Homework #1

Image Processing and Affine Transformations

Assigned: Wednesday, January 30

Due: Wednesday, February 6
at the beginning of class

Directions: Please provide short written answers to the following questions, using this page as a cover sheet. Feel free to discuss the problems with classmates, but please ***answer the questions on your own*.**

Name:_____

Problem 1: Image processing (50 points)

Suppose you have two digital filters, *a* and *b*, and you want to apply both of them to an image *f*:

$$g[i,j] = a[i,j] * b[i,j] * f[i,j]$$

In fact, convolution is associative, so we can perform this operation in two different orders. One option is to first convolve *b* with *f*, and then convolve *a* with the result:

$$g[i,j] = a[i,j] * (b[i,j] * f[i,j])$$

Another option is to first convolve *a* with *b*, effectively creating a new filter, $a * b$, and then convolve that new filter with *f*:

$$g[i,j] = (a[i,j] * b[i,j]) * f[i,j]$$

The answer you get in either case is identical. (When implemented with finite precision, there will of course be small differences, but we'll neglect this.)

Recall that convolution entails repeated multiplication and addition. For this problem, we'll count a multiplication and addition as one operation called a multiply-accumulate (MAC); in fact, newer processors can perform this operation in one clock cycle. When counting up all the MAC's in a convolution, we'll assume the "accumulator" is initialized to zero and begin adding to it after each multiply. When analyzing the cost of a convolution, you should compute the number of MAC's involved.

Note that for this problem, you can assume that the boundaries of the filters *a* and *b* are implicitly padded with zeroes, but one should of course not count the cost of a MAC involving a product with a zero. For the image *f*, assume that some non-zero extension of the boundary has been defined so that you do you have to count MAC's for portions of a filter that extend outside of the boundary of the image.

You do not need to worry about flipping any of the filters or images in this problem. Justify your answers to the following questions.

a) (4 points) Compute $a * b$ where:

$$a = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \qquad b = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

What kind of filter do you get from this convolution? [Aside: a 2D filter that can be written as the convolution of a 1D horizotnal filter and a 1D vertical filter, as in this sub-problem, is called a "separable" filter.]

b) (4 points) More generally, suppose *a* is a 1D horizontal filter of length *L* and *b* is 1D vertical filter of length *M*. What is the cost of computing $a * b$? How many rows does the resulting filter have? How many columns?

c) (4 points) For the filters in **b**), what is the cost of computing $(a * b) * f$, where *f* is $N \times N$ in size? Let's assume that *N* is much greater than *L* and *M*, and ignore the cost of first computing $a * b$.

Problem 1 (cont'd)

d) (6 points) For the filters in **b**), what is the cost of computing $a * (b * f)$? Assuming L and M are both greater than 1, is this approach more or less efficient than computing $(a * b) * f$ as in **c**)?

e) (6 points) Now suppose *a* is a 2D filter of size $L \times L$ and *b* is a 2D filter of size $M \times M$. After computing $a * b$, how many rows and columns will the resulting filter have?

f) (4 points) For the filters in **e**), what is the cost of computing $(a * b) * f$, where *f* is $N \times N$ in size? Again, let's assume that *N* is much greater than *L* and *M*, and ignore the cost of first computing $a * b$.

g) (6 points) For the filters in **e**), what is the cost of computing $a * (b * f)$? Assuming L and M are both greater than 1, is this approach more or less efficient than computing $(a * b) * f$ as in **f**)?

h) (6 points) An alternative to spatial convolution is to compute the Fourier transforms of the inputs, multiply them, and then take the inverse Fourier transform of the result. For discrete images, we would use the 2D Discrete Fourier Transform (DFT) and its inverse:

$$H[k,l] = \sum_{n=0}^{P-1} \sum_{m=0}^{Q-1} h[n,m]\, e^{-i2\pi\left(\frac{kn}{P}+\frac{lm}{Q}\right)} \qquad h[n,m] = \frac{1}{PQ} \sum_{k=0}^{P-1} \sum_{l=0}^{Q-1} H[k,l]\, e^{i2\pi\left(\frac{kn}{P}+\frac{lm}{Q}\right)}$$

where the dimensions of image *h* are $P \times Q$. The DFT can be computed efficiently using the Fast Fourier Transform (FFT). For real-valued *h*, we can use a standard "black box" FFT algorithm to compute the DFT (or its inverse) with $2PQ\left(\log_2 P + \log_2 Q\right)$ MACs. Now suppose we were to compute $a * b * f$ using FFTs (and *without* doing any spatial convolutions) for filter sizes defined in **e**). To do so, we will first pad *a* and *b* with zeros until they are $N \times N$ in size; we're assuming $L, M < N$ and also that the black box FFT algorithm does not take advantage of the zero padding. When multiplying spectra, you will have to account for the cost of multiplying complex numbers; you can assume they require 4 MACs, since $(x + i\,y)(z + i\,w) = (xz\text{-}yw) + i\,(xw+yz)$. (The cost of the FFT stated above already includes the cost of doing any complex arithmetic.) What then is the cost of computing the convolution $a * b * f$?

i) (3 points) Suppose $L = M$. Given your answers in **g**) and **h**), how big must *L* be in order for the FFT-based approach to be more efficient than doing spatial convolutions?

j) (3 points) Even if the FFT is more efficient than spatial convolution under some circumstances, it can introduce visual artifacts, particularly at the boundaries. What artifacts might you expect to see and why?

k) (4 points) To address this issue, one could increase the size of the image *f* by appending to it a reflected copy across its right edge, another reflected copy across its top edge, and a third copy rotated by 180 degrees and placed in the upper right corner. Why would this be an improvement? What is the cost of computing the FFT for this new, "reflection-augmented" version of the image? (Note: for this last part, you're just computing the FFT of the image, no filtering is involved.)

