

**NVIDIA**®

**GeForce 8800 OpenGL Extensions**

**Evan Hart**

# Roadmap



- **What's different**
- **The programmable core**
- **Feeding it**
- **New pathways**
- **The backend**

# GeForce 8800 Differences

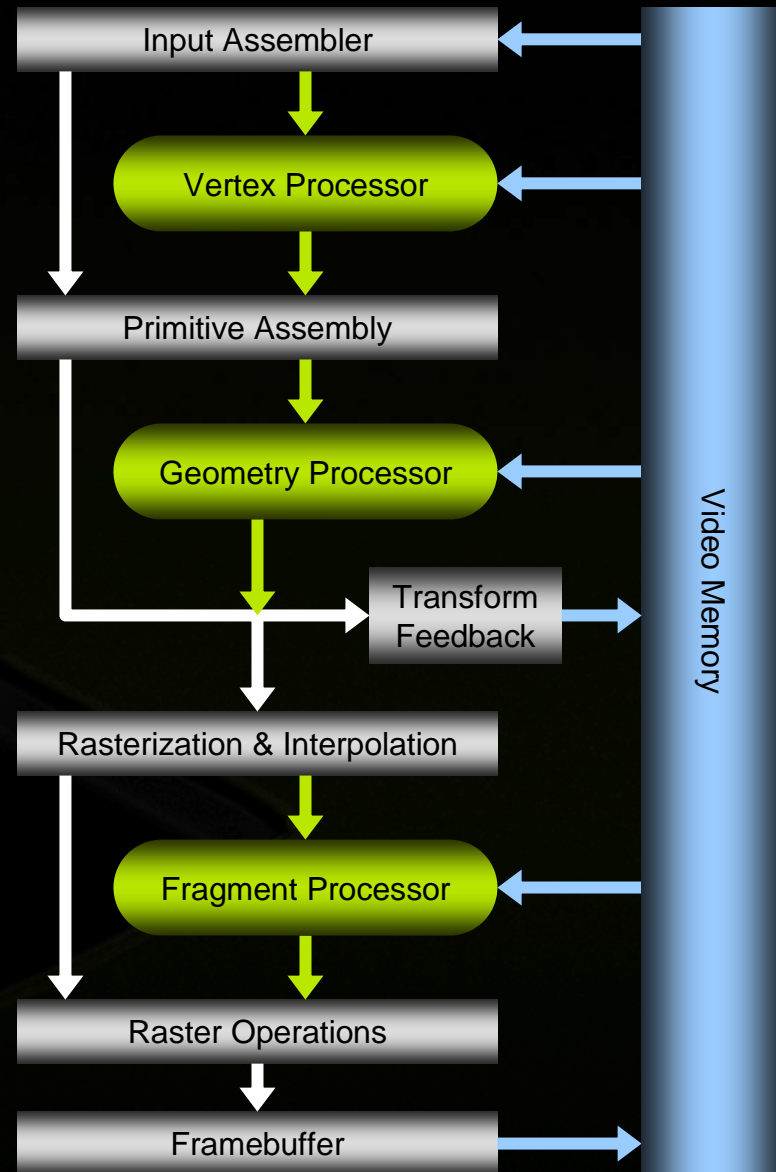
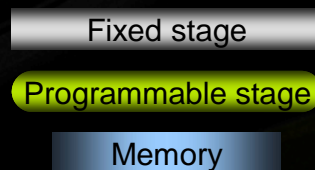
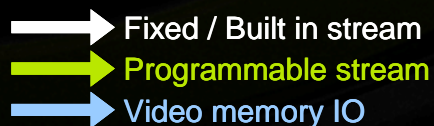


- **Pipeline modifications**
  - Additional geometry shader stage
  - Feedback available midstream
- **Unified shading hardware**
  - Same instructions and characteristics across shaders

