

Isotropic Diffusion via Heat Equation

import library

In [68]:

```
import numpy as np
import matplotlib.image as img
import matplotlib.pyplot as plt
import matplotlib.colors as colors
```

load image

In [69]:

```
l_color = img.imread('ccc.jpg') # load color image
```

check the size of image and convert 3D to 2D if necessary

In [70]:

```
def rgb2gray(rgb):
    r, g, b = rgb[:, :, 0], rgb[:, :, 1], rgb[:, :, 2]
    gray = 0.3333 * r + 0.3333 * g + 0.3333 * b
    return gray

n_dimension = l_color.shape[2] # number of dimension of data
print(l_color.shape[2])

if n_dimension == 3:
    l = rgb2gray(l_color)

n_dimension = 2 # number of dimension of the converted data
n_row       = l.shape[1] # number of rows of data
n_column    = l.shape[0] # number of columns of data

print("number of dimension = ", n_dimension)
print("number of rows = ", n_row)
print("number of columns = ", n_column)
```

```
plt.figure(figsize=(16,8))

plt.subplot(121)
plt.imshow(l_color)
plt.title('input image in color')
plt.axis('off')

plt.subplot(122)
plt.imshow(l, cmap = 'gray')
plt.title('input image in grey')
plt.axis('off')
```

3

number of dimension = 2
number of rows = 563
number of columns = 541

Out[70]: (-0.5, 562.5, 540.5, -0.5)

input image in color



input image in grey



```
git commit -a -m "load image and check its size"
```

```
git push origin master
```

normalize input image so that the range of image is [0, 1]

```
In [71]:
```

```
I /= 255
```

```
git commit -a -m "normalise data"
```

```
git push origin master
```

define a function to compute the first-order derivatives with Neumann boundary

condition

In []:

```
In [72]: def compute_derivative_first_order(data, axis, scheme, boundary):

    if boundary == 'neumann':
        D = np.pad(data , ((1,1),(1,1)), 'constant', constant_values=0)    # use numpy.pad function

    if axis == 'x':
        if scheme == 'forward':
            derivative = np.roll(D ,1, axis=1 ) - D   # use numpy.roll function
        elif scheme == 'backward':
            derivative = np.roll(D,-1, axis=1 ) - D # use numpy.roll function
        else:    # scheme == central
            derivative = np.roll(D,-1, axis=1 ) - D   # use numpy.roll function
    elif axis == 'y':
        if scheme == 'forward':
            derivative = np.roll(D,1, axis=0 ) - D   # use numpy.roll function
        elif scheme == 'backward':
            derivative = np.roll(D,-1, axis=0 ) - D   # use numpy.roll function
        else:    # scheme == central
            derivative = np.roll(D,-1, axis=0 ) - D   # use numpy.roll function
    dD = derivative    # remove the first and the last rows and columns for the boundary condition
    return(dD)
```

```
git commit -a -m "define a function for computing the first-order derivative"
```

```
git push origin master
```

define a function to compute the second-order derivatives with Neumann boundary condition

In [73]:

```
def compute_derivative_second_order(data, axis):  
  
    D_forward = compute_derivative_first_order(data, axis, 'forward', 'neumann')  
    D_backward = compute_derivative_first_order(data, axis, 'backward', 'neumann')  
  
    dDdD = D_forward - D_backward # use the above D_forward and D_backward  
  
    return(dDdD)
```

```
git commit -a -m "define a function for computing the second-order derivative"
```

```
git push origin master
```

define a function to compute heat equation

In [74]:

```
def heat_equation(data, delta_t, number_iteration):  
  
    u = np.pad(data , ((1,1),(1,1)), 'constant', constant_values=0) # initialisation  
  
    for t in range(number_iteration):  
  
        laplace = compute_derivative_second_order(data, axis='x') +compute_derivative_second_order(data, axis='y') # use the above  
        u += delta_t*laplace # compute the heat equation u(t+1) = u(t) + delta_t * laplace(u)  
  
    return(u)
```

In [75]:

```
delta_t = 0.1 # delta_t in the computation of heat equation  
  
u10 = heat_equation(l, delta_t, 10) # compute heat equation with 10 iterations  
  
u100 = heat_equation(l, delta_t, 100) # compute heat equation with 100 iterations
```

```
u1000 = heat_equation(l, delta_t, 1000) # compute heat equation with 1000 iterations

plt.figure(figsize=(16,16))

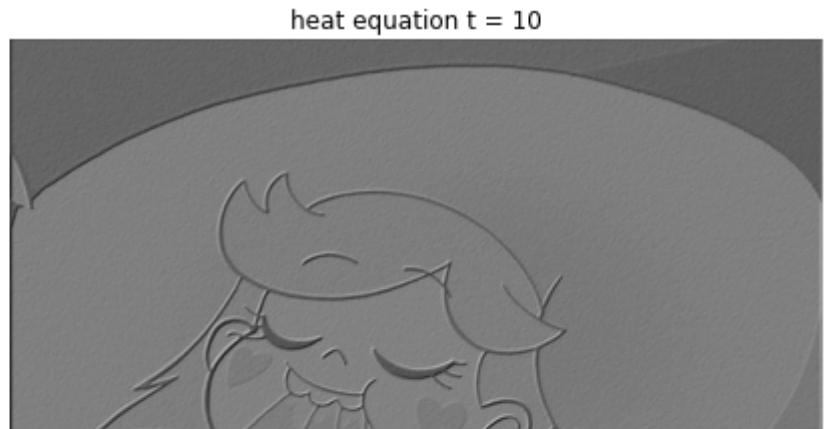
# visualise the original image
plt.subplot(221)
plt.imshow(l, cmap='gray')
plt.title('input image')
plt.axis('off')

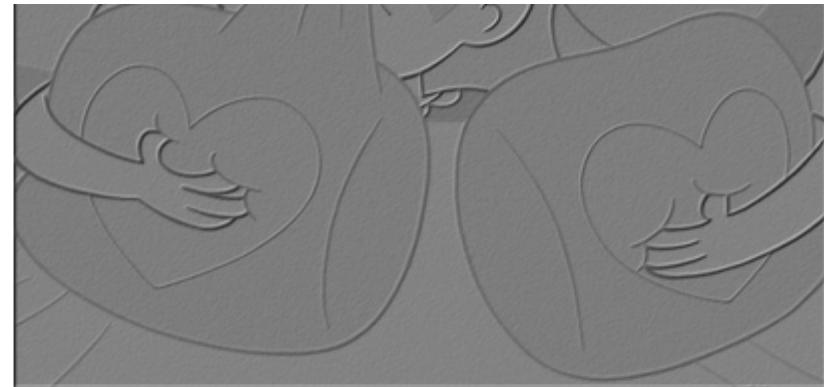
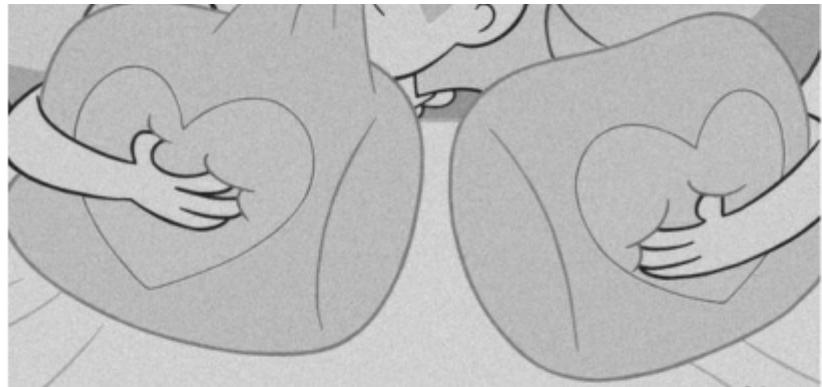
# visualise the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')
```

