Technical Report

Nature Scene

Abdul Bezrati
abezrati@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

Abstract

This paper describes few techniques that, when used together, can produce a realistic looking natural scene. The featured techniques include:

- Antialised Moving Grass
- Water with Reflection and Refraction
- Dynamic Sky Dome with Horizon Tinting
- Multi-Layer Terrain

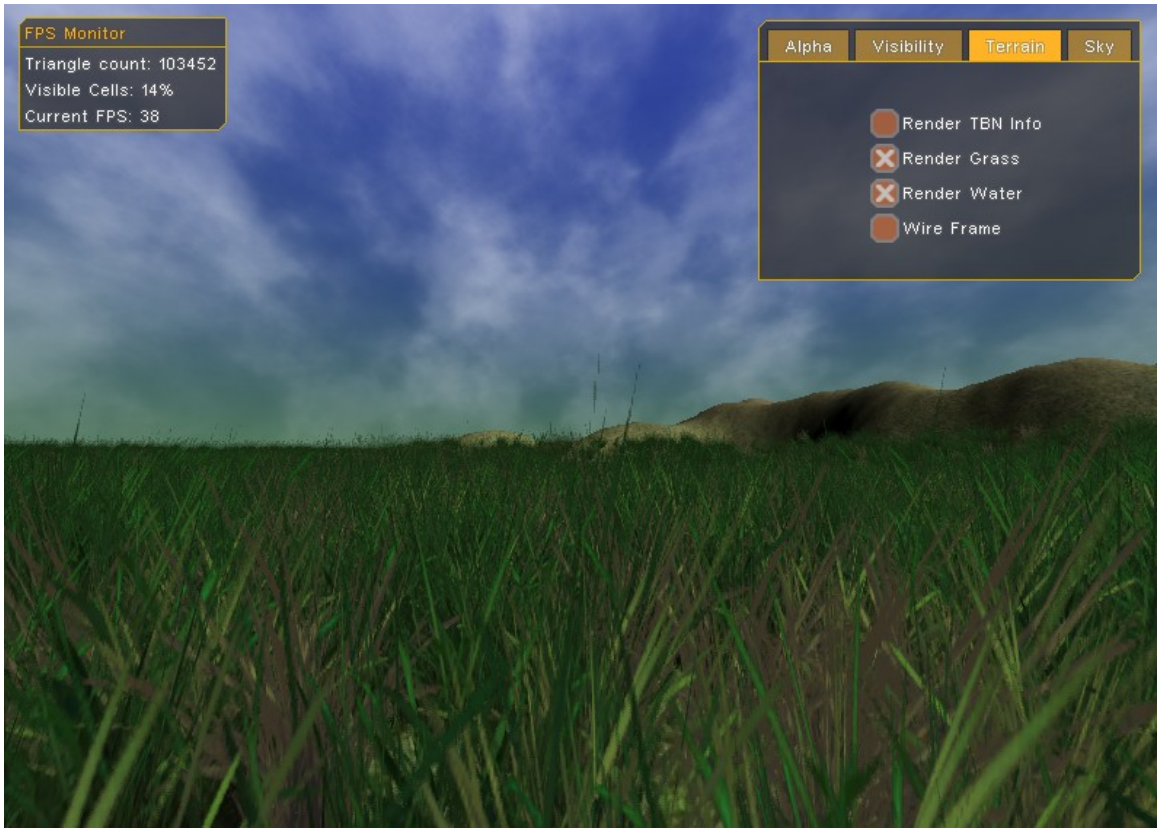
Use W, A, S, D, Q, and E to move around in the scene. Drag the mouse to change the orientation of the camera. Press H to toggle display of the GUI.

Antialiased Moving Grass

The techniques used in this code sample to render and animate the grass blades is described in *GPU Gems* and *GPU Gems II*. The basic approach is to create a flexible structure to control the orientation and animation of grass blades in a vertex shader.

```
typedef struct grassVertex
{
  Tuple3f uvJitter;
  Tuple3f binormal;
  Tuple3f tangent;
  Tuple3f normal;
  Tuple3f offset;
  Tuple3f vertex;
} GVertex;
```

The animation only affects the top vertices of a grass object, to differentiate those from the bottom vertices, we analyze the y component of the texture coordinates stored in uvJitter; if it's greater than 0, we move the vertex using a simple cosine function. The z component of uvJitter determines the stiffness and wind resistance of the grass objects.