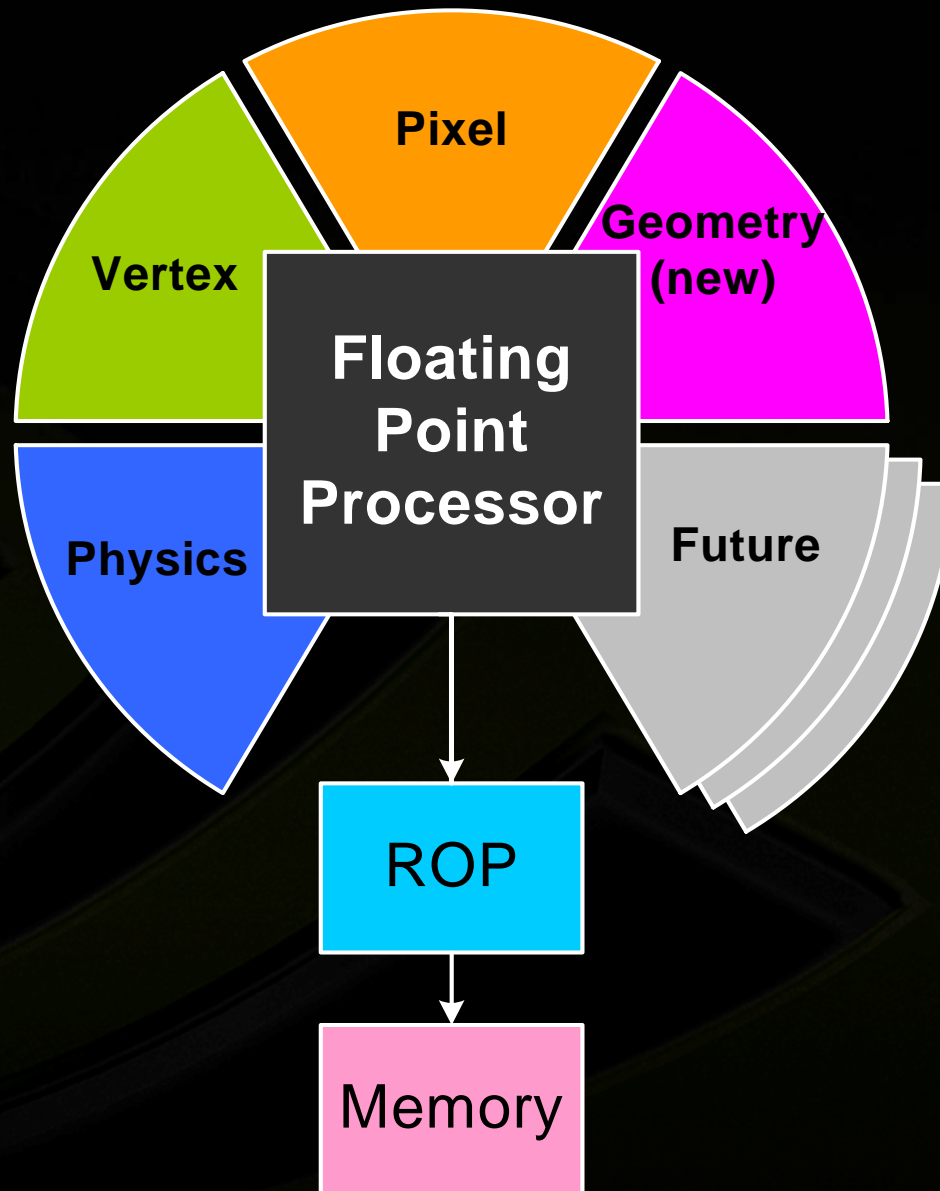
# GeForce 8800 OpenGL Pipeline



- **More flexible memory access model**
  - Multiple ways to read and write
- **Additional pipeline stages**
  - Fundamentally new capabilities



# Unified Shaders



# Programmability



- **#1 design concern of all extensions**
  - Efficient for input
  - Efficient for computation
  - Efficient for output
- **No concessions for fixed-function**
  - Only the right choices for programmability
  - Most will not work with the fixed function pipeline

# New Capabilities



- **Unified instruction set for programs**
- **Integer instructions and data types**
- **Uniform set of structured branching constructs**
- **Indexable constants and temporaries**
- **New texture fetching instructions**
- **Attribute interpolation control**
  - **Flat shaded, perspective-incorrect or centroid sampled**

# New OpenGL Program Extensions



- **OpenGL Shading Language**
  - **EXT\_gpu\_shader4**
    - GLSL extension for fourth generation shaders
- **ARB\_vertex\_program style asm-like programs**
  - **NV\_gpu\_program4**
    - **NV\_vertex\_program4**
    - **NV\_fragment\_program4**
    - **NV\_geometry\_program4**
- **Cg 2.0**
  - **Not a GL extension**
  - **Will support the capabilities**



