Programmability - a New Frontier in Graphics Hardware
A Revolution in Graphics Hardware

- Moving from graphics accelerators to processors
- Full hardware OpenGL and DirectX pipelines
Programmability Changes the World

- Graphics hardware pipelines are becoming massively programmable
- Will fundamentally change graphics
- Allows hyper-realistic characters, special effects, and lighting and shading
3D Graphics is about

- Animated films (Bug’s Life, Toy Story, etc.)
- Special Effects in live action movies (The Matrix)
- Interactive Entertainment (Video games)
- Computer Models of real world objects
  - Or, objects that haven’t been invented yet
- Making reality more fantastic
- Making fantasies seem real
Why are Movie Special Effects Exciting and Interesting?

• Suspension of Disbelief
  • Something amazing is happening
  • But, you believe it, because it is “real”
• Realistic and detailed characters
  • Motion, and emotion
• Realistic and recognizable materials
  • Chrome looks like chrome
  • Skin looks like skin
• Action!
The Year 2000 Graphics Pipeline

- T & L: vertex transform and lighting
- setup rasterizer
- texture blending: per-pixel texture
- fb anti-alias
Pixar’s Geri – A Believable Old Man

- Not a real person
- Geri is built from Curved Surfaces
- Curved surfaces are broken down into triangles
- Each triangle is transformed into position
- Each pixel in each triangle is shaded
- Every frame
  - 24 (movie) or 60 (PC) times per second
3D Movie Special Effects Come to PC and Console Graphics

- Lots of Geometry – lots of stuff going on
  - GeForce does this – hardware Transform & Lighting
  - The next generation makes the pipeline programmable
- Lots of Lighting and Shading
  - GeForce (year 2000)
    - Hardwired vertex lighting
    - Little “shader programs” run for every pixel
  - Taking Shading to the next level (year 2001)
    - Powerful “vertex programs” run for every vertex
    - Powerful “shader programs” run for every pixel
The Year 2001 Graphics Pipeline

- curved surfaces
- vertex shaders
  - per-vertex shading
- setup rasterizer
- tex-addr ops
  - shadows 3d tex
  - per-pixel shading
- texture blending
- fb antialias

NVIDIA
Microsoft \textit{xbox} \textit{Powered by NVIDIA}

- \textbf{Next-Generation GPU from NVIDIA}
  - 10X Graphics performance Playstation2
  - World’s First Tera-Op Processor
    - Over one Trillion Operations Per Second (1.2TOPS)
  - World’s first Programmable Shading Engine
- \textbf{NVIDIA Custom Media/Communication Processor}
  - Broadband
  - Unparalleled 3D Audio Capabilities
- \textbf{Additional features include:}
  - 733MHz x86 compatible CPU
  - 64MB of RAM (Unified memory architecture)
  - 8GB hard drive
  - 4X DVD drive with movie playback
  - Four game controller ports
  - Expansion port
Effects Explained

• (1) Shadows
  • Raven’s arm casts a shadow on her body

• (2) Reflections
  • Robot reflects Raven and the world

• (3) Lighting, shading and materials
  • Raven’s clothing looks like cloth with wrinkles and shape

• (4) Programmable Vertex Shading
  • Raven’s arms and body bend smoothly, like real arms

• (5) Anti-aliasing
  • Edges are smooth, not jagged
Programmable Vertex Processing

- GeForce family introduced hardware T&L to the PC
  - Transform and Lighting
- Next generation makes T&L user programmable
  - Vertex programs
- Developers can write custom
  - Vertex Transformation
  - Vertex Lighting
  - Special effects (layered fog, volumetric lighting, morphing…)
Developers Have Been Asking For…

• Complete control of the transformation and lighting hardware
• Complex vertex operations performed in hardware
• Custom vertex lighting
• Custom skinning and blending
• Custom texgen
• Custom texture matrix operations
• <your request goes here>
Custom Substitute for Standard T&L

Vertex Input
- 128 bits
- 4 floats
- 16 entries

Programmable Vertex Processor
- 128 instructions

Vertex Output
- 128 bits
- 4 floats
- 13 entries

Constant Memory
- 128 bits
- 4 floats
- 96 entries

Registers
- 128 bits
- 4 floats
- 12 entries

A0
What does it do?

- Per **vertex** calculation
- Processing of:
  - Colors – true color, pseudo color
  - 3D coordinates - procedural geometry, blending, morphing, deformations
  - Texture coordinates – texgens, set up for pixel shaders, tangent space bumpmap setup
  - Fog – elevation based, volume based
  - Point size
- Vertex program accepts one input vertex, generates one output vertex
Plus: Novel Effects… (Demos Later!)

- Irregular view transformation
  - Fish-Eye lens, ...
- Novel texture coordinate calculations
  - Projected textures
- Paletted skinning with 20 or more bones!
  - Now you can be much more efficient than with DirectX7™
- Geometry morphing
  - Blending multiple meshes
- Procedural Geometry Deformations
Vertex Programs
Physics on the GPU
Programmable Shaders make possible materials, lighting, reflections, shadows.
Evolution of Hardware Shading

- Hardware Rasterizers and perspective-correct texture mapping (RIVA 128)
- Single Pass Multitexture (TNT / TNT2)
- Register Combiners: a generalization of multitexture (GeForce 256)
- Per-pixel Shading (Geforce 2 GTS)
- Programmable Hardware Pixel Shading
Single Texture Programming Model

Texture Blender

Source

Result

Texture 0
Register Combiner Programming Model

Texture Combiner

Texture 0
Texture 1

Source(s)

Result(s)

Registers
Pixel Shading Pipeline

- Triangle Rasterizer
- 4 Pixel Shader Stages
- 8 Combiner Stages
- Specular / fog Combiner
- ROP & Frame buffer
Pixel Shaders

A pixel shader converts a set of texture coordinates \((s, t, r, q)\) into a color \((ARGB)\), using a shader program.

Pixel shaders use:

- Floating point math
- Texture lookups
- Results of previous pixel shaders
Simple Dependent Textures

The results of one shading program can be interpreted as the texture coordinates for a subsequent texture lookup.

- AR → \((s, t)\)
- GB → \((s, t)\)

Texture lookups become arbitrary functions.
Register Combiners / Texture Blending

- Strict superset of framebuffer alpha blending capabilities
  - \( a \times b + c \times d \)
- Register-based programming
  - All textures and colors available for each and every texture blending stage
  - 8 Stages
  - Signed color arithmetic
A “processor model” for Per-pixel Shading

• Computation primitives:
  • Texture addressing
  • Cube maps
  • Volume textures
  • Comparison & muxery
  • Register combiners
  • Vector math (dot3, reflection, etc)

• Hardware shading is now
  • Programmable
  • Extensible
Bumpy Shiny Patch

- The bumpy_shiny_patch demo illustrates three key new extensions working together
  - NV_evaluators
  - NV_vertex_program
  - NV_texture_shader
- The goal of bumpy_shiny_patch is to render a bumpy, mirrored, and deformable patch -- with an RGB glossmap to boot
The Bump Map and Gloss Map
The Environment Map
The Results
Future Graphics “Pipeline”

- Host interface
- Primitives
- Setup & rasterizer
- Tex-addr ops
- Texture blending
- Frame buffer anti-alias
- Per-primitive and per-vertex programs
- Shadows 3d tex
- Per-pixel shading

Smart memory interface
Acknowledgements

• Thanks to Mark Kilgard, Sim Dietrich, Matthew Papakipos, Simon Green, Cem Cebonoyan, Erik Lindholm, Doug Rogers, Cass Everitt, and other NVIDIA wizards for contributing slides, demos, and images.