NVIDIA

Technical Report

Antialiasing with Transparency

DEVELOPMENT

Antialiasing with Transparency

Introduction

Alpha testing is a simple and relatively easy way to simulate sparse geometry such as vegetation without necessitating complex models. Geometric detail is encoded into a fourth component of an RGBA texture to form a cutout of the pixels to be rendered. This allows us to produce a complex image by just rendering a texture-mapped quad.

The problem with using alpha testing to virtually simulate geometry is the hard edge that is produced where the test occurs. This is because the alpha test compares a fragment's alpha value against a reference value and either keeps or discards the fragment based on the test result. What's worse is that the granularity of the edge matches the size of the texels displayed on the screen. Therefore, magnified textures will suffer horribly from this cookie cutter approach.

The conventional solution for dealing with this problem is to use alpha blending. Instead of using a single reference value to decide whether a pixel will be rendered or not, the alpha value is used to decide how much the pixel being rendered will contribute to the final image. The current pixel's alpha value represents its own contribution whereas the inverse value (one minus the current alpha) is considered what the previous rendered pixel's contribution will be. This is very effective at producing soft edges, but has the disadvantage of being prohibitively slow. It also necessitates sorting all objects from back to front causing further CPU intervention in the GPU's work.

If we could render transparent objects with antialiasing, we would also achieve smooth virtual edges. One approach would be to supersample the scene; this always works, but has major disadvantages with respect to performance because four times the original number of pixels are rendered.

Transparency Supersampling

On GeForce 7800 GTX, NVIDIA has exposed a Supersample renderstate allowing *individual primitives* to be supersampled (in combination with a multisampled backbuffer). This is especially effective for textures with hard edges like a chain link fence or lattice work.

Transparency supersampling provides optimal flexibility: because transparency supersampling can be controlled on a per-primitive basis, you get the advantage of extremely high quality supersampling for alpha-tested textures but you can avoid this cost for other geometry.

Transparency Multisampling

Multisampling, on the other hand, is based on geometric pixel coverage, and will not work with fragments where the underlying geometry does not match the actual pixel's geometry. Because multisampling computes coverage before the pixel shader is run we need a way to indicate sample coverage based on a fragment's alpha value. With GeForce 7800 GTX, NVIDIA has exposed a render state that allows individual primitives to be multisampled. Multisampling is accomplished by using the computed alpha as a measure of how many samples the pixel will take. A texel with an alpha value of one will contribute to all samples in the destination pixel, while texels with successively lower alpha values will contribute to fewer samples. Transparency multisampling is especially effective for textures where soft edges are desirable (as is the case with most vegetation).

When using transparency multisampling, your art assets must have a gradual falloff towards their edges in the alpha channel. The alpha channel should be created or scaled in the shader such the computed alpha goes from the alpha reference value to 0.0f at the boundaries. This ensures that the coverage values will fall off nicely, producing smooth antialiased edges.

In Figure A we have an alpha tested leaf against a tan colored background. With 4x supersampling, we will have 4 unique results as the pixel shader is run four times (since each original pixel now becomes four pixels in the blown-up image for supersampling). With MSAA, the results of all four samples will be dictated by the red sample point. This is incorrect as the upper right sample is not covered by the leaf and should remain the background color. The problem is that as far as the hardware is concerned all four samples are covered since we are rendering an alpha tested quad. With transparency multisampling, we still run the pixel shader once. However, the samples that are covered are computed based on the computed alpha value and not geometric coverage.