# GL\_EXT\_gpu\_shader4



## ● New integer support

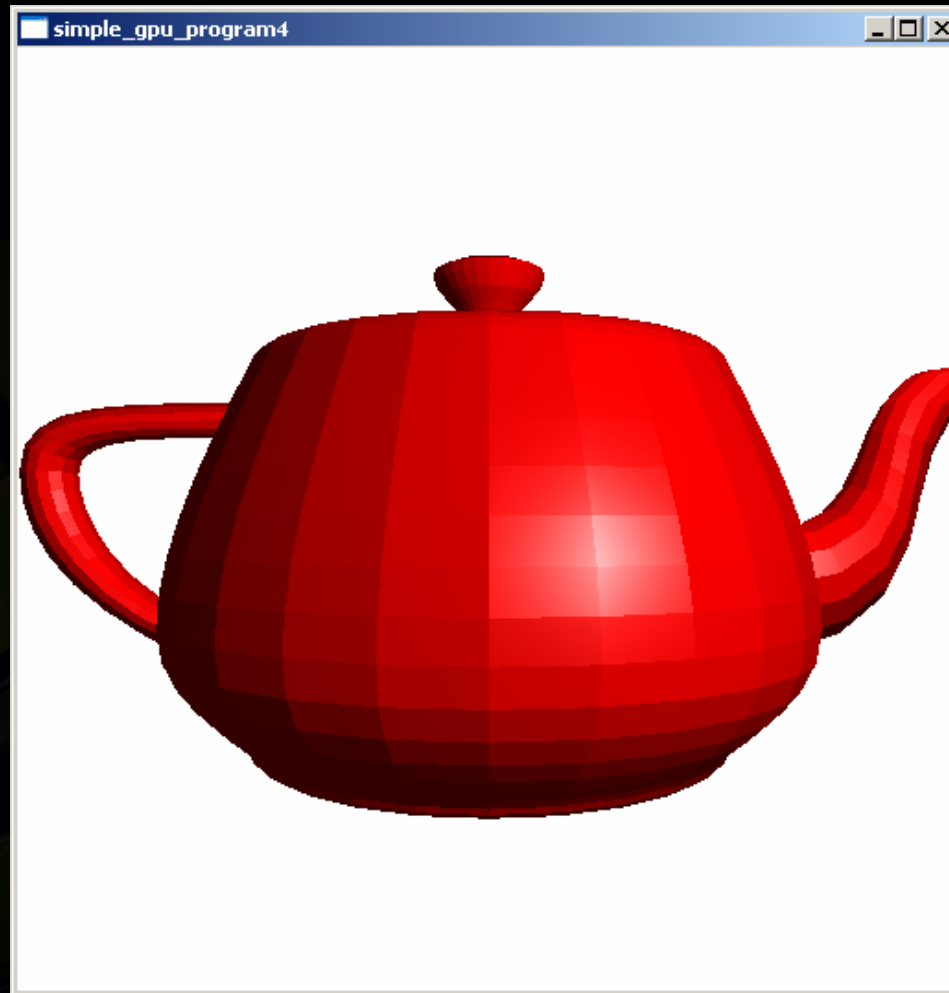
- unsigned int, uvec2, uvec3, uvec4
- Integer varying and attributes
- Integer texture samplers: `isampler*`, `usampler*`
- Real integer ops
  - `%`, `&`, `|`, `^`, `>>`, `<<`, `~`

# GL\_EXT\_gpu\_shader4 cnt'd



- **Extends varying type qualifiers**
  - **Centroid** – keeps the value inside the covered region
  - **Flat** – no interpolation, like flat shading
  - **Noperspective** – interpolate in screen-space

# Flat interpolation



- **Instancing and procedural generation**
  - **gl\_VertexID**
    - Integer index derived from glDrawElements, etc
    - Only with vertex arrays (no display lists)
    - Only when using VBOs
  - **gl\_InstanceID**
    - Integer index of the current primitive
    - Only available when using DrawElementsInstancedEXT
  - **gl\_PrimitiveID**
    - Integer describing the primitive number

# GL\_EXT\_gpu\_shader4 cnt'd



- **User-defined output variables**
  - Declared as “varying out”
  - Allow more flexible outputs from the fragment shader
    - Integers
    - Integers and floats simultaneously
  - Used instead of `gl_FragColor` or `gl_FragData`

# GL\_EXT\_gpu\_shader cnt'd



## ● Exact texel fetches

- `texelFetch*( sampler, icoord, lod)`
- Treats a texture as a directly addressable array of texels

## ● Texture size query

- `textureSize*( sampler, lod)`
- Returns the dimensions of the texture level

## ● Shadow cubemaps

- Depth is compared against the 4<sup>th</sup> component

## ● Texture Gradient fetches

- `texture*Grad( sampler, coord, dx, dy)`
- Allows custom lod/anisotropy control

## ● Offset texture fetches

- `texture*Offset( sampler, coord, offset)`
- Offset must be compile-time constant expression
- Allow convenient shortcut for building filter kernels
- Offset size has an implementation dependent limit

# GL\_NV\_gpu\_program4



- **Composed of three sub-extensions**
  - GL\_NV\_vertex\_program4
  - GL\_NV\_fragment\_program4
  - GL\_NV\_geometry\_program4
- **Still based on 4-wide registers**
  - No longer really matches HW
  - Enhances backward compatibility
- **Provides same capabilities as EXT\_gpu\_shader4**



