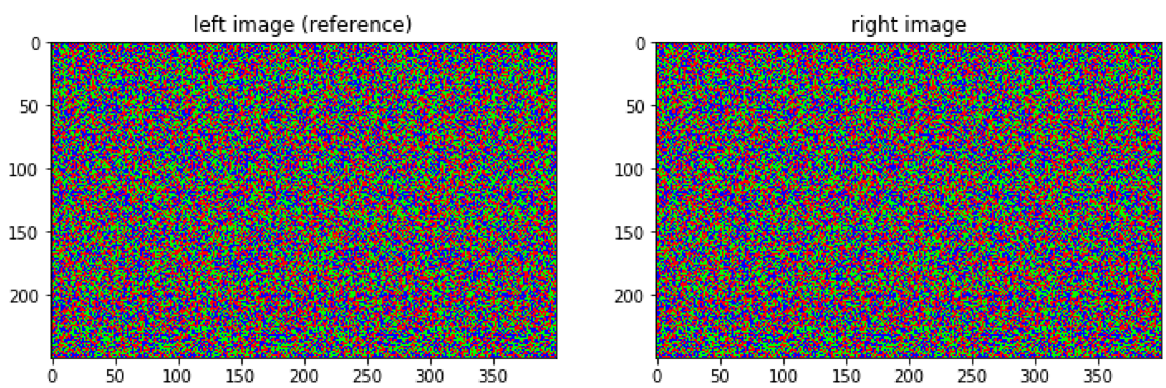
Assignment 3

Part I: Windows-based stereo

In [1]:
```
%matplotlib inline
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as image
from matplotlib.colors import LogNorm
from skimage import img_as_ubyte
from skimage.color import rgb2grey
```

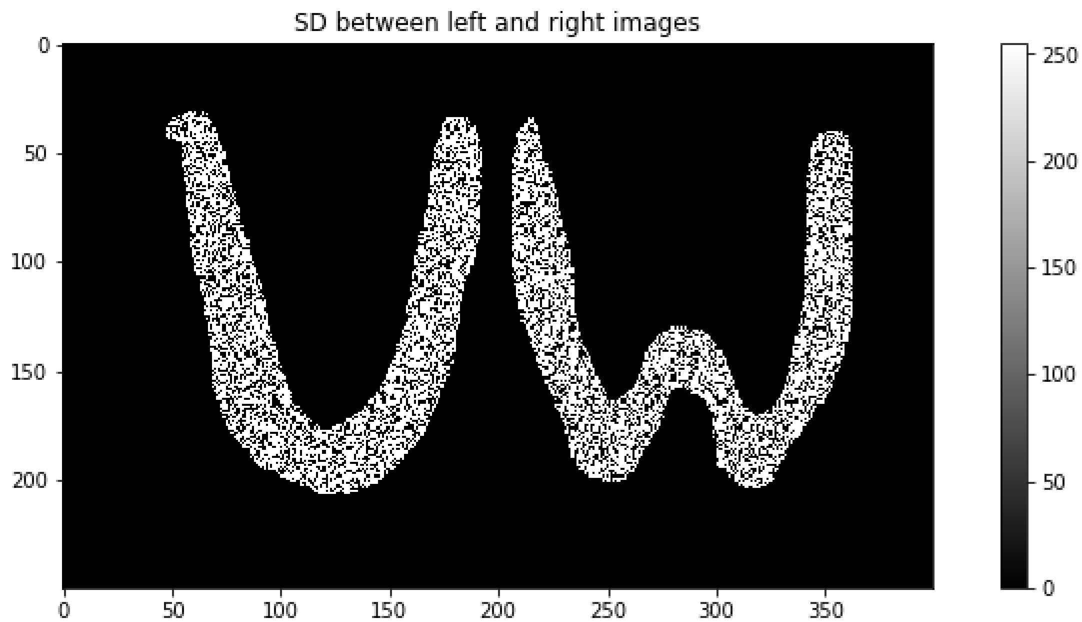
In [3]:
```
# you should use this random dot stereo pair for code developing/test
ing in Probelms 1-5
im_left = image.imread("images/stereo_pairs/rds_left.gif")
im_right = image.imread("images/stereo_pairs/rds_right.gif")

# Image dimentions
n, m, r = im_left.shape

fig = plt.figure(figsize = (12, 5))
plt.subplot(121)
plt.title("left image (reference)")
plt.imshow(im_left)
plt.subplot(122)
plt.title("right image")
plt.imshow(im_right)

# the range of disparities for this random dot stereo pair
d_min = 0
d_max = 2
```



