How is the photoshop cutout filter implemented?

Asked 16 years, 1 month ago Modified 4 years, 3 months ago

Viewed 12k times

10

Photoshop has a lot of cool artistic filters, and I'd love to understand the underlying algorithms.

One algorithm that's particularly interesting is the Cutout filter (number 2 at the link above).

It has three tunable parameters, Number of Levels, Edge Simplicity, and Edge Fidelity. Number of levels appears to drive a straightforward posterization algorithm, but what the other sliders do technically eludes me.

I would think that they're doing something related to Vornoi diagrams or k-means partitionion, but poking around on wikipedia hasn't resulted in anything that maps obviously to what Photoshop is doing, especially considering how fast the filter renders itself.

Is there any source for technical descriptions of the Photoshop filters? Alternatively, do you have any thoughts about how this particular filter might be implemented?

algorithm language-agnostic image-processing reverse-engineering photoshop

Share

Improve this question

Follow

edited Oct 26, 2008 at 4:57

Martin Beckett

96k ● 28 ● 195 ● 268

asked Oct 26, 2008 at 1:56

fastcall

2,584 ● 4 ● 20 ● 22

Wow, those are cool. I appreciate image processing much more these days after working on a project which involved some (though my tasks were less complex than the others). – Ed Swangren Mar 6, 2009 at 7:03

9 Answers

Sorted by: Highest score (default)

7

✓

Very old question but maybe someone searching for an answer and maybe this helps. Opencv's findcontours and approxPolyDP functions can do this. But we need to prepare the image before main process. First; find most used N colors with k-means. For example find 8 colors.Find contours for each color and then calculate contourArea for all colors one by one (We will have N=8 layers). After that draw filled contours after approxPolyDP for each color from biggest ContourArea to smaller with its pre-calculated color. My another suggestion is eliminate very small contours while calculating contourArea.

Photoshop cutout effects parameters; Number Of Levels=K-Means-find most used N colors. Edge Simplicity=I guess gaussian blur or other removing noise filters like bilateral filter or meanshift filter with edge preserving will be useful for this step.This step can be executed after K-Means and before finding contours. Edge fidelity=openCV's approxPolyDP epsilon parameter.

Share Improve this answer Follow

answered Aug 29, 2020 at 13:45

M wiseman s
98 ● 1 ● 7

gist.github.com/TACIXAT/c25dd24f9af40e5cd0ff91a3178c4dcb - Implemented here. Changed the order of some of the steps. blur -> cluster -> remap colors -> find contours -> draw. Never used most of these functions before, so big thanks for this answer! – douggard Sep 6, 2022 at 8:46

7

Edge detection is usually a Sobel or Canny filter then the edges are joined together with a chain code.
Look at something like the OpenCV library for details

Share Improve this answer Follow

edited Sep 20, 2010 at 13:50

+125

answered Oct 26, 2008 at 4:57

Martin Beckett
96k ● 28 ● 195 ● 268

6

Did you see this post. It explains how to get the same result using ImageMagic, and IM is opensource.

Share Improve this answer Follow

answered Mar 3, 2009 at 23:09

Sunny Milenov
22.3k ● 6 ● 82 ● 107

3

I'm not sure it could be some kind of cell shading, but it also looks like a median filter with a very big kernel size or which was applied several times.

The edge simplicity/fidelity might be options which help decide whether or not to take in account an adjacent pixel (or one which falls inside the kernel) based on difference of color with the current pixel.

Share Improve this answer Follow

answered Nov 26, 2008 at 19:05

Ismael C

0

Maybe not exactly what you are looking for, but if you like knowing how filters work, you could check out the source code of GIMP. I can't say if GIMP has an equivalent of cutout filter you mentioned, but it's worth taking a look if you are truly interested in this field.

Share Improve this answer Follow

answered Oct 26, 2008 at 2:56

Kibbee
66.1k ● 28 ● 144 ● 184

1 This was a good jumping off point for additional googling, but unfortunately GIMP doesn't have a close analog built in. I did find a cool GIMP plugin called pspi, which can load photoshop plugins, but Cutout is one that isn't implemented this way :(– fastcall Oct 26, 2008 at 4:12

0

The number of levels seems to resemble how cell-shading is done and this is how I'd implement that part in this case: you simply take this histogram of the image and divide it into the "No. of levels" amount of sections then calculate an average for each section. Each color in the histogram will then use that average in stead of their original color.

The other two parameters require some more thinking but 'Edge simplicity' seems to detonate the number of segments the shapes are build up off. Or rather: the number of refinements applied to some crude Image Segmentation Algorithms. The fidelity slider seems to do something similar; it probably controls some kind of threshold for when the refinements should take place.

[This might help]

answered Oct 28, 2008 at 2:28

Jasper Bekkers
6,809 ● 36 ● 46

0

Got a simple solution, which would theoretically produce something similar to that filter. Somehow similar to what Ismael C suggested.

Edge *Simplicity* controls window size. Maybe window should be weighted.

But unlike it happens for regular windowed filters this one would take only a fixed size portion of random pixels from this window. The size of the portion is controlled with *Fidelity* parameter.

Set the pixel color to the median of the sample.

Given we have some posterization algorithm, it is applied afterwards.

Here we go!

Please report results if you implement it.

PS. I really doubt that segmentation is used at all.

Share Improve this answer Follow

answered Mar 3, 2009 at 19:31

Maleev

863 ● 3 ● 12 ● 22

0

I imagine it's probably some thresholding, edge-detection (Sobel/Canny/Roberts/whatever) and posterisation.

Share Improve this answer

answered Mar 3, 2009 at 19:48

Follow

James B
8,343 ● 4 ● 34 ● 40

0

From tinkering with it I've found out that:

- it's deterministic
- it doesn't do any kind of pixel based posterization to achieve final effect
- it probably doesn't use any kind of pixel based edge detection, it seems to work rather with areas then edges.
- it calculates the shapes closed polygons to draw (some of the polygon edges might overlap with image edges).
- when the edges of polygons are known then color of each area enclosed in edges (not necessarily belonging to one polygon) is colored with average color of pixels of original image that area covers.
- edge of polygon can intersect with itself. *Especially visible for high edge simplicity.*
- as 'line simplicity' drops, the number of polygon edges increases, but also number of polygons increases.
- edge fidelity influences line polygon edge count but does not influence polygon count

- high edge fidelity (=3) causes single polygon to have very long and very short edges at the same time, low fidelity (=1) causes single polygon to have all edges roughly the similar length
- high edge simplicity and low edge fidelity seem to prefer polygons anchored at edges of image, even at cost of sanity.

Altogether it looks like simplified version of Live Trace algorithm from Adobe Illustrator that uses polygons instead of curves.

... or maybe not.

edited Sep 20, 2009 at 17:08

answered Sep 20, 2009 at 14:57

Kamil Szot
17.8k ● 8 ● 64 ● 65