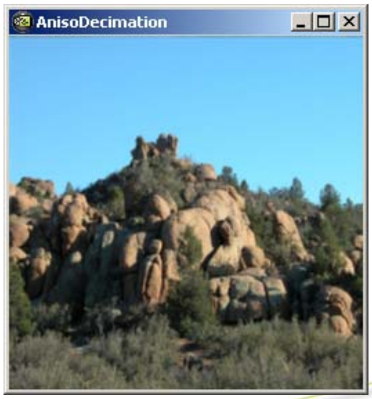
Technical Report

Accelerated Decimation Using Anisotropic Filtering



DEVELOPMENT

Abstract

[abstract]

Decimation filters are commonly performed by applications prior to performing large low-pass filters for post-processing special effects. This code sample demonstrates a variety of different mechanisms for performing these decimation filters, including using the hardware's built-in anisotropic texture filtering capabilities.

Gary King
sdkfeedback@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

February 11, 2005

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

Detailed Description

Decimation filters are commonly used to reduce the amount of time (and fillrate) required to perform large post-processing effects in games, and to scale the effective filter size with screen resolution. These allow applications to use one set of shaders (for example, 15-tap Gaussian filters) for post-processing effects, independent of screen resolution (the example Gaussian filter is applied to a decimated 256x256 render target, rather than the full-size 1600x1200 frame buffer). There are two common types of decimation filters currently used in applications:

1. A single point- or bilinear- filtered sample is taken from the full-size image per pixel in the decimated image (this can be performed using the `StretchRect` API call in DirectX 9).
2. Multiple point- or bilinear- filtered samples are taken from the full-size image per pixel in the decimated image, and averaged. This is commonly performed using a pixel shader, and many applications use 4 bilinearly filtered samples.

The first option is obviously the fastest; however, it can cause aliasing artifacts due to undersampling. The second option eliminates most aliasing; however, because each sample is fetched independently, it can be a source of texture cache thrashing.

An alternative to option #2 that avoids texture cache thrashing is to use the hardware's built-in anisotropic filtering capabilities to automatically average multiple samples in one lookup. This can be performed trivially in 1-2 passes (one for >2:1 horizontal anisotropy and one for >2:1 vertical anisotropy), resulting in a performance increase of >30% as measured on a GeForce 6800GT. The implementation of this is as follows, assuming a 1600x1200 source render target, a 256x600 intermediate render target, and a 256x256 render target for the final, decimated image:

```
    Bind source as tex0

    Set tex0 filtering to aniso, max aniso to 8

    Render a full-screen quad to intermediate target, offset by ½ vertical texel

    Set intermediate target as tex0

    Render a full-screen quad to decimated target
```

The two-pass approach is necessary so that the hardware computes the anisotropic ratio properly (since 1600/256 and 1200/256 is approximately square, no anisotropic filtering would be performed in a single decimation pass). By offsetting the vertical texture coordinate by ½ texel in the first pass, we can use standard bilinear filtering to average 2 scanlines of samples simultaneously; this reduces the amount of filtering work that will need to be repeated in the second pass (and reduces the number of pixels that need to be filled in the first pass).

This technique provides high-quality decimated images much faster than the conventional 4-tap pixel shader approach, and is extremely simple to implement.