

Higher level functionality

Jesper Mosegaard

Based primarily on SIGGRAPH 2004 GPGPU COURSE and Visualization 2004 Course



Lecture schedule

- Today
 - Higher level GPGPU functionality (JM)
- 18/11
 - Spring-mass + ? (JM)
- 25/11
 - Fluid simulation + linear algebra (TSS)



Higher level functionality

- Data structures
 - pointers
 - 1d and 3d arrays
- Scatter and Reduction
- Sorting and searching
- GPGPU Languages

- Branching
- Multi surface PBuffers

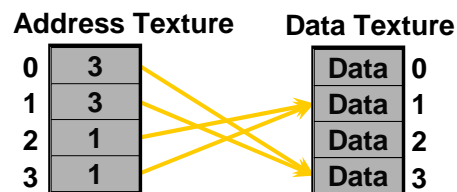


Data structures

Pointers

- Store addresses in texture
- **Dependent** texture read

```
float2 addr = tex2D( addrTex, texCoord );  
float2 data = tex2D( dataTex, addr );
```



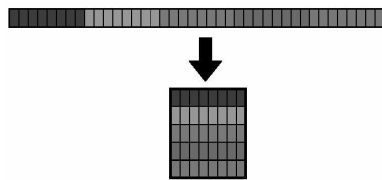
Usage of pointers

- If your problem does not map to
 - Grid based approach with regular offsets to "neighbours"
- Use a secondary texture to encode addresses to "neighbours"

GPU Arrays

- Large 1D arrays

- 1D textures limited to size 2048 (or 4096)
- Pack 1D array into 2d texture



3D Arrays

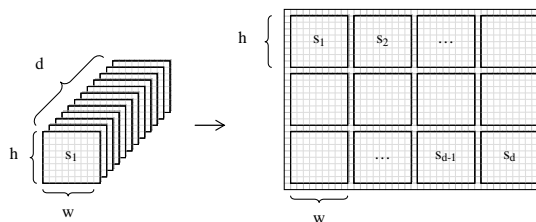
- There are 3d textures, but...

- Problem

- No 3d frame buffer
- No RTT to a 3d texture with Pbuffers

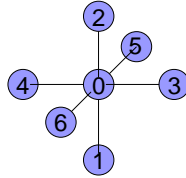
- Solution

- Map 3d arrays into 2d textures (*flat 3d-texture*)

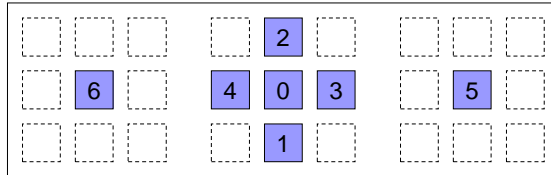


Addressing in 3D grid

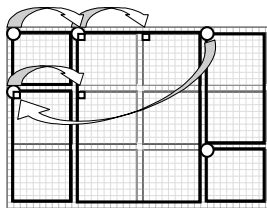
3d grid



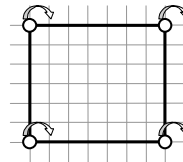
Layout in texture:



Addressing in 3D grid



a)



b)



GPU Arrays

- Higher Dimensional Arrays
 - Pack into 2D buffers
 - N-D to 2D address translation
 - Same problems as 3D arrays if data does not fit in a single 2D texture

- Conclusions
 - Fundamental GPU memory primitive is a fixed-size 2D array (not 1d)
 - GPGPU needs more general memory model



Scatter and Reduction

Computational primitives

- Kernel operations
 - Read only memory *Texture sample*
 - Random access read-only *Texture sample*
 - Per-data-element interpolants *Varying parameters*
 - Temporary storage (no saved state) *Local register*
 - Read-only constants *Uniform parameters*
 - Write-only memory (!=read only memory) *Render to Texture*
 - Floating point ALUops

Computation Primitives, what is missing ?

- No stack
- No heap
- No integer or bitwise operations
- No native scatter ($a[i]=b$)
- No native reduction operations (max, min, sum)
- Data-dependent conditionals
- Limited number of outputs

- WHY ?
 - No demand for it in games
 - It is a GRAPHICS processing unit

Emulating scatter

```
i = foo();  
a[i] = bar();
```

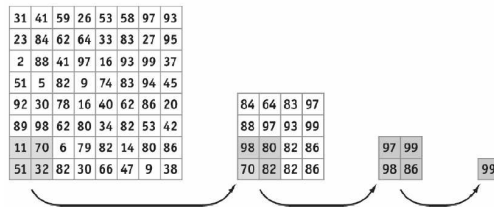
- Solution 1
 - Reformulate algorithm to gather instead of scatter
- Solution 2
 - Render point-sized points and move them according to texture information
 - With vertex texture fetches (SM 3.0)
 - Render-to-vertex array
 - Rendering points for each data-element is slow
- Solution 3
 - Sorting

Reduction

- Reduction
 - Operation that requires all data elements
 - Min, max, sum etc.
- Example:
 - Tone mapping, dot product.