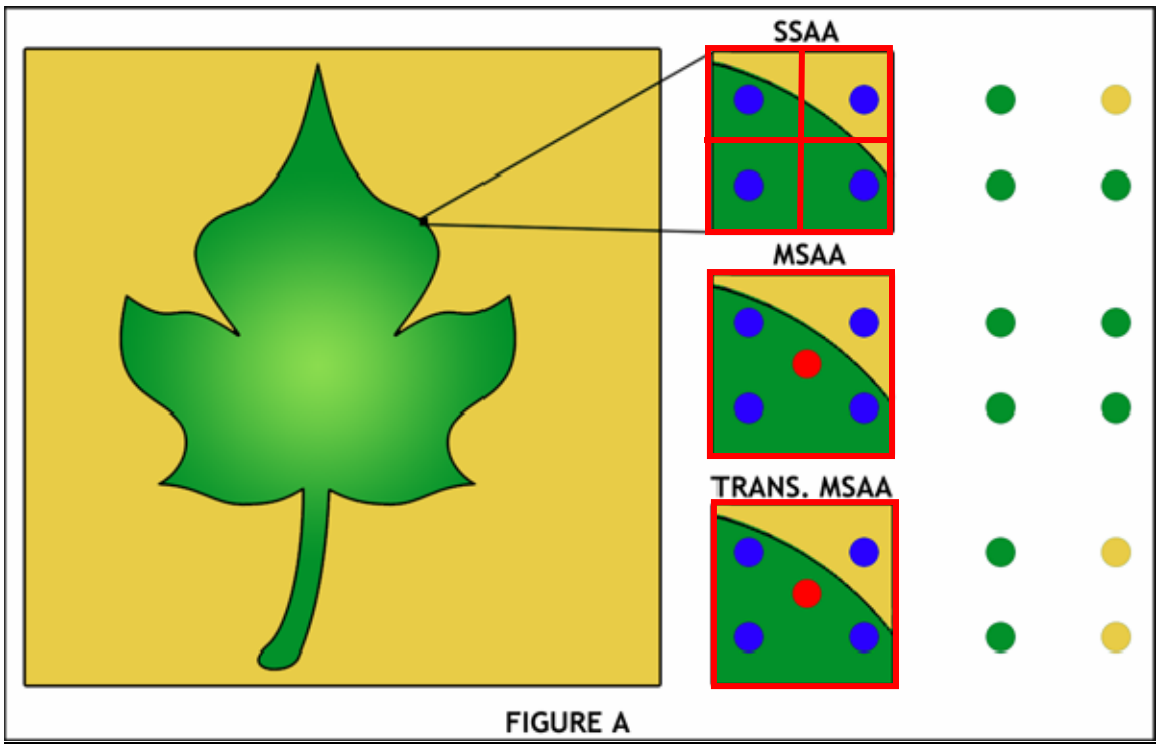

FIGURE A

The "Antialiasing with Transparency" sample that accompanies this whitepaper illustrates how to use these new renderstates to effectively antialias textures that contain transparent texels. The sample renders several thousand grass objects similar to those described by Kurt Pelzer in his GPU Gems chapter entitled "Rendering Countless Blades of Waving Grass".

Sample Code

For quick reference here is the bit of code that activates the two states:

To enable Transparency Multisampling in Direct3D first check if it is supported:

```
(pd3d->
CheckDeviceFormat(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
                  D3DFMT_X8R8G8B8, 0,D3DRTYPE_SURFACE,
       (D3DFORMAT)MAKEFOURCC('A', 'T', 'O', 'C'))) == S_OK);
```

When rendering alpha tested fragments multisampling can be turned on by setting:

```
pd3dDevice->
SetRenderState(D3DRS_ADAPTIVETESS_Y,
               (D3DFORMAT)MAKEFOURCC('A', 'T', 'O', 'C'));
```

To enable Transparency Supersampling first check if it is supported:

```
(pd3d->
CheckDeviceFormat(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
                  D3DFMT_X8R8G8B8, 0, D3DRTYPE_SURFACE,
       (D3DFORMAT)MAKEFOURCC('S', 'S', 'A', 'A'))) == S_OK);
```

When rendering alpha tested fragments supersampling can be turned on by setting:

```
pd3dDevice->
SetRenderState(D3DRS_ADAPTIVETESS_Y,
               (D3DFORMAT)MAKEFOURCC('S', 'S', 'A', 'A'));
```

Both modes can be turned off by setting:

```
pd3dDevice->SetRenderState(D3DRS_ADAPTIVETESS_Y,
                           D3DFMT_UNKNOWN);
```

Bibliography

1. Kurt Pelzer "Rendering Countless Blades of Waving Grass" in "GPU Gems" ed. Randima Fernando, Addison Wesley: 2004, ISBN 0-321-22832-4, p 343
http://developer.nvidia.com/object/gpu_gems_home.html

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