# Feeding the Shader



## ● Instancing

- Optimized rendering of multiple copies of an object

## ● New texture types

- Texture arrays

## ● New texture formats

- Integer textures
- Additional HDR formats

## ● Data buffers

- Fast flexible ways to swap blocks of constants

## ● Texture buffers

- Massive data store for shaders

# EXT\_draw\_instanced



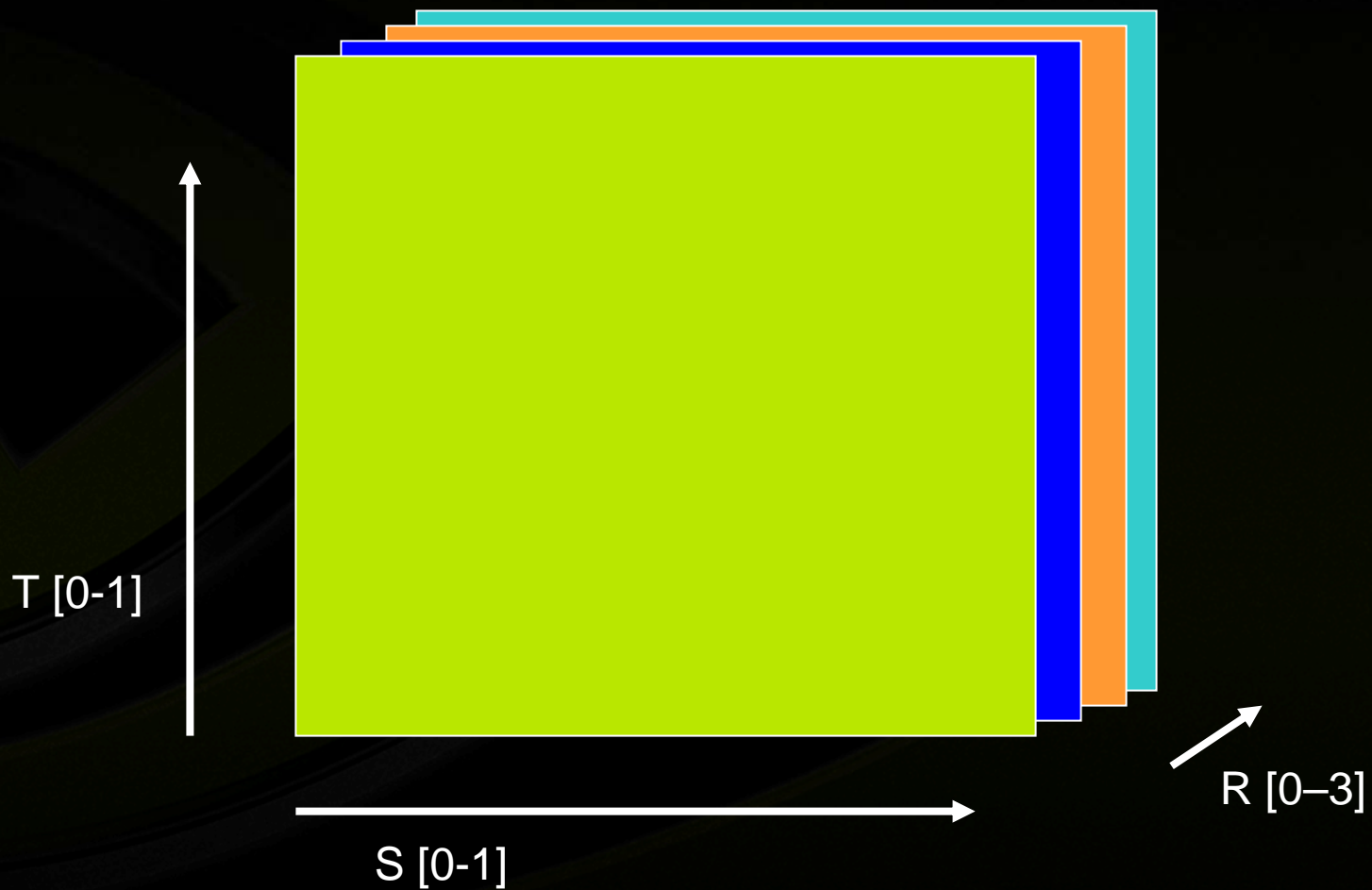
- **Efficient rendering for large numbers of objects**
- **Vertex array only**
  - **glDrawArraysInstancedEXT**
  - **glDrawElementsInstancedEXT**
- **Draw calls take one additional parameter**
  - **# of instances to draw**
- **Each instance has a separate instance ID**
  - **Used by the shader to change behavior**
    - **Select transform matrix**
    - **Select material**
- **Clever shaders can even draw different objects**

# EXT\_texture\_array



- **Array Textures**
- **Generalizes 1D and 2D textures to consist of an array of 1D or 2D textures**
  - 1D array loaded using `glTexImage2D`
  - 2D array loaded using `glTexImage3D`
- **Array indexable from shader program**
  - Access layer using `r` texture coordinate
- **Layers must be same size and format**
- **No filtering between layers**
- **Arrays of cubemaps not currently supported**
- **Removes need for texture atlases**
  - Useful for instancing, terrain texturing