Problem 1: compute and visualize (as an image) an array of "squared differences" between RGB pixel values in the left and the right images. Treat each RGB pixel value as 3-vector and interprete "squared difference" as squared L2 norm of the difference between the corresponding vectors. HINT: convert R, G, B values to floats. Note that $imshow$ function can display (as an image) any 2D array of floats.

In [4]:
```
SD = np.zeros(np.shape(im_left))
SD = np.linalg.norm(im_left - im_right, axis = 2)

fig = plt.figure(figsize = (12, 5))
plt.title("SD between left and right images")
plt.imshow(SD, cmap = "gray")
plt.colorbar()
```

Out[4]:
```
<matplotlib.colorbar.Colorbar at 0x7f40a3f21be0>
```



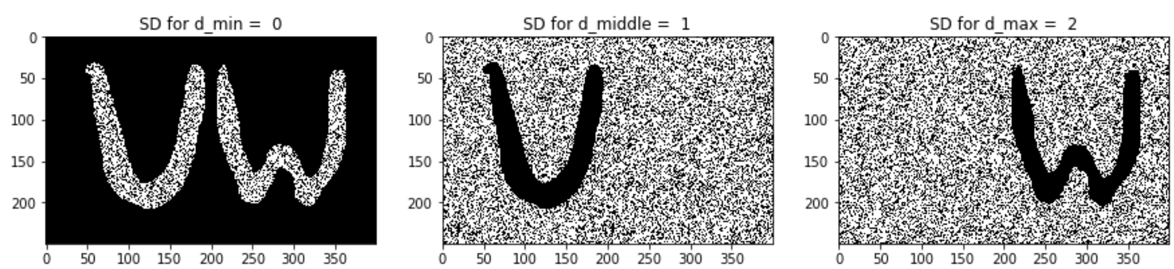
Problem 2: write function for computing squared differences between RGB pixel values in the reference (left) image and the "shifted" right image for ALL shifts/disparities Δ in the range $\Delta \in [d_{min}, d_{max}]$. You should think about the correct direction of the shift. The output should be array SD such that $SD[i]$ is an image of Squared Differences for shift $\Delta = d_{min} + i$ for any $i \in [0, d_{max} - d_{min}]$.

In [5]:
```
def SD_array(imageL, imageR, d_minimum, d_maximum):
    # initialization of the array of "squared differences" for differ
ent shifts
    imageL = imageL.astype('float64')
    imageR = imageR.astype('float64')
    SD = np.zeros((1 + d_maximum - d_minimum, np.shape(imageL)[0], np
.shape(imageL)[1]))

    for delta in range (d_maximum - d_minimum + 1):
        imageRRoll = np.roll(imageR.astype(float), delta + d_minimum,
axis = 1)
        SD[delta] = np.linalg.norm(imageL.astype(float) - imageRRoll,
axis = 2)

    return SD
```

Use $\mathrm{SD\_array}$ function to compute SD images for the random dot stereo pair. Visualize such squared difference images for $\Delta = d_{min}$, $\Delta = d_{mid} \approx \frac{d_{min}+d_{max}}{2}$, and $\Delta = d_{max}$. Note that the first image should be identical to the one in Problem 1. (fully implemented)

In [7]:
```
SD = SD_array(im_left, im_right, d_min, d_max)

fig = plt.figure(figsize = (15, 4))
plt.subplot(131)
plt.title('SD for d_min = {:>2d}'.format(d_min))
plt.imshow(SD[0], cmap = "gray")
plt.subplot(132)
d_middle = round((d_min+d_max)/2)
plt.title('SD for d_middle = {:>2d}'.format(d_middle))
plt.imshow(SD[d_middle-d_min], cmap = "gray")
plt.subplot(133)
plt.title('SD for d_max = {:>2d}'.format(d_max))
plt.imshow(SD[d_max-d_min], cmap = "gray")
#plt.colorbar(cax=plt.axes([0.91, 0.25, 0.01, 0.5]))
```

