EEE598: Mobile Systems Architecture Fall 2017

**Project assignment 1**: Speedy Tilt Shift Acceleration

Official Release Date: *2017 September 1*
Checkpoint Review Date: *2017 September 22*
Due Date: *2017 October 6*
Send any requests for clarifications or errata to likamwa@asu.edu

## ***Learning Objective:***

*Analyze the performance implications of mobile image processing implementations.*

Fancy image transformations -- colloquially, "filters" -- have dramatically captured the attention of the mobile market, to the point that operating systems and devices take special measures to promote efficient image processing. This project assignment is intended for you to learn about system bottlenecks to image processing and tricks to optimize performance.

For this project assignment, you will be building "Tilt-Shift" functionality, simulating macro photography for full-size scenes. You will design and benchmark your implementation using Java, C++ and ARM NEON intrinsics.

### Honor code

Thoroughly read and acknowledge this affidavit BEFORE you open the assignment.

- My team will work independently on this project.
- I will **NOT** look up other tilt-shift implementations on the Internet.
- I will **NOT** share or discuss *source code* with other teams. (I am allowed to discuss concepts)
- I will **NOT** copy/paste code from any sources.
- I will **NOT** debug others' code.
- I will **NOT** receive help debugging my own code.
- I will **NOT** copy/paste any information from other sources into my documentation.
    - Any referenced information will be sufficiently paraphrased and cited.

### Optional reading

- Gaussian blur: https://en.wikipedia.org/wiki/Gaussian_blur
- Examples of Tilt shift blur: https://inst.eecs.berkeley.edu/~cs194-26/fa14/upload/files/projFinalUndergrad/cs194-ko/part1/

### Github

Use Github to maintain a ***private*** repository of your code. Github education pack allows unlimited private repositories. Name your git repository "eee598-tiltshift-[lastname1]-[lastname2]-[lastname3]", e.g., "eee598-tiltshift-likamwa-kodukula". Add "roblkw-asu" and "kodukulav" as collaborators.

You should commit to the git at least once a week.

**Application hooks**

For convenience and standardization, this assignment will provide an application template that includes application infrastructure to process an image file through transformations.

You will be provided with three dummy functions:

- tiltshift_java(float sigma_near, float sigma_far, int a0, int a1, int a2, int a3)
- tiltshift_cpp(float sigma_near, float sigma_far, int a0, int a1, int a2, int a3)
- tiltshift_neon(float sigma_near, float sigma_far, int a0, int a1, int a2, int a3)

It is your job to populate the three functions.

All three functions take in two sigma float values: sigma_near ($\sigma_1$) and sigma_far ($\sigma_2$), representing the blur of near and far objects, and four integers, a0, a1, a2, a3 representing the location and gradient of transition between blurry and non-blurry regions.

**Tilt-shift algorithm**

The tilt-shift effect simulates the focal blur effects created when photographing miniature scenes. A simple way to create the tilt-shift effect is to apply a Gaussian blur kernel of different sigma value (blurriness) depending on the assumed physical depth of the subject. For this assignment, we will use the y-coordinate to assume the depth of the image contents.

The tilt shift filter should take in 4 integer arguments representing the image rows for the progression of blur. The filter should also take in two float arguments, $\sigma_1$ and $\sigma_2$ representing the blur of the "near" and "far" regions respectively. See Figure 1 for specification on how to assign blur operations.

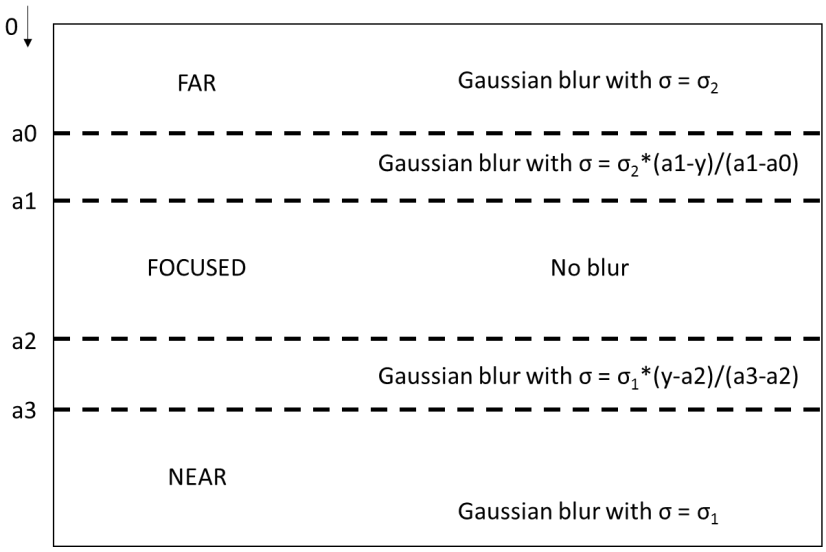


Figure 1: Tilt shift blur algorithm Gaussian blur kernels

**Gaussian blur algorithm**

In this section, we provide the details of the Gaussian blur algorithm. The algorithm takes an image region and kernel matrices as inputs and outputs a blurred image region. Each output pixel is a Gaussian-weighted combination of nearby input pixel values. Gaussian blur should be performed on each color channel (R, G, B) independently. There are two approaches to computing Gaussian blur: (i) a weight matrix approach; and (ii) a weight vector approach. You may choose to implement either.

| Parameter | Symbol |
|---|---|
| (y, x) | Pixel indices |
| Standard deviation | $\sigma$ |
| Radius of the kernel | r |
| Input image region | p |
| Gaussian weight kernel matrix | G |
| Gaussian weight vector | G |
| Output image region | P |

NOTE: Input pixels with out-of-bound indices should be treated as black pixels. e.g., p(-1, -1) = 0.

