Texture Compression in Real-Time Using the GPU

Jason Tranchida
Senior Programmer
THQ | Volition Inc.

Agenda

- Why would I want to use the GPU?
- DXT1/BC1 Primer
- How do we do it?
- Platform tricks
- Make it fast!

Prior Work

Real-Time DXT Compression

J.M.P. van Waveren

Intel Software Network, October 2006

http://www.intel.com/cd/ids/developer/asmo-na/eng/324337.htm

FastDXT

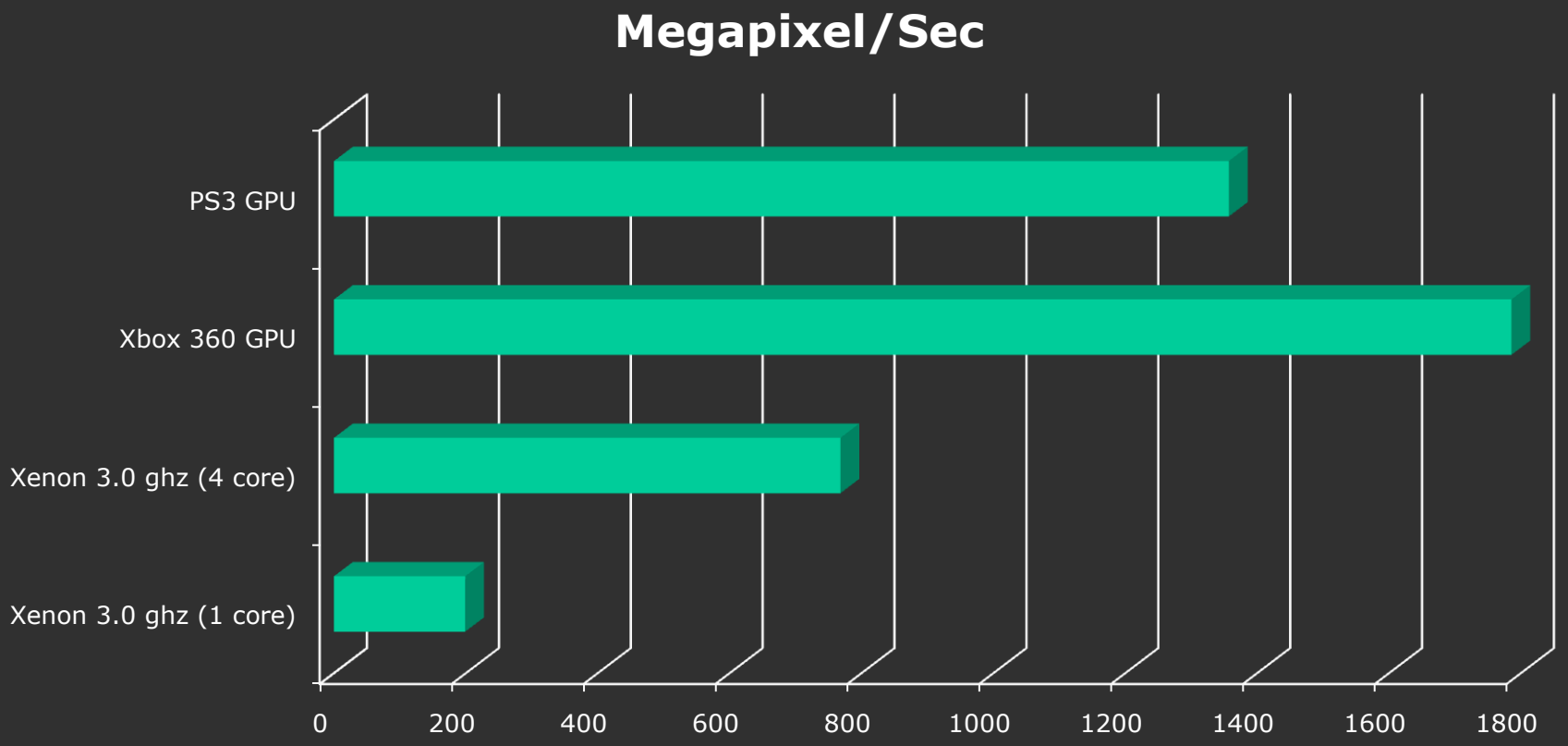
Luc Renambot

http://www.evl.uic.edu/cavern/fastdxt/

Why Use The GPU

- Games are using more run-time generated content
  - Blended Maps
  - Dynamic Cube Maps
  - User generated content
- CPU compression is slower
- CPU compression requires extra synchronization & lag