Emulating Reduction

- Solution
 - Perform iterative gather operations
 - Reading in 2x2 blocks (or bigger) to a texture of half the size.
- Problem
 - $\log(n)$ passes, for a $n \times n$ texture



GPGPU Languages

- Why do we want them?
 - Make programming GPUs easier!
 - Don't need to know OpenGL, DirectX, or ATI/NV extensions
 - Simplify common operations
 - Focus on the algorithm, not on the implementation
- Sh
 - University of Waterloo
 - <http://libsh.sourceforge.net>
 - <http://www.cgl.uwaterloo.ca>
- Brook
 - Stanford University
 - <http://brook.sourceforge.net>
 - <http://graphics.stanford.edu>

Sh Features

- Implemented as C++ library
 - Use C++ modularity, type, and scope constructs
 - Use C++ to metaprogram shaders and kernels
 - Use C++ to sequence stream operations
- Operations can run on
 - GPU in JIT compiled mode
 - CPU in immediate mode
 - CPU in JIT compiled mode
- Can be used
 - To define shaders
 - To define stream kernels
- No glue code
 - To set up a parameter, just declare it and use it
 - To set up a texture, just declare it and use it
- Memory management
 - Automatically uses puffers and/or ubuffers
 - Textures are shadowed and act like arrays on both the CPU and GPU
 - Textures can encapsulate interpretation code
 - Programs can encapsulate texture data
- Program manipulation
 - Introspection
 - Uniform/varying conversion
 - Program specialization
 - Program composition
 - Program concatenation
 - Interface adaptation
- Free and Open Source
 - <http://libsh.sourceforge.net>

Sh Fragment Shader

```
fsh = SH_BEGIN_PROGRAM("gpu:fragment") {
    ShInputNormal3f  nv;           // normal (VCS)
    ShInputVector3f lv;           // light-vector (VCS)
    ShInputVector3f vv;           // view vector (VCS)
    ShInputColor3f  ec;           // irradiance
    ShInputTexCoord2f u;         // texture coordinate

    ShOutputColor3f fc;           // fragment color

    vv = normalize(vv);
    lv = normalize(lv);
    nv = normalize(nv);
    ShVector3f hv = normalize(lv + vv);
    fc = kd(u) * ec;
    fc += ks(u) * pow(pos(hv|nv), spec_exp);
} SH_END;
```

Streams and Channels

- `ShChannel<element_type>`
 - Sequence of elements of given type
- `ShStream`
 - Sequence of channels
 - Combine channels with `&`:
`ShStream s = a & b & c;`
 - *Refers* to channels, does *not* copy
 - Single channel also a stream
- Apply programs to streams with `<<`:
`ShStream t = (x & y & z);`
`s = p << t;`
`(a & b & c) = p << (x & y & z);`

Brook: general purpose streaming language

- stream programming model
 - enforce data parallel computing
 - streams
 - encourage arithmetic intensity
 - kernels
- C with stream extensions
- GPU = streaming coprocessor

Brook language streams

■ streams

- collection of records requiring similar computation

- particle positions, voxels, FEM cell, ...

```
float3 positions<200>;
float3 velocityfield<100,100,100>;
```

- similar to arrays, but...

- index operations disallowed: `position[i]`
- read/write stream operators

```
streamRead (positions, p_ptr);
streamWrite (velocityfield, v_ptr);
```

- encourage data parallelism

Brook language kernels

■ kernels

- functions applied to streams

- similar to `for_all` construct

```
kernel void foo (float a<>, float b<>,
                 out float result<>) {
    result = a + b;
}
```

```
float a<100>;
float b<100>;
float c<100>;
```

```
for (i=0; i<100; i++)
    c[i] = a[i]+b[i];
```

```
foo(a,b,c);
```



Sorting on the GPU



Sorting

- Given an unordered list of elements, produce
 - list ordered by key value
 - Fundamental kernel: compare and swap
- GPUs constrained programming environment limits viable algorithms
 - The sequence of comparisons is not data-dependent
 - Bitonic merge sort [Batcher 68] (sorting networks)

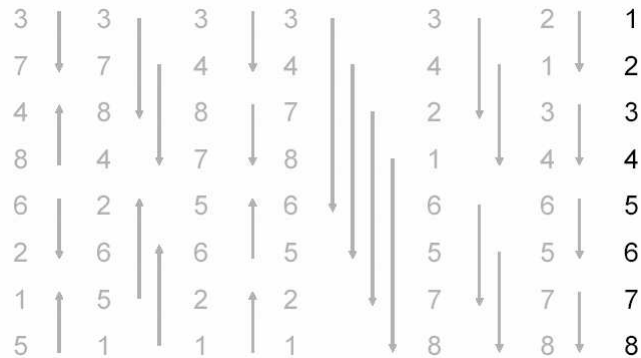
Bitonic sequence

$$\blacksquare a_1 \leq a_2 \leq \dots \leq a_k \geq a_{k+1} \geq \dots \geq a_n$$

Bitonic sort

- Sort a bitonic set
 - Create (split into) two **bitonic** sequences (c_i) and (d_i)
 - $c_i = \min(a_i, a_{i+k})$
 - $d_i = \max(a_i, a_{i+k})$
 - $c_i \leq d_i$
 - Proceed by recursively splitting c_i and d_i
 - The result is a sorted set
- Bitonic sort on arbitrary input
 - Create bitonic sets of size four (pairwise comparison)
 - Sort the bitonic sets alternating between increasing and decreasing the order to create a new bitonic set of twice the size.

