

Large Histograms & Non-separable Filters

General Purpose GPU – GLSL/CUDA/OpenCL

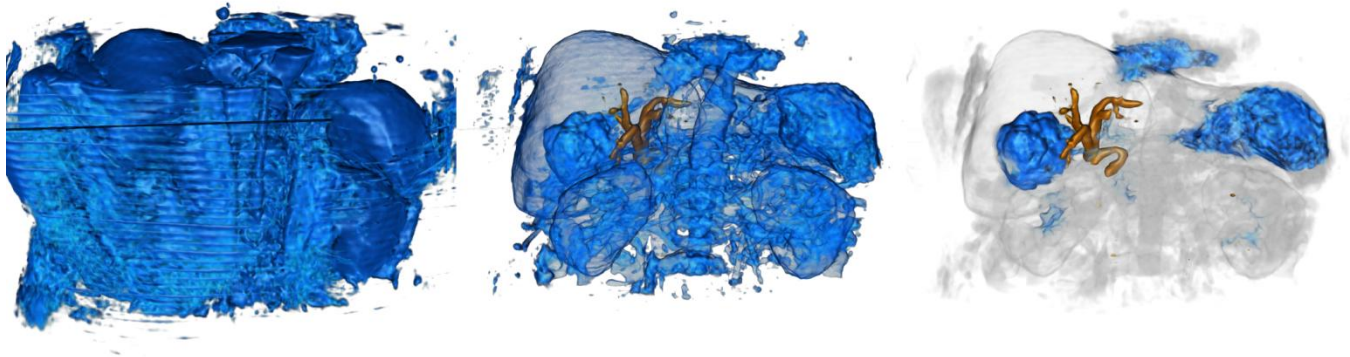
Stefan Lindholm

ITN 2010-06-03

Questions:

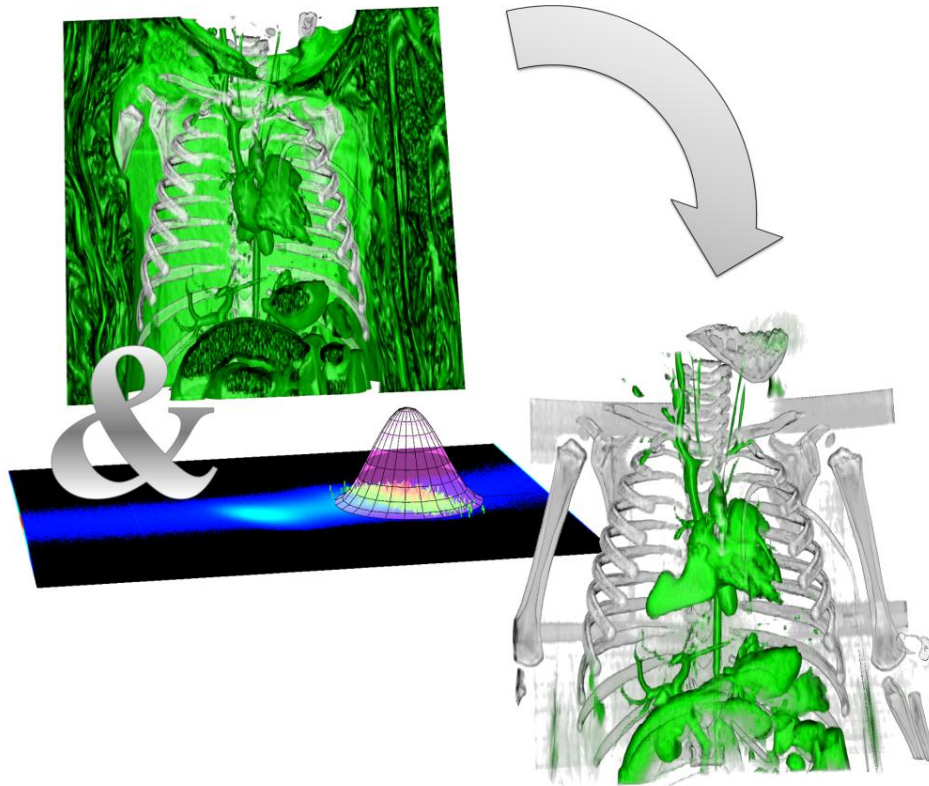
- 1) Why is CUDA/OpenCL easier to optimize for small histograms?
- 2) What allows the actual system speedup to surpass the benchmarked speedup for the Large Histogram implementation?
- 3) Name one reason CUDA might be a better option than OpenCL for Non-separable Filters?