Out[75]: (-0.5, 564.5, 542.5, -0.5)





heat equation t = 100



heat equation t = 1000



git commit -a -m "compute the heat equation"

```
git push origin master
```

compute the (forward) first-order derivative of the solutions obtained by heat equation

```
In [77]:  
lx      = compute_derivative_first_order(l, 'x', 'forward', 'neumann') # compute l_x using forward scheme  
ly      = compute_derivative_first_order(l, 'y', 'forward', 'neumann') # compute l_y using forward scheme  
  
u10_x   = compute_derivative_first_order(u10, 'x', 'forward', 'neumann') # compute u(10)_x using forward scheme  
u10_y   = compute_derivative_first_order(u10, 'y', 'forward', 'neumann') # compute u(10)_y using forward scheme  
  
u100_x  = compute_derivative_first_order(u100, 'x', 'forward', 'neumann') # compute u(100)_x using forward scheme  
u100_y  = compute_derivative_first_order(u100, 'y', 'forward', 'neumann') # compute u(100)_y using forward scheme  
  
u1000_x = compute_derivative_first_order(u1000, 'x', 'forward', 'neumann') # compute u(1000)_x using forward scheme  
u1000_y = compute_derivative_first_order(u1000, 'y', 'forward', 'neumann') # compute u(1000)_y using forward scheme  
  
plt.figure(figsize=(16,16))  
  
# visualise the first-order derivative in x-axis of the original image  
plt.subplot(221)  
plt.imshow(lx, cmap='gray')  
plt.title('input image')  
plt.axis('off')  
  
# visualise the first-order derivative in x-axis of the results of heat equation with 10 iterations  
plt.subplot(222)  
plt.imshow(u10_x, cmap='gray')  
plt.title('heat equation t = 10')  
plt.axis('off')  
  
# visualise the first-order derivative in x-axis of the results of heat equation with 100 iterations  
plt.subplot(223)  
plt.imshow(u100_x, cmap='gray')  
plt.title('heat equation t = 100')  
plt.axis('off')  
  
# visualise the first-order derivative in x-axis of the results of heat equation with 1000 iterations  
plt.subplot(224)  
plt.imshow(u1000_x, cmap='gray')  
plt.title('heat equation t = 1000')  
plt.axis('off')
```

```
plt.figure(figsize=(16,16))

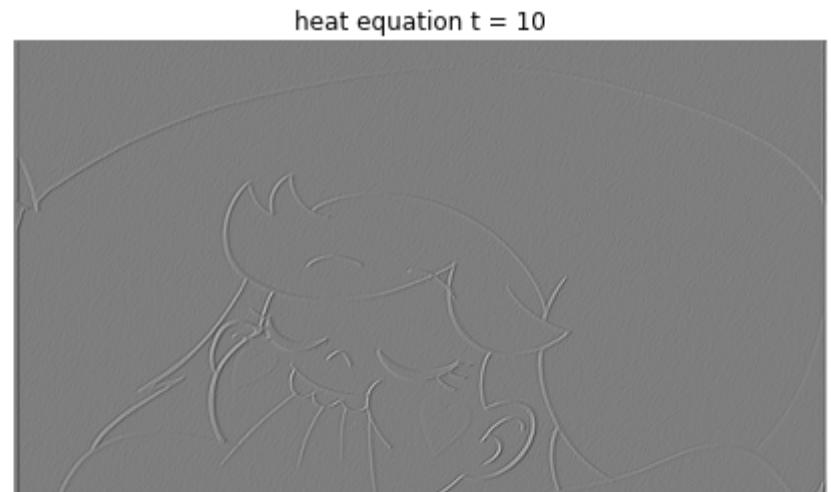
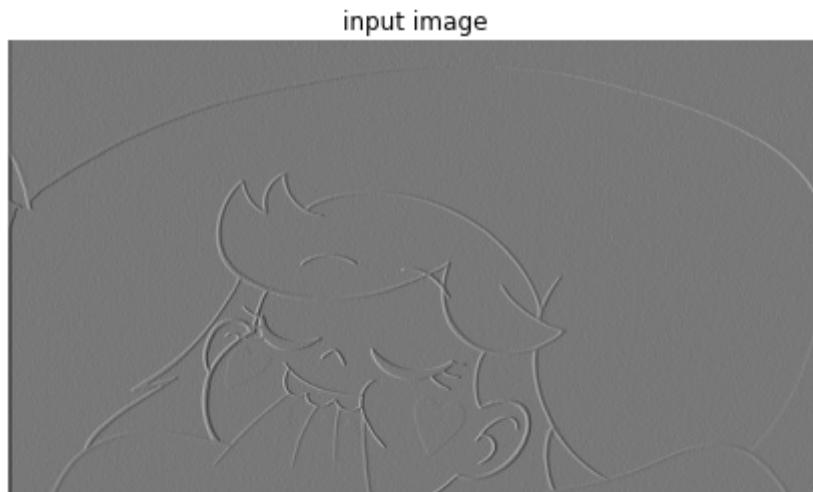
# visualise the first-order derivative in y-axis of the original image
plt.subplot(221)
plt.imshow(Iy, cmap='gray')
plt.title('input image')
plt.axis('off')

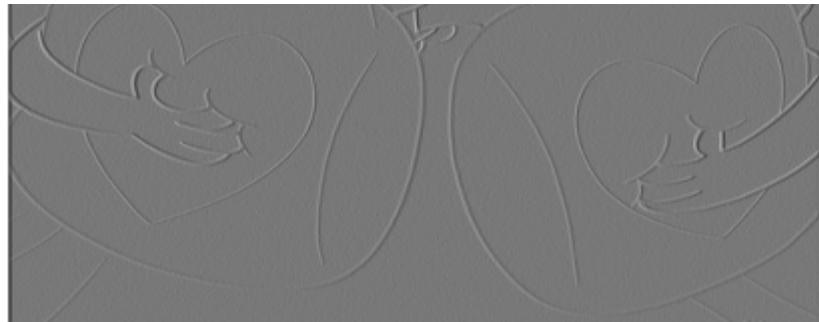
# visualise the first-order derivative in y-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_y, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the first-order derivative in y-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_y, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the first-order derivative in y-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_y, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')
```

Out[77]: (-0.5, 566.5, 544.5, -0.5)

