Assignment 2: Computer Vision and Pattern Recognition

Wei Huang

1 Problem

Checking code: Running *ex1.m* in ex1 folder.

- Explicit scheme:
  1. Convert input of RGB into gray with range of $[0, 1]$
  2. Computing diffusion term for each pixels by calling central difference function *gradient* in Matlab.
  3. Applying central difference function *gradient* again for $c(x, y; t)\nabla I(x, y; t)$, and compute divergence.
  4. Forward Euler formula update image in next time.
- Semi-implicit scheme:
  1. Convert input of RGB into gray with range of $[0, 1]$
  2. Computing diffusion term for each pixels by calling central difference function *gradient* in Matlab.
  3. Computing tridiagnal matrix along x and y by calling *aos.m* function.
  4. Applying Thomas algorithm to solve linear system by calling function *Thomas.m* which is implemented by myself.

Conclusion: The stability of explicit scheme is dependent by the value of Δt. When $\Delta t = 5$, explicit scheme does not converge and image become unstable. However for semi implicit scheme, convergence is independent with Δt. Besides, when we use larger K, the diffusion term will beome larger and the filtered image will beome more blurred.

Figure 1: Original Image

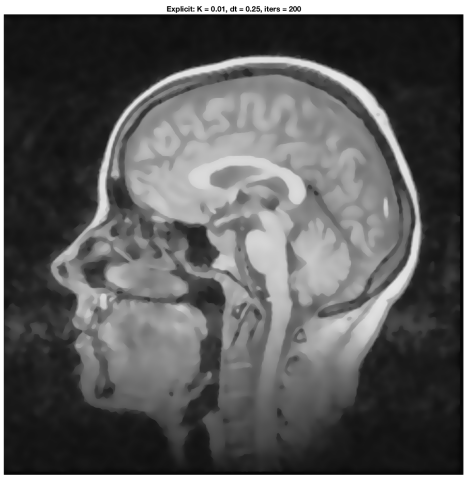

Figure 2: Explicit: k = 0.01, dt = 0.25, iters = 200

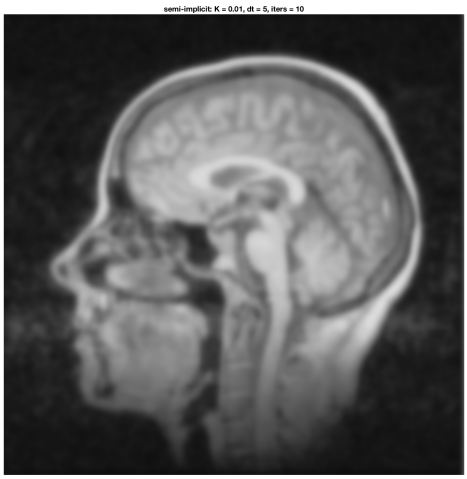

Figure 3: Implicit: k = 0.01, dt = 5, iters = 10

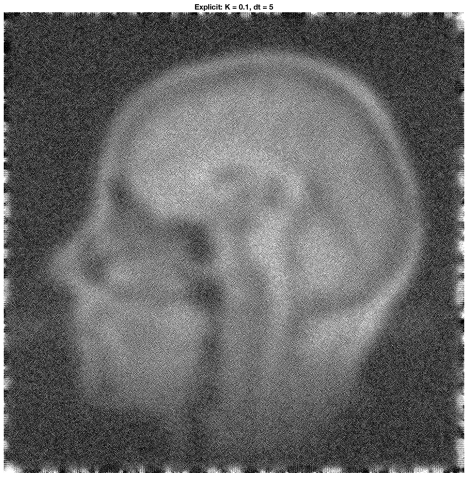

Figure 4: Explicit: k = 0.1, dt = 5, iters = 200

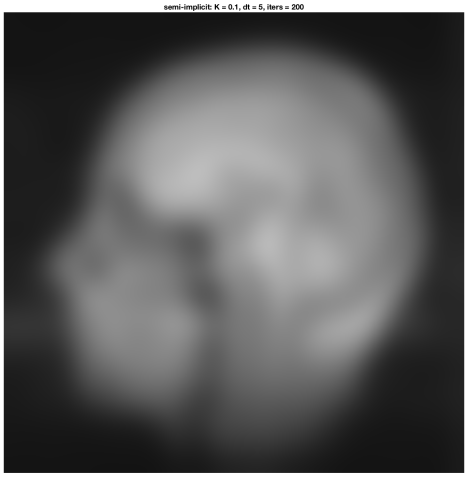

Figure 5: Implicit: k = 0.1, dt = 5, iters = 200

2 Problem

Checking code: Running *ex2.m* in ex2 folder.