Performance

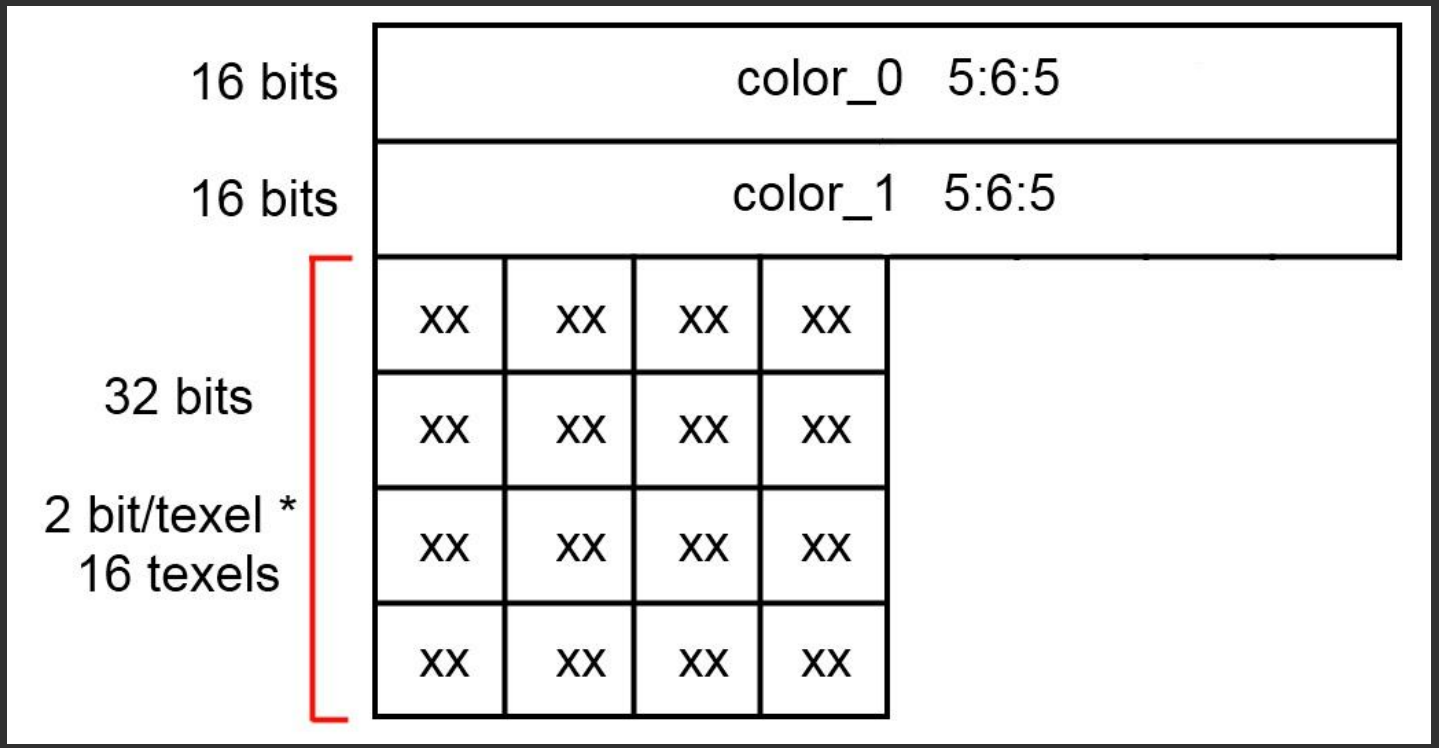


* CPU Performance Numbers from Real-Time DXT Compression Paper

volition inc THQ

DXT1/BC1 Primer

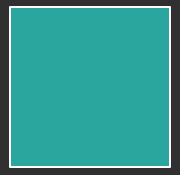
- 64bit block representing 4x4 texels
  - 4 color values, 2 stored, 2 interpolated



Color Indices

- Index 00 = color_0
- Index 01 = color_1
- Index 10 = 2/3 * color_0 + 1/3 * color_1
- Index 11 = 1/3 * color_0 + 2/3 * color_1
- Note: if color_1 C> color 0 then
  - Index 10 = ½ color_0 + ½ color_1
    Index 11 = “Transparent”

R: 179
G: 191
B: 17



R: 42
G: 166
B: 159

R: 133
G: 182
B: 64

R: 87
G: 174
B: 111

volition inc THQ

Basic DXT Compression

- Get a 4x4 grid of texels
- Find the colors that you would like to use as the stored colors
- Match each of the 4x4 texels to the best fitting color
- Create binary representation of block
- Get the results into a texture

Getting Results

- Method varies per-platform
- Render target should be ¼ dimensions of source
  - 1024x1024 source = 256x256 target
- Use a 16:16:16:16 unsigned short format

Get a 4x4 Grid of Texels

```
float2 texel_size = (1.0f / texture_size);
texcoord -= texel_size * 2;

float4 colors[16];
for (int i = 0; i < 4; i++) {
   for (int j = 0; j < 4; j++) {
        float2 uv = texcoord + float2(j, i) * texel_size;
        colors[i*4+j] = uv;
    }
}
```

Find Endpoint Colors

This can be very expensive … or very cheap!

```
float3 min_color = samples[0];
float3 max_color = samples[0];

for(int i=1; i<16; i++) {
   min_color = min(min_color, samples[i]);
   max_color = max(max_color, samples[i]);
}
```

But… there are some caveats that I'll get to later.

Building Endpoint Values

- Convert color_0 & color_1 to 5:6:5 encoded unsigned short
  - No bitwise operations available, replace with arithmetic operations
  - Dot product makes an excellent bit shift + add operation

```
int3 color_0  = min_color*255;
color_0 = color_0 / int3(8, 4, 8);
int color_0_565 = dot(color_0, float3(2048, 32, 1));

int3 color_1  = max_color*255;
color_1 = color_1 / int3(8, 4, 8);
int color_1_565 = dot(color_1, float3(2048, 32, 1));
```

Taking Care of Alpha

- Check for solid color, early out
- Check for needing to swap endpoints based on 5:6:5 value

```
float3 endpoints[2];
if(color_0_565 == color_1_565) {
   float4 dxt_block;
   dxt_block.r = color_0_565+1;
   dxt_block.g = color_0_565;
   dxt_block.b = dxt_block.a = 21845; // hard code to 01
   return dxt_block;
} else {
   bool swap = color_0_565 <= color_1_565;
   endpoints[0] = swap ? min_color : max_color;
   endpoints[1] = swap ? max_color : min_color;
}
```

Find Indices For Texels

```
float3 color_line = endpoints[1] - endpoints[0];
float color_line_len = length(color_line);
color_line = normalize(color_line);

int2 indices = 0;
for(int i=0; i<8; i++) {
   int index = 0;
   float i_val = dot(samples[i] - endpoints[0], color_line) /
                 color_line_len;
   float3 select = i_val.xxx > float3(1.0/6.0, 1.0/2.0, 5.0/6.0);
   index = dot(select, float3(2, 1, -2));
   indices.x += index * pow(2, i*2);
}
```

Repeat for the next 8 pixels

Build the block

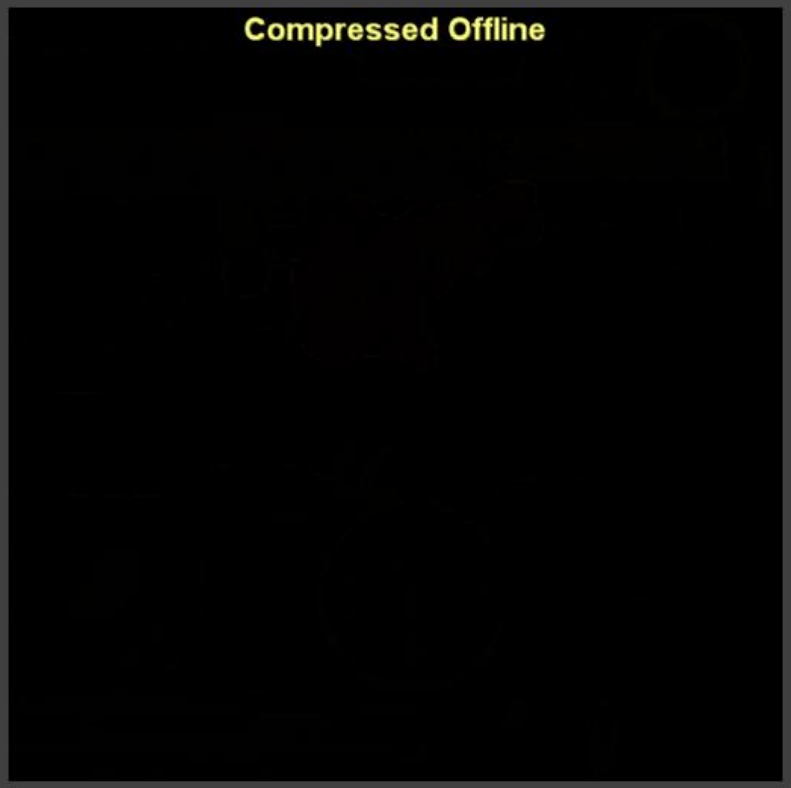
```
dxt_block.r = max(color_0_565, color_1_565);
dxt_block.g = min(color_0_565, color_1_565);
dxt_block.b = indices.x;
dxt_block.a = indices.y;

return dxt_block;
```