**Gaussian blur through a weight matrix approach**

The size of kernel matrix, G, is **(2r+1) by (2r+1)**, where **r = ceil (3 $\sigma$)**
Then, each entry G(y,x), where $x \in [-r, r]$ and $y \in [-r, r]$ can be generated as follows:

$$\mathbf{G(y,x)} = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2+y^2}{2\sigma^2})$$

Then, each pixel in the output image P is obtained by applying G as a weighted sum on a patch of pixels of the input image p. Mathematically, for each (y, x) in P:

$$\begin{aligned}\mathbf{P(y,x) = G(-r, -r)*p(y-r, x-r), + \dots + G(0, -r)*p(y, x-r),+ \dots + G(r, -r)*p(y+r, x-r) +}\\ \mathbf{G(-r, 0)*p(y-r, x), \quad + \dots + G(0, 0)*p(y, x), \quad + \dots + G(r, 0)*p(y+r, x) +}\\ \mathbf{G(-r, r)*p(y-r, x+r), + \dots + G(0, r)*p(y, x+r),+ \dots + G(r, r)*p(y+r, x+r)}\end{aligned}$$

**Gaussian blur through a weight vector approach**

It is often faster to perform the Gaussian blur as two independent transforms across each x-y dimension.
To do this, generate Gaussian weight vector G, where $k \in [-r, r]$ as follows:

$$\mathbf{G(k)} = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{k^2}{2\sigma^2})$$

Then, a first transformation generates the intermediate matrix, q(y, x):

$$\mathbf{q(y, x) = G(-r)*p(y-r, x), + \dots + G(0)*p(y, x),+ \dots + G(r)*p(y+r, x)}$$

In a second transformation, this intermediate matrix is used to generate the final output:

$$\mathbf{P(y, x) = G(-r)*q(y, x-r), + \dots + G(0)*q(y, x),+ \dots + G(r)*q(y, x+r)}$$

**Benchmarks**

Provide comparisons against your 3 tilt-shift implementations through the use of benchmark operations. The results of these benchmarks should be charted and discussed in your documentation.

For each implementation, you should:

- Use timestamps to measure elapsed time of operations.
- Vary image sizes to see relationships between elapsed time.
- Set the tablet into performance mode, optimized mode, see if there are changes.
- Change the clock frequency of the tablet.

**Documentation**

- Goals
    - State the high-level objectives you sought to achieve.
    - Restate the assignment description in your own way.
- Design/Implementation
    - Describe your application components, including any activities or classes
    - Describe interfaces between application components.
- Provide an evaluation of benchmarks
    - Charts and/or tables of your benchmarks
    - Speculate on reasons for performance differences between different implementations
- Describe challenges encountered along the way
    - How you overcame the challenges.
    - Include any missteps or misunderstandings.

**Full Deliverables (October 6)**

Please follow these submission directions. If you do not follow these directions, you will be docked 5 points.

1) Source code
    a. Print to pdf your source code files. One pdf per source code file
    b. Package your app into an .apk file
    c. Zip your Android project folder, including your source code.
2) Documentation
    a. Include your documentation as a pdf.
3) Declaration
    a. In a text file, type out the honor code verbatim. Include a statement declaring that you and your team fully abides by the honor code.
4) Package all the above files in to a zip. Use the filename "Lastname1-Lastname2.zip", e.g., "Kodukula-LiKamWa.zip". Submit the file via Blackboard.
5) Git
    a. Invite "roblkw-asu" and "kodukulav" as collaborators to your Git repository.

**Checkpoint Deliverable (Sept 22)**

For the checkpoint, you need to complete a Java OR C++ implementation of the tilt-shift blur. Use the same deliverable instructions above. No documentation is required for this checkpoint.

The checkpoint is worth 10% of your assignment 1 grade.

**Full Deliverable Rubric**

- 15%**:** Tilt-shift blur in Java
    - Gaussian correctly working and applied correctly
    - File-level and function-level comments
- 15%: Tilt-shift blur in C/C++
    - Gaussian correctly working and applied correctly
    - File-level and function-level comments
- 15%: Tilt-shift blur in NEON intrinsics
    - Gaussian correctly working and applied correctly
    - File-level and function-level comments
- 10% (Extra Credit): Tilt-shift blur in NEON Assembly
    - Gaussian correctly working and applied correctly
    - File-level and function-level comments
- 15%: Performance measurements
- 30%: Thorough Documentation
- 10%: Consistent git commits to the repository.
    - Aim to update once a week or more frequently.