Out[7]: <matplotlib.image.AxesImage at 0x7f40a3d43908>



Problem 3: write function to compute an "integral image" for any given "scalar" image

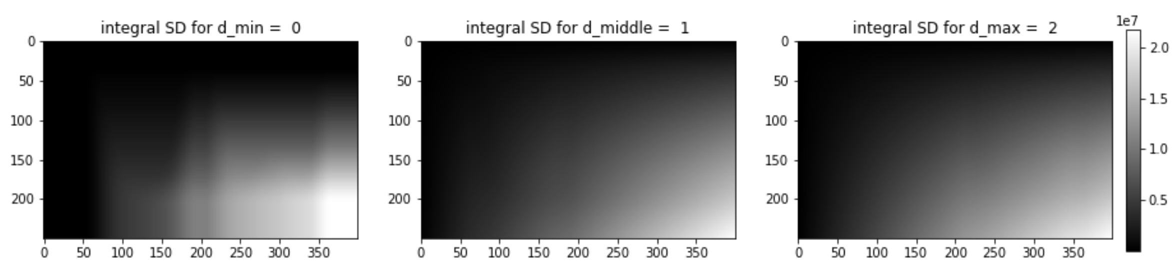
In [8]:
```
# Function integral_image can be applied to any scalar 2D array/imag
e.
# This function should return a double/float64 (precision) array/imag
e of the same size.
# NOTE: it is safer to explicitly specify double/float64 precision fo
r integral images since
# later we will be adding/subtracting ("differenting") their values i
n nearby pixels .

def integral_image(img):
    img = img.astype('float64')
    imgShape = img.shape
    img = np.cumsum(img, axis = 0, dtype = float)
    img = np.cumsum(img, axis = 1, dtype = float)
    img = img.reshape(imgShape)
    return img
```

apply intergal_image function to the "squared differences" (SD) for each disparity (fully implemented)

In [9]:
```
integral_SD = np.zeros(np.shape(SD))
print(np.shape(integral_SD),np.shape(SD))

for Delta in range(1+d_max-d_min):
    integral_SD[Delta] = integral_image(SD[Delta])

fig = plt.figure(figsize = (15, 4))
plt.subplot(131)
plt.title('integral SD for d_min = {:>2d}'.format(d_min))
plt.imshow(integral_SD[0], cmap = "gray")
plt.subplot(132)
d_middle = round((d_min+d_max)/2)
plt.title('integral SD for d_middle = {:>2d}'.format(d_middle))
plt.imshow(integral_SD[d_middle-d_min], cmap = "gray")
plt.subplot(133)
plt.title('integral SD for d_max = {:>2d}'.format(d_max))
plt.imshow(integral_SD[d_max-d_min], cmap = "gray")
plt.colorbar(cax=plt.axes([0.91, 0.2, 0.01, 0.6]))
```

```
(3, 250, 400) (3, 250, 400)
```