Bitonic sorting example



Binary search

- Again, GPUs constrained programming environment limits viable algorithms
- Each fragment searches in the sorted set.

Branching Techniques

- Fragment program branches can be expensive
 - No true fragment branching on GeForce FX or Radeon
 - SIMD branching on GeForce 6 Series
 - Incoherent branching hurts performance
 - Excluding fragment computation
- Sometimes better to move decisions up the pipeline
 - Replace with math
 - Occlusion Query
 - Static Branch Resolution
 - Z-cull
 - Pre-computation

Branching with Occlusion Query

- Use it for iteration termination

```
Do
{ // outer loop on CPU
  BeginOcclusionQuery
  {
    // Render with fragment program that
    // discards fragments that satisfy
    // termination criteria
  } EndQuery
} While query returns > 0
```

- Can be used for subdivision techniques

OQ extensions

- HP_occlusion_test
 - True/false if any pixels are rendered
- NV_occlusion_query
 - How many pixels are rendered ?
 - (also on the radeon 9800)

```
for (i = 0; i < N; i++) {
    glBeginOcclusionQueryNV(occlusionQueries[i]);
    // render bounding box for object I
    glEndOcclusionQueryNV();
} // at this point, if possible, go and do some other computation

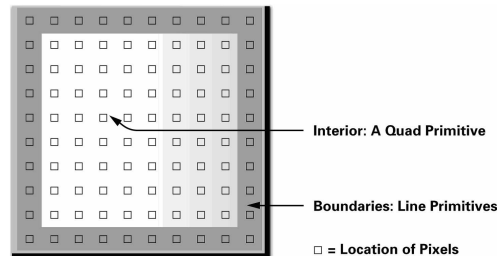
for (i = 0; i < N; i++) {
    glGetOcclusionQueryiivNV(occlusionQueries[i], GL_PIXEL_COUNT_NV, &pixelCount);
}
```

OQ material

http://www.nvidia.com/dev_content/nvopenglspecs/GL_NV_occlusion_query.txt
<http://www.cs.unc.edu/~coombe/research/radiosity/>

Static Branch Resolution

- Avoid branches where outcome is fixed
 - One region is always true, another false
 - Separate FPs for each region, no branches
- Separate expensive computation
- Example:
boundaries



Z-Cull

- In early pass, modify depth buffer
 - Clear Z to 1
 - Draw quad at Z=0
 - Discard pixels that should be modified in later passes
- Subsequent passes
 - Enable depth test (GL_LESS)
 - Draw full-screen quad at z=0.5
 - Only pixels with previous depth=1 will be processed
- Can also use early stencil test
- Not available on NV3X
 - Depth replace disables Zcull

Z-Cull, tested code

```
glEnable(GL_DEPTH_TEST);
glClearDepth(1);
glClear(GL_DEPTH_BUFFER_BIT);
glDepthFunc(GL_LESS); // GL_ALWAYS Does not work?

//render fragments that will later represent excluded fragment-
processing

glDepthFunc(GL_LESS);
glDepthMask(GL_FALSE);
```

Z-cull drawing geometry at a specific depth

- Depth to z coordinat defined by
 - glOrtho(0.0, texWidth, 0.0, texHeight, 1, 10);
 - Near: 1, far: 10
 - Z values from -1 to -10
 - Not a linear mapping

- Draw in negative z
 - glVertex3f(x,y,-2);
 - Depends on Model View

GeForce 6 Series Branching

- True, SIMD branching
 - Lots of incoherent branching can hurt performance
 - Should have coherent regions of ≈ 1000 pixels
 - That is only about 30×30 pixels, so still very useable!
- Don't ignore overhead of branch instructions
 - Branching over < 5 instructions may not be worth it
- Use branching for early exit from loops
 - Save a lot of computation

Pre-computation

- Pre-compute anything that will not change every iteration!



Multiple surfaces



Performance tips

- Multi surface Pbuffers
 - Avoid context switches by doing ping-pong between different surfaces of the same PBuffer



Multi Render Target

- Render to more than one surface of a render-target.
- OpenGL:
 - `void DrawBuffersATI(sizei n, const enum *bufs);`
- Fragment program
 - Specify: `ATI_draw_buffers` option
 - Use: `result.color[n]`

http://oss.sgi.com/projects/ogl-sample/registry/ATI/draw_buffers.txt