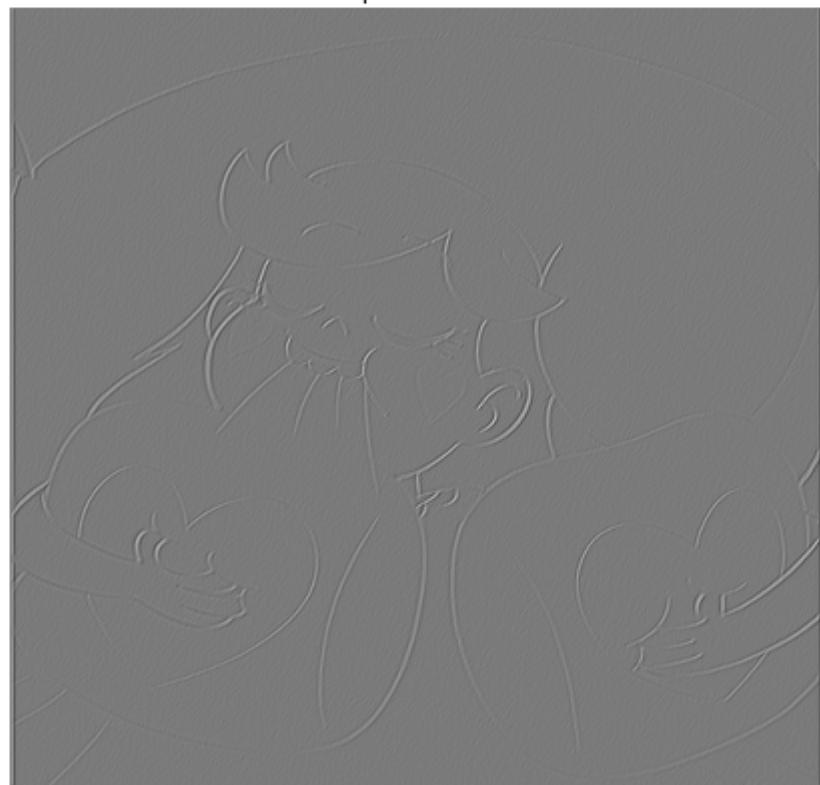




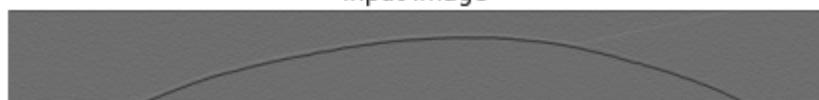
heat equation $t = 100$



heat equation $t = 1000$

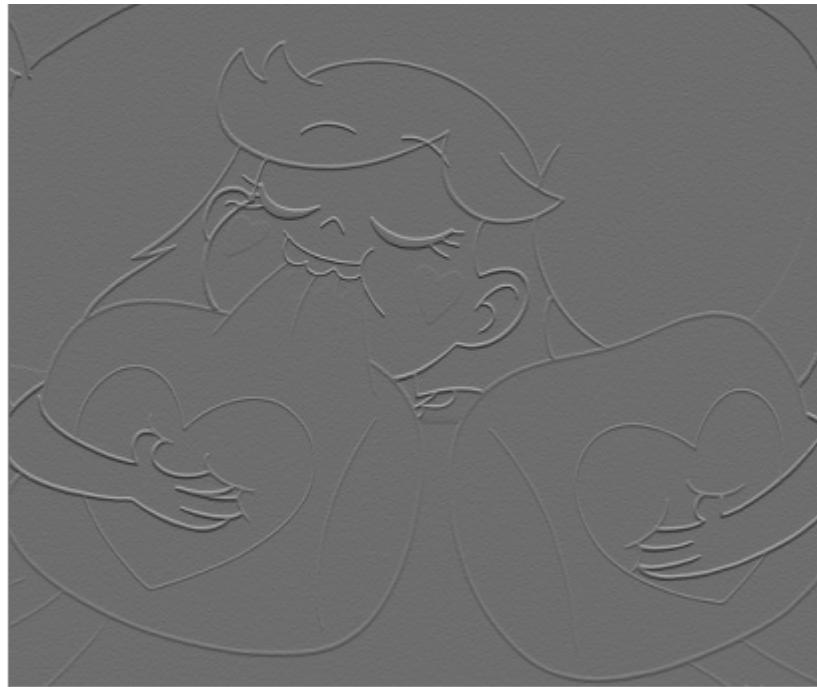


input image

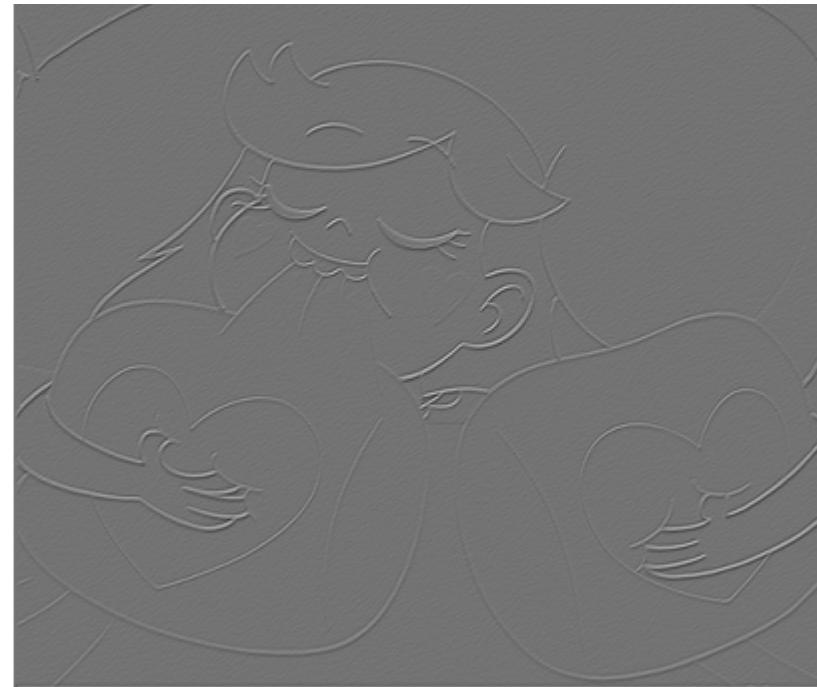


heat equation $t = 10$

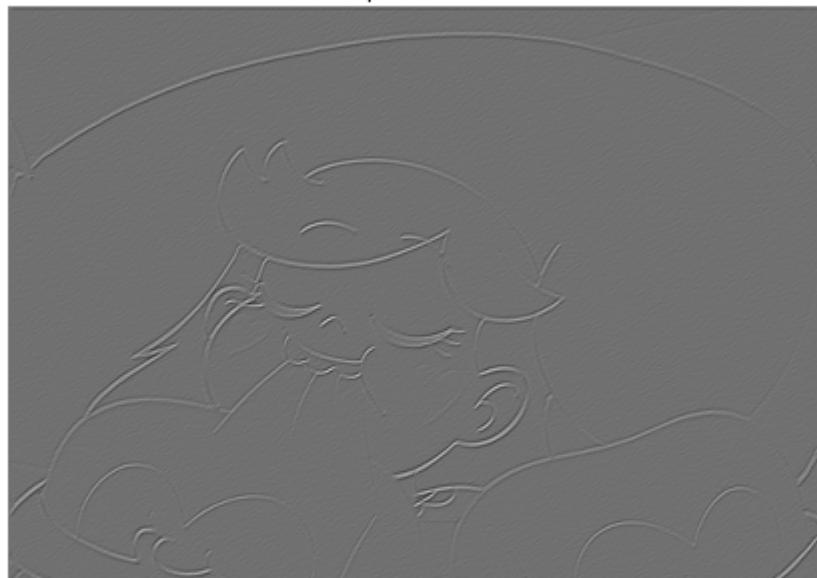




heat equation $t = 100$



heat equation $t = 1000$





```
git commit -a -m "compute the first-order derivative of the solution from the heat equations"
```

```
git push origin master
```