Out[9]:
```
<matplotlib.colorbar.Colorbar at 0x7f40a3c1d358>
```



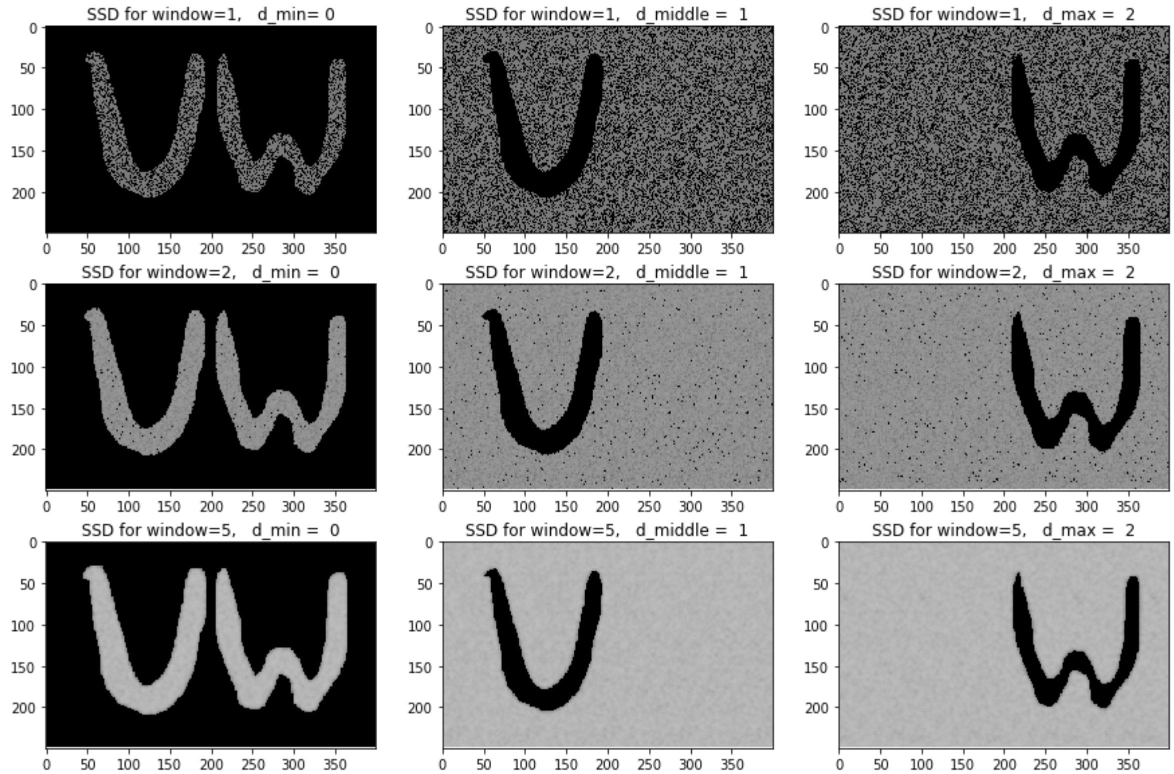
Problem 4: write function that sums the elements of the input image within fixed-size windows around image pixels. Note that this function should work for any (odd or even) values of parameter $\text{window\_width} \in \{1, 2, 3, 4, 5, \dots\}$ according to the windows illustration below:

In [10]:
```python
# function windSum can be applied to any scalar 2D array/image. It should return an array/image where the value of
# each element (pixel p) is the "sum" of the values in the input array/image within a window around element p.
# The return image should be of the same size/type and have its margins (around half-window width) filled with INFTY.
# NOTE: you should use function integral_image implemented earlier.
# HINT: you should use standard np.roll function to avoid double or triple for loops.
# |-------------------
# |
# |       tl         tr
# |        o---------
# |        |    w   |
# |      bl|________|br
# f = br - bl - tr + tl
INFTY = np.inf
def windSum(img, window_width):
    img = img.astype('float64')
    integral = integral_image(img)
    n = img.shape[0]
    m = img.shape[1]
    br = np.pad(integral, ((window_width, 0), (window_width, 0)), 'constant')
    bl = np.roll(br, window_width, axis = 1)
    tr = np.roll(br, window_width, axis = 0)
    tl = np.roll(br, window_width, axis = (0, 1))
    f = br -bl - tr + tl

    f = f[window_width:, window_width:]

    marginTandL = (window_width - 1) // 2
    marginBandR = window_width // 2

    img = np.full(img.shape, np.inf)

    img[marginTandL : (n - marginBandR), marginTandL : (m - marginBandR)] = f[window_width - 1:, window_width - 1:]
    return img
```

Compute SSD images for windows of different widths and for different disparities by applying $\mathrm{windSum}$ function to the "squared differences" SD. Note that the results for windows of width 1 (the first row of the figure below) should look identical (except for the narrow "margin") to the results in Problem 2. (fully implemented)

