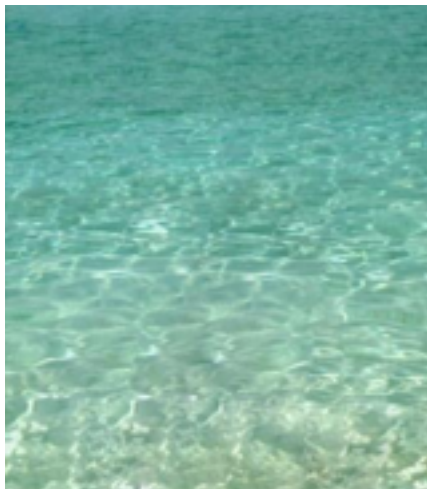




LAB 3: DUE 27 JANUARY 2016

Poisson image editing enables seamless compositing. The reference on the topic is Perez et al.'s article, "Poisson image editing," published at Siggraph 2003 and available at <http://dl.acm.org/citation.cfm?id=882269>

Given a background (target) and a new foreground (source), as well as a binary mask, we seek to replace the target region inside the mask by the content of the source. Below, we show a target (water), a source (bear), and the corresponding mask.



Task 1: Naive Compositing (10 pts)

Implement a function `naiveComposite(bg, fg, mask, y, x)` that simply copies the pixel values from an image `fg` into a target `bg` when `mask` is equal to 1. `fg` is assumed to be smaller than `bg` and has the same size as `mask`. `y` and `x` specify where the upper left corner of `fg` goes in the source image `bg`. In python you should be able to do it without a for loop, using multiplications between images and the mask or `1-mask`, and using the slicing operator `:`.



Task 2: Poisson Gradient Descent (40)

Write a function `Poisson(bg, fg, mask, niter)` implements gradient descent to solve for Poisson image compositing. `bg`, `fg`, and `mask` are assumed to have the same size. Your code will be a more or less direct implementation of the math in the supplementary material.

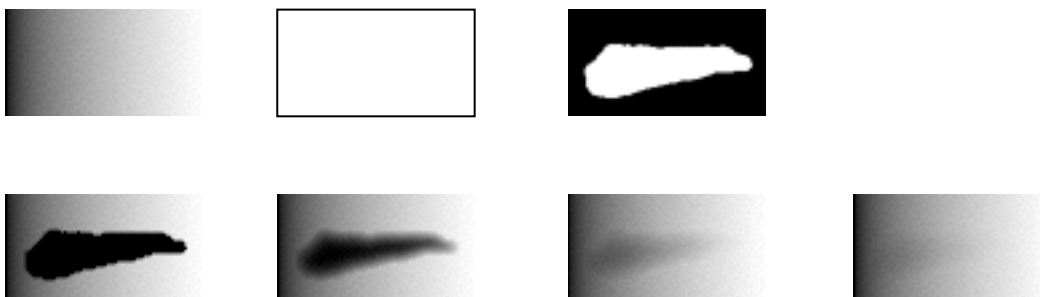
Implement a Laplacian operator and a dot product for images functions.

The Laplacian operator is a convolution of the image with $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$. The dot product is the sum of all elements in a element wise multiplication (images, of course, need to be same size).

In a nutshell, first compute the constant image `b` by applying the Laplacian operator to the image `fg`. Initialize `x` with a black image and iterate: compute the residual `r` by applying the Laplacian to your current estimate and subtract it from `b`. Then solve for `alpha` using the dot product you have implemented for images. Finally, update your estimate of `x`.

Besides replacing the matrix-vector operations by their image implementation, the other special thing you need is to make sure that pixels outside the that mask have values from `bg` and are not updated. For this, first initialize the pixel in `x` outside the masked area by copying those in `bg`. See below for examples of `x`, and in particular `x0`. You can do all this easily with array subtractions and multiplications. Next, you need to multiply your residual by the mask as soon as you compute it. This way, your updates will only consider pixels inside the mask.

Test this function on the small inputs because it is slow to converge. I highly recommend the provided `testRamp` which seeks to paste a flat source into a background composed of a greyscale ramp. Input and in the first row, and `x0`, `x50`, `x100`, `x200`.



Task 3: Poisson Conjugate (50)

Write a function `PoissonCG(bg, fg, mask, niter)` that implements the above conjugate gradient algorithm to solve Poisson image editing. The specifications are the same as for gradient descent. Similarly, `x0`, `ri` and `di` need to be masked appropriately. Note how much faster the conjugate method converges.

Extra

Have fun with it.

Deliverables

Naive Compositing image. Task 2: 50, 100, and converged composited. Task 3: Final Bear composite, and your composite.

README file.

- How long did the assignment take?
- Issues and descriptions of your partial solution (for partial credit)
- Any extras?
- Collaboration acknowledgment.
- What was most unclear/difficult?
- What was most exciting?