compute the second-order derivative of the solutions obtained by heat equation

In [78]:

```
lxx      = compute_derivative_second_order(l, 'x') # compute second derivative l_xx in x-direction
lyy      = compute_derivative_second_order(l, 'y') # compute second derivative l_yy in y-direction

u10_xx   = compute_derivative_second_order(u10, 'x') # compute second derivative u(10)_xx in x-direction
u10_yy   = compute_derivative_second_order(u10, 'y') # compute second derivative u(10)_yy in y-direction

u100_xx  = compute_derivative_second_order(u100, 'x') # compute second derivative u(100)_xx in x-direction
u100_yy  = compute_derivative_second_order(u100, 'y') # compute second derivative u(100)_yy in y-direction

u1000_xx = compute_derivative_second_order(u1000, 'x') # compute second derivative u(1000)_xx in x-direction
u1000_yy = compute_derivative_second_order(u1000, 'y') # compute second derivative u(1000)_yy in y-direction

plt.figure(figsize=(16,16))

# visualise the second-order derivative in x-axis of the original image
plt.subplot(221)
plt.imshow(lxx, cmap='gray')
plt.title('input image')
plt.axis('off')

# visualise the second-order derivative in x-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_xx, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the second-order derivative in x-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_xx, cmap='gray')
```

```

plt.title('heat equation t = 100')
plt.axis('off')

# visualise the second-order derivative in x-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_xx, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')

plt.figure(figsize=(16,16))

# visualise the second-order derivative in y-axis of the original image
plt.subplot(221)
plt.imshow(lyy, cmap='gray')
plt.title('input image')
plt.axis('off')

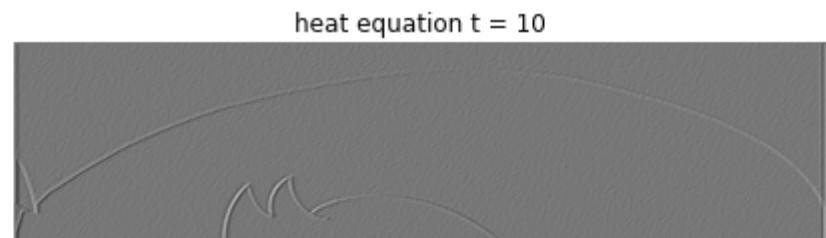
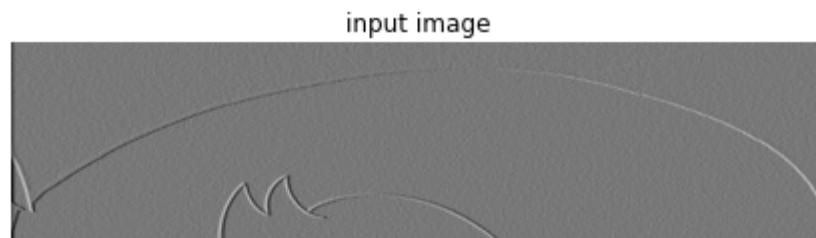
# visualise the second-order derivative in y-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_yy, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the second-order derivative in y-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_yy, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the second-order derivative in y-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_yy, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')

```

Out[78]: (-0.5, 566.5, 544.5, -0.5)

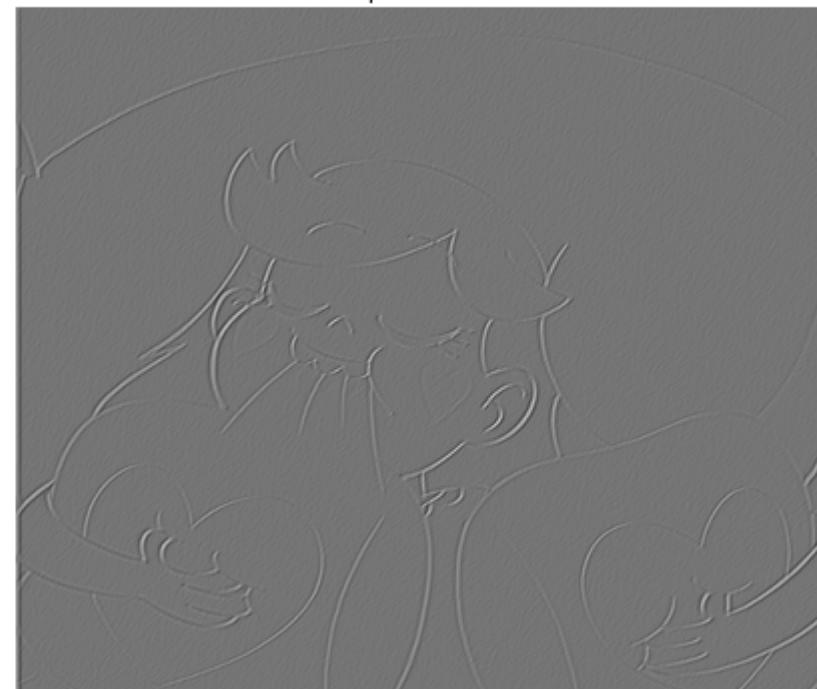
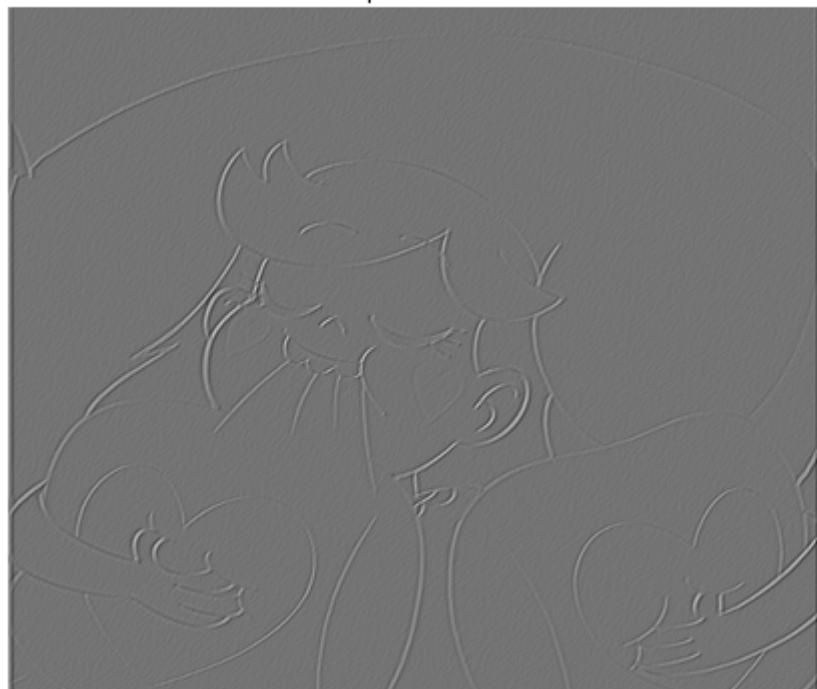




