Analysis of Blur

**Summary**

The company has asked internal startup Hung & Nick Associates to analyze the runtime of the Blur filter designed for the CS61BL project. We conclude that Blur is a time intensive filter that can be made more efficient by improving the speed of color parsing in the Pixel class provided by the company. A significant amount of runtime (80-90% in our test case at Blur Threshold 3) came from requesting color and alpha data from individual pixels.

**Introduction**

Blur is one of our Picture program’s most time intensive tasks. The Blur algorithm works by going through the image one pixel at a time. A grid of pixels is collected around the individual pixel (the size of this grid is determined by the Blur Threshold), and the color and alpha data is collected from each collected pixel. An average color and average alpha is calculated from the surrounding pixels, and the current pixel’s color and alpha are set to these averages. As you can imagine, it requires many computations to iterate through each pixel, collect batches of pixels, and parse individual pixels for color and alpha data.

There are several key factors that have a significant effect on how fast our filter blurs an image. We have listed them in order of “impact level,” and summarized how each item affects the runtime. We have also listed what user behaviors affect these factors.

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **Purpose** | **Factor’s Impact on Runtime** | **How User Behavior Can Affect Factor** |
| Iteration through every pixel in the image | We must iterate through individual pixels to manipulate the canvas of pixels. | Low Impact | Factor’s impact is linearly proportional to number of pixels in image. Larger pictures will have larger runtimes. |
| Collecting a set of pixels surrounding each pixel in the image. This forms a “pixel group.” | The blur color and blur alpha is calculated from average values from pixels surrounding our pixel of interest. | Medium Impact | Lower blur thresholds can be executed significantly faster than higher blur thresholds because smaller groups of pixels will be analyzed, per pixel. |
| Parsing pixels for color/alpha and calculating average color and alpha based on the results. | Perform arithmetic to average the colors for our pixel of interest. | Significant Impact | Factor is proportional to (2\*Threshold + 1)2; thus higher thresholds will have much higher runtimes. |

**Methods**

To determine the runtime impact of each factor, we embedded three lines of code in the blur function in Java:

**long** startTime = System.*currentTimeMillis*();

[blur Java code goes here]

**long** endTime = System.*currentTimeMillis*();

System.*out*.println("That took " + (endTime - startTime) + " milliseconds");

The three lines of code check the machine time before blur’s math executes, then checks the time after blur is finished working. Then the time difference is outputted in millisecond form. The System.currentTimeMillis() approach uses the computer’s clock to measure start and end times. Because Java requires time to check the clock, our method can be inaccurate up to 50 milliseconds. Such a margin of error is tolerable in our tests because runtimes are orders of magnitude larger than this potential inaccuracy.

To measure each factor’s impact, we performed a battery of tests for two images (attached in Appendix):

* Image 1: Bikers.jpg (a 1224x816 color image; 1 megapixel)
* Image 2: Bird.jpg (a 3008x2000 color image; 6 megapixels)

The photo program was closed before each new test to make sure subsequent tests were performed in an identical program starting state.

We selectively disabled features in order to gauge their contribution to runtime. We ran the blur function on a computer “at rest” (not running any CPU intensive background activities). Disabling any feature in blur causes blur to return incorrect images, but we ignored this because we are only interested in runtime. Three time measurements were collected per image, per disabled feature. Thus we collected 6 time measurements total per disabled feature. Times were averaged and are reported in the Evidence section. The blur threshold for each test was set to 3.

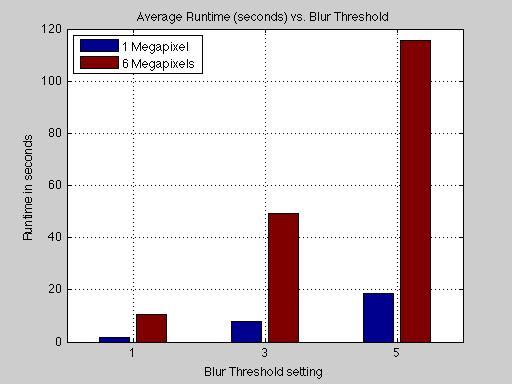
Note that some features could not be disabled. The Border Checking feature checks each pixel near a border to make sure we do not grab pixels from outside the image borders (this would throw an error). We do not believe Feature 3 affects runtime significantly because it is a simple if/else check that is performed for each pixel. The time impact from this feature is small and linearly proportional to the number of pixels in our image.

We performed an additional test where we ran blur (all features enabled) at different blur thresholds as well. This helps us gauge how Blur Threshold affects runtime.

**Computer Details:** Consumer-grade Macbook Pro, 2.8 gHZ Intel Core i7 with 8 GB 1067 mHZ DDR3 RAM. Running OS X 10.7.4 Lion. Running Picture.Java through Eclipse at default run settings.

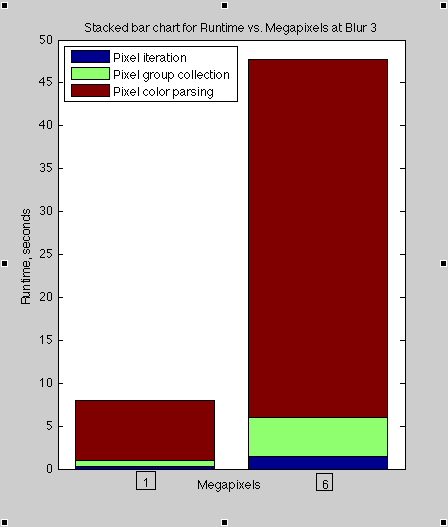
**Evidence**

**Figure 1:** Average runtimes for two different pictures, at three thresholds



Note: All raw data is reproduced in the appendix at the end of this paper.

**Figure 2:** A stacked chart showing the approximate runtime contributions from pixel iteration, pixel group collection, and color parsing of pixels for average color/alpha calculation. Relatively trivial runtime sources like arithmetic and creation of new objects are not included.



**Conclusions**

Runtime increase is (approximately) linearly proportional to the number of megapixels in the image; if a one megapixel image can blur in N time, a six megapixel image will blur in approximately 6N time at the same threshold.

Runtime increase for an image is proportional to (2\*Threshold+1)2. In other words, if it takes N time to blur an image at Threshold 1 ((2\*1+1)2 = 9), it will take approximately 5.4 times as long to blur the image at Threshold 3 ((2\*3+1)2 = 49; 49/9 = 5.4).

Ultimately, the runtime of blur is dominated by the time it takes to request color and alpha data from pixels. Parsing pixels for color and alpha is responsible for 80-90% of the runtime in blur at Threshold 3. For the data in Figure 2, parsing pixels occupied 85-90% of runtime for both one and six megapixel images. While experiments were not performed to test runtimes at other blur thresholds, we still expect pixel color/alpha parsing to contribute the lion’s share of runtime.

Should the company wish to improve runtime of Blur, Hung and Nick Associates recommends that the company invest in improving efficiency in the Pixel class.

**Appendix**

Raw Data

**For Bikers.jpg (time is in milliseconds), all features enabled**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Blur Threshold | Runtime Trial 1 | Runtime Trial 2 | Runtime Trial 3 | Mean Runtime |
| 1 | 1709 | 1690 | 1696 | 1698 |
| 3 | 7897 | 7837 | 8233 | 7989 |
| 5 | 18323 | 18792 | 18767 | 18627 |

**For Bird.jpg (time is in milliseconds), all features enabled**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Blur Threshold | Runtime Trial 1 | Runtime Trial 2 | Runtime Trial 3 | Mean Runtime |
| 1 | 10368 | 10488 | 10543 | 10466 |
| 3 | 52973 | 47498 | 47674 | 49382 |
| 5 | 115725 | 112777 | 117810 | 115437 |

**For Bikers.jpg (time is in milliseconds), some features disabled**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Description | Runtime Trial 1 | Runtime Trial 2 | Runtime Trial 3 | Mean Runtime |
| Pixel iteration: ON  Pixel collection: OFF  Pixel color/alpha parsing: OFF | 242 | 256 | 245 | 248 |
| Pixel iteration: ON  Pixel collection: ON  Pixel color/alpha parsing: OFF | 973 | 972 | 947 | 964 |

**For Bird.jpg (time is in milliseconds), some features disabled**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Description | Runtime Trial 1 | Runtime Trial 2 | Runtime Trial 3 | Mean Runtime |
| Pixel iteration: ON  Pixel collection: OFF  Pixel color/alpha parsing: OFF | 1453 | 1475 | 1500 | 1476 |
| Pixel iteration: ON  Pixel collection: ON  Pixel color/alpha parsing: OFF | 5870 | 6074 | 6168 | 6037 |

**Formulas used to calculate approximate individual feature runtimes:**

IterationTime\* = Measured directly by disabling all higher level features

PixelCollectionTime = (mean runtime with no color/alpha parsing) – IterationTime

PixelColorAlphaParseTime = (Mean runtime with all features enabled) – IterationTime – PixelCollectionTime

\*Note: We understand that IterationTime also contains runtime contributions from other activities in the Java code, but we are grouping everything at this lowest level of activity into the runtime for iteration.