White Paper

Denoising

February 2007
WP-03020-001_v01

Document Change History

Version	Date	Responsible	Reason for Change
_v01	February 15, 2007	AK, TS	Initial release

Go to sdkfeedback@nvidia.com to provide feedback on Denoising.

Denoising

Abstract

Image denoising algorithms may be the oldest in image processing. Many methods, regardless of implementation, share the same basic idea – noise reduction through image blurring. Blurring can be done locally, as in the Gaussian smoothing model or in anisotropic filtering; by calculus of variations; or in the frequency domain, such as Weiner filters. However a universal "best" approach has yet to be found.

Among others, neighborhood filters – filters that restore a pixel by taking an average of the values of neighboring pixels with a similar color – are of great interest due to the simplicity of implementation and the quality of the result. A GPU based technique is to store an image in a texture and use a pixel shader to implement a kernel.

Alexander Kharlamov
NVIDIA Corporation

Motivation

White noise is one of the most common problems in image processing. Even a high-resolution photo is bound to have some noise in it. But, whereas for a high-resolution photo a simple box blur may be sufficient, because even a tiny features like eyelashes or cloth texture will be represented by a large group of pixels. Unfortunately, this is not the case with video where real-time noise reduction is still a subject of many researches. However, current DX10 hardware allows us to implement high quality filters that run at acceptable frame rate.

Figure 1. Filtering

How It Works

The main idea of any neighborhood filter is to calculate pixel weights depending on how similar their colors are. We describe two such methods: the *K Nearest Neighb*ors and *Non Local Means* filters

K Nearest Neighbors Filter

The K Nearest Neighbors filter was designed to reduce white noise and is basically a more complex Gaussian blur filter. Let **u(x)** be the original noisy image, and **KNNh,ru(x)** be the result produced by the **KNN** filter with parameters **h** and **r**. Let **Ω(p)** be the spatial neighborhood of a certain size surrounding pixel **p**. We will consider it to be a block of pixels of size **NxN**, where **N = 2M + 1** – so that **p** is the central pixel of **Ω(p).** Then let

```
KNNh,ru(x) =   where C(x) is the normalizing coefficient.
```

Figure 2. Original, Noisy and KNN Restored Picture

Non Local Means Filter

The Non Local Means filter is a more complex variation of the KNN filter. Using the same notation as for KNN, let **NLMh,r,Bu(x)** be the restored image, Let **B(q)** be the spatial neighborhood of a certain size surrounding pixel **q**. We will consider it to be a block of pixels of size **KxK**, where **K = 2L + 1** – so that **q** is the central pixel of **B(q).** Then let

$$NLM_{h,r,B}u(\mathbf{x}) = \frac{1}{C(\mathbf{x})}\int_{\Omega(\mathbf{x})} u(\mathbf{y}) e^{-\frac{|\mathbf{y}-\mathbf{x}|^2}{r^2}} e^{-\frac{ColorDistance(B(x),B(y))}{h^2}} d\mathbf{y}$$ where C(x) is the normalizing coefficient, and

$$ColorDistance(\mathbf{B}(\mathbf{x}),\mathbf{B}(\mathbf{y})) = \frac{1}{S(\mathbf{B})}\int_{\mathbf{B}(\mathbf{x})} |u(\mathbf{y}+(\mathbf{x}-\boldsymbol{\alpha}))-u(\boldsymbol{\alpha})|^2 \, d\boldsymbol{\alpha}$$

where **S(B)** is **B**'s area. Thus **ColorDistance(B(x),B(y))** represents a normalized sum of absolute differences between blocks around pixel **u(x)** and around pixel **u(y).**

Figure 3. Original, Noisy and NLM Restored Picture.

Note: NLM can even fix some flaws in original images (Figure 3)

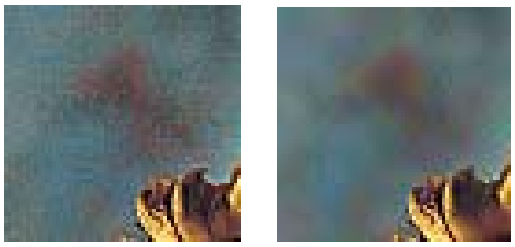

Figure 4. Original and NLM Restored Picture.

Choosing Parameters

There is always the question of choosing the best parameters for KNN and NLM. As evident from the previous equations the weights for surrounding pixels depend on the following parameters

- **r** – can be considered to be a traditional Gaussian blur coefficient. If then KNN transforms into Gaussian blur. As with Gaussian blur **r** should be equal to **N**.
- **h** – is a lot more tricky to choose. In fact, the best way to select h is to make it user defined as estimation of quality of the image is highly subjective.
- The size of Ω depends on the size of the image, but good visual results are usually achieved with a 7x7 block of pixels.
- NLM has one additional parameter – the size of **B**. **B** is usually a 7x7 block of pixels.

Implementation Details

Both methods can be easily implemented in DX10. Original and noisy pictures are loaded as textures and filter kernel is implemented in a pixel shader. The restored image is rendered into a texture.

Running the Sample

KNN has the following parameters that can be altered by the user:

- Noise Level corresponds to in the formula for **KNNh,ru(x).**
- Lerp Coefficient, Weight Threshold and Counter Threshold are used in a simple modification:

  Once a weight is counted, it can be compared to **g_WeightThreshold**. The number of weights that are greater than **g_WeightThreshold** are stored in the **fCounter** variable. The **g_WeightThreshold** should lie between (0.66f, 0.95f).

```
fCounter += (fWeight > g_WeightThreshold) ? 1.0f: 0.0f;
```

  Once all the weights are counted, and a restored pixel has been computed, we blend between the original and restored pixels. The blending coefficient is **g_LerpCoefficient**, its values should lie between (0.00f 0.33f). A simple check determines if the block is smooth: if **fCounter** exceeds **g_CounterThreshold** (usually more then 66% of all pixels in the block) then the block should be considered smooth, otherwise we presume that the block contains edges or small features. The blending coefficient is altered to preserve restored pixels in the smooth areas and original pixels otherwise.

```
f4Result.rgb = (fCounter > (g_CounterThreshold * iWindowArea))?
  lerp (f4Result, f4Tex00, 1-g_LerpCoefficient):
  lerp (f4Result, f4Tex00,   g_LerpCoefficient);
```

- Gaussian Sigma is a traditional Gaussian blur coefficient. It corresponds to in the formula for **KNNh,ru(x).**
- Window Radius determines the size of Ω. If Ω is a block of pixels of size **NxN**, where **N = 2M + 1** then **M** = Window Radius.

NLM has one additional parameter:

- Block Radius determines the size of **B**. If **B** is a block of pixels of size **KxK**, where **K = 2L + 1** then **L** = Block Radius.

Performance

Traditionally, image filtering is much faster on the GPU than on the CPU. Both KNN and NLM run faster on the GPU. On the CPU, KNN runs in real time while the NLM is much slower.

Figure 5 shows what noise does to an image. Figure 6 shows repaired images.

Original Image	Noisy Image

Figure 5. Noise in an Image

Image restored with KNN, runs at ~ 290 FPS with *Window Radius = 3*

Image restored with NLM, runs at ~ 27 FPS with *Window Radius = 3* and *Block Radius = 3*

Figure 6. Restored Images

References

[1] A. Buades, B. Coll, and J. Morel. *"Neighborhood Filters and PDE's"*. Technical Report 2005-04, CMLA, 2005.

[2] L. Yaroslavsky. *"Digital Picture Processing - An Introduction"*. Springer Verlag, 1985.

.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com