heat equation $t = 100$

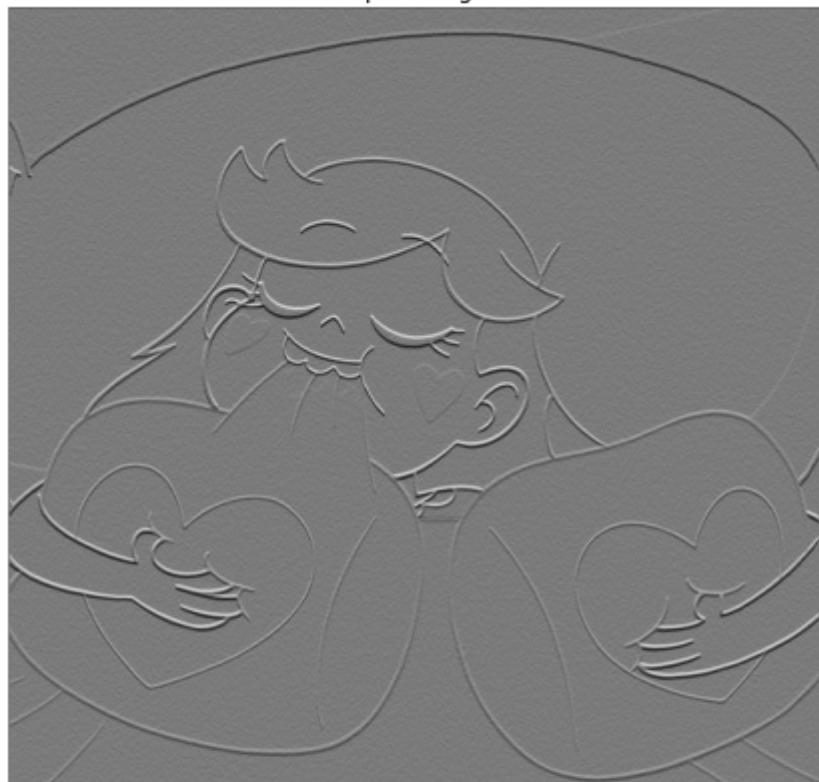


heat equation $t = 1000$

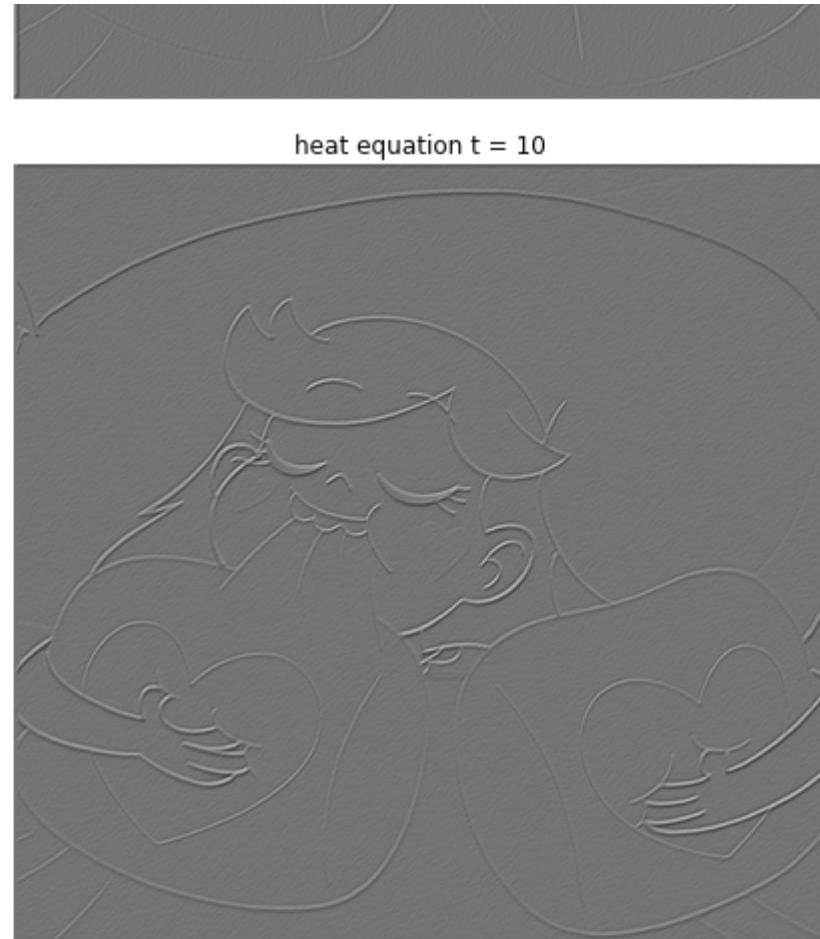




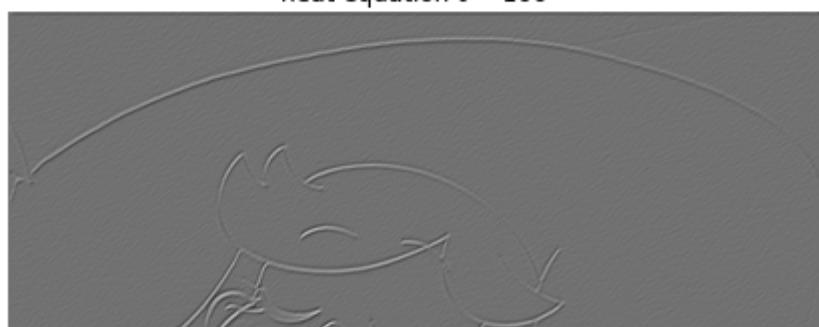
input image



heat equation $t = 10$

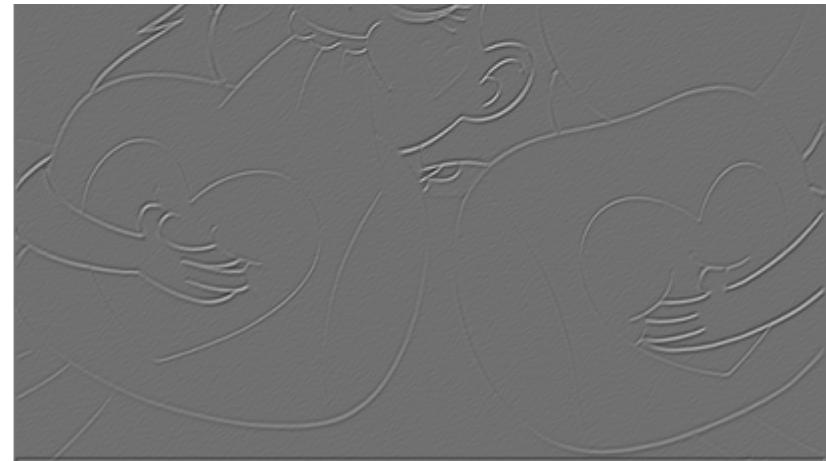
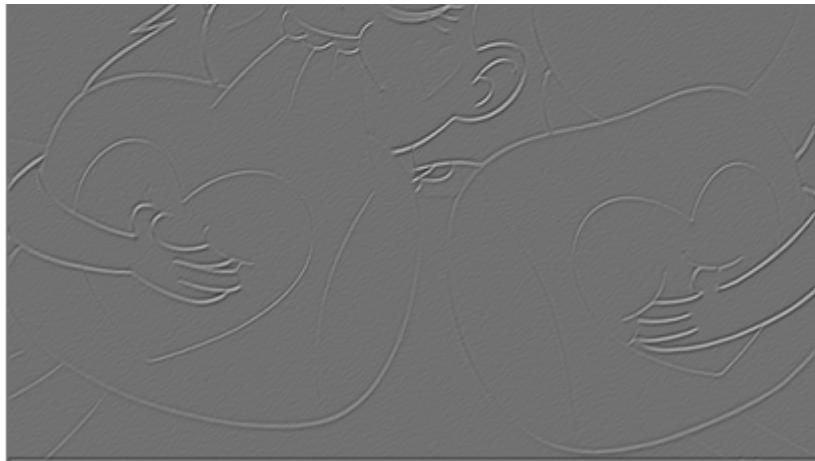


heat equation $t = 100$



heat equation $t = 1000$





```
git commit -a -m "compute the second-order derivative of the solution from the heat equations"
```

```
git push origin master
```

define functions for the visualization

In [79]:

```
def plot_image_color(l, title = 'title'):

    plt.figure(figsize=(8,8))
    plt.imshow(l)
    plt.title(title)
    plt.axis('off')

def plot_image_gray(l, title = 'title'):

    plt.figure(figsize=(8,8))
    plt.imshow(l, cmap='gray')
    plt.title(title)
    plt.axis('off')

def plot_image_gray_2x1(l1, l2, title1 = 'title 1', title2 = 'title 2'):

    plt.figure(figsize=(16,8))

    plt.subplot(121)
    plt.imshow(l1, cmap = 'gray')
    plt.title(title1)
```

```

plt.axis('off')

plt.subplot(122)
plt.imshow(l2, cmap = 'gray')
plt.title(title2)
plt.axis('off')

def plot_image_gray_2x2(l1, l2, l3, l4, title1 = 'title 1', title2 = 'title 2', title3 = 'title 3', title4 = 'title 4'):

    plt.figure(figsize=(16,16))

    plt.subplot(221)
    plt.imshow(l1, cmap = 'gray')
    plt.title(title1)
    plt.axis('off')

    plt.subplot(222)
    plt.imshow(l2, cmap = 'gray')
    plt.title(title2)
    plt.axis('off')

    plt.subplot(223)
    plt.imshow(l3, cmap = 'gray')
    plt.title(title3)
    plt.axis('off')

    plt.subplot(224)
    plt.imshow(l4, cmap = 'gray')
    plt.title(title4)
    plt.axis('off')