# Array Textures Cont'd



# Texture Format Extensions



- **EXT\_texture\_integer**
  - Adds integer texture formats
- **EXT\_packed\_float**
  - Space-efficient float format
  - Relatively low precision
    - More than good enough most times
- **EXT\_texture\_shared\_exponent**
  - Space-efficient float format
  - Variable accuracy
    - Can be more or less accurate than packed float
    - Also more than good enough

# Integer texture formats



## ● EXT\_texture\_integer

- Adds integer texture formats
- 8, 16, and 32 bit per component
- Signed and unsigned
- RGB, RGBA, Luminance, Alpha, Intensity, and LA
- Lack filtering support
- Only available with EXT\_gpu\_shader4 / NV\_gpu\_program4

## ● Uses

- Bitfields
- Color index emulation
- Lookup tables

# Packed Float Textures



- **EXT\_packed\_float**
  - **11/11/10 floating point format**
  - **5 bit exponent per component**
    - **Bias of -15**
  - **Only supports positive values (no sign bit)**
  - **Can be used as framebuffer format**
  - **Max values**
    - **R/G – 65024**
    - **B – 64512**
  - **Size advantage can make it much faster than float16**
  - **Supports filtering, blending, and MSAA**

# Packed Float Texture Usage





# RGBE / Shared Exponent textures



- **EXT\_texture\_shared\_exponent**
  - **9/9/9/5 RGBE format**
  - **Similar to Radiance 8/8/8/8 RGBE format**
  - **Shared 5 bit exponent (bias of -15)**
  - **Source texture format only (not renderable to)**
  - **Only supports positive values (no sign bit)**



# New compressed texture formats



- **EXT\_texture\_compression\_latc**
  - Good format for greyscale w/ alpha compression
  - 8-bits per texel
  - Stored in 4x4 blocks (like DXT formats)
  - Components are compressed independently
  
- **EXT\_texture\_compression\_rgtc**
  - Same properties as latc
  - Returns (r,g,0,1) instead of (l,l,l,a)
  - Useful for normal map compression
    - $(x,y) = 2 * (r,g) - 1$
    - $Z = \text{sqrt}(1 - (x^2 + y^2))$

# OpenGL Data Buffer Extensions



- **EXT\_bindable\_uniform**
  - Used with EXT\_gpu\_shader4
  - Allows a uniform to be bound to a buffer object
  - Quickly switch all values in a large structure or array
  
- **NV\_parameter\_buffer\_object**
  - Used with NV\_gpu\_program4
  - Defines new object containing banks of uniform parameters
  - Enables rendering to parameter buffers
  - Useful for instancing
  - Provides a very large parameter store

# EXT\_texture\_buffer\_object



- **Bind buffer object as a texture**
- **Jumbo 1D texture**
  - **Larger size limit (128 Mtexels)**
- **Does not support filtering**
- **Addressed by element**
  - **Not normalized [0-1]**
- **Limited format support**
  - **No RGB support (RGBA is OK)**
  - **Minimum of 1 byte per component**
  - **No compressed formats**

# Texture buffer usage



- **Large constant store**
  - Significantly larger than EXT\_bindable\_uniform
  - Jumbo bone list for skinning
- **Instancing data**
  - Transform matrices
  - Materials
- **Custom indexing**
  - Separate 'index' for position, normal, texture coordinate
  - Less efficient than normal vertex fetching
    - Not as coherent
  - Can be useful interactive editing due to reduced sw cost

# New Pathways



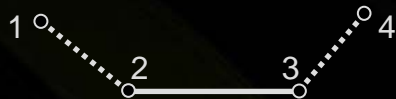
- **Geometry Shaders**
  - New pipeline stage
- **Render target arrays**
  - Geometry shader indexing of texture arrays
- **Transform feedback**
  - Export vertices mid-stream

# Geometry Shader Basics

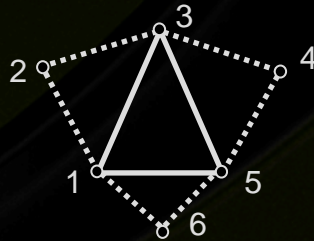


## Input

- Standard primitives
  - point, line, triangle...
- New primitive types include neighboring vertices
  - Line with adjacency

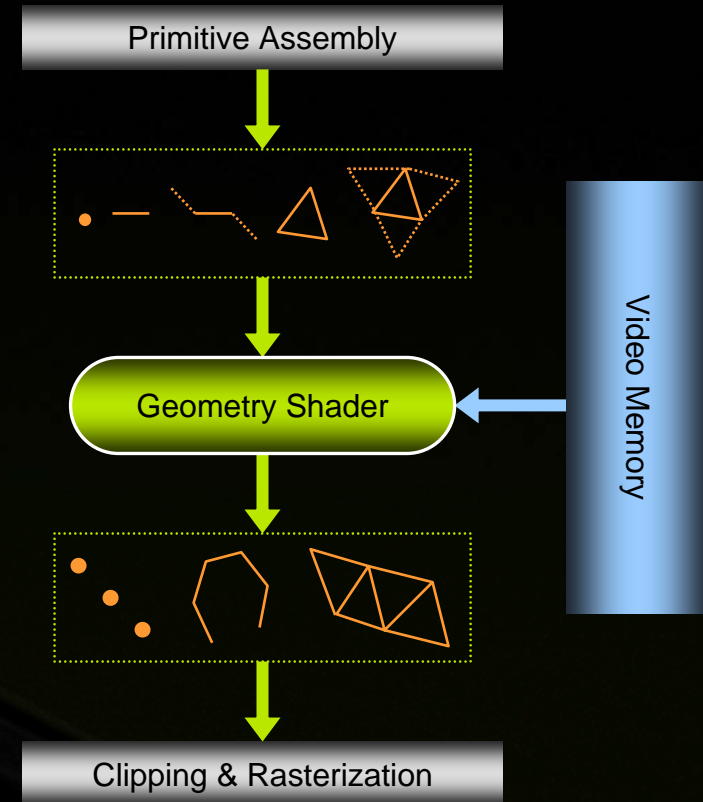


- Triangle with adjacency



## Output

- Unique output type (independent from input type)
- Points, line strips or triangle strips
- Can output zero or more primitives
- Generated primitive stream is in the same order as inputted



# Geometry Shader Applications



- **Better point sprites**
  - Rotation, non-square, motion blur
- **Simple subdivision**
- **Single pass cube map creation**
- **Automatic stencil shadow polygon generation**
- **Fur rendering**
  - Fin generation
- **Curve rendering**
  - 2D rendering, hair/fur
- **Particle systems**
- **GPGPU**
  - Data amplification – variable number of outputs



# Geometry Shader Extensions



- **EXT\_geometry\_shader4**
  - GLSL geometry shaders
- **NV\_geometry\_shader4**
  - Adds to EXT\_geometry\_shader4
- **NV\_geometry\_program4**
  - ARB\_vertex\_program style
  - Part of NV\_gpu\_program4

# EXT\_geometry\_shader



## ● New link-time parameters

- Primitive input and output type
- Max vertices output

## ● New shader variables

- `gl_VerticesIn` – number of input vertices
- `gl_Layer` – texture array layer target
- `gl_PrimitiveID` – Set primitive ID seen by the fragments
- `gl_PrimitiveIDIn` – Primitive ID based on input prims

# EXT\_geometry\_shader code



```
varying in vec3 eyeNormal[gl_VerticesIn];

varying out vec3 oEyeNormal;

for ( int i = 0; i < gl_VerticesIn; i++) {
    oEyeNormal = eyeNormal[i];
    //causes all output varying to submit
    EmitVertex();
}

//Start a new strip
RestartPrimitive();
```

# NV\_geometry\_shader4



- **Relax restrictions of EXT\_geometry\_shader4**
  - Changing input/output primitive without a relink
  - Changing max output vertices without a relink
- **Defines additional behavior**
  - Quads and polygons are turned into triangles
    - Shader accepting triangle accepts quads

# NV\_geometry\_program4



- Same capabilities as geometry\_shader4
- ARB\_vertex\_program style syntax
- Inputs are 'ATTRIB'
- Outputs are 'RESULT'
- Input primitive, output primitive, and max vertices
  - Declared in the shader

# Rendering to Texture Arrays



- **Can select destination layer for each primitive in geometry program**
  - Write to “gl\_Layer” or “result.layer”
- **Can be used for single-pass render-to-cubemap**
  - Read input triangle
  - Output to 6 cube map faces, transformed by correct face matrix
  - No real savings for GPU
    - Same transform and rasterization load
    - Additional geometry shader work
  - Simple culling may help

# EXT\_transform\_feedback



- **Allows storing output from a vertex program or geometry program to buffer object**
- **Enables multi-pass operations on geometry, e.g.**
  - **Store results of animation (skinning) to buffer, reuse for multiple lights**
  - **Recursive subdivision**
- **Provides queries for number of primitives generated by geometry program**

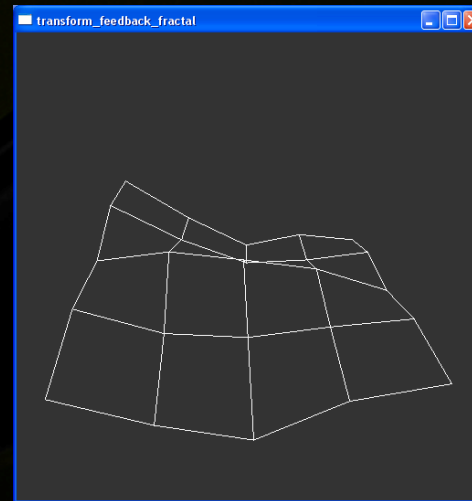
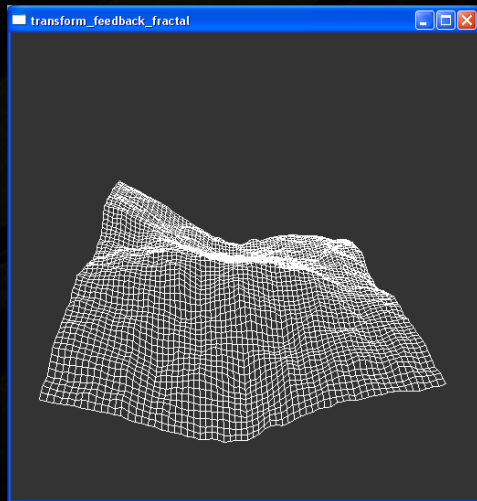
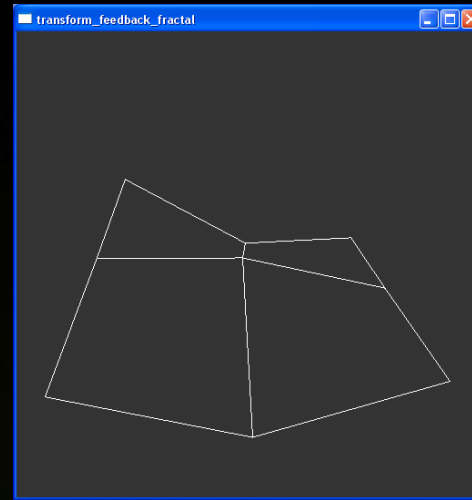
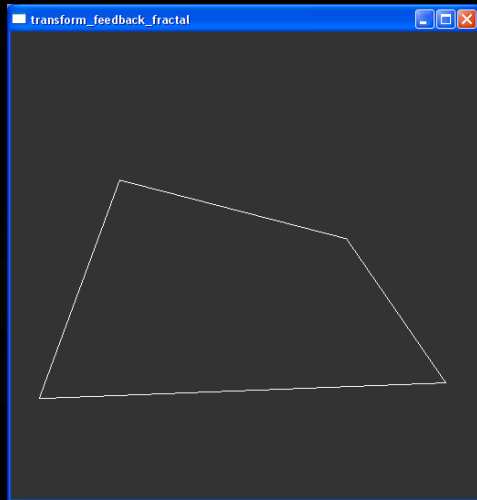
# Example: Terrain Subdivision



- Takes quad as input (actually `line_adj` primitive)
- Geometry program subdivides into 4 new quads using diamond-square subdivision
- Uses two VBOs, reads from one, writes to the other using transform feedback
- Then swap



# Terrain Subdivision



# The backend



- **Floating point depth buffers**
- **EXT\_draw\_buffers2**
- **sRGB Framebuffers**
- **Coverage sample anti-aliasing**

# NV\_depth\_float



- Provides floating point depth buffers and textures
- Range extended to  $[-MAX\_FLOAT, MAX\_FLOAT]$
- New functions for non-normalized values
  - `glDepthRangedNV`
  - `glClearDepthdNV`
  - `glDepthBoundsdNV`
- Multiple formats
  - `DEPTH_COMPONENT32F_NV`
  - `DEPTH32F_STENCIL8_NV`
- FBO only
- Care must be taken in using the extra precision

# Float depth precision



- **Normal [0-1] mapping is ineffective**
  - Precision bunches up at 0
    - Logarithmic distribution of floats
    - 24 bits of precision [0.5 ( $2^{-1}$ ) – 1.0 ( $2^0$ )]
  - Perspective projection pushes scene toward 1
    - 2x near maps to roughly 0.5
    - 90%+ of scene [0.5 – 1.0] range
  - Effectively a 25-bit depth buffer
- **Changing to [0-256] is no better**
  - [128 – 256] has the same 24-bits

# Depth Precision Distribution



Integer distribution



Floating point distribution



↑  
2x near

# The Solution



- **Reverse the depth mapping**
  - Far = 0.0
  - Near = 1.0 (or higher)
- **Precision bunching now reversed**
  - Compression and expansion line up
  - Results in essentially linear depth buffering

# A Caveat



- **Small precision cliff near 0**
  - FP numbers often use denormals to fill it
  - Graphics hardware typically avoid it
    - Often CPU's too, denorms are slow
- **Problem is minimal due to exponent range**
  - Infinite far plane completely fixes it

# EXT\_draw\_buffers2



- Provides per draw buffer control of
  - Blend Enable
  - Color mask
- Does not provide independent control of
  - Blend Function
  - Blend Equation
  - Blend Color