In [11]:
```
SSD1 = np.zeros(np.shape(SD))
SSD2 = np.zeros(np.shape(SD))
SSD5 = np.zeros(np.shape(SD))

for Delta in range(1+d_max-d_min):
    SSD1[Delta] = windSum(SD[Delta],1)
    SSD2[Delta] = windSum(SD[Delta],2)
    SSD5[Delta] = windSum(SD[Delta],5)

d_middle = round((d_min+d_max)/2)

fig = plt.figure(figsize = (15, 10))
plt.subplot(331)
plt.title('SSD for window=1,   d_min={:>2d}'.format(d_min))
plt.imshow(1+SSD1[0], cmap = "gray", norm=LogNorm(vmin=1, vmax=200000
))
plt.subplot(332)
plt.title('SSD for window=1,   d_middle = {:>2d}'.format(d_middle))
plt.imshow(1+SSD1[d_middle-d_min], cmap = "gray", norm=LogNorm(vmin=1
, vmax=200000))
plt.subplot(333)
plt.title('SSD for window=1,   d_max = {:>2d}'.format(d_max))
plt.imshow(1+SSD1[d_max-d_min], cmap = "gray", norm=LogNorm(vmin=1, v
max=200000))
plt.subplot(334)
plt.title('SSD for window=2,   d_min = {:>2d}'.format(d_min))
plt.imshow(1+SSD2[0], cmap = "gray", norm=LogNorm(vmin=1, vmax=200000
))
plt.subplot(335)
plt.title('SSD for window=2,   d_middle = {:>2d}'.format(d_middle))
plt.imshow(1+SSD2[d_middle-d_min], cmap = "gray", norm=LogNorm(vmin=1
, vmax=200000))
plt.subplot(336)
plt.title('SSD for window=2,   d_max = {:>2d}'.format(d_max))
plt.imshow(1+SSD2[d_max-d_min], cmap = "gray", norm=LogNorm(vmin=1, v
max=200000))
plt.subplot(337)
plt.title('SSD for window=5,   d_min = {:>2d}'.format(d_min))
plt.imshow(1+SSD5[0], cmap = "gray", norm=LogNorm(vmin=1, vmax=200000
))
plt.subplot(338)
plt.title('SSD for window=5,   d_middle = {:>2d}'.format(d_middle))
plt.imshow(1+SSD5[d_middle-d_min], cmap = "gray", norm=LogNorm(vmin=1
, vmax=200000))
plt.subplot(339)
plt.title('SSD for window=5,   d_max = {:>2d}'.format(d_max))
plt.imshow(1+SSD5[d_max-d_min], cmap = "gray", norm=LogNorm(vmin=1, v
max=200000))
```

Out[11]: <matplotlib.image.AxesImage at 0x7f40a39218d0>



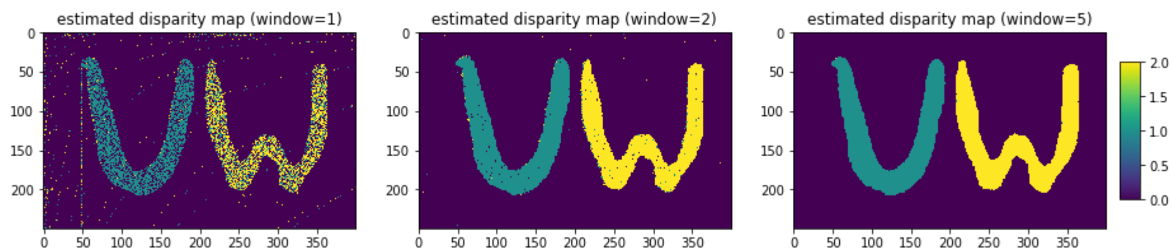
Problem 5: write code for function computing "disparity map" from SSD arrays (as above) for each disparity in the range specified by integers d_{min}, d_{max}. It should return a disparity map (image). At each pixel, disparity map image should have disparity value corresponding to the minimum SSD at this pixel. For pixels at the margins, the disparity map should be set to zero. HINT: margin pixels are characterized by $INFTY$ values of SSD.

