Shared memory on the GPU

Jesper Mosegaard

Some slides are based on SIGGRAPH 2004 GPGPU COURSE and Visualization 2004 Course

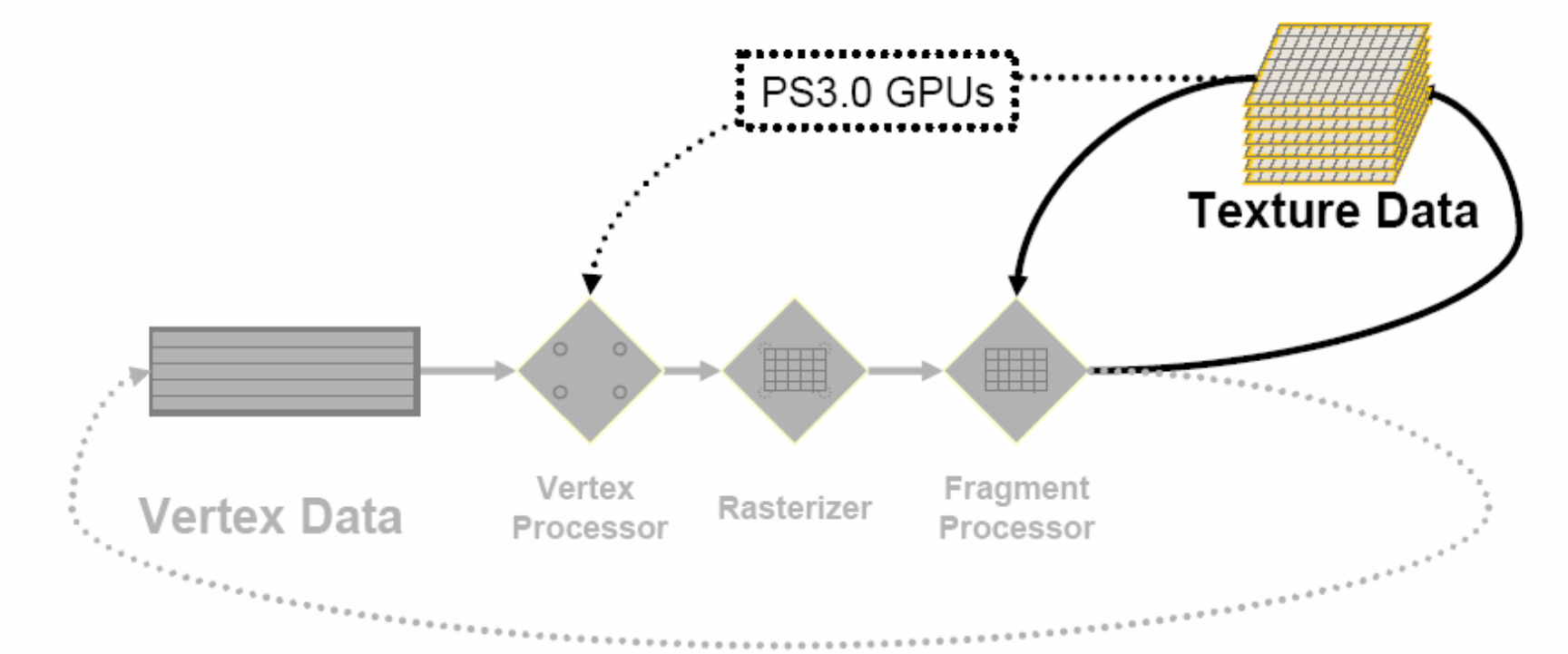
Plan

- Shared Memory on the GPU
  - PBuffer
  - FBO
- Memory addressing
- Memory write
  - MRT

Shared memory on the GPU

- Problem
  - How do fragment communicate ?
  - How can calculation depend on previous calculations ?
- Solution
  - “Render-to-texture”
  - Reading from texture

Shared Memory



Solutions to rtt

- glCopyTexImage2D
  - Copy of current frame-buffer to a texture
  - Slow…
- PBuffer and renderTarget based
  - Render to offscreen Pixel buffer, bind buffer as texture.
  - Faster, but each PBuffer has an OpenGL context
  - WGL extension
- Framebuffer Objects
  - Attaching texture images to framebuffer objects
  - Not dependant on WGL extensions
  - Potentially faster than PBuffers

