filler Namespace Reference

filler namespace: specifies a set of functions for performing flood fills on images. More...

Namespaces

bfs
bfs namespace: specifies a set of filler functions for doing fills on images employing a breadth-first-search approach.

dfs
dfs namespace: specifies a set of filler functions for doing fills on images employing a depth-first-search approach.

Functions

template<template< class T > class OrderingStructure>
animation **fill** (**PNG** &img, int x, int y, **colorPicker** &fillColor, int tolerance, int frameFreq)
General filling function: a general helper that should be invoked by all of the public fill functions with the appropriate color picker for that type of fill. More...

Detailed Description

filler namespace: specifies a set of functions for performing flood fills on images.

Adapted from MP4 authored by CS225 staff, Fall 2010.

Author
Chase Geigle

Date
Fall 2012

Function Documentation

template<template< class T > class OrderingStructure>

```
animation filler::fill ( PNG &          img,
                         int            x,
                         int            y,
                         colorPicker &  fillColor,
                         int            tolerance,
                         int            frameFreq
                       )
```

General filling function: a general helper that should be invoked by all of the public fill functions with the appropriate color picker for that type of fill.

For example, **filler::dfs::fillSolid** should call fill with the correct ordering structure as its template parameter, passing in a **solidColorPicker**.

filler::bfs::fillGradient should call fill with a different ordering structure as its template parameter, passing in a **gradientColorPicker**.

Parameters

img	Image to do the filling on.
x	X coordinate to start the fill from.
y	Y coordinate to start the fill from.
fillColor	The **colorPicker** to be used for the fill.
tolerance	How far away colors are allowed to be to still be included in the fill: this is given as squared euclidean distance in RGB space (i.e. $(c1.red - c2.red)^2 + (c1.green - c2.green)^2 + (c1.blue - c2.blue)^2$).
frameFreq	How frequently to add a frame to the animation, in pixels. For instance, if frameFreq == 1, a frame is added when every pixel is filled. If frameFreq == 10, a frame is added after every 10 pixels is filled.

Returns

An animation that shows the fill progressing over the image.

Todo:

You need to implement this function!

This is the basic description of a flood-fill algorithm: Every fill algorithm requires an ordering structure, which is passed to this function via its template parameter. For a breadth-first-search fill, that structure is a **Queue**, for a depth-first-search, that structure is a **Stack**. To begin the algorithm, you simply place the given point in the ordering structure. Then, until the structure is empty, you do the following:

1. Remove a point from the ordering structure.

   If it has not been processed before and if its color is within the tolerance distance (up to and **including** tolerance away in square-RGB-space-distance) to the original point's pixel color [that is, $(currentRed - OriginalRed)^2 + (currentGreen - OriginalGreen)^2 + (currentBlue - OriginalBlue)^2 \leq tolerance$], then:

   2d bool array

   a. indicate somehow that it has been processed (do not mark it "processed" anywhere else in your code)
   b. change its color in the image using the appropriate **colorPicker**
   c. add all of its neighbors to the ordering structure, and
   d. if it is the appropriate frame, send the current **PNG** to the animation (as described below).
2. When implementing your breadth-first-search and depth-first-search fills, you will need to explore neighboring pixels in some order.

While the order in which you examine neighbors does not matter for a proper fill, you must use the same order as we do for your animations to come out like ours! The order you should put neighboring pixels **ONTO** the queue or stack is as follows: **RIGHT(+x), DOWN(+y), LEFT(-x), UP(-y). IMPORTANT NOTE: *UP* here means towards the top of the image, so since an image has smaller y coordinates at the top, this is in the *negative y* direction. Similarly, *DOWN* means in the *positive y* direction.** To reiterate, when you are exploring (filling out) from a given pixel, you must first try to fill the pixel to it's RIGHT, then the one DOWN from it, then to the LEFT and finally UP. If you do them in a different order, your fill may still work correctly, but your animations will be different from the grading scripts!

3. For every k pixels filled, **starting at the kth pixel**, you must add a frame to the animation, where k = frameFreq.

   For example, if frameFreq is 4, then after the 4th pixel has been filled you should add a frame to the animation, then again after the 8th pixel, etc. You must only add frames for the number of pixels that have been filled, not the number that have been checked. So if frameFreq is set to 1, a pixel should be filled every frame.

4 colors:
tolerance:
- current color;
- original color;

fillColor:
- color when constructed the colorPicker
- color that returns by the colorPicker