# EXT\_framebuffer\_sRGB



- **Allows framebuffer to be stored in sRGB space**
  - Provides perceptual color compression
    - 8 bits sRGB is similar to 10 bits linear RGB
  - Converts to/from sRGB on access to framebuffer
  - Implements a gamma of 2.2
    - This matches most monitors relatively well
- **Controlled via a simple enable**
  - Not available on all formats (>8 bit per component)

- **Coverage Sample Anti-Aliasing**
  - `GL_NV_framebuffer_multisample_coverage`
- **Decouples primitive coverage from depth/color**
  - Increases edge quality with little cost
  - Memory and bandwidth overhead scale with # depth samples
  - Worst case is as good as # of depth samples

# CSAA usage



- **Extremely simple**

- **MSAA**

```
glRenderbufferStorageMultisampleEXT(  
    GL_RENDERBUFFER_EXT, 4, GL_RGBA8, width,  
    height)
```

- **CSAA**

```
glRenderbufferStorageMultisampleCoverageNV(  
    GL_RENDERBUFFER_EXT, 16, 4, GL_RGBA8, width,  
    height)
```

# CSAA Modes



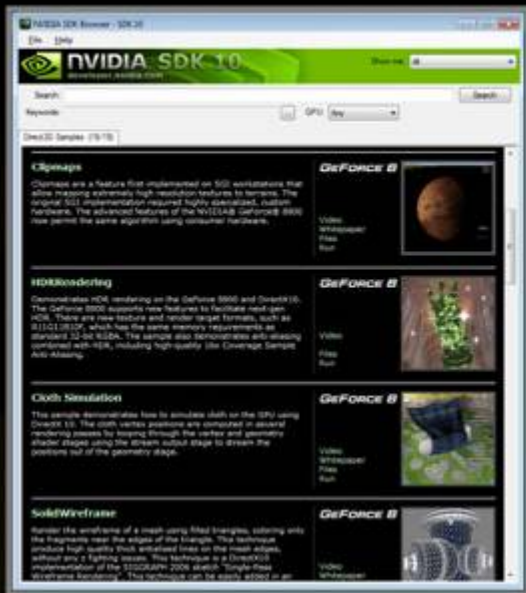
<b>Name</b>	<b>Coverage Samples</b>	<b>Color/Depth Samples</b>
<b>2x</b>	<b>2</b>	<b>2</b>
<b>4x</b>	<b>4</b>	<b>4</b>
<b>8x</b>	<b>8</b>	<b>4</b>
<b>8xQ</b>	<b>8</b>	<b>8</b>
<b>16x</b>	<b>16</b>	<b>4</b>
<b>16xQ</b>	<b>16</b>	<b>8</b>

# Thanks

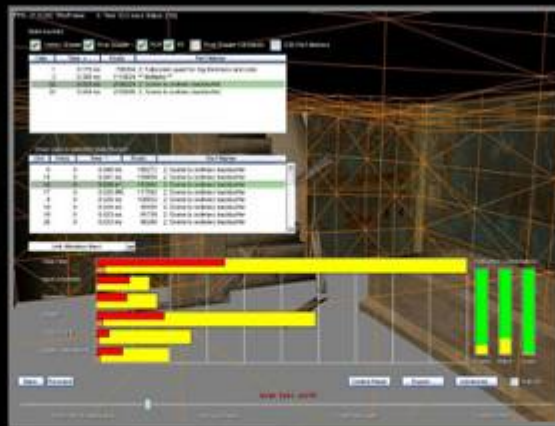


- **Simon Green**
- **Samuel Gateau**
- **Henry Moreton**
- **NVIDIA DevTech team**

# New Developer Tools at GDC 02007



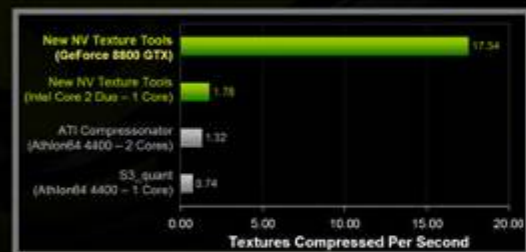
SDK 10



PerfKit 5



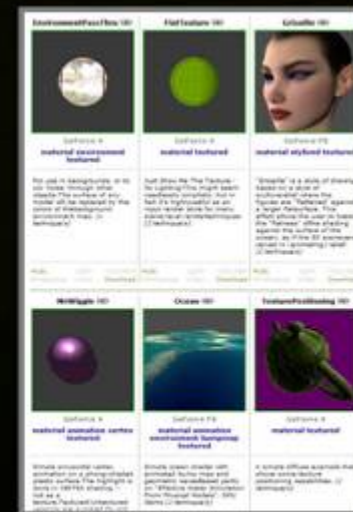
FX Composer 2



GPU-Accelerated Texture Tools



ShaderPerf 2



Shader Library