PBuffer Material

http://www.ati.com/developer/ATIpbuffer.pdf
http://developer.nvidia.com/object/gdc_oglrtt.html

http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl_pbuffer.txt
http://oss.sgi.com/projects/ogl-sample/registry/ARB/wgl_render_texture.txt

http://www.gpgpu.org/developer/

http://www.opengl.org/resources/features/GL_EXT_render_target.txt
http://www.opengl.org/discussion_boards/cgi_directory/ultimatebb.cgi?ubb=get_topic;f=3;t=011406

RTT through PBuffer

- WGL_ARB_pbuffer
  - Pixel buffers are additional non-visible rendering buffers for an OpenGL renderer
- WGL_ARB_render_texture
  - Allows a color buffer to be used for both rendering and texturing
- A way to implement render-to-texture
  - (WGL = on windows only)

Creating the PBuffer

- Query device and rendering contexts
  - *wglGetCurrentDC() wglGetCurrentContext()*
- Choose pixel format supporting bind to texture
  - *wglChoosePixelFormatARB()*
- Create the PBuffer
  - *wglCreatePbufferARB()*

Rendering to the Pbuffer

- Switch rendering context, frame buffer writes.
  - *wglMakeContextCurentARB()*

Binding PBuffer as texture

- Create Texture Object
  - *glGenTextures()*
- Bind Texture Object
  - *glBindTexture()*
- Bind PBuffer to currently bound texture object
  - *wglBindTexImageARB()*
- Release PBuffer
  - *wglReleaseTexImageARB()*

PBuffer problems

- Pbuffers Give us Render-To-Texture
  - Designed to create an environment map or two
  - Never intended to be used for GPGPU (100s of pbuffers)

  - Problem
    - Each pbuffer has its own OpenGL render context
    - Each pbuffer may have depth and/or stencil buffer
    - Changing OpenGL contexts is slow

  - Solution
    - Many optimizations to avoid this bottleneck…
    - Allocate several surfaces

PBuffer Tips

- Ping-Pong
  - Create a PBuffer with a front and back buffer
    - Switch between the two, rendering and reading
    - Avoids context switch
  - Use auxillary buffers for more surfaces

Floating point textures, material

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

http://www.nvidia.com/dev_content/nvopenglspecs/GL_NV_float_buffer.txt

Floating point textures

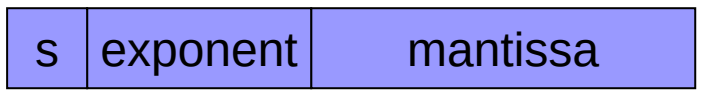
- Vendor Specific Extensions
  - ATI_texture_float (Also supported by NV40)
  - NV_float_buffer
- General
  - ARB_texture_float (only on NV)
- Support for filtering may be limited / implemented in software

Key Differences

- ARB_texture_float
  - 16bit and 32bit textures
- Older:
  - ATI_texture_float
    - Available in Fixed Function and Programs
  - NV_float_buffer
    - Rectangle Textures
    - Available ONLY in Programs

Floating point texture formats

- FP32 (IEEE standard)
  - s23e8 (largest counting number: 16,777,217)
- FP16
  - s10e5 (largest : 2,049)
- Remember, the ATI Radeon 9800 can ONLY work with 24 bit internally
  - s16e7 (largest : 131,073)



$$\text{sign} * 1.\text{mantissa} * 2^{(\text{exponent}+\text{bias})}$$

Using PBuffers for your projects

- PBuffers have been wrapped up nicely in the RenderTexture library – see RenderTexture.h

- Construct new RenderTexture
  - `rt = new RenderTexture();`
- Setup format
  - `rt->Reset("rgba=32f tex2D rtt");`
- Bind for rendering
  - `rt->BeginCapture();`
- Bind to the current active texture
  - `glActiveTextureARB(GL_TEXTURE0_ARB);`
  - `rt->BindBuffer(WGL_FRONT_LEFT_ARB);`

Framebuffer Object

- EXT_framebuffer_object
- Advantages
  - Single OpenGL context
    - Switching framebuffers is potentially faster than switching OpenGL context (*wglMakeCurrent*)
  - Share renderbuffer images and texture images between framebuffers
  - Easier to use.