```

```
git commit -a -m "define functions for the visualization"
```

```
git push origin master
```

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

results

```
# #####
```

01. plot the input image in color

In [80]:

```
plot_image_color(l_color, 'input image in color')
```

input image in color



02. plot the input image in gray

In [81]:

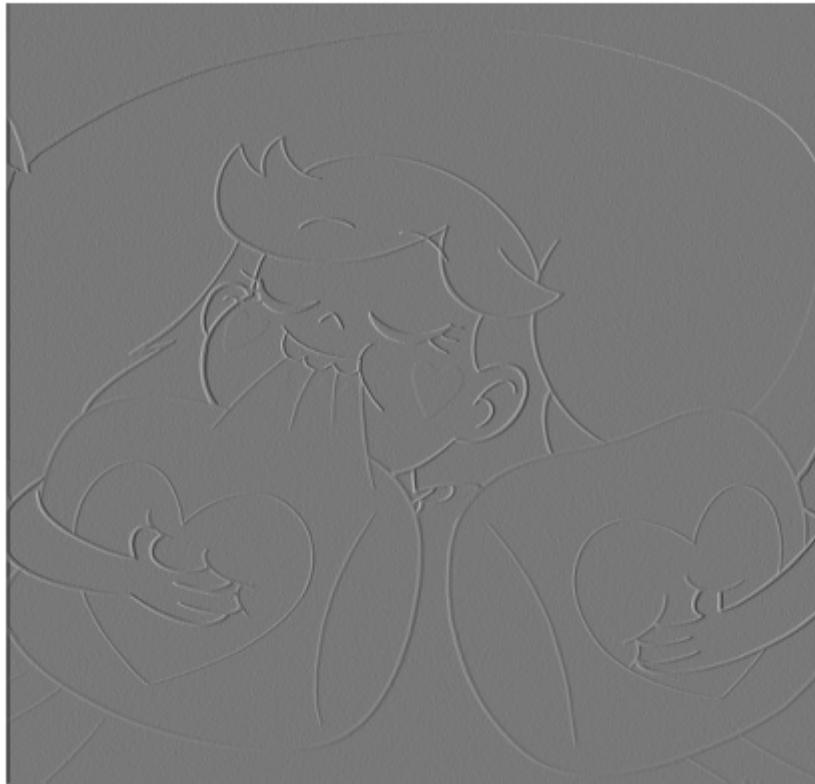
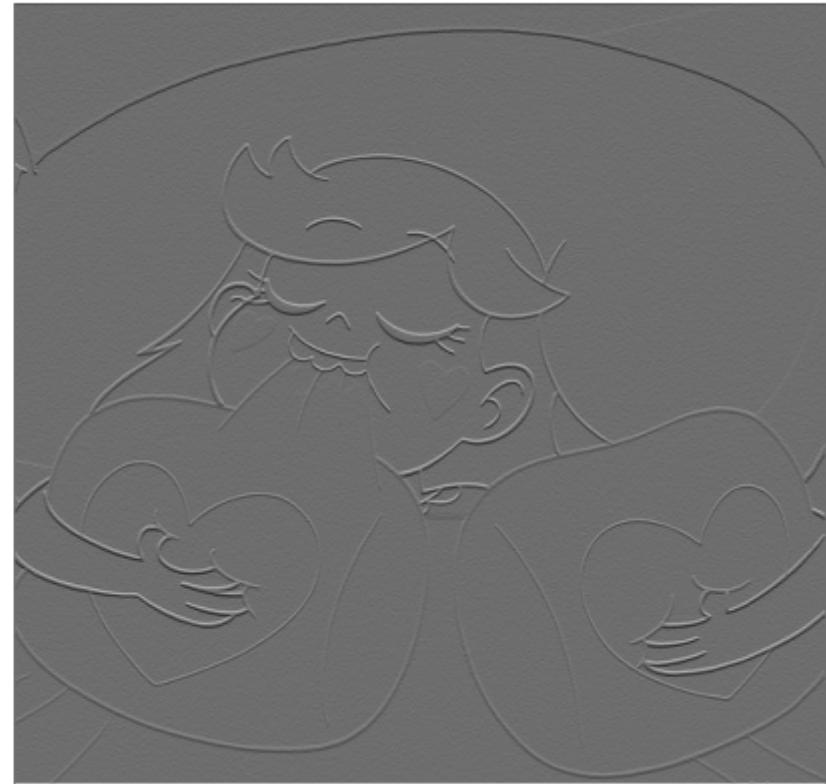
```
plot_image_gray(l, 'input image in grey')
```

input image in grey



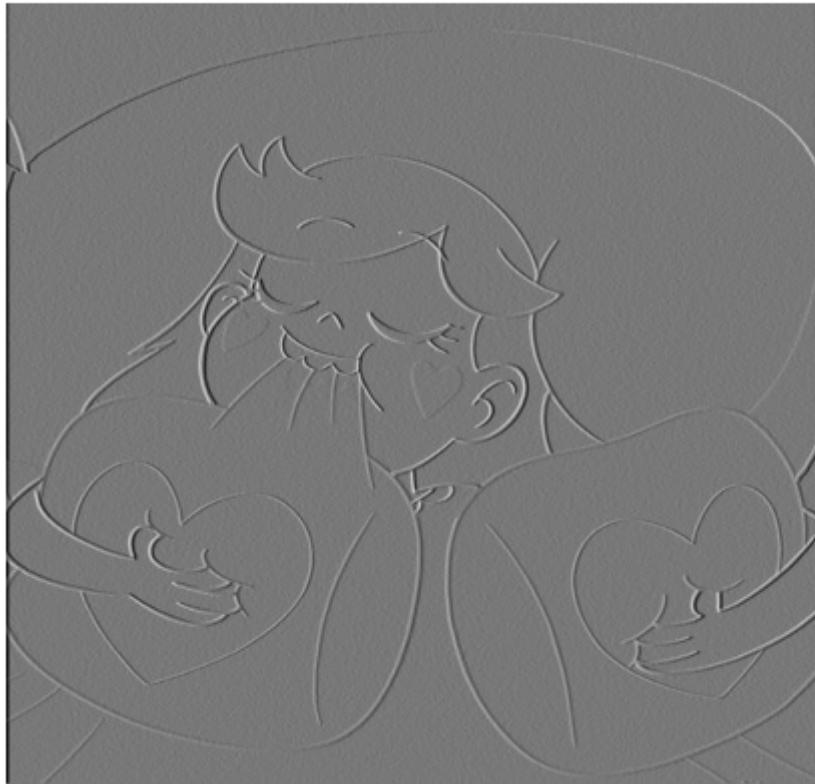
03. plot the (forward) first derivative I_x and I_y of input gray image I with Neumann boundary condition

In [82]:
plot_image_gray_2x1(Ix, Iy, '\$I_x\$', '\$I_y\$')

I_x  I_y 

04. plot the second derivative I_{xx} and I_{yy} of input gray image I with Neumann boundary condition

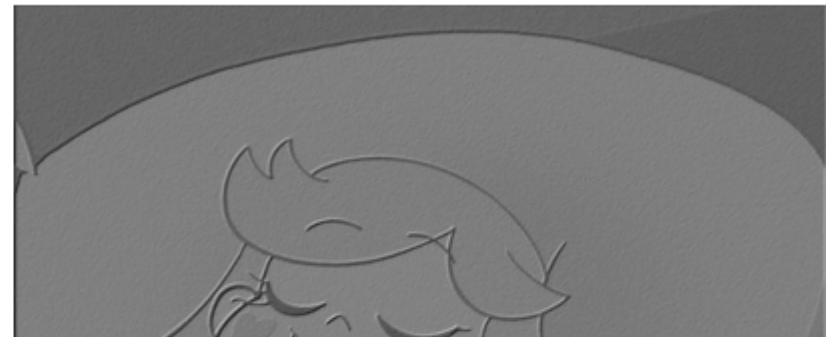
```
In [63]: plot_image_gray_2x1(Ixx, Iyy, '$I_{xx}$', '$I_{yy}$')
```

