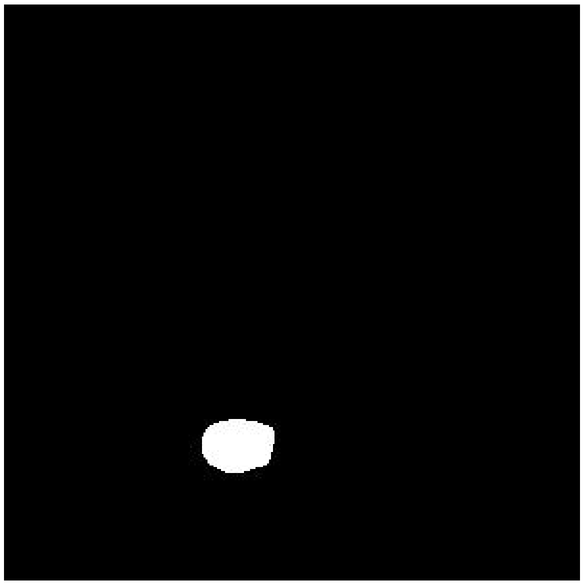


Figure: Mask

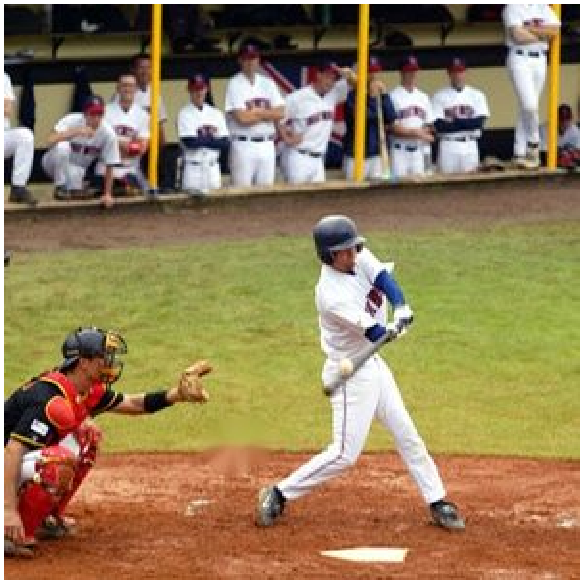


Figure: inpainted image

The separation of grass and ground is maintained inside masked area.

2. Exemplar based image inpainting:

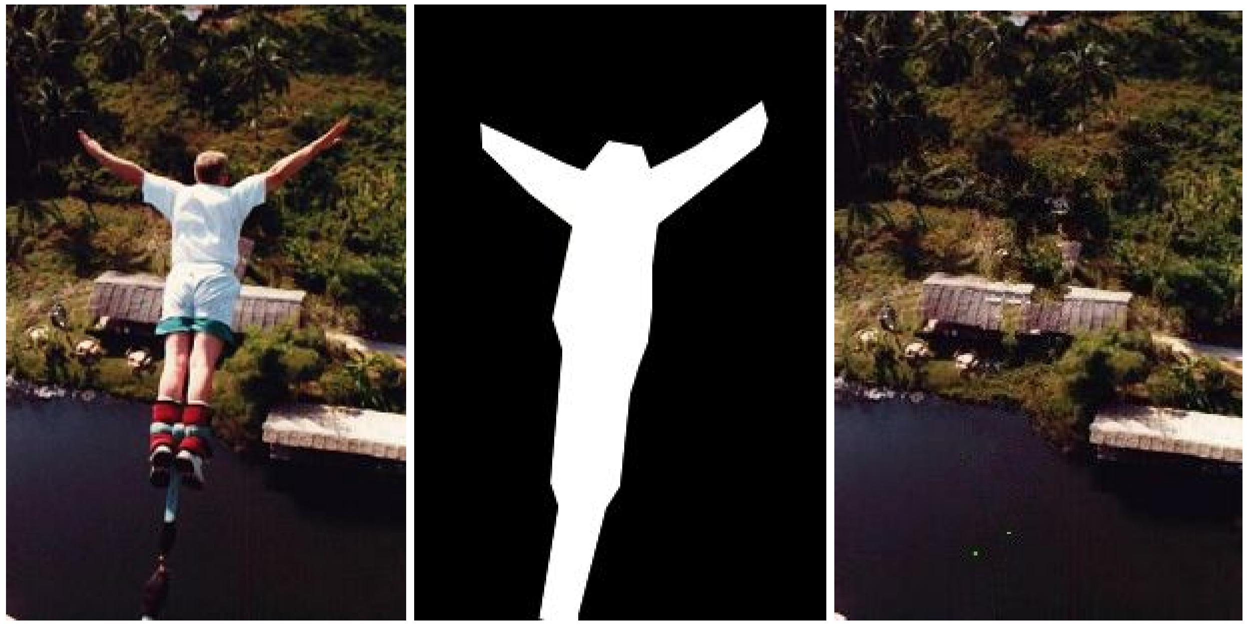


Figure: (a) Input Image. (b) Mask image (c) Output image.

2. Comparison

To measure the effectiveness of our implementation, the results obtained by the implementation of our proposed algorithms are compared with the built-in inpainting functions. The openCV has two built-in functions for inpainting - Navier stokes method and fast marching method.

1. *Navier stokes method:* This methompares image inpainting problem to an incompressible fluid problem. Algorithm for image inpainting is directly related to Navier-Stokes (NS) equations for an incompressible fluid. The NS method introduces ideas from fluid dynamics into the areas of computer vision and image analysis. This method has an immediate advantage due to the availability of well developed numerical and theoretical results. The Navier stokes algorithm is enable by using the flag, cv2.INPAINT_NS in opencv.
2. *Fast marching method:* The fast marching method (FMM), an image inpainting method, reconstructs images without the implementation of numerically elaborate methods. The FMM method can be enable in openCV directly by enabling the flag cv2.INPAINT_TELEA.

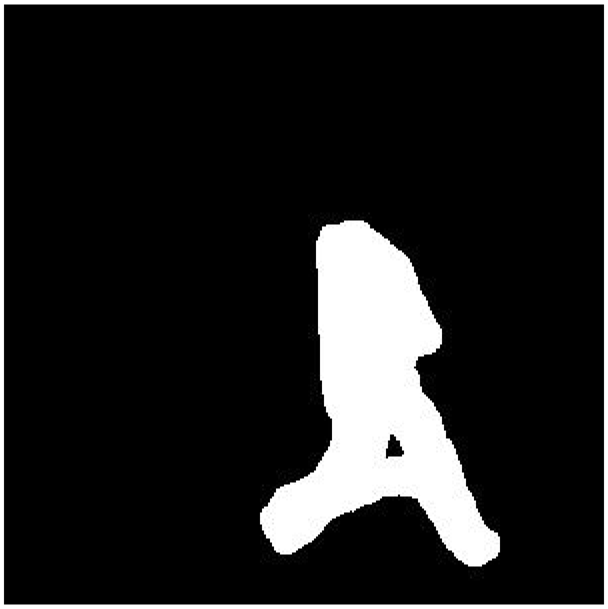


Figure: mask

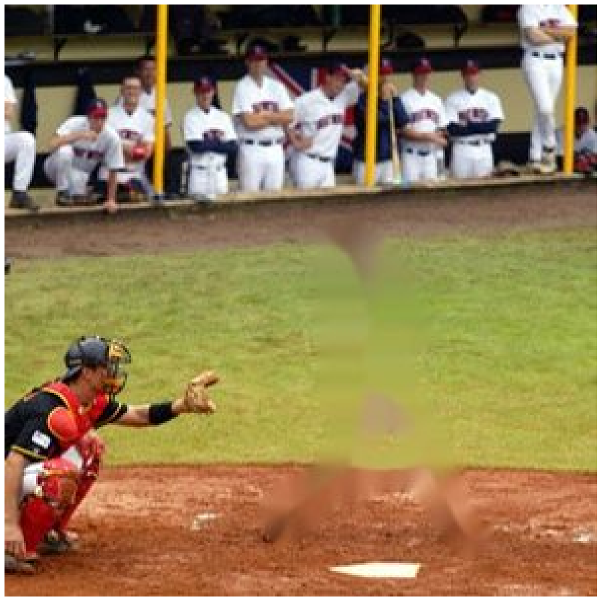


Figure: our implementation

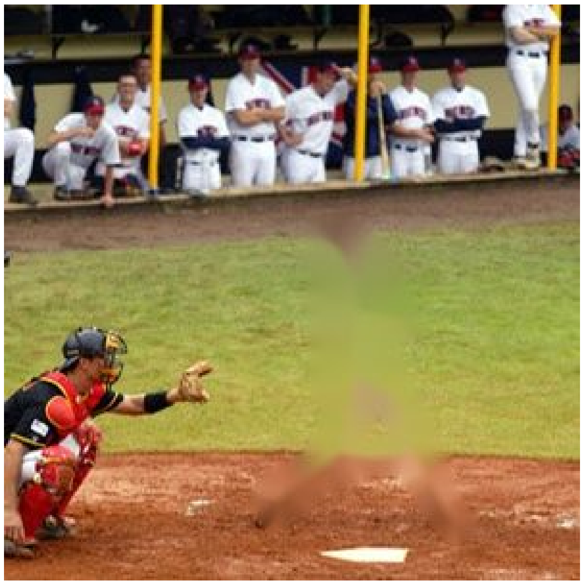


Figure: FMM implementation

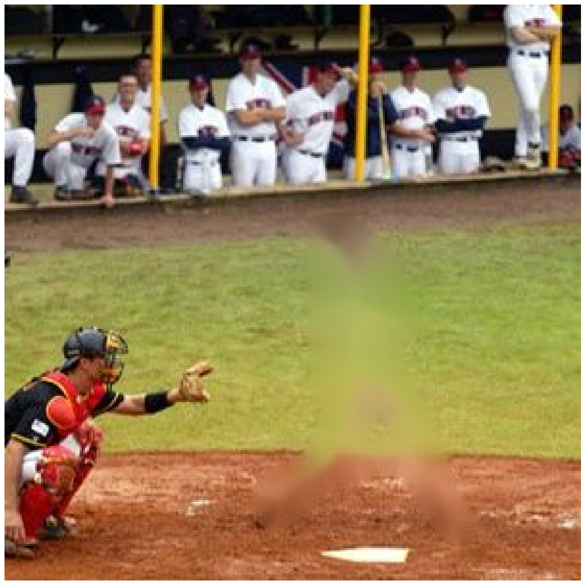


Figure: Navier stokes implementation

As we can see, the results of our implementation match significantly with the implementation using inbuilt opencv functions.

3. **Effect of parameters:**

The only changeable parameter in the implementation of FMM is radius. This decides the distance to look for information farther from a pixel to inpaint given pixel. If radius is more, pixels that are far away also contribute to the inpainting of given pixel. This affects the smoothness of the inpainted region. For optimum results radius is set to 5. This can easily be changed in the source code

.

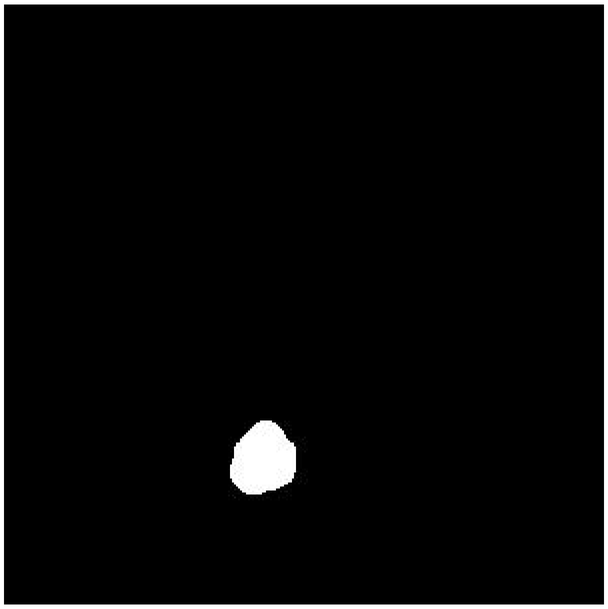
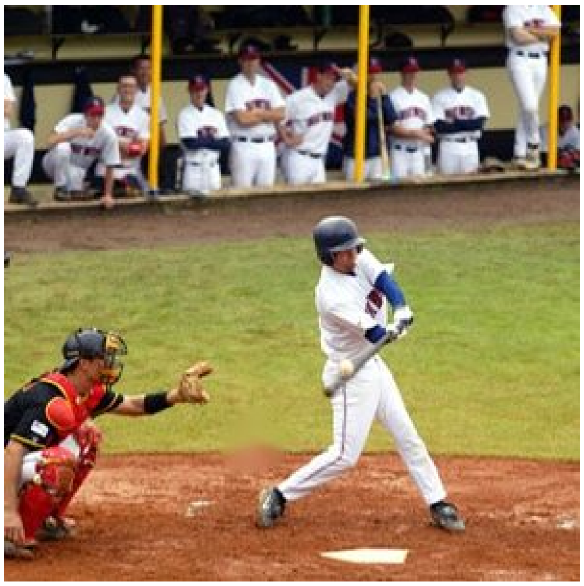


Figure: Mask



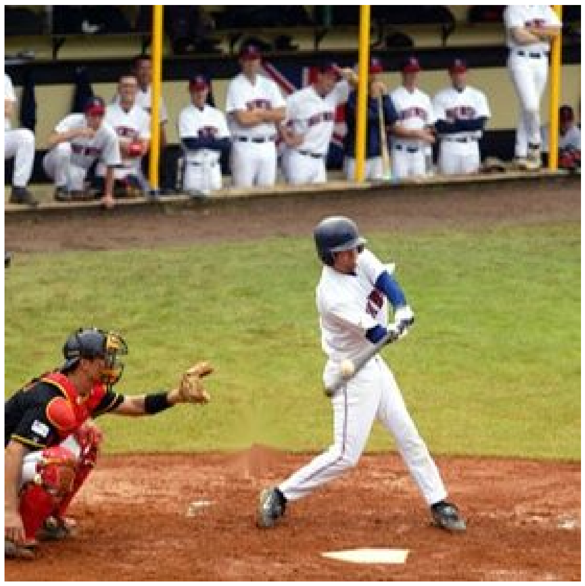


Figure:inpainted image(radius = 10) **Figure:inpainted image(radius = 2)**

Figure shows inpainting of same mask with different radiuses. Notice that inpainting with radius 10 is smoother than that with 2.

4. **Shortcomings**

a. **Of our implementation**

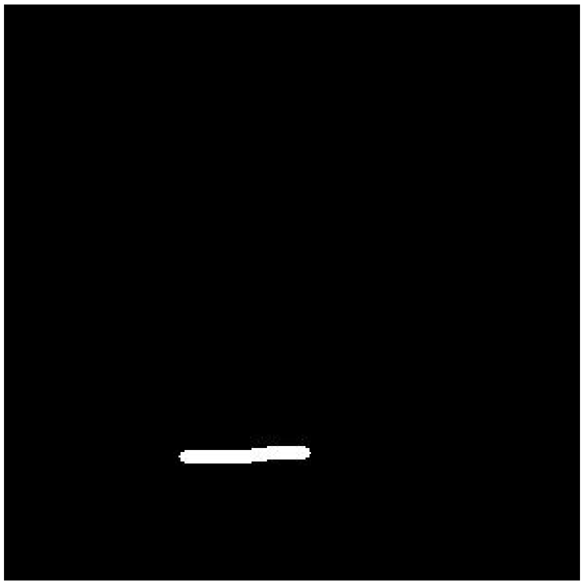


Figure: Mask

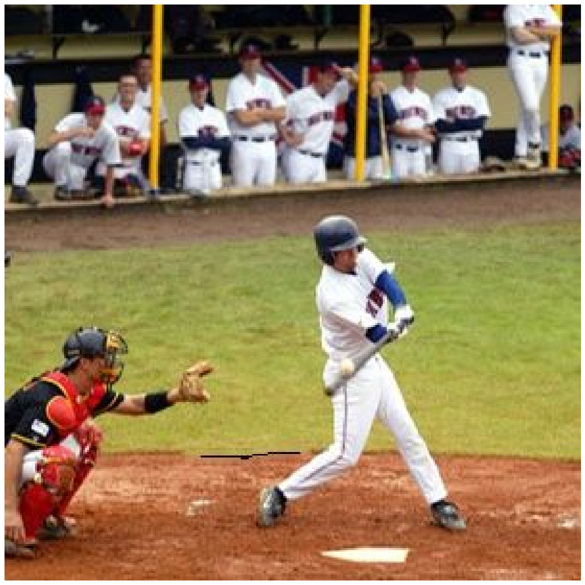


Figure: inpainted image

If the mask is thin, a small part of the image is not inpainted. This problem is related to radius of structuring element used during initialization of FMM.

b. Of inpainting using FMM

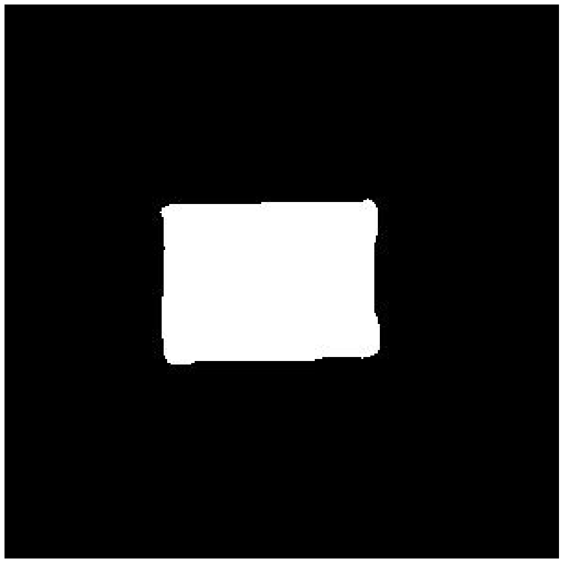


Figure: Mask

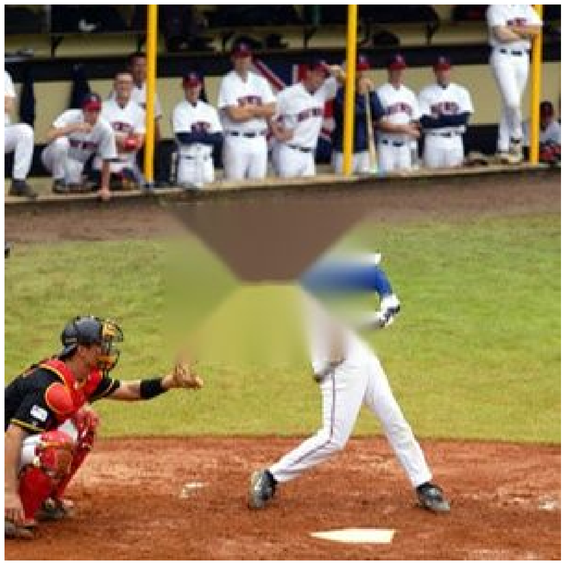


Figure: Inpainted image

If the mask does not have continuity in curve, boundary is propagated from each side and produces unnatural results along their common line of propagation. Similar results are observed using inbuilt function

5. Future Development

- Create an user interface for selecting image region and generating mask automatically.
- Resolve the issue of selecting the correct exemplar patch.
- Improve isophote calculations for defining sharper edges.
- Resolve the problem of smaller holes in the image.
- White box testing of boundary normal computation.

V. References

1. Telea, Alexandru. "An Image Inpainting Technique Based on the Fast Marching Method" Journal of graphics tools 9.1 (2004)
2. Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro. "Navier-stokes, fluid dynamics, and image and video inpainting." In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1, pp. I-355. IEEE, 2001.
3. A. Criminisi, P. Perez, K. Toyama, "Region filling and object removal by exemplar-based inpainting", IEEE Transactions on Image Processing, Vol. 13, No. 9, pages 1200-1212, 2004.
4. M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In Proc. ACM Conf. Comp. Graphics (SIGGRAPH), pages 417–424, New Orleans, LU, July 2000. http://mountains.ece.umn.edu/ ~guille/inpainting.htm.
5. R. Bornard, E. Lecan, L. Laborelli, and J-H. Chenot. Missing data correction in still images and image sequences. In ACM Multimedia, France, December 2002.
6. H. Igehy and L. Pereira. Image replacement through texture synthesis. In Proc. Int. Conf. Image Processing, pages III:186–190, 1997.
7. M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. "Simultaneous structure and texture image Inpainting." In Proc. Conf. Comp. Vision Pattern Rec., Madison, WI, 2003.
8. Given Cython tutorial
9. https://docs.opencv.org/3.2.0/df/d3d/tutorial_py_inpainting.html
10. https://en.wikipedia.org/wiki/Inpainting
11. http://www.ifp.illinois.edu/~yuhuang/inpainting.html
12. http://math.berkeley.edu
13. http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_reconstruction_Inpainting_Interpolation.php
14. Stackoverflow.com
15. https://docs.opencv.org/3.0-beta/