Dual-Energy
Visualization ITN Spatial Conditioning
DVR Siemens Corporate Research
MPR Research Ultra Sound Radiology Computed Tomography
Medical CMIV



DVR

- Direct Volume Rendering (DVR) for visualization of medical data
- Transfer Function classification of tissues
- Current research
 - Dual-energy Computed Tomography (DECT)
 - Spatial Conditioning (material presence functions)
 - Boundary preserving smoothing filters (anisotropic diffusion)

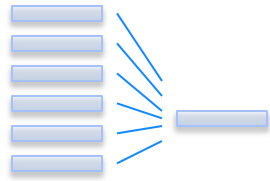


- Large Histograms
- Non-separable Filters

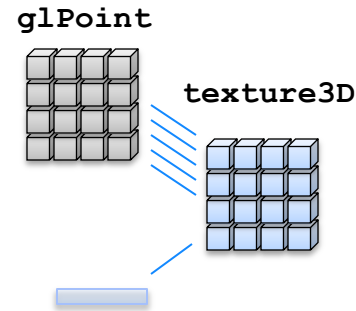


CRESEARCH

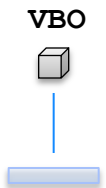
Large Histograms



- 64bin or 256bin histograms: Calculate and sum multiple small histograms, multiple examples by Nvidia
- Arbitrary size/MD histograms: Local memory overflow (OpenCL/CUDA), incrementations of global memory requires atomic operations



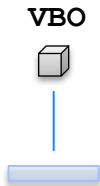
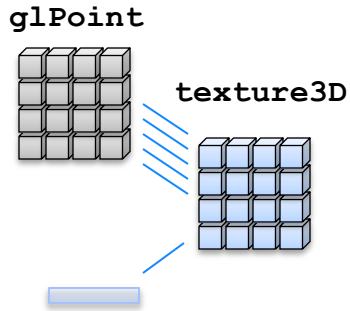
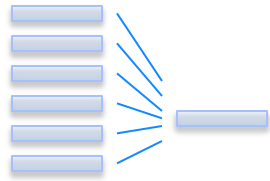
- Fragments increase buffers and are atomic (GLSL)
 - "Render" one glPoint per voxel and use vertex shader for bin transformations
 - Requires ~ 134'000'000 GL calls and 134'000'000 texture lookups
- Reduce GL calls
 - Put 134'000'000 glPoints in a Display List or VBO
 - Reduces GL calls but costs 134 MB of VRAM for non-information (0.0.0, 0.0.1, 0.0.2 ...)
 - Geometry shaders? Instancing?



- Reduce texture lookups
 - Interpret the volume as a VBO and get the voxel values directly in gl_Vertex
 - Reduces both GL calls and texture lookups
 - Total call reduction by more than 268'000'000 : 1
 - ~20X benchmarked speedup versus CPU
 - >8000X actual system speedup by in-thread processing

GLSL

Large Histograms

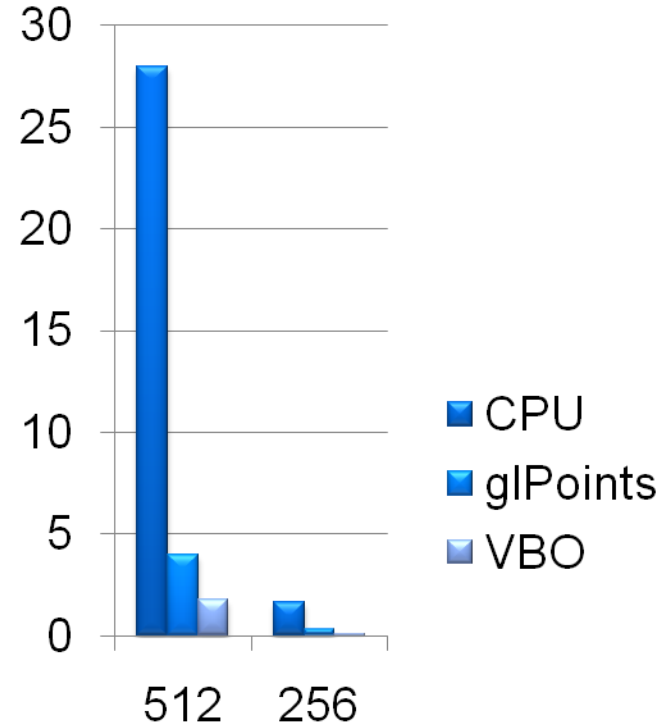


```
// C++
glBegin(GL_POINTS);
for (int i = 0; i < int(w); i++)
for (int j = 0; j < int(h); j++)
for (int k = 0; k < int(d); k++)
{
    glVertex3f( float(i) / w + off[0],
               float(j) / h + off[0],
               float(k) / d + off[0] );
}
glEnd();

// GLSL vertex shader
void main(void)
{
    float bitsStored = (bitsUsed > 8.0) ? 16.0 : 8.0;
    vec3 pos = gl_Vertex.xyz;
    vec3 val;
    val.x = texture2D( volume0, pos.xy ).x * (bitsStored/bitsUsed);
    gl_Position = vec4( 2.0*val.x-1.0, 0.5, 0.0, 1.0 );
}

// C++
glBindBuffer(GL_ARRAY_BUFFER_ARB, pboId);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 1, im_type, GL_TRUE, im_bitsStored/8, 0);
glDrawArrays(GL_POINTS, 0, im_w * im_h * im_d);

// GLSL vertex shader
void main(void)
{
    float val = gl_Vertex.x;
    val = val * bitsStored/bitsUsed;
    val = 2.0 * val - 1.0;
    val = val + 1.0 / pow(2.0, bitsUsed);
    gl_Position = vec4( val, 0.5, 0.0, 1.0 );
}
```

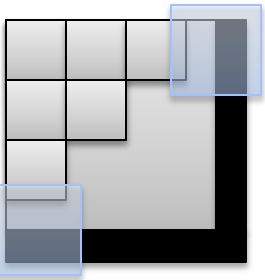
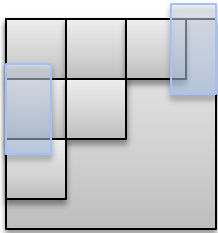
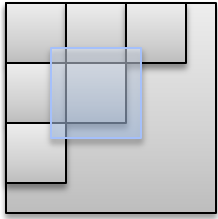
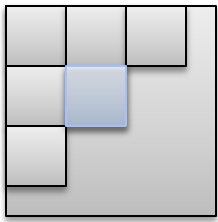


- Verdict: OpenCL/CUDA implementations not trivial for arbitrary size histograms - GLSL works fine for histograms up to 2D



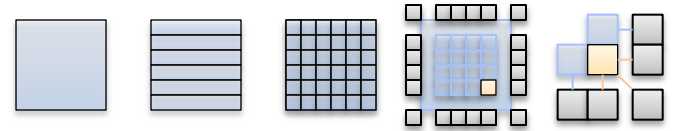
GLSL

Non-separable Filters



- **Separable:** Treat data as 1D and filter, keep track of boundary points, multiple examples by Nvidia
- **Non-separable:** Copy MD patch ($N_x * N_y * N_z$) to shared mem - problematic since patches need local padding that easily breaks optimal memory layouts (x16)

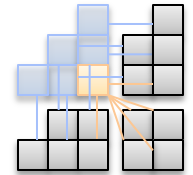
- `async_work_group_copy`
 - $N_y * N_z$ calls (possible to minimize)
 - Slow performance even for "ideal" cases



- **Manual copy**
 - 1 copy per thread, works fairly well

- **Filter size**
 - Due to limitations (border conditions, padding) implementation becomes booring

- **Local padding**
 - Easily introduces costly offsets to optimal mem layout (x16)
 - 26 additional border conditions (6 sides 12 edges 8 corners)
 - Up to 8 copys (1 self 3 sides 3 edges 1 corner) for a single thread



- **Global padding**
 - Non pow2 images need padding in at least two dims (mem layout)
 - `row_pitch / slice_pitch` requires 2D/3D image types in OpenCL (works with buffers in CUDA)
 - Image types (no buffer indexing, vector returns, etc.) or convert/copy
 - Pad on the cpu (can be costly)



Non-separable Filters

```

event_t e_copy;
for (int i = 0; i < patch_height; i++)
{
    e_copy = async_work_group_copy(t_data+i*patch_width, i_ptr+offset+i*W, patch_width, (i==0)?(event_t)0:e_copy);
}
wait_group_events(1, &e_copy);

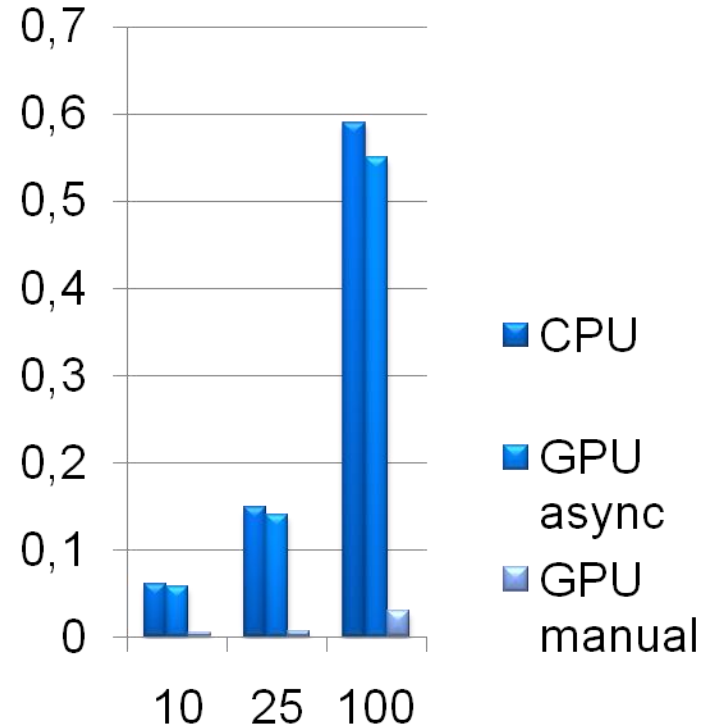
```

```

////////////////////////////////////

// Center point
t_data[tdx] = i_ptr[gdx];
// Sides
if (u == 0 && !isBorderX) t_data[xxxT(0,1,1)] = i_ptr[xxxG(0,1,1)];
if (u == s && !isBorderX) t_data[xxxT(2,1,1)] = i_ptr[xxxG(2,1,1)];
if (v == 0 && !isBorderY) t_data[xxxT(1,0,1)] = i_ptr[xxxG(1,0,1)];
if (v == t && !isBorderY) t_data[xxxT(1,2,1)] = i_ptr[xxxG(1,2,1)];
if (w == 0 && !isBorderZ) t_data[xxxT(1,1,0)] = i_ptr[xxxG(1,1,0)];
if (w == r && !isBorderZ) t_data[xxxT(1,1,2)] = i_ptr[xxxG(1,1,2)];
// Borders
if (u == 0 && v == 0 && !isBorderX && !isBorderY) t_data[xxxT(0,0,1)] = i_ptr[xxxG(0,0,1)];
if (u == s && v == 0 && !isBorderX && !isBorderY) t_data[xxxT(2,0,1)] = i_ptr[xxxG(2,0,1)];
if (u == 0 && v == t && !isBorderX && !isBorderY) t_data[xxxT(0,2,1)] = i_ptr[xxxG(0,2,1)];
if (u == s && v == t && !isBorderX && !isBorderY) t_data[xxxT(2,2,1)] = i_ptr[xxxG(2,2,1)];
if (u == 0 && w == 0 && !isBorderX && !isBorderZ) t_data[xxxT(0,1,0)] = i_ptr[xxxG(0,1,0)];
if (u == s && w == 0 && !isBorderX && !isBorderZ) t_data[xxxT(2,1,0)] = i_ptr[xxxG(2,1,0)];
if (u == 0 && w == r && !isBorderX && !isBorderZ) t_data[xxxT(0,1,2)] = i_ptr[xxxG(0,1,2)];
if (u == s && w == r && !isBorderX && !isBorderZ) t_data[xxxT(2,1,2)] = i_ptr[xxxG(2,1,2)];
if (v == 0 && w == 0 && !isBorderY && !isBorderZ) t_data[xxxT(1,0,0)] = i_ptr[xxxG(1,0,0)];
if (v == t && w == 0 && !isBorderY && !isBorderZ) t_data[xxxT(1,2,0)] = i_ptr[xxxG(1,2,0)];
if (v == 0 && w == r && !isBorderY && !isBorderZ) t_data[xxxT(1,0,2)] = i_ptr[xxxG(1,0,2)];
if (v == t && w == r && !isBorderY && !isBorderZ) t_data[xxxT(1,2,2)] = i_ptr[xxxG(1,2,2)];
// Corners
if (u == 0 && v == 0 && w == 0 && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(0,0,0)] = i_ptr[xxxG(0,0,0)];
if (u == s && v == 0 && w == 0 && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(2,0,0)] = i_ptr[xxxG(2,0,0)];
if (u == 0 && v == t && w == 0 && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(0,2,0)] = i_ptr[xxxG(0,2,0)];
if (u == s && v == t && w == 0 && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(2,2,0)] = i_ptr[xxxG(2,2,0)];
if (u == 0 && v == 0 && w == r && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(0,0,2)] = i_ptr[xxxG(0,0,2)];
if (u == s && v == 0 && w == r && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(2,0,2)] = i_ptr[xxxG(2,0,2)];
if (u == 0 && v == t && w == r && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(0,2,2)] = i_ptr[xxxG(0,2,2)];
if (u == s && v == t && w == r && !isBorderX && !isBorderY && !isBorderZ) t_data[xxxT(2,2,2)] = i_ptr[xxxG(2,2,2)];

```



- Verdict: OpenCL is fast but easily becomes hardcoded or non-intuitive



OpenCL



- Future
 - More pipeline steps in CUDA/OpenCL (converted from GLSL)
 - Reduction processes / Filter kernels
 - Static / dynamic memory
 - GLSL vs. CUDA/OpenCL
- GPGPU in medical visualization
 - Medical workstations tend to have fancy GPUs
 - Size of datasets is an issue
 - OpenCL / CUDA are very very optimized

- Large Histograms
- Non-separable Filters



CRESEARCH



Linköping University

expanding reality

www.liu.se