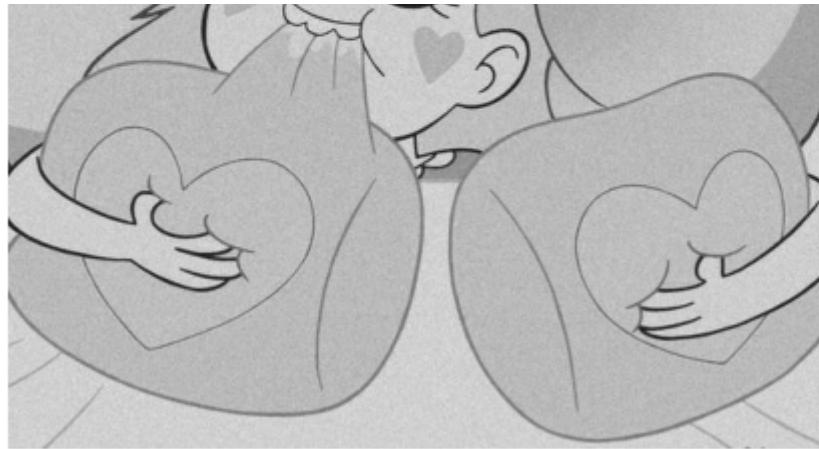
I_{xx}  I_{yy} 

05. plot the original image and its solution of the heat equation with 10, 100, 1000 iterations

In [83]:

```
plot_image_gray_2x2(I, u10, u100, u1000, 'original $I$', '$u(t=10)$', '$u(t=100)$', '$u(t=1000)$')
```

original I  $u(t = 10)$ 



$u(t=100)$



$u(t=1000)$

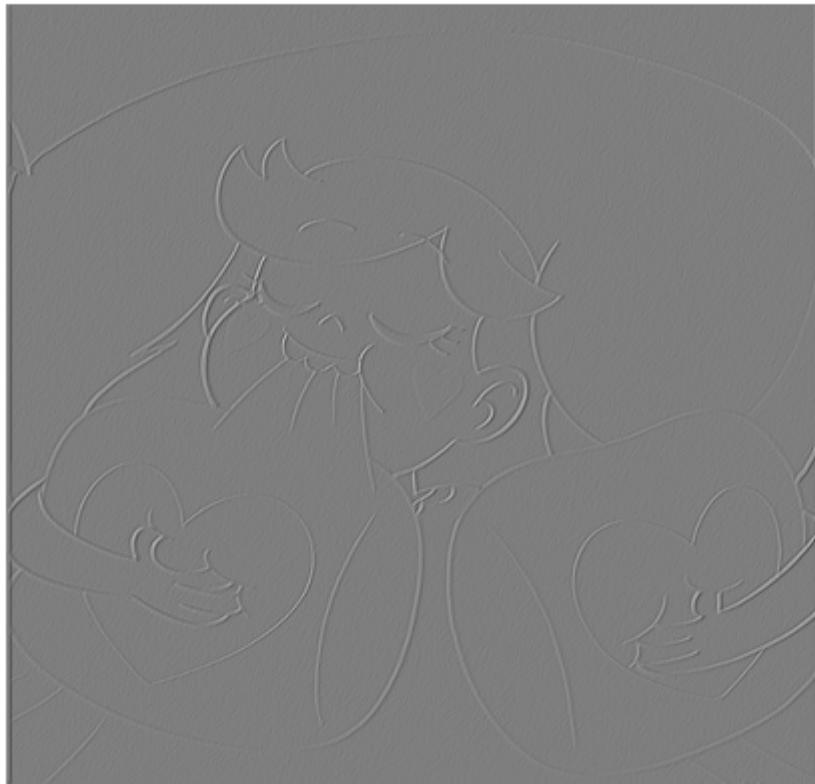


06. plot the (forward) first derivative of the solution of the heat equation with 10 iterations

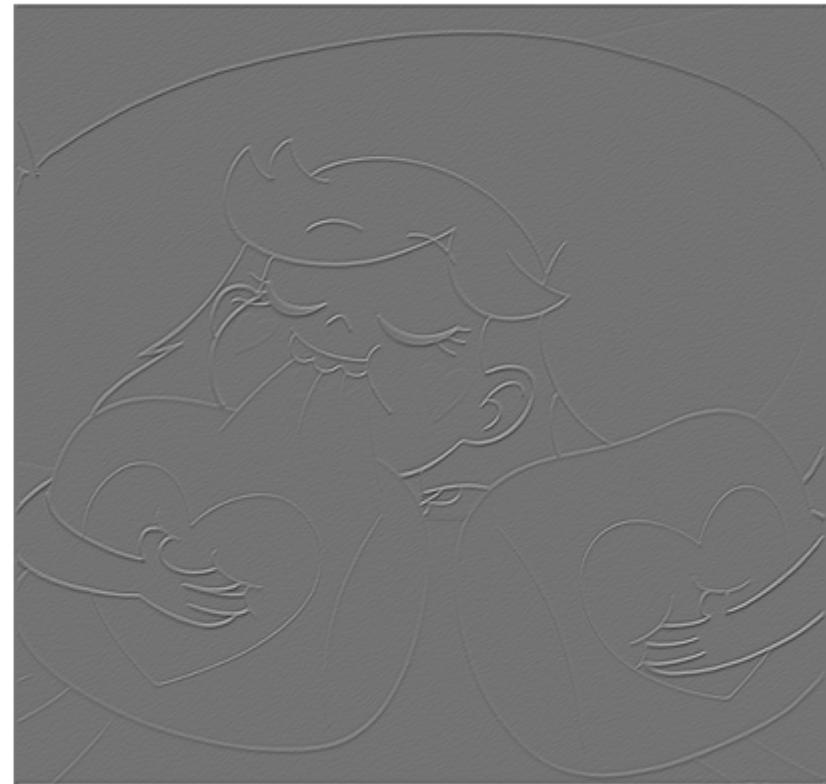
In [65]:

```
plot_image_gray_2x1(u10_x, u10_y, '$u_x(t=10)$', '$u_y(t=10)$')
```

$u_x(t = 10)$



$u_y(t = 10)$

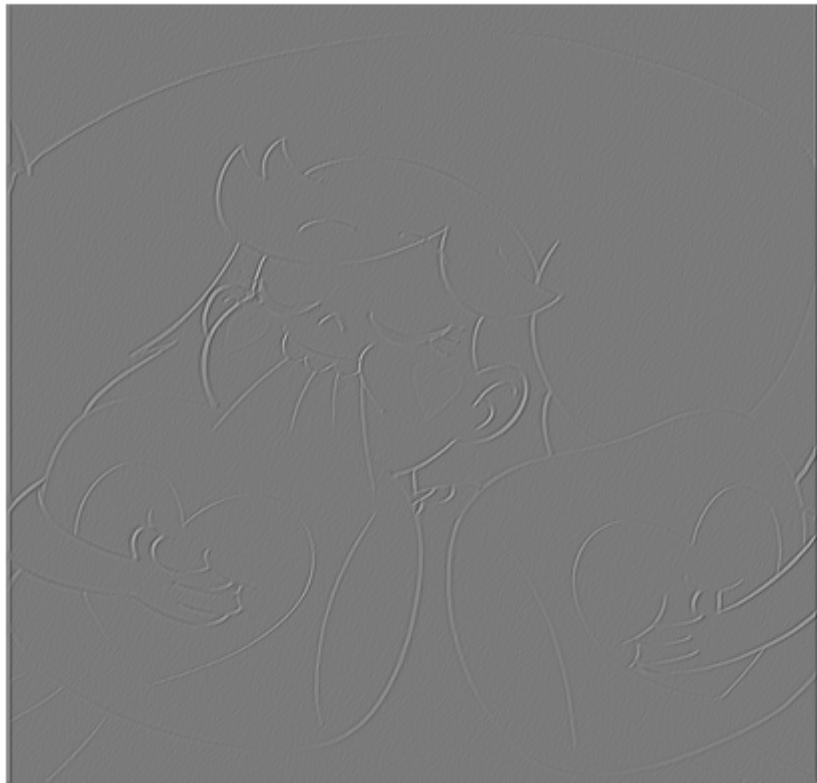


07. plot the (forward) first derivative of the solution of the heat equation with 100 iterations

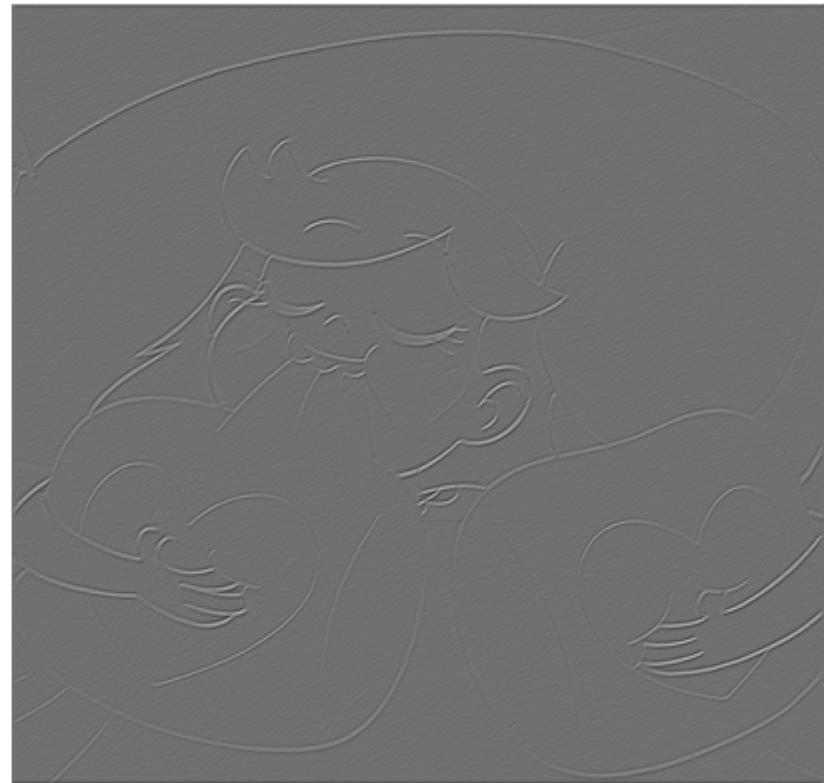
In [84]:

```
plot_image_gray_2x1(u100_x, u100_y, '$u_x(t=100)$', '$u_y(t=100)$')
```

$u_x(t = 100)$



$u_y(t = 100)$

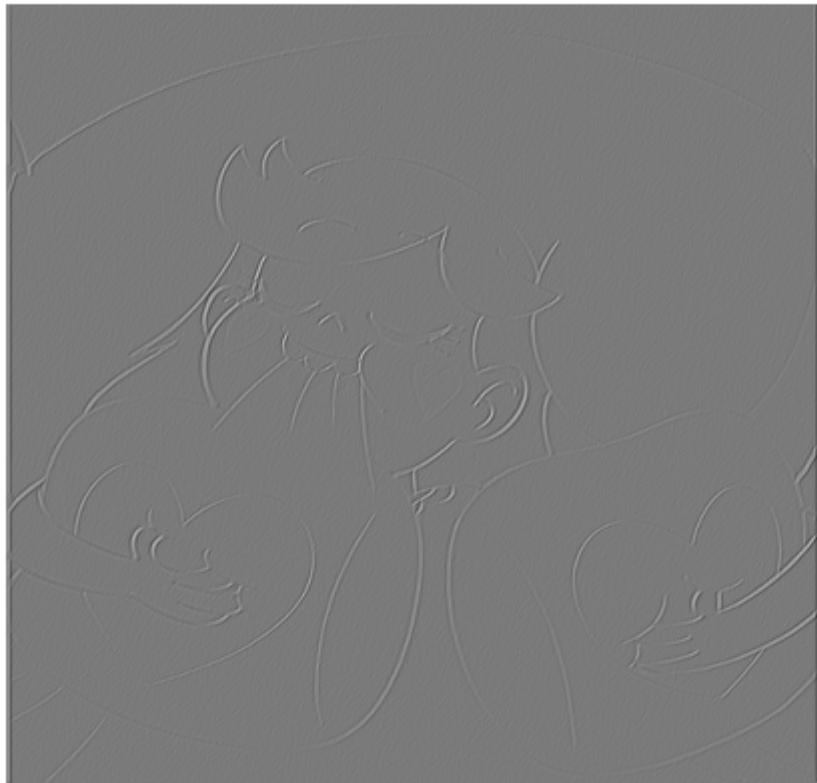


08. plot the (forward) first derivative of the solution of the heat equation with 1000 iterations

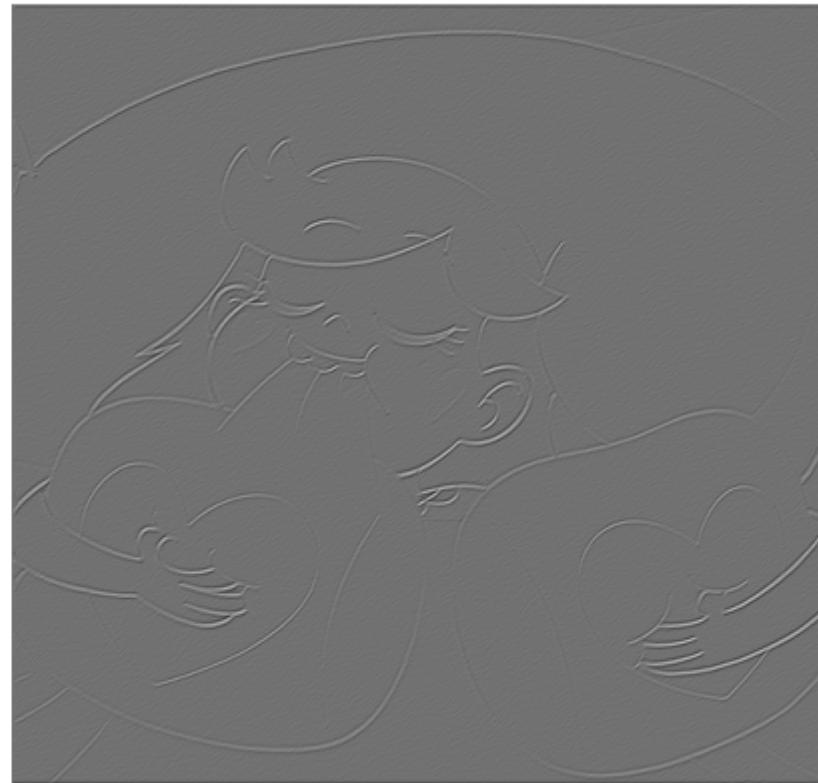
In [85]:

```
plot_image_gray_2x1(u1000_x, u1000_y, '$u_x(t=1000)$', '$u_y(t=1000)$')
```

$u_x(t = 1000)$



$u_y(t = 1000)$



In []: