Stephen’s Programmers Blog

My Undo/Redo functionality and Make File testing functionality

Purpose:The purpose of the code snippets is to basically undo and redo every buffer, image or filter applied on the canvas. The second snippet represents the testing of some of the MIA command line functionality.

My Undo/redo functionality is very similar to what was performed in previous iteration 2. The coding style for this iteration also improved in the sense that better naming conventions were used for better understanding of the program. I also tried to document very descriptively by making a comment at each line where I felt a comment was strongly needed

Code-snippet 1

/\*

\* Returns the most recent PixelBuffer pushed onto the undo stack/vector, and moves it to the redo stack

\*/

PixelBuffer\* BufferHistoryHandler::undoPixelBuffer(){

redoVector.push\_back(undoVector.back()); //Add the most recent PixelBuffer to be undone to the redo vector so we have it saved when it to be undone

undoVector.pop\_back();// Delete the most recent PixelBuffer from the undo Vector.

return undoVector.back(); //Return the new most recent PixelBuffer from the undo vector

}

/\*

\* Returns the most recent PixelBuffer pushed onto the redo stack/vector, and moves it to the undo stack

\*/c

PixelBuffer\* BufferHistoryHandler::redoPixelBuffer(){

PixelBuffer\* getBuffer = redoVector.back(); // Store the most recent PixelBuffer from the redo Vector into a value.

undoVector.push\_back(redoVector.back()); // Get the most recent PixelBuffer from the redo Vector and add it to the Undo Vector

redoVector.pop\_back(); // remove the most recent PixelBuffer from the redo vector. The second most recent is now the most recent pixel buffer

return getBuffer; // return the most recent pixel buffer that was stored in a variable

}

Code snippet 2

RUN\_MIA = ../Mia/./Mia

CLEAR\_IMG = house\*.jpg

#This is the default image

DEFAULT\_IMG = default\_Img.jpg

DEFAULT\_IMG\_PNG = default\_Img.png

#thresh and detect image

THRESH\_DETECT\_IMAGE = house\_threshdetect.jpg

THRESH\_DETECT\_COMPARE\_IMAGE = test\_threshdetect.jpg

#saturate and quantize image

SATURATE\_QUANTIZE\_IMAGE = house\_saturatequantize.jpg

SATURATE\_QUANTIZE\_COMPARE\_IMAGE = test\_saturatequantize.jpg

#different Image

DIFF\_IMAGE = stamp.jpg

#compare

COMPARE = -compare

COMPARE\_IMAGE = compare\_Img.jpg

#sharpen

SHARPEN = -sharpen

SHARPEN\_VALUE = 5

SHARPEN\_WRONG\_VALUE = -5

SHARPEN\_IMAGE = house\_sharpen.jpg

SHARPEN\_COMPARE\_IMAGE = test\_sharpenImg.jpg

DIFF\_SHARPEN\_COMPARE\_IMAGE = test\_sharpen\_diff.jpg

Test\_sharpen:

@test -s $(DEFAULT\_IMG) || { echo "File Does not exist. Please enter a valid file"; exit 1; }

@echo "Please wait, Mia is loading";\

sleep 1s;\

$(RUN\_MIA) $(DEFAULT\_IMG) $(SHARPEN) $(SHARPEN\_VALUE) $(SHARPEN\_IMAGE);\

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SHARPEN MIA COMMAND LINE TEST\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";\

echo "$(DEFAULT\_IMG) is going to be sharpened with value 5";\

#tests for if the files are equal. Test is passed if true;\

if cmp -s $(SHARPEN\_IMAGE) $(SHARPEN\_COMPARE\_IMAGE) ; then \

echo "TEST PASSED!";\

else \

echo "TEST FAILED!";\

fi

The undo/redo functionality makes use of two vectors called a redo vector and an undo vector. The way an undo works is that when the undo button is pressed. The second to last action (pixel buffer) performed on the canvas should be the action displayed when the undo is called. In this case we also need to keep track of the last pixel buffer in the case a redo is called.  
According to my undo Implementation, I get the most recent pixel buffer and add it to the redo vector, so that way when a redo is called, I can easily access that pixel buffer. After it is added to the redo vector, the most recent pixel buffer now has to be deleted from the undo vector in order to make the second most recent pixel buffer now the most recent pixel buffer. In that sense, when the undo is called the pixel buffer that would be displayed on the canvas would be the second most recent pixel buffer made on the canvas. We simply just return the new most recent pixel buffer on the undo stack.   
  
  
The redo functionality just simply works in a reverse order. In order to redo, the most recent pixel buffer on the redo vectored needs to be stored inside a variable so we can just return that variable which would be the most recent pixel buffer. In order to call multiple different recent pixel buffers, we need to add the most recent version on the redo vector to the undo vector and then remove that pixel buffer from the redo vector in order to contain the next most recent pixel buffer when the redo is called. We return the most recent pixel buffer we stored earlier in a variable since it contains the most recent version of the pixel buffer.

 The purpose of section 2 is to accomplish the testing of the MIA tools in a make file something similar to automated testing. A way this is accomplished is that a picture is assigned to a variable which is then later called in a target. The MIA command that is being used is called and then an if statement is used to compare the outputs of the image based on how mia functions. Something very important about this makefile script is the variable assignment which in my opinion I consider a very good approach to adopt. The fact I stored the images in a variable such as SHARPEN\_IMAGE = house\_sharpen.jpg instead of calling the image in the target made it very easy to manipulate and also increased the rate of efficiency at which I tested the MIA command line tools.