Problem 2: Rotations (30 points)

Recall that a linear transformation can be determined by mapping the unit Cartesian axes (e.g., the x-, y-, and z-axes in 3D) to new axes in the transformed space. The vectors that define these new axes can simply be entered into the columns of the transformation matrix. Also recall that a rotation maps the set of unit-length, orthogonal vectors pointing along these Cartesian axes into a new set of unit-length, orthogonal vectors; i.e.,, for a 3D rotation matrix $\mathbf{R} = [\mathbf{u}\ \mathbf{v}\ \mathbf{w}]$, we know that $\mathbf{u} \cdot \mathbf{u} = \mathbf{v} \cdot \mathbf{v} = \mathbf{w} \cdot \mathbf{w} = 1$, and $\mathbf{u} \cdot \mathbf{v} = \mathbf{u} \cdot \mathbf{w} = \mathbf{v} \cdot \mathbf{w} = 0$.

For each of the sub-problems below, you may assume linear transformation matrices, i.e., no extra bottom row or right column used for affine transformations with translations. Thus, a 3D rotation is represented by a 3x3 matrix, and an *N*-D rotation is represented by an *N*x*N* matrix.

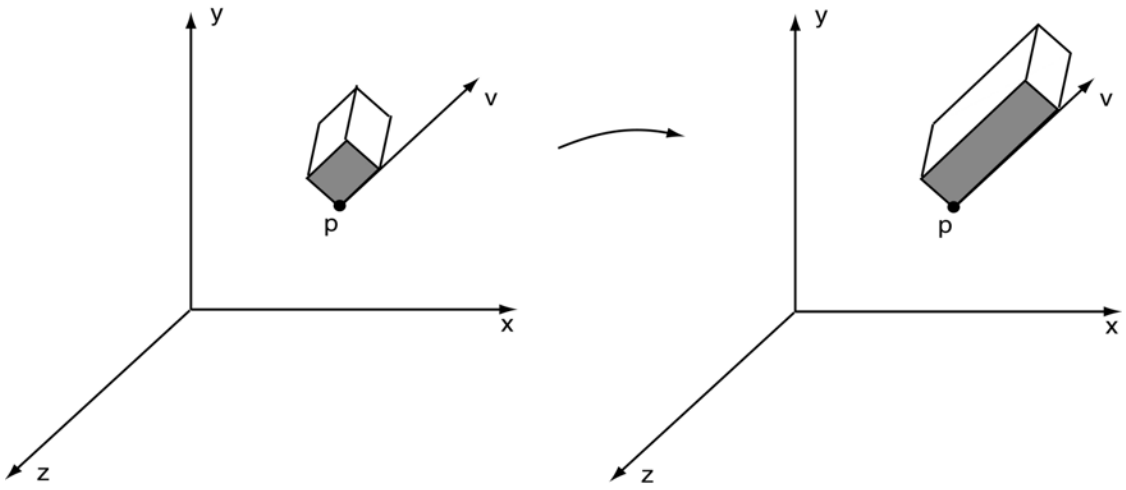
a) (10 points) Given an arbitrary 3D rotation matrix R and its transpose R^T, what does the matrix product R^T R equal? Justify your answer. What does this say about the relationship between R and its transpose?

b) (10 points) A matrix comprised of column vectors that are each of unit length and orthogonal to all the other columns is called an *orthogonal matrix*. A rotation is thus an orthogonal matrix; however, the converse is not generally true. In particular, a reflection is an orthogonal matrix, but not a rotation matrix. Using cross products and/or dot products among the **u**, **v**, and **w** vectors, devise a test to determine whether a 3x3 orthogonal matrix is a rotation matrix and not a reflection matrix. *Justify your answer* using geometrical reasoning.

c) (10 points) Suppose we have an *N*-D space. We can still define a rotation matrix that has the general properties of 3D rotations (i.e., orthogonality). How many degrees of freedom does an *N*-D rotation matrix have? Justify your answer.

Problem 3. 3D affine transformations (20 points)

The basic scaling matrix discussed in lecture scales only with respect to the x, y, and/or z axes. Using the basic translation, scaling, and rotation matrices, one can build a transformation matrix that scales along a ray in 3D space. This new transformation is determined by the ray origin $\mathbf{p} = (p_x, p_y, p_z)$ and direction vector $\mathbf{v} = (v_x, v_y, v_z)$, and the amount of scaling s_v. For clarity, a diagram has been provided, showing a box being scaled with respect to a given ray.



Now consider a ray that is parallel to the *y*-*z* plane. The ray direction will then have the form $\mathbf{v} = (0, \sin\theta, \cos\theta)$ for some θ. Solve for the product of matrices that will scale with respect to this ray.

You may use any of the following standard matrices (from lecture) as building blocks: canonical axis rotations $R_x(\alpha)$, $R_y(\beta)$, $R_z(\gamma)$, scales $S(s_x, s_y, s_z)$, and translations $T(t_x, t_y, t_z)$. You don't need to write out the entries of the 4x4 matrices. It is sufficient to use the symbols given above, supplied with the appropriate arguments. All scale factors are strictly positive. *Show your work*, using words and drawings as needed to support your answer.