In [12]:
```
# You should use functions np.where (pointwise "if, then, else" opera
tion) and np.minimum (pointwise "minimum" operation)
# These functions will help to avoid double loops for traversing the
 pixels.
# WARNING: there will be a deducton for double-loops traversing pixel
s, but single loop to traverse disparities is OK.

def SSDtoDmap(SSD_array, d_minimum, d_maximum):

    minVal = SSD_array[0]
    dMap = np.full(np.shape(SD[0]), d_minimum)
    for i in range(1, SSD_array.shape[0]):
        current = d_minimum + i
        dMap = np.where(SSD_array[i] < minVal, current, dMap)
        minVal = np.where(SSD_array[i] < minVal, SSD_array[i], minVal
)
    #set pixel at margins to 0
    dMap = np.where(SSD_array[0] == np.inf, 0, dMap)
    return dMap
```

Compute and show disparity map (fully implemented)

In [13]:
```
dMap1 = SSDtoDmap(SSD1,d_min,d_max)
dMap2 = SSDtoDmap(SSD2,d_min,d_max)
dMap5 = SSDtoDmap(SSD5,d_min,d_max)

fig = plt.figure(figsize = (15, 3))
plt.subplot(131)
plt.title("estimated disparity map (window=1)")
plt.imshow(dMap1, vmin = 0, vmax = d_max)
plt.subplot(132)
plt.title("estimated disparity map (window=2)")
plt.imshow(dMap2, vmin = 0, vmax = d_max)
plt.subplot(133)
plt.title("estimated disparity map (window=5)")
plt.imshow(dMap5, vmin = 0, vmax = d_max)
plt.colorbar(cax=plt.axes([0.91, 0.25, 0.015, 0.5]))
```

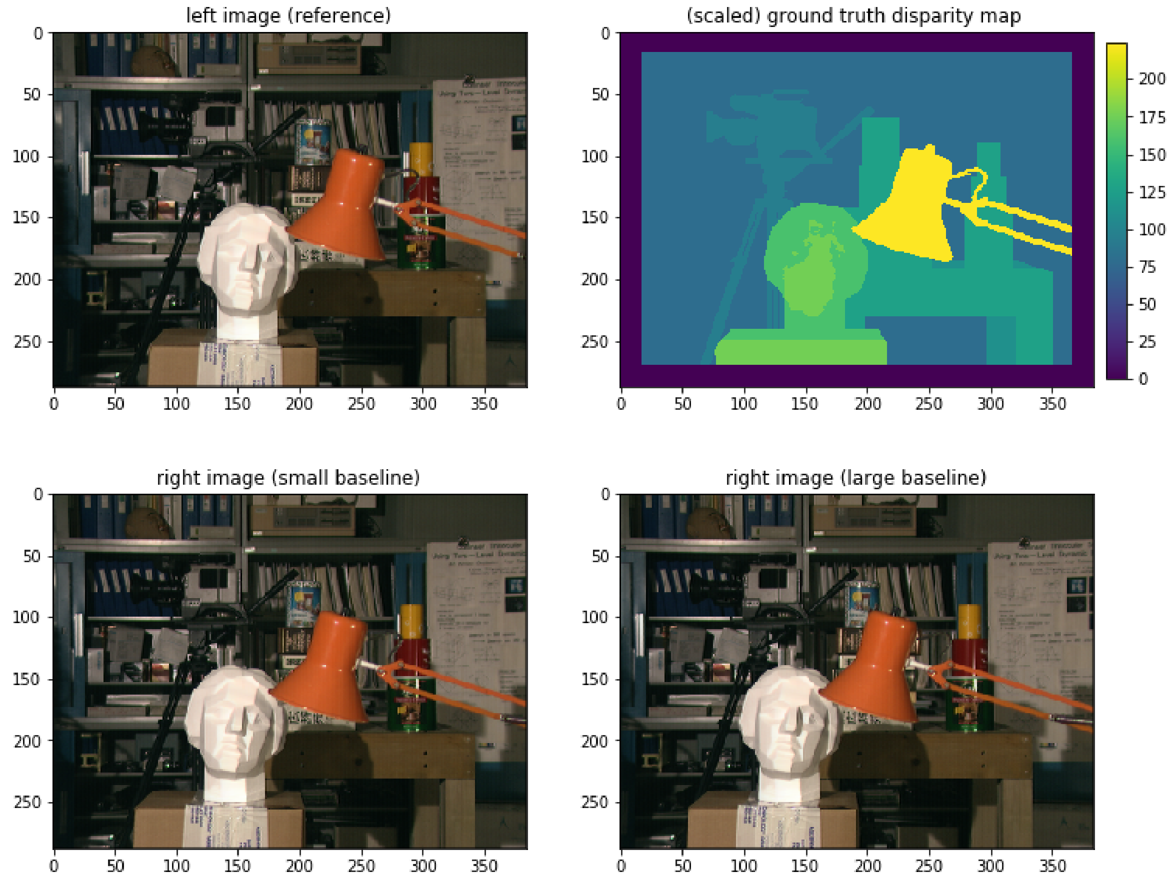
Out[13]:
```
<matplotlib.colorbar.Colorbar at 0x7f40a37fd588>
```



