

ME290R Assignment 4 part 2

What is the fastest way to calculate reductions?

Due date: Thursday 3/5.

Your assignment is to implement a reduction that sums all the elements in a floating point texture. Don't worry about overflow for this assignment, which is looking only at speed. You should implement at least the following variations and compare their running times. Be sure to say what GPU you used and turn in a printout of your source code.

- Vary the offset distance
 - E.g. adding up texels (n,m) , $(n+1,m)$, $(n+1, m+1)$, $(n, m+1)$ vs. texels (n,m) , $(n+d/2,m)$, $(n+d/2, m+d/2)$, $(n, m+d/2)$ for a $d \times d$ texture.
- Vary the number of summations per fragment program pass (and number of passes)
 - E.g. adding together 2 texels at a time, or 4, or 16, etc. (with fewer passes if more texels get added per pass, of course).
- Vary the size of a square input texture
 - Does the running time vary linearly with the size? Does the running time vs. size behavior change depending on whether the size is a power of two?
- Vary the aspect ratio for a rectangular texture.
 - Compare summing the same number of values as you vary the aspect ratio. E.g. an $n \times n$ texture, a $2n \times .5n$ texture, a $3n \times .333n$ texture, etc.

Make sure to do the helloGPGPU and GPGPU Reduction Tutorials first. You may also find the following sample fragment program useful -- it adds up all the values in a floating point texture. Even though you don't need to do anything special in your vertex program compared to what the fixed function pipeline does, you might need to define a vertex program as well (a not-well-documented feature of at least some older cards).

```

// Fragment program for adding a texture
// It takes a floating point texture as an input.
// The texture is assumed to be square in this case with
// the size of a side equal to some power of two.
// The z attribute of the texture coordinates contains the
// offset which is added and subtracted from the x and y
// attributes to get the location of four fragments to be added.

struct input
{
    float4 position : POSITION;
    float4 color : COL0;
    float4 texCoord : TEXCOORD0;
};
void main(input IN,
          out float4 color : COLOR,
          uniform samplerRECT tex /* Floating point texture*/
)
{

    float4 texc1 = texRECT(tex,IN.texCoord.xy);
    IN.texCoord.x = IN.texCoord.x + IN.texCoord.z;
    float4 texc2 = texRECT(tex,IN.texCoord.xy);
    IN.texCoord.y = IN.texCoord.y + IN.texCoord.z;
    float4 texc3 = texRECT(tex,IN.texCoord.xy);
    IN.texCoord.x = IN.texCoord.x - IN.texCoord.z;
    float4 texc4 = texRECT(tex,IN.texCoord.xy);

    color = texc1 + texc2 + texc3 + texc4;
    color.w=1;
}

```