Antialiased Moving Grass

The binormal, tangent, normal and offset will be used to position and orient the grass object according to the slope and altitude of the underlying terrain geometry.

To add more diversity to the look and feel of the grass, we pack four different grass images into a single texture and then we use the texture coordinates to determine what region should we map onto the object.

This sample also shows how to use the Transparency Antialiasing mode of the `GL_ARB_multisample` extension to provide higher quality, order independent antialiasing. This mode converts the alpha value of the pixel into a 16-level (dithered) coverage mask, which is logically ANDed with the raster-generated coverage mask.

Water with Reflection and Refraction

The water texture is first rendered into the back-buffer to properly capture the anti-aliased grass fragments before copying it using the `glCopyTexSubImage2D` function.

In a pixel shader, using a normal map we slightly offset the projected texture coordinates to create a ripple effect.

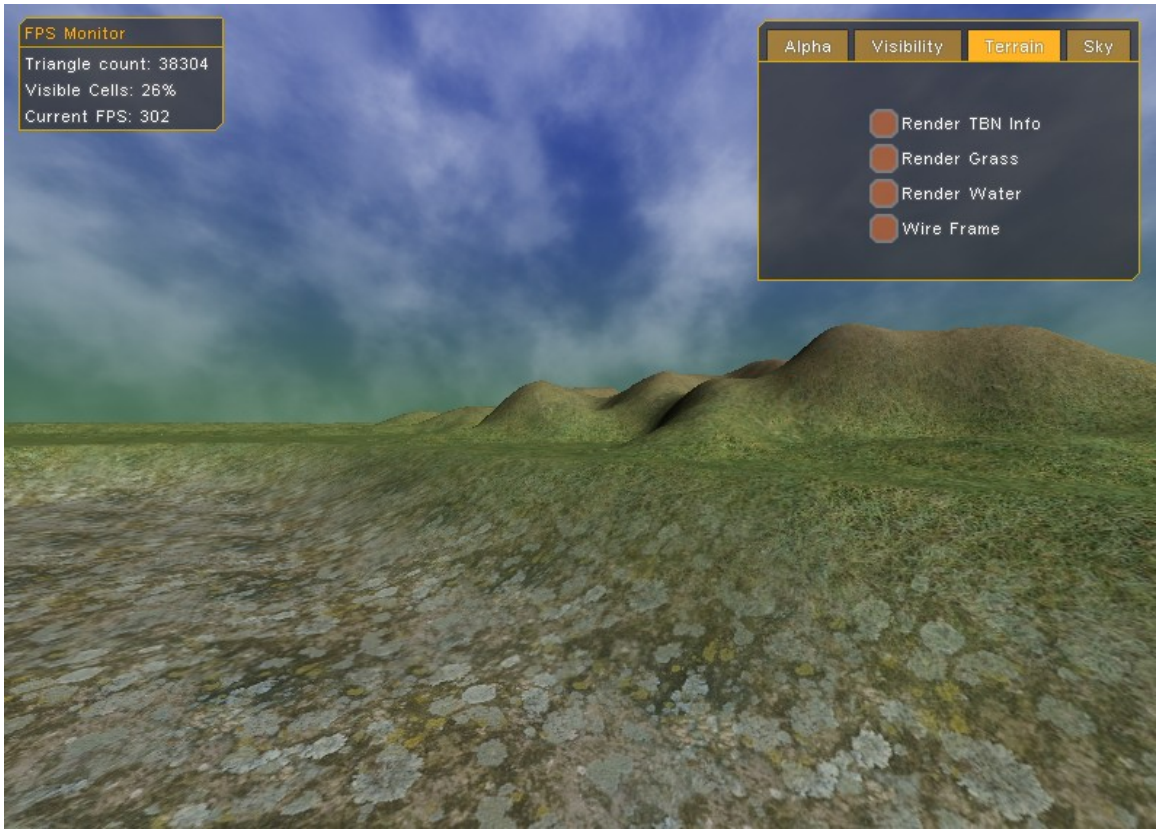
We also use a deepness map to control the refraction of the water; the deeper it gets, the less refracted color we see.

