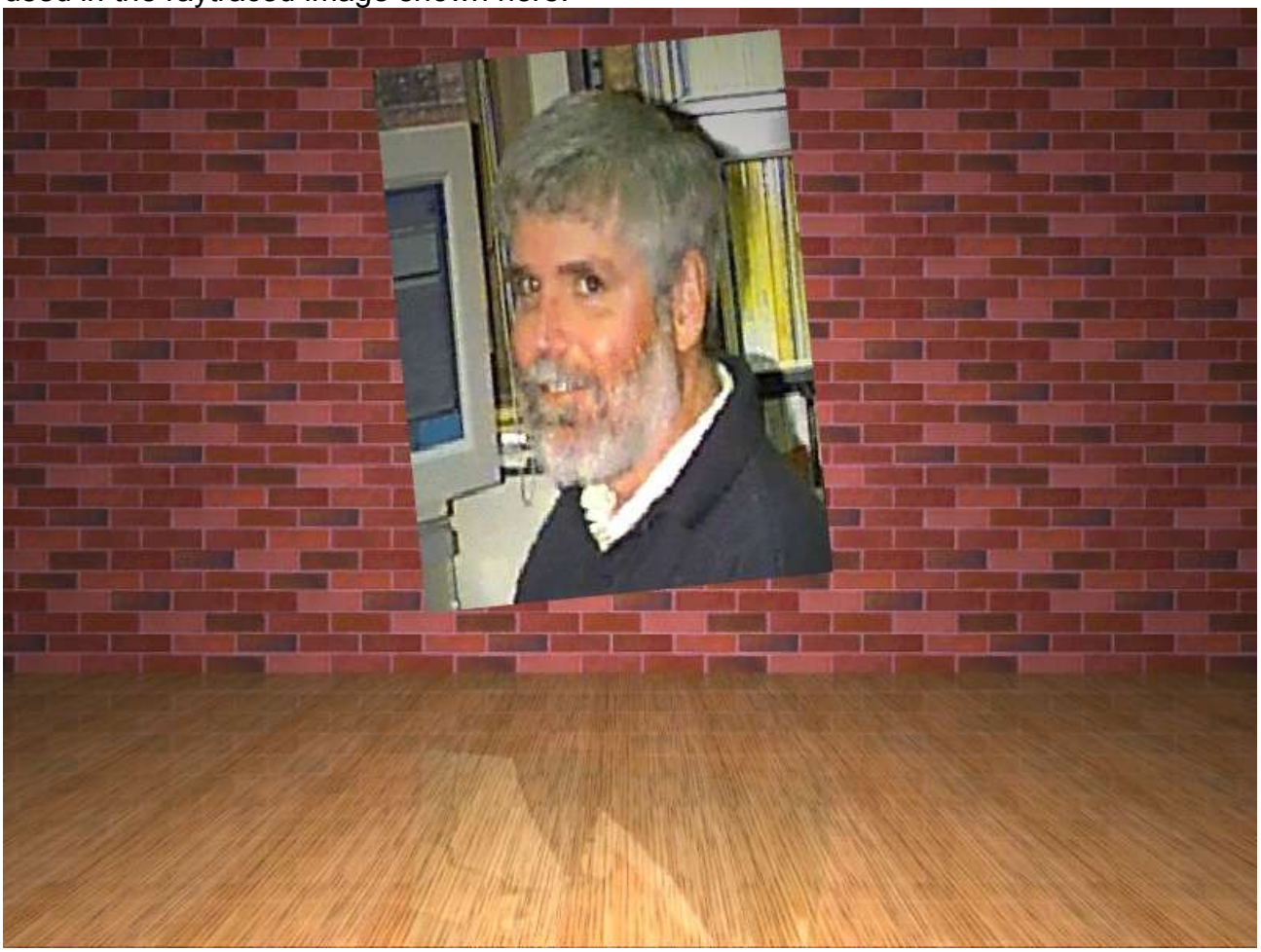
Short Program 3: .ppm texture mapper

Due: Friday, Sep 16 at 11:59 pm

Overview

Texture mapping is a computer graphics procedure in which images are mapped onto synthetic objects such as those used in a ray tracing system. There are two variations on this basic idea. We call these tile and stretch to fit. Both of these are used in the raytraced image shown here.



The brick wall and the floor consist of repeated tilings of an image which is considerably smaller than the area shown. The photograph has been fit to the desired dimensions by stretching the original image. Such stretching can cause both (1) loss of detail and (2) xy distortion. Note that the gradient in the lighting and the reflection from the floor are not present in the textures but are produced by the raytracing procedure.

Your program is to implement the *stretch to fit* form of texture mapping. Thus your mission will be to use an input .ppm file as a texture to create an output .ppm of dimension specified on the command line.

You *must name* your source code *sp3.c*. You may use the solution to short program 2 as a starting point. The source code is in

/home/westall/class/215/assns/sp2data/sp2.c

Program operation

As usual your program will read the input image from *stdin* and write the output image to *stdout.* It will be invoked with a command line similar to the following:

```
a.out  800 200 < mike.ppm  > newmike.ppm
```

In this case the program should load the image *mike.ppm* and then map it onto an output image having dimesions 800 *columns* by 200 *rows*.

The basic algorithm should be as follows:

acquire output dimensions from the command line parameters (see page 40 in the notes for a refresher on this -- don't worry about dealing with missing or broken parameters)

read ppm header of input image

malloc space for reading input image

compute size of output image (3 * (outcols * outrows)) from command line.

malloc space for building output image

use a single call to fread to read the *input* image

for each row of the *output image*

for each column of the *output image*

compute the (row, col) of corresponding pixel in *input image*

copy the (r,g,b) components of the corresponding pixel in the input image to output image

---

At end of the doubly nested loop, use fprintf(stdout, “ “) to write the .ppm header of the output image. Be sure to (a) use P6 format (b) use the *output* dimensions (c) get the dimensions in the correct order with number of columns *before* number of rows.

use a single call to fwrite to write the output image.

Computing the location of the "corresponding pixel".

The following discussion is column based but row computations are analogous

Suppose the *output* image has 914 colums, the *input* image has 644 columns, and we seek to find the "corresponding column" for *output* image column 833.

833/914 = 0.9114 meaning that pixel is at a distance 91.14% of the width from the left edge.

Thus the "corresponding column" in the input image is 0.9114 * 644 = 586.93 = 586

Notes: You must use floating point to compute the fractional amount. You must convert back to integer to obtain the final column. You should *not* try to round to the nearest integer lest you compute column 644.

Additional notes

Use the *xv* program to ensure that your output image looks correct.

Deductions will be made if warnings remained when compiled with -Wall

Use pointer, not array notation in processing the image arrays.

There is a sample input texture *mike.ppm* in the *sp3data* along with a couple of ouput images.

How to submit your program:

NOTE: This procedure has NOTHING in common with "handin" nor "sendlab"
Do NOT even TRY to think about how they fit into this procedure because
THEY DON'T!!

<<<Do NOT turn in any image files, core files, makefiles etc.>>>

You must turn in 1 file: sp3.c. Use the same procdure that was described in the sp1 assignment to turn the program in.

After you think you have turned your programs in, its a good idea to

cd /local/jmw2/215/sp3/wjsmith

and make sure your files are their and they still compile and work correctly.