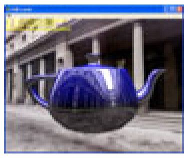
HDRFormats Sample

⊟ Collapse All

High dynamic range lighting effects require the ability to work with color values beyond the 0 to 255 range, usually by storing high range color data in textures. Floating point texture formats are the natural choice for HDR applications, but they may not be available on all target systems. This sample shows how high dynamic range data can be encoded into integer formats for compatibility across a wide range of devices.



Path

Source	*SDK root*\Samples\C++\Direct3D\HDRFormats
Executable	*SDK root*\Samples\C++\Direct3D\Bin\x86 or x64\HDRFormats.exe

Sample Overview

High dynamic range lighting techniques add realism to your application; furthermore, these techniques are fairly straightforward to implement, integrate well into an existing render pipeline, and require a modest amount of processing time; however, most HDR techniques rely in floating-point texture formats, which may not be available on the target system. One way to enable HDR techniques on devices without floating-point texture support is to encode high range floating-point data into an integer format, decoding on-the-fly as needed.

Be aware that integer textures are not perfect replacements for floating-point textures; the disadvantages of integer encoding are a loss of precision and/or a loss a range from the floating-point source, plus a performance hit resulting from the encoding/decoding process. This encoding/decoding happens seemlessly within the pixel shaders, so it's not prohibitively expensive, but it does use some of the available shader instruction count and can cause a slight frame-rate drop. For static elements, such as a skyboxes, the encoding can be done once during initialization, saving some cycles during run-time.

Implementation

The application uses two high dynamic range techniques: tone mapping and blooming. If you are not familiar with how these techniques work or want more information about high dynamic range images in general, look at HDR Lighting programming guide first, and view the HDRLighting and HDRCubeMap samples included with the SDK.

For the techniques used in this sample, the ability to store HDR data in textures is critical. The scene must be rendered to an HDR render target, the environment map texture needs to be HDR, and the temporary textures used to measure scene luminance all need to store HDR data. A device which supports floating-point textures will offer the best performance and simplest implementation, but with a little extra work integer textures can be used to store HDR data. This sample implements two encoding schemes:

RGB16

Using a 16-bit per channel integer format, 65536 discrete values are available to store per-channel data. This encoding is a simple linear distribution of those 65536 values between 0.0f and an arbitrary maximum value (100.0f for this sample). The alpha channel is unused. Here is the formula for decoding the floating-point value from an encoded RGB16 color:

```
decoded.rgb = encoded.rgb dot max_value
```

RGBE8

This encoding is more sophisticated than RGB16 and allows for a far greater range of color data by using a logarithmic distribution. Each channel stores the mantissa of the color component, and the alpha channel stores a shared exponent. The added flexibility of this encoding comes at the cost of extra computation. Here is the formula for decoding the floating-point value from an encoded RGBE8 color:

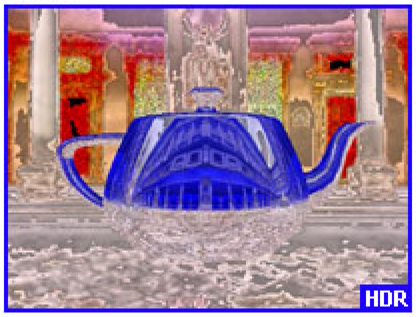
$$\text{decoded.rgb} = \text{encoded.rgb} * 2^{\text{encoded.a}}$$

The images below show the textures used to render the scene in top-down order according to when they're used. Images with a blue border contain high dynamic range data, shown here with RGBE8 encoding.

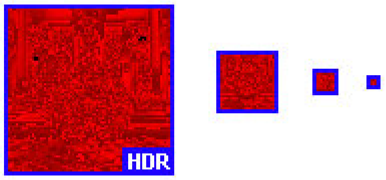
Skybox

Env Map

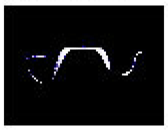
Scene



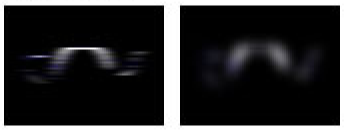
Luminance Calculation



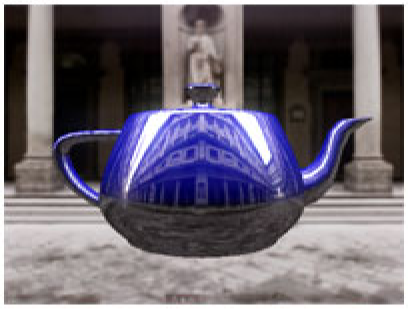
Bright Pass



Bloom Filter



Final



The process of rendering the scene is nearly identical between the native floating-point texture mode and the encoded integer texture modes. Pixel shaders which read from high dynamic range textures simply require an extra decoding step when reading from encoded integer textures and require a similar encoding step when writing HDR data to integer textures. By using uniform arguments to the shader, the same shader can be compiled differently and used by every encoding scheme. For example, here is the pixel shader that lights the model:

```
//-----------------------------------------------------------------------------
// Name: ScenePS
// Type: Pixel Shader
// Desc: Environment mapping and simplified hemispheric lighting
//-----------------------------------------------------------------------------
float4 ScenePS( SceneVS_Output Input,
                uniform bool RGBE8,
                uniform bool RGB16 ) : COLOR
{
    // Sample the environment map
    float3 vReflect = reflect( Input.ViewVec_World, Input.Normal_World );
    float4 vEnvironment = texCUBE( CubeSampler, vReflect );

    if( RGBE8 )
        vEnvironment.rgb = DecodeRGBE8( vEnvironment );
    else if( RGB16 )
        vEnvironment.rgb = DecodeRGB16( vEnvironment );

    // Simple overhead lighting
    float3 vColor = saturate( MODEL_COLOR * Input.Normal_World.y );

    // Add in reflection
    vColor = lerp( vColor, vEnvironment.rgb, MODEL_REFLECTIVITY );

    if( RGBE8 )
        return EncodeRGBE8( vColor );
    else if( RGB16 )
        return EncodeRGB16( vColor );
    else
        return float4( vColor, 1.0f );
}
```