Problem 6: test your code on a real stereo pair with ground truth (Tsukuba)

In [14]:
```
# images/tsukuba subdirectory contains (a subset of) "Tsukuba" stereo images that are probably
# the oldest stereo data with dense ground-truth produced at the University of Tsukuba in 2001.
# The full Tsukuba dataset and many other stereo images with ground-truth disparity maps can be
# downloaded from well-known Middlebury repository  http://vision.middlebury.edu/stereo/
im_left = image.imread("images/stereo_pairs/tsukuba/scene1.row3.col3.ppm")
im_gt = image.imread("images/stereo_pairs/tsukuba/truedisp.row3.col3.pgm")
im_right = image.imread("images/stereo_pairs/tsukuba/scene1.row3.col4.ppm")
im_right2 = image.imread("images/stereo_pairs/tsukuba/scene1.row3.col5.ppm")

fig = plt.figure(figsize = (12, 10))
plt.subplot(221)
plt.title("left image (reference)")
plt.imshow(im_left)
plt.subplot(222)
plt.title("(scaled) ground truth disparity map ")
plt.imshow(im_gt)
plt.colorbar(cax=plt.axes([0.91, 0.557, 0.015, 0.3]))
plt.subplot(223)
plt.title("right image (small baseline)")
plt.imshow(im_right)
plt.subplot(224)
plt.title("right image (large baseline)")
plt.imshow(im_right2)
```

```
Out[14]: <matplotlib.image.AxesImage at 0x7f40a0666a20>
```



Note that the integer-valued ground truth image above represents scaled disparity values for the pixels in the reference (left) mage. The scale w.r.t. the small baseline right image ($\mathrm{im\_right}$) is 16, while the scale for the large baseline right image ($\mathrm{im\_right2}$) is 8. Below, you should use the small-baseline right image ($\mathrm{im\_right}$).

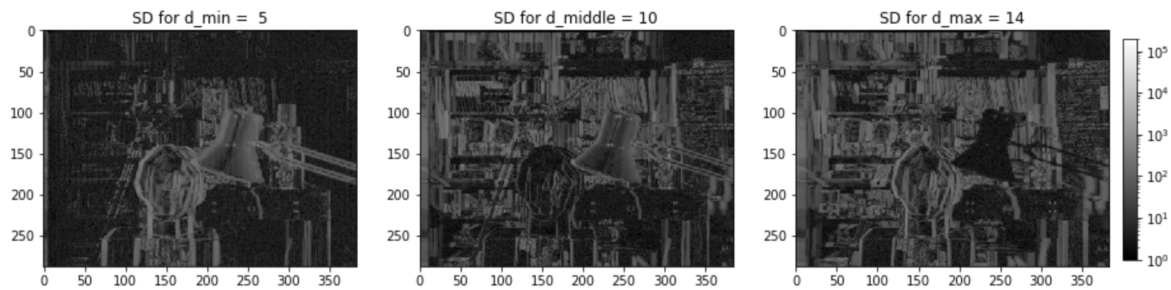
Problem 6a: Using ground truth disparity map, estimate the range of disparity values between pixels in the left image ($\mathrm{im\_left}$) and the right image ($\mathrm{im\_right}$).

```
In [15]: # Solution: use standard functions to find min and max values in the
          ground truth disparity map.
         # You should ignore 0-valued margin!

         # d_min = 0
         d_min = int(np.amin(im_gt[np.where(im_gt > 0)]))//16
         d_max = int(np.max(im_gt)//16)
```

Compute squared differences using $\mathrm{SD\_array}$ function and visualize the results using logarithmic scale. Note that linear scale would make it hard to see smaller squared differences since there are many very large ones. (fully implemented)