The following slides on framebuffer objects are based partly on NVIDIA's GDC2005 slides on framebuffer objects

Framebuffer Object

- OpenGL Framebuffer, logical buffers
  - Color, depth, stencil, accumulation
- EXT_framebuffer_object
  - Attach and detach texture and renderBuffers as logical framebuffer objects

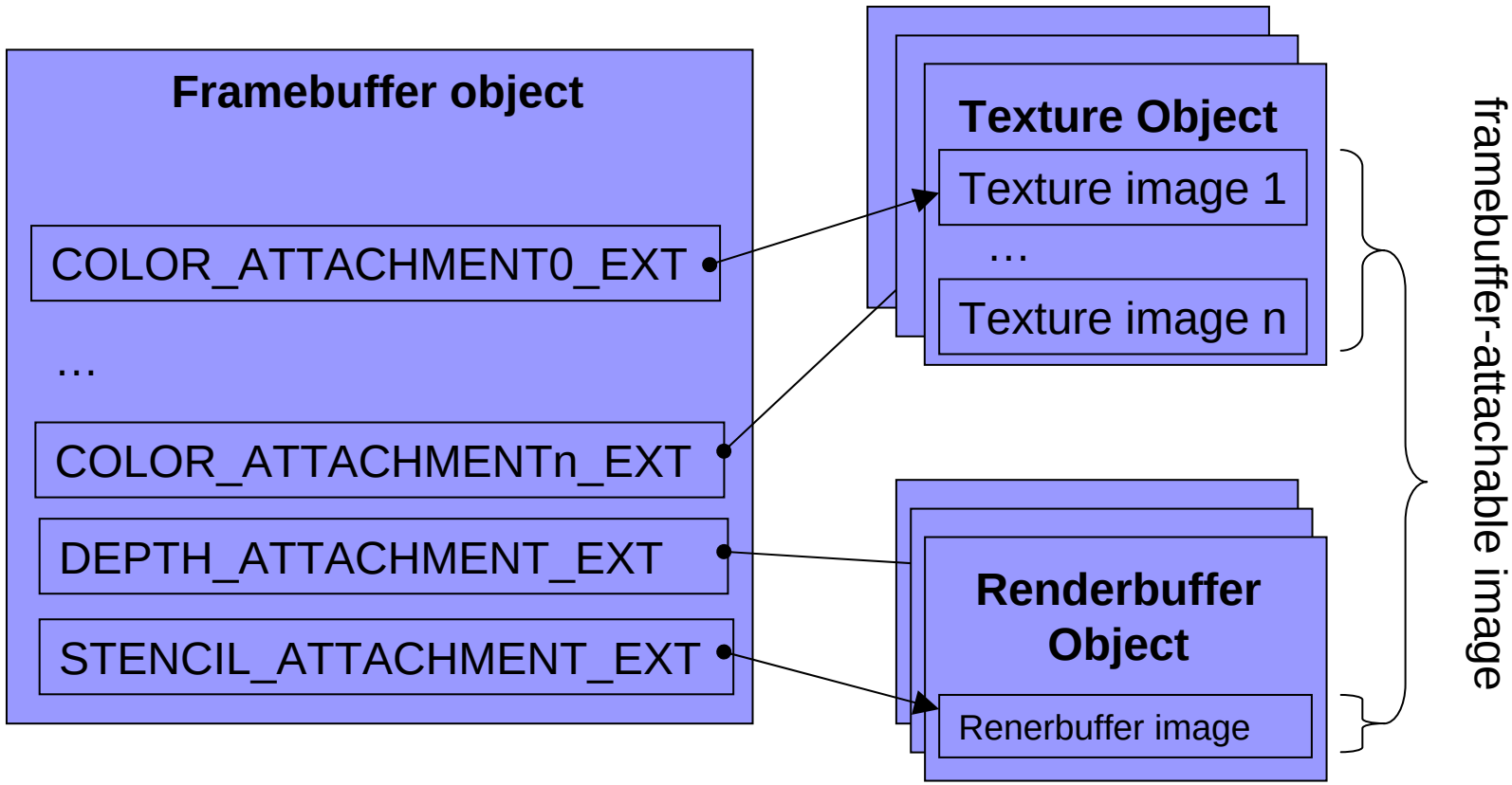
Framebuffer Object, material

http://www.opengl.org/documentation/extensions/EXT_framebuffer_object.txt
http://download.nvidia.com/developer/presentations/2005/GDC/OpenGL_Day/OpenGL_FrameBuffer_Object.pdf

Framebuffer_object

- Application-generated framebuffer object
  - Collection framebuffer-attachable image
    - Offscreen buffers (renderbuffer)
    - Textures (texture image)

Object architecture



New Terminology

- *Framebuffer*
  - Collection of framebuffer attachable images
  - State (direction of GL rendering)
  - Windows-system framebuffer and application-generated
- *Renderbuffer* – offscreen image and state
  - *Renderbuffer image* – 2D array of pixels. Part of a renderbuffer.
- *Framebuffer-attachable image* – 2D array of pixels that can be attached to a framebuffer. (e.g. texture image and renderbuffer image)
- *Attachement point* – state that references a framebuffer-attachable image in a framebuffer. (color, depth, stencil)
- *Attach* – connecting one object to another. Similar to bind.