- a)

$$\begin{aligned} F(f, f_x, f_y) &= \int_\Omega \mathcal{I}_{\Omega/\Omega_0}(f(x,y) - g(x,y))^2 dxdy + \lambda \int_\Omega (f_x^2 + f_y^2 + \epsilon)^{1/2} dxdy \\ &= \int_\Omega \mathcal{I}_{\Omega/\Omega_0}(f(x,y) - g(x,y))^2 + \lambda (f_x^2 + f_y^2 + \epsilon)^{1/2} dxdy \end{aligned}$$

and we have:

$$\frac{\partial F}{\partial f} = 2\mathcal{I}_{\Omega/\Omega_0}(f(x,y) - g(x,y)) \qquad \frac{\partial F}{\partial f_x} = \lambda \frac{f_x}{\sqrt{f_x^2 + f_y^2 + \epsilon}} \qquad \frac{\partial F}{\partial f_y} = \lambda \frac{f_y}{\sqrt{f_x^2 + f_y^2 + \epsilon}}$$

By E-L equation,

$$\frac{\partial F}{\partial f} - \frac{\partial F}{\partial x \partial f_x} - \frac{\partial F}{\partial y \partial f_y} = 0$$

we have:

$$2\mathcal{I}_{\Omega/\Omega_0}(f(x,y) - g(x,y)) - \lambda div\left(\frac{\nabla f}{(\|\nabla f\|^2 + \epsilon)^{1/2}}\right) = 0$$

- b)

We could design the following update formula:

$$\frac{\partial f(x,y;t)}{\partial t} = \lambda div\left(\frac{\nabla f}{(\|\nabla f\|^2+\epsilon)^{1/2}}\right) - 2\mathcal{I}_{\Omega/\Omega_0}(f(x,y;t)-g(x,y;t))$$

where stable f(x,y,t) satisfy E-L equation. Because:

$$\frac{\partial F}{\partial x \partial f_x} = \frac{f_{xx}f_y^2 - f_x f_y f_{yx} + \epsilon f_{xx}}{(f_x^2+f_y^2+\epsilon)^{3/2}}$$

$$\frac{\partial F}{\partial y \partial f_y} = \frac{f_{yy}f_x^2 - f_x f_y f_{xy} + \epsilon f_{yy}}{(f_x^2+f_y^2+\epsilon)^{3/2}}$$

and

$$div\left(\frac{\nabla f}{(\|\nabla f\|^2+\epsilon)^{1/2}}\right) = \frac{f_{xx}f_y^2 - 2f_{xy}f_x f_y + f_{yy}f_x^2 + \epsilon(f_{xx}+f_{yy})}{(f_x^2+f_y^2+\epsilon)^{3/2}}$$

By discretizing we have:

$$\begin{aligned} f(x,y;t+\Delta t) =& f(x,y;t) + \Delta t\lambda \frac{f_{xx}f_y^2 - 2f_{xy}f_x f_y + f_{yy}f_x^2 + \epsilon(f_{xx}+f_{yy})}{(f_x^2+f_y^2+\epsilon)^{3/2}} \\ & - 2\Delta t\mathcal{I}_{\Omega/\Omega_0}(f(x,y;t)-g(x,y;t)) \end{aligned}$$

- c) Alghough increated time step Δt could speed up our algorithm, our algorithm will be more likely to be unstable. In this experiment, when $\Delta t < 0.7$, the algorithm can guarantee convergence. If I chose $\Delta t \geq 0.7$, Our implicit scheme will become unstable.

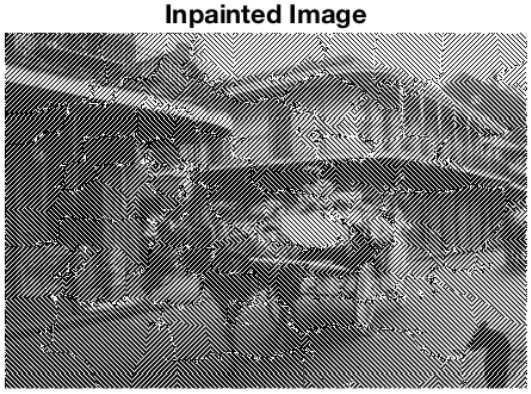

Figure 6: $\epsilon = 1.0, \Delta t = 0.7, \lambda = 1.0$, iters $= 500$

Theoretically speking, in my case, I chose $\epsilon = 1$ so diffusion coefficients $\frac{1}{(\|f\|^2+\epsilon)^{1/2}} < 1$ in our PDE. As we known, for 2D heat equation with constant diffusion coefficients 1:

$$f_t = div(1 \cdot \nabla f)$$

Implicit scheme is stable when $\Delta t < 0.25$. Thus, we could conclude when $\Delta t < 0.25$, we can be sure about stability of implicit scheme. In my following experiment, I chose $\Delta t = 0.2$. If we use smaller ϵ, we should chose little smaller Δt in order to make sure stability.

- d) Please see *ex2.m*

  Parameters: $\epsilon = 1.0, \Delta t = 0.2, \lambda = 1.0$, number of iterations = 500

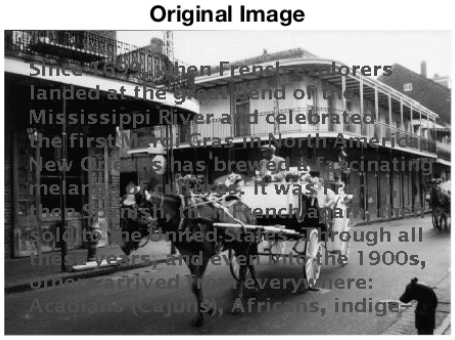

Figure 7: Painted Image

Figure 8: Inpainted Image

3 Problem

Checking code: Running *ex3.m* in ex3 folder.

- a)
  - Convert RGB image into gray image.
  - Smoothing original image with Gaussian kernel of size of 15×15 with $\sigma = 2.0$. And compute g by the following formula:

$$g = \frac{1}{1 + \|\nabla(G_\sigma * I)\|^2}$$

Figure 9: image of g

– initialize Level set function ϕ with signed distance function by using *bwdist* and *imfill* in Matlab. When $\phi_0 = 0$, the curve is rectangle.

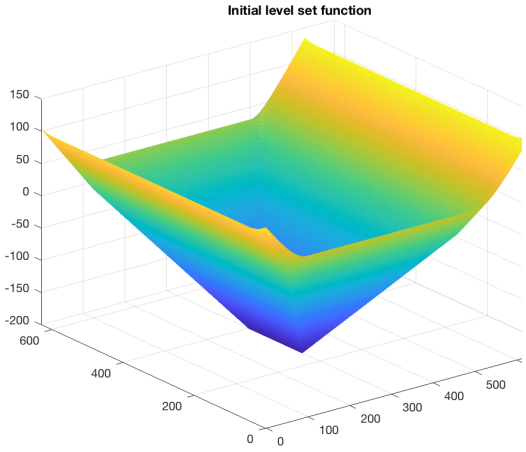

Figure 10: Initial level set function

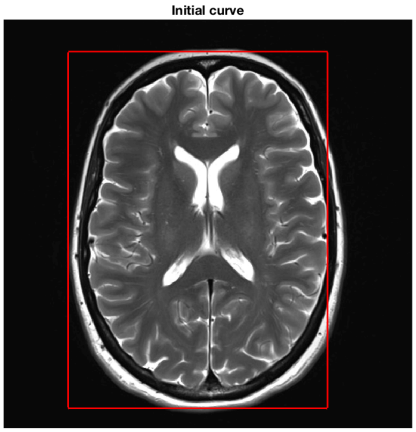

Figure 11: Initial curve

– Apply AOS method to solve diffusion equation:

$$\phi_t = div(g(\nabla I)\nabla\phi)$$

– Do level set re-distancing by solving ODE every 10 iterations which is implemented by *redistance_phi.m*:

$$\phi_t = sign(\phi_0)(1 - \|\nabla\phi)\|$$

The final result is stated below after 120 iterations:

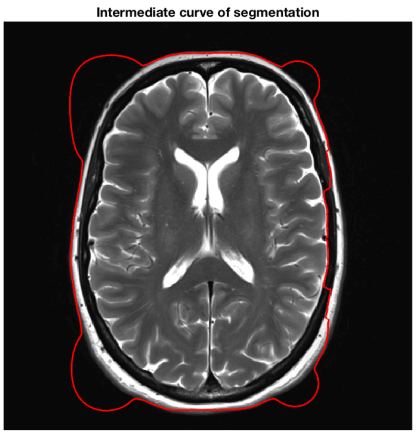

Figure 12: Intermediate image

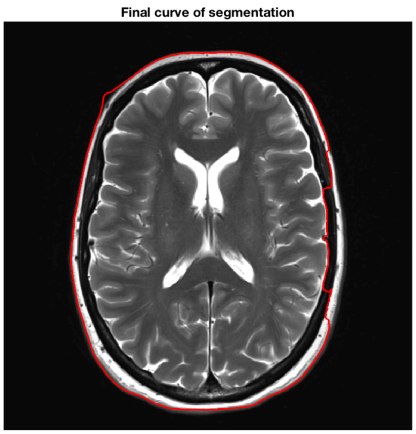

Figure 13: Final image

and the final level set function is below:

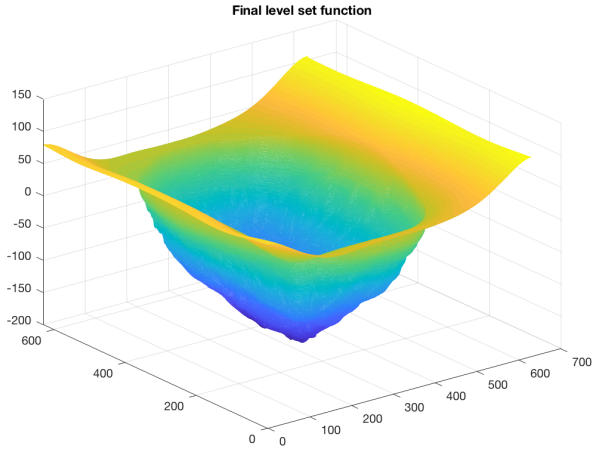

Figure 14: Final level set function

- b) for each iteration with AOS methof it will take about 0.02 second. In my opinion, we could use Gaussian Seidel method to solve diffusion PDE. Because the corresponding matrix is sparse so time complexity is still linear function of the number of pixels. Besides it is easy to implement and does not require additional memory.

4 Seam Carving

Checking code: Running *seam_carving.m* in seam carving folder.
Seam carving is technique which can resize image without distortion. I summarize procedure below:

- Compute the energy value in each pixels, usually use normalized norm of gradient.

$$E(i,j) = \|\nabla I\|_2 \,/\, \max_{i,j}(\|\nabla I(i,j)\|_2)$$

- Using the dynamic programming method to find the path from left to right or from top to bottom along which the sum of energy is minimal. Dynamic programming is implemented by *dynamic_find.m*

$$T(i,j) = E(i,j) + min(T(i-1,j-1), T(i-1,j), T(I-1,j+1))$$

  where T is the cumulative minimum energy matrix.

- Finding the seam with minimal energy and delete it. For example:

Figure 15: The first detected seam

- repeat until you get image with desired size.

I would like to convert original image of size of 186 $\times$274 into size of 186 $\times$174. The main Matlab function is inside in *seam_carving.m*, and the result is stated below:

Figure 16: Original image with size of 186 ×274

Figure 17: Seam carved image with size of 186 × 174

5 Possion image editing

Checking code: Running *possion_editing.m* in possion editing folder.

Possion image editing can be summarized as the following minimization problem:

$$\min_f \int\int_\Omega \|\Delta f - v\|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Where v is vector filed of to be cloned image. By E-L equation,

$$f_{xx} - \frac{\partial v_1}{\partial x} + f_{yy} - \frac{\partial v_2}{\partial y} = 0$$

Thus, the original problem is equivalent to the following problem:

$$\Delta f = div(v) \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

By discretizing, we have the following linear equation system:

$$f_{i+1,j} + f_{i,j+1} + f_{i-1,j} + f_{i,j-1} - 4f_{i,j} = v_{i,j}$$

In order to solve this linear system, we could use Gauss-Seidel iteration method:

$$f_{i,j}^{n+1} = \frac{f_{i+1,j}^n + f_{i,j+1}^n + f_{i-1,j}^n + f_{i,j-1}^n - v_{i,j}}{4}$$

In order to improve performance on boundary of interpolated domain Ω. I implement a weight template which linearly decay to 0 on boundary and value 1 in interior point. The plot is stated below:

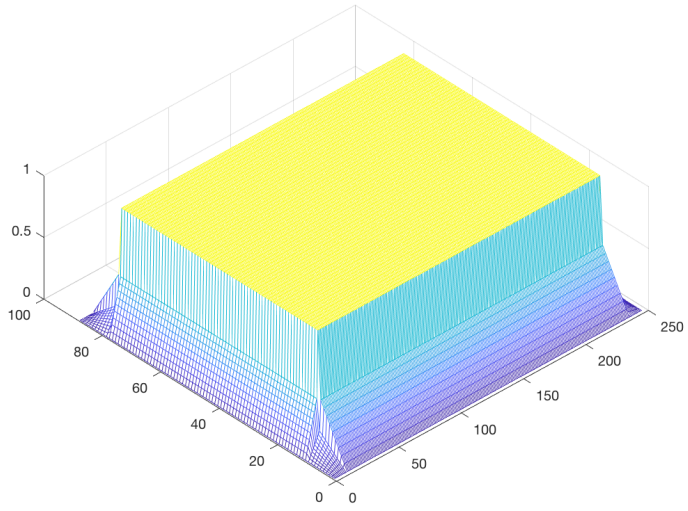

Figure 18: weight template

and then we take weighted average of f and orginal image region in target image which could smooth boundary:

$$f = temp. * f + (1 - temp). * origin;$$

The main Matlab function is inside in *possion_editing.m*, and result is the follwoing after 10000 iterations:

Figure 19: cloning

Figure 20: seamless cloning