In [16]:
```
SD = SD_array(im_left, im_right, d_min, d_max)

fig = plt.figure(figsize = (15, 4))
plt.subplot(131)
plt.title('SD for d_min = {:>2d}'.format(d_min))
plt.imshow(im_left)
plt.imshow(1+SD[0], cmap = "gray", norm=LogNorm(vmin=1, vmax=200000))
plt.subplot(132)
d_middle = round((d_min+d_max)/2)
plt.title('SD for d_middle = {:>2d}'.format(d_middle))
plt.imshow(1+SD[d_middle-d_min], cmap = "gray", norm=LogNorm(vmin=1,
vmax=200000))
plt.subplot(133)
plt.title('SD for d_max = {:>2d}'.format(d_max))
plt.imshow(1+SD[d_max-d_min], cmap = "gray", norm=LogNorm(vmin=1, vma
x=200000))
plt.colorbar(cax=plt.axes([0.91, 0.2, 0.01, 0.6]))
```

Out[16]: <matplotlib.colorbar.Colorbar at 0x7f40a0482be0>



Problem 6b: Explain the differences you observe above:

answer:

We notice that disparities are inversely proportionate to the distance of objects. Smaller disparitiers target far away objects, while larger disparities target more closer objects

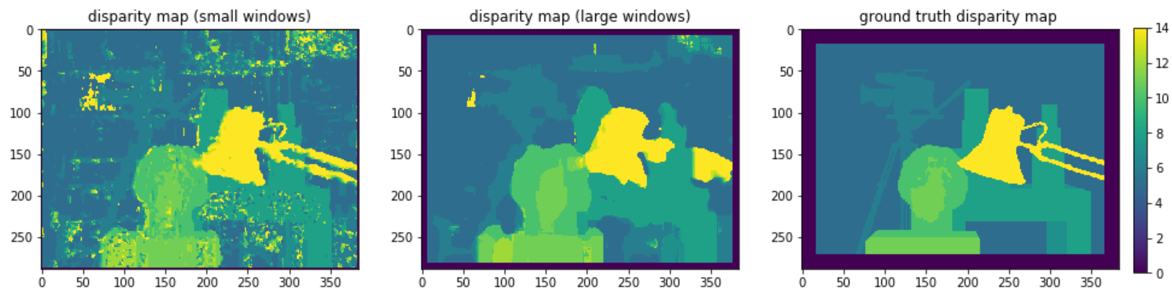
Problem 6c: Write function $\mathrm{Dmap\_Windows}$ that returns disparith map from a given stereo pair (left and right image), specified disparity range, and window size. Your implementation should combine functions implemented and debugged earlier ($\mathrm{SD\_array}$, $\mathrm{windSum}$, and $\mathrm{SSDtoDmap}$).

In [17]:
```
#This is what the prof did for q4 to Compute SSD images for windows
def Dmap_Windows(imageL, imageR, d_minimum, d_maximum, window_width):
    SD = SD_array(imageL, imageR, d_minimum, d_maximum)
    SSD = np.zeros(np.shape(SD))
    for Delta in range(1 + d_maximum - d_minimum):
        SSD[Delta] = windSum(SD[Delta], window_width)
    return SSDtoDmap(SSD, d_minimum, d_maximum)
```

Compute and show disparity maps for Tsukuba using small and large windows. (fully implemented)

In [18]:
```
dispMap_small = Dmap_Windows(im_left, im_right, d_min, d_max, 4)
dispMap_large = Dmap_Windows(im_left, im_right, d_min, d_max, 15)

fig = plt.figure(figsize = (16, 7))
plt.subplot(131)
plt.title("disparity map (small windows)")
plt.imshow(dispMap_small, vmin = 0, vmax = d_max)
plt.subplot(132)
plt.title("disparity map (large windows)")
plt.imshow(dispMap_large, vmin = 0, vmax = d_max)
plt.subplot(133)
plt.title("ground truth disparity map ")
plt.imshow(im_gt/16, vmin = 0, vmax = d_max)
plt.colorbar(cax=plt.axes([0.91, 0.3, 0.01, 0.4]))
```

Out[18]: <matplotlib.colorbar.Colorbar at 0x7f40a0219d68>



In [ ]: