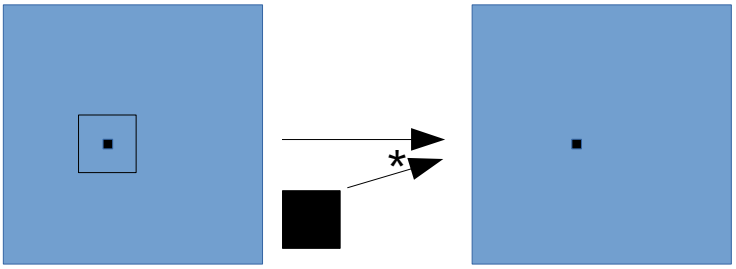
Blur Filter on a COMPUTE Shader

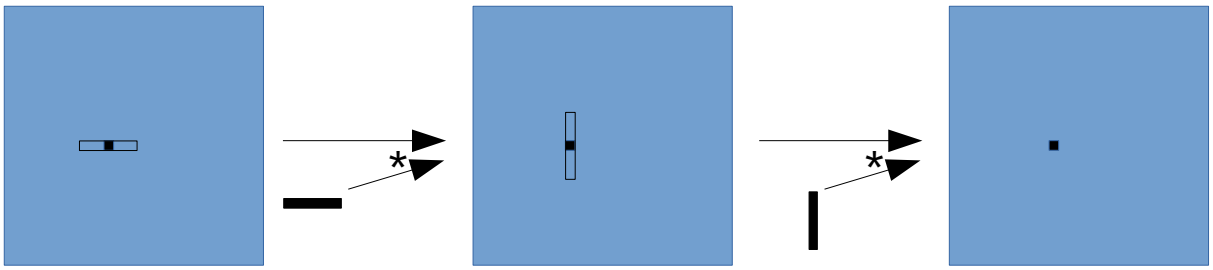
Filter

Input 2D image, rectangular $n \times n$ kernel of weights (which sum to one)
Output 2D image, where each output pixel is a weighted sum of
corresponding rectangle of input pixels
times rectangle of weights.
This is $O(n^2)$



Separable filter

Write kernel as product of row and column weights
Perform filter in two steps, one horizontal, one vertical
This is $O(2n)$



Building the kernel weights

Want bell-shaped curve across $2w+1$ of pixels:
(w is the half width of an odd number of pixels)
w=0: 1 pixel at offset 0
w=1: 3 pixels at offsets -1,0,1
w=2: 5 pixels -2,-1,0,1,2
in general: $2w+1$ pixels at offsets -w,...-2,-1,0,1,2,...w
Compute using the Gaussian exponential curve
problem: this is never zero
solution: choose width so it is mostly zero on outer pixels
Let s be bell width:
$s=w/2$ (recommended for realtime graphics),
$s=w/3$ (the usual recommendation)
visually, s is curve half-width at its half-height
Compute $2w+1$ weights
$e^{-\frac{1}{2}\left(\frac{i}{s}\right)^2}$ for i in range $-w \ldots w$
then normalize them to sum to one.

Efficient filtering in a compute shader

See page 16 of presentation:
http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Efficient%20Compute%20Shader%20Programming.pps

Idea

128 threads will: (virtually arranged into a 128x1 row)

read 128+2w pixels into shared memory

one pixel each thread, plus one extra for first 2w threads

compute 128 output pixels

one each by weighted sum of 2w+1 weights and pixels

write 128 output pixels

one each

Dispatch

Tile image with 128×1 sized blocks

That is: dispatch $width/128 \times height$ thread groups

Shader

Uniform inputs:

src, dst images

w: half-size of kernel

weights: array of $2w+1$ floats

Declare thread group to be 128×1 threads

Declare thread-group-shared-memory **v[128+2*w+1]** floats

actually must be constant size: **v[128+<largest filter size>]**

Compute global-position **gpos**

Compute local-index **i** in thread group

Read and store this threads **first** pixel from **src** image:

v[i]=imageLoad(src, gpos+(-w,0))

first 2*w threads load extra pixel out beyond 128

v[i+128]=imageLoad(src, gpos+(128-w,0))

Force memory synchronization.

Compute sum of **weights[0 ... 2w]** times corresponding pixels **v[i ... i+2w]**

Store sum at **gpos** in **dst** image

imageStore(dst, gpos, sum)

Sample CPU code:

Create compute shader

Same as other shaders,
but use **GL_COMPUTE_SHADER** in **glCreateShader** call
cannot coexist with other shaders in a shader program

CPU invokes computer shader enough times to tile an image:

```
glUseProgram(programID)
// Set all uniform and image variables
glDispatchCompute(W/128, H, 1) // Tiles WxH image with groups sized 128x1
glUseProgram(0)
```

Send block of weights to shader (as a **uniform block**)

```
glGenBuffers(1, &blockID) // Generates block
bindpoint = 0; // Start at zero, increment for other blocks

loc = glGetUniformBlockIndex(programID, “blurKernel”)
glUniformBlockBinding(programID, loc, bindpoint)

glBindBufferBase(GL_UNIFORM_BUFFER, bindpoint, blockID)
glBufferData(GL_UNIFORM_BUFFER, #bytes, data, GL_STATIC_DRAW)
```

Send a texture to the shader as an **image2D**

```
imageUnit = 0;  // Increment for other images

loc = glGetUniformLocation(programID, “src”)
glBindImageTexture(imageUnit, textureID, 0, GL_FALSE, 0, GL_READ_ONLY, GL_R32F)

glUniform1i(loc, imageunit)
// Change GL_READ_ONLY to GL_WRITE_ONLY for output image
// Change GL_R32F to  GL_RGBA32F for 4 channel images
```

Sample Compute shader code

```
#version 430  // Version of OpenGL with COMPUTE shader support
layout (local_size_x = 128, local_size_y = 1, local_size_z = 1) in; // Declares thread group size

uniform blurKernel {float weights[101]; }; // Declares a uniform block

layout (r32f) uniform readonly  image2D src; // src image as single channel 32bit float readonly
layout (r32f) uniform writeonly image2D dst; // dst image as single channel 32bit float writeonly

shared float v[128+101]; // Variable shared with other threads in the 128x1 thread group

void main() {
   ...
   ivec2 gpos = ivec2(gl_GlobalInvocationID.xy); // Combo of groupID, groupSize and localID

   uint i = gl_LocalInvocationID.x; // Local thread id in the 128x1 thread groups128x1

   v[i] = imageLoad(src, gpos+...);  // read an image pixel at an ivec2(.,.) position
   if (i<2*w) v[i+128] = imageLoad(src, gpos+...);  // read extra 2*w pixels

   barrier(); // Wait for all threads to catchup before reading v[]

   ...

   imageStore(dst, gpos, ...); // Write to destination image
}
```