FBO: Create framebuffer

glGenFramebuffersEXT(); // generate a framebuffer

glGenTextures(); // generate a texture

// bind the generated framebuffer
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, framebufferID);

glBindTexture(GL_TEXTURE_2D, textureID); // setup texture parameters

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, w, h, 0, GL_RGBA,
GL_FLOAT, NULL);

// Attaching the two Texture Images to the Framebuffer
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, textureID, 0);

FBO: Render to image of framebuffer

// Bind application-generated framebuffer ID
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, ID);
glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT); // render attachment 0

DRAW OPENGL STUFF

// Bind window-system framebuffer object
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,0);

FBO: Bind Texture

- Just like usual !

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, TEXID);

Performance tips

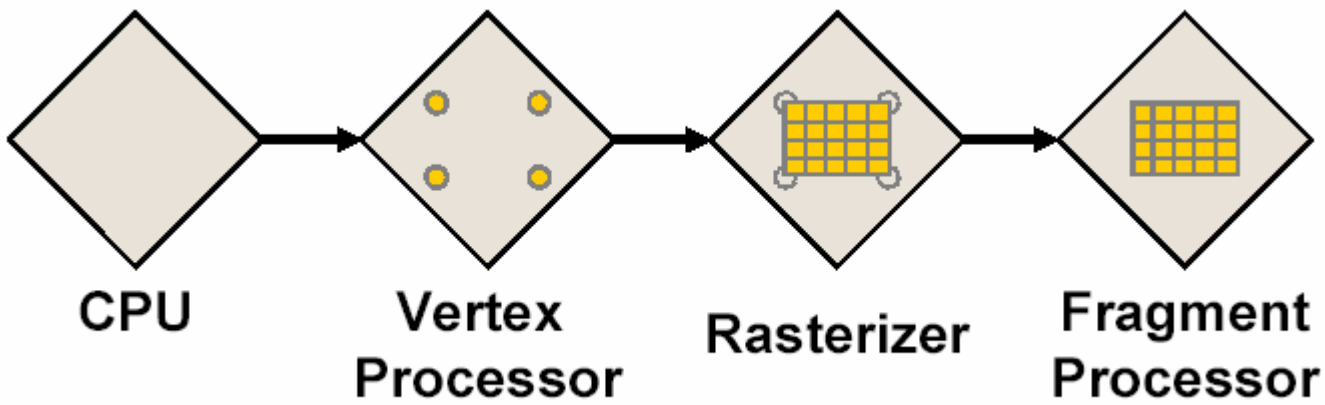
- Don’t create and destroy FBO’s every frame
- Try to avoid modifying textures used as rendering destinations using TexImage, CopyTexImage etc.

FBO Usage Scenarios

- FBO allows several ways of switching between rendering destinations. In order of increasing performance:
  - Multiple FBOs
    - create a separate FBO for each texture you want to render to
    - switch using BindFramebuffer()
      - can be 2x faster than wglMakeCurrent() in beta NVIDIA drivers
  - Single FBO, multiple texture attachments
    - textures should have same format and dimensions
    - use FramebufferTexture() to switch between textures
  - Single FBO, multiple texture attachments
    - attach textures to different color attachments
    - use glDrawBuffer () to switch rendering to different color attachments

Memory addressing

- CPU defined vertex attributes
  - Define few vertices
- Vertex based calculations
- Interpolated by rasterizer
  - Exploit this if possible
- Fragment based calculations
  - Creates dependent texture lookups (problem on radeon 9800)



Texture lookup

- Tip:
  - Move as much independent math after a texture fetch to hide the latency

Warning

- Floating-point can leave unaddressable texels
  - Nvidia FP32 (16,777,217)
  - Nvidia FP16 (2,049)
  - Ati 24-bit float: (131,073)

The tradeoff of parameters

- Draw quad with four texture coordinats
  - Fast drawing, interpolation not flexible
- Draw pixel sized points with individual texture coordinats
  - Very slow draw, very flexible

Memory write

- Output from fragment program
  - Only one 4-vector (rgba)
- Multi Render Target (MRT)
  - Write to several ”surfaces” (or framebuffer attached images)

MRT: Material

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

MRT

- Render to several (up to four at the moment) surfaces at the same position

- OpenGL:
  - void **DrawBuffersATI**(sizei n, const enum *bufs);
- Fragment program
  - Specify: ATI_draw_buffers option
  - Use: result.color[n]
- Cg
  - COLOR1 through COLOR3 binding sematics