Water with Reflection and Refraction

Multi-Layer Terrain

We use several textures and a pixel shader to create a diverse, multi-layered terrain surface:

- A height map that describes the location of the lake and the altitude of the terrain
- A fungus layer that is only visible under the water
- A grass layer and a dirt layer that will be mixed together according to the terrain altitude



Multi-Layer Terrain

Dynamic Sky with Horizon Tinting

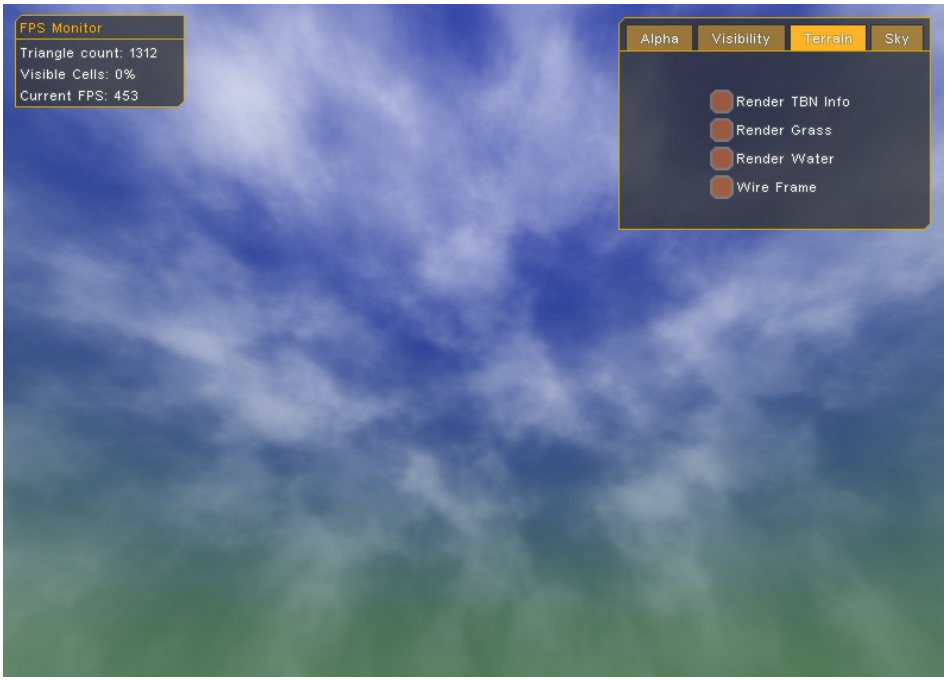
We use a dome made in a 3D modeling software and we import it to our scene. The texture coordinates of the 3D model are unspecified at compile time; instead we compute them in a vertex shader and pass them later to the pixel shader to correctly map our cloud textures.

To generate the texture coordinates, we compute the axis aligned bounding box encapsulating the dome mesh on the CPU and then pass the resulting axis extents as uniforms to the vertex shader.

These uniforms are used later on in the vertex shader to offset the incoming vertices so that they all lay in the positive region of the coordinates system before scaling them to the [0-1] range. Depending on how cloudy or clear we want the sky to be, we multiply our clamped texture coordinates by another uniform scalar.

So now we have clamped values for x, y and z to pass to the pixel shader, but before we do so, we take the y coordinate and compute its exponential value. This value will determine later on in the pixel shader the color and intensity of the horizon. Since the `exp` function maps [0-1] to [1 – 2.73], we divide our result by 2.75 to obtain yet another interval [0.36 – 0.999]. We will also need to store the square of the intensity to boost the color of fragments in the upper part of the sky dome.

We store our new texture coordinates plus some timer offset in the xy and zw components of `gl_Texcoord[0]` to simulate the scrolling of the clouds. In the pixel shader we take two samples of a tileable noise texture and add them together before multiplying the resulting color by the intensity squared that we previously computed in a vertex shader. The color of the sky is controlled by a set of uniforms accessible in the sample through the Sky tab.



Dynamic Sky with Horizon Tinting