Diffuse Compression

volition inc THQ

Diffuse Compression Variance



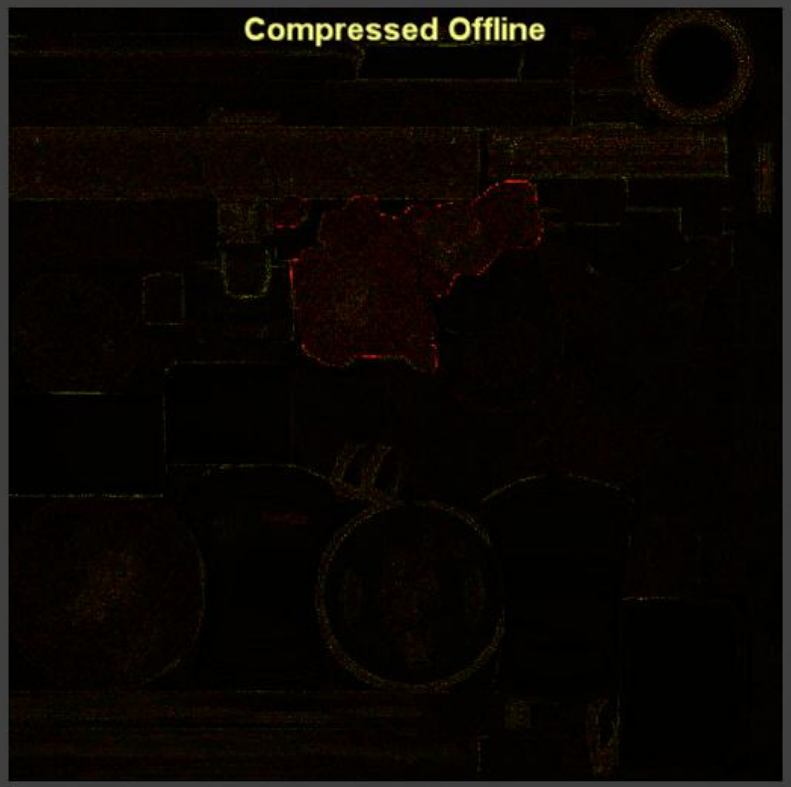
Compress DXT1 1024x1024

511.85 fps

Difference Scale: 1.0
Compress: 0.5728 ms
Reg: 22 Cycles: (73.3 - 169.3)

volition inc THQ

Diffuse Compression Variance



Compress DXT1 1024x1024

511.70 fps

Difference Scale: 10.0
Compress: 0.5727 ms
Reg: 22 Cycles: (73.3 - 169.3)

volition inc THQ

DirectX 10.1

- Easiest platform to work with
- Render to 64-bit fixed point target
  - DXGI_FORMAT_R16G16B16A16_UINT
- Use CopyResource to copy render target data to a BC1 texture.

Xbox 360 Magic

- Two methods for handling output
- Render to 16:16:16:16 render target
  - Resolve output to 16:16:16:16 buffer that shares memory with a DXT1 texture
- Use memexport
  - Doesn’t require EDRAM
  - Saves ~100 us not having to do a resolve
  - Slightly harder to use a tiled DXT1 target, must calculate tiling memory offsets

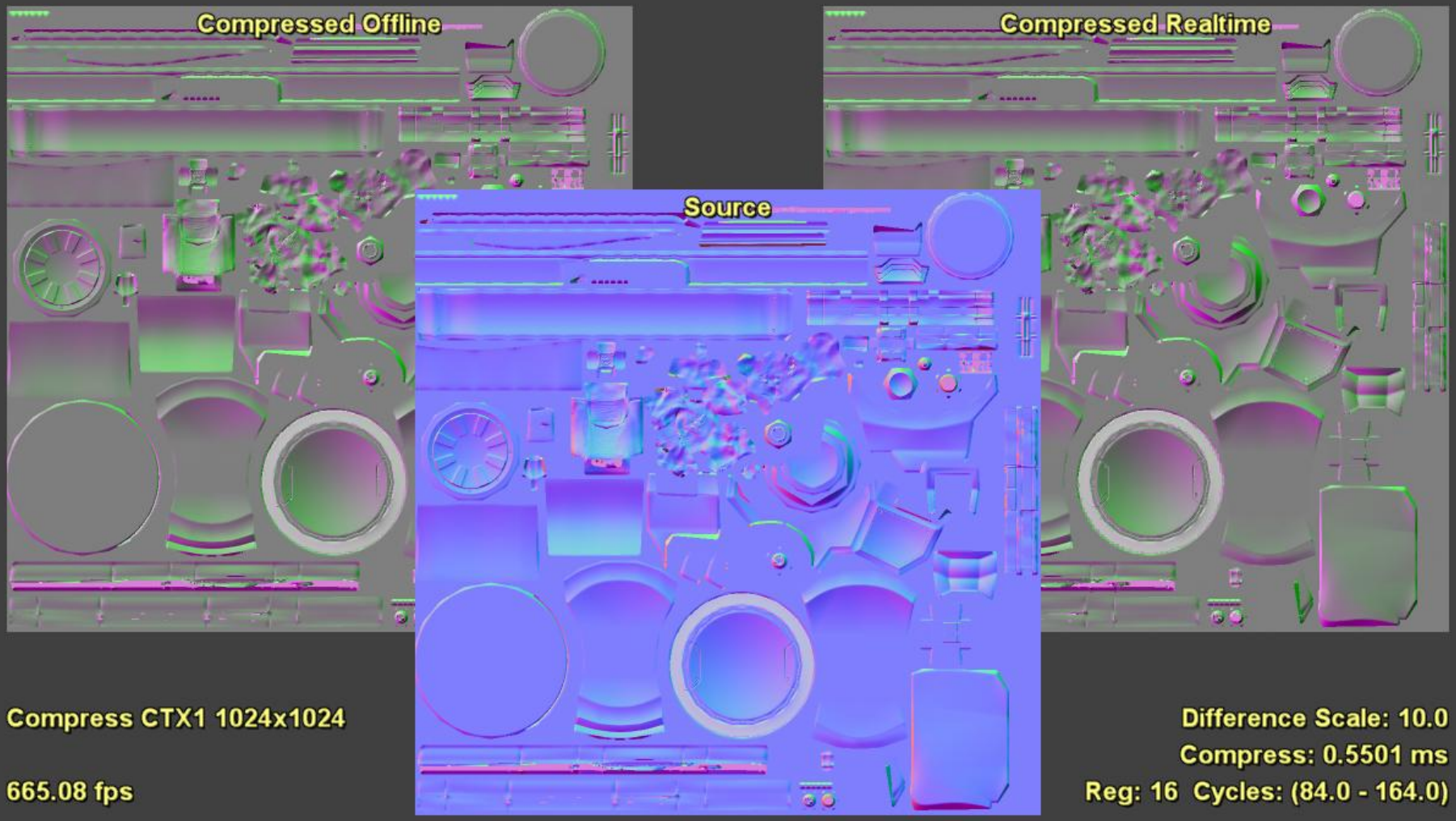
Taming the PS3

- PS3 lacks a 16:16:16:16 fixed point format
- Work around this by using a 16:16 target with double width
  - 1024 x 1024 source = 512 x 256 target
- Alternate writing out colors & indices
- 25% cost overhead for doing part of the work twice

Tweaking performance

- Shader compilers are smart, but not perfect
- Make sure you test unrolling vs. looping
- Create variant shaders for your target format
  - Normal maps can be cheaper if you’re only working with 2 components

Normal Compression



Normal Compression Variance

Compress CTX1 1024x1024

567.85 fps

Difference Scale: 1.0
Compress: 0.5497 ms
Reg: 16 Cycles: (84.0 - 164.0)

volition inc THQ

Normal Compression Variance

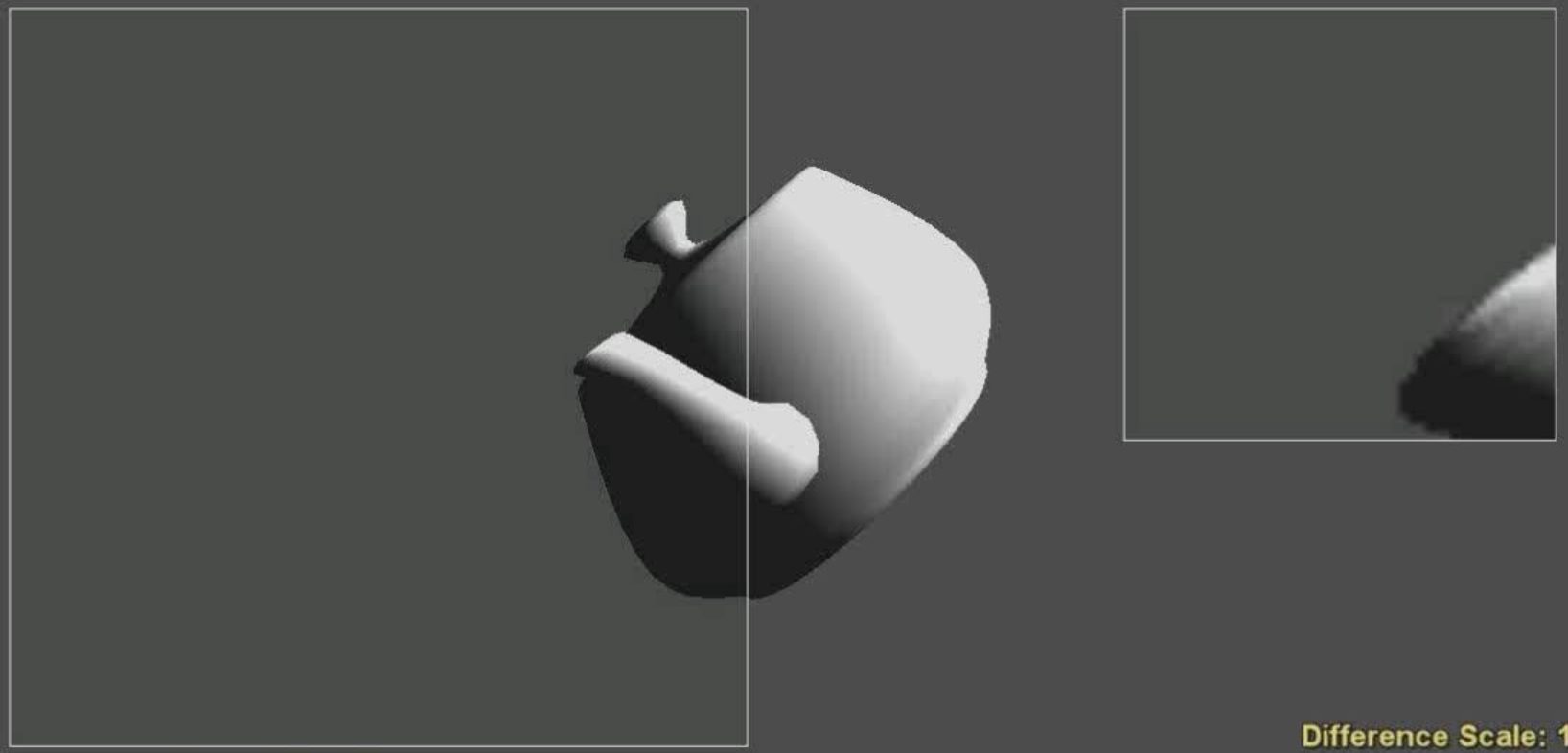
Compress CTX1 1024x1024

567.79 fps

Difference Scale: 10.0
Compress: 0.5496 ms
Reg: 16 Cycles: (84.0 - 164.0)

volition inc THQ

In Action!

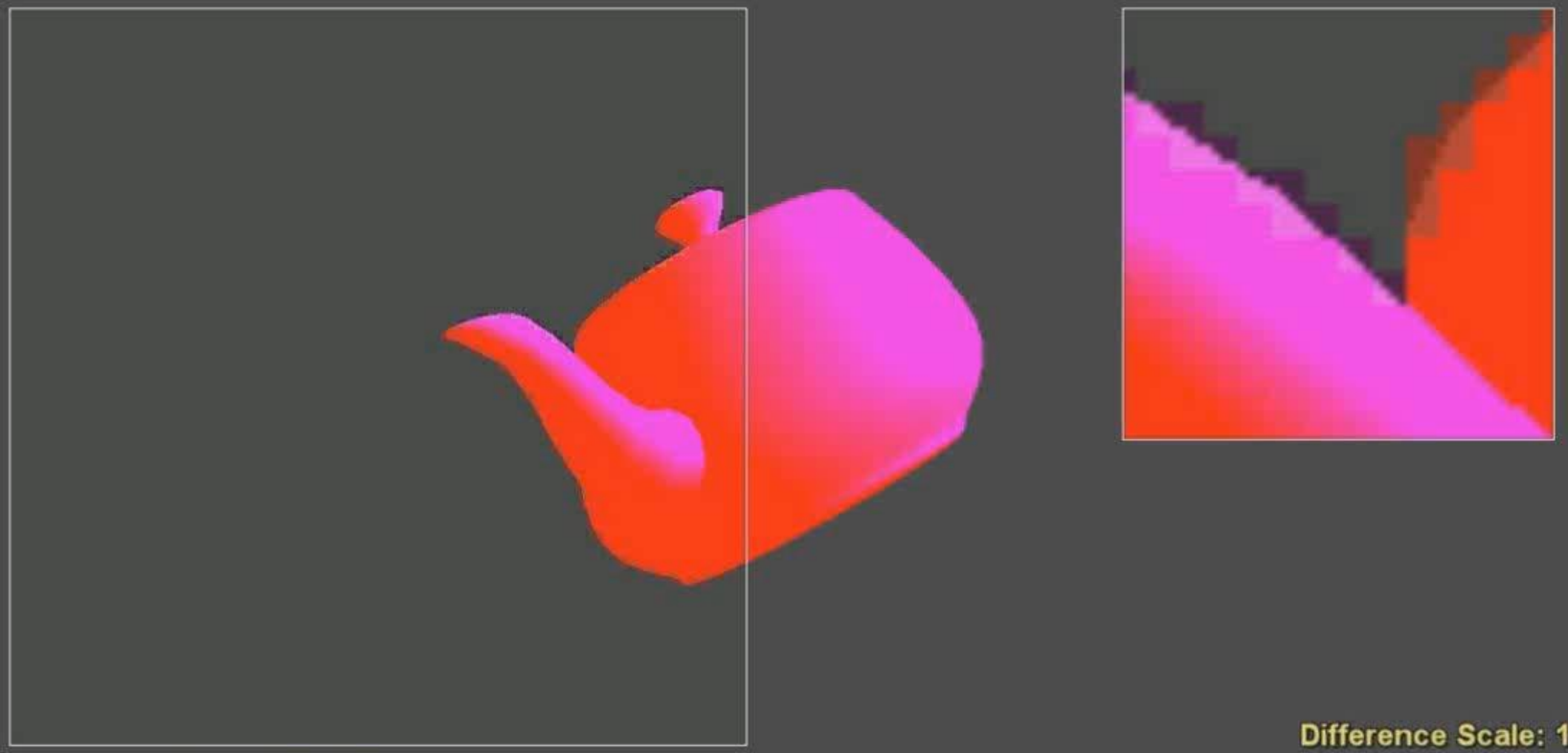


Compress Render Target 512x512
1201.52 fps

Difference Scale: 1.0
Compress: 0.1511 ms
Reg: 22 Cycles: (73.3 - 169.3)

volition inc THQ

In Action!



Compress Render Target 512x512
1203.74 fps

Difference Scale: 1.0
Compress: 0.1514 ms
Reg: 22 Cycles: (73.3 - 169.3)

volition inc THQ

Questions?

Email me at:

jtranchida@volition-inc.com

volition inc THQ