

## A User-Programmable Vertex Engine

Erik Lindholm  
Mark Kilgard  
Henry Moreton

NVIDIA Corporation

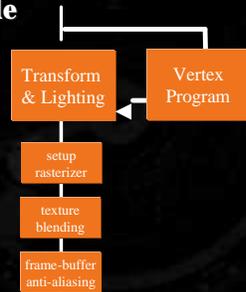


## Introduction

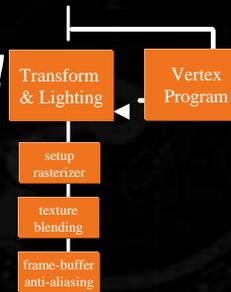
- Programmability vs. Configurability
- Programmability is trend
- Very efficient vertex engine
- Simple yet powerful programming model
- Supports fixed function pipeline
- Delivered in the GeForce3



## Pipeline Role



Same HW!



## Previous Work: Geometry Engine

- High bandwidth + lots of Flops
- Low clock rate
- No architectural continuity
- VERY hard to program
- Some high-level language support (maybe)
- A compromise solution (vtx,prim,pix,...)



## Alternative: The CPU

- Low bandwidth + reasonable Flops
- High clock rate
- Excellent architectural continuity
- VERY hard to use efficiently
- Excellent high-level language support
- Flexible, but often too slow



## New Design: The Vertex Engine

- Simple hardware for a commodity GPU
- Allows user to manipulate vertex transform
- Simple to use programming model
- Superset of fixed function mode



2001

## Why Vertex Processing?

- Very parallel
- Use single vertex programming model
- Hardware can batch or interleave
- KISS



2001

## Why Not Primitive Processing?

- Face culling and clipping break parallelism
- Complicates memory accesses
- Inefficient (control takes time)
- Let hardware designers optimize



2001

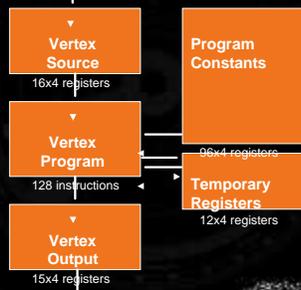
## Programming Model: Vertex I/O

- Streaming vertex architecture
- Source data converted to floats
- Source data loaded
- Run program
- Destination data drained
- Destination data re-formatted for hw



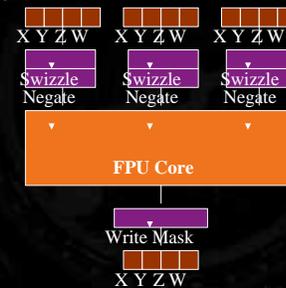
2001

## Programming Model



2001

## Data Path



2001

## Instruction Set: The Core Features

- Immediate access to sources
- Swizzle/negate on all sources
- Write mask on all destinations
- DP3, DP4 most common graphics ops
- Cross product is MUL+MAD with swizzling
- LIT instruction implements phong lighting



2001

## Cross Product Coding Example

```
# Cross product R2 = R0 x R1
```

```
MUL R2, R0.zxyw, R1.yzxw;  
MAD R2, R0.yzxw, R1.zxyw, -R2;
```



2001

## Sample OpenGL Vertex Program

```
static const GLubyte vpgm[] = "\!VP1. 0\  
DP4 o[HPOS].x, c[0], v[0]; \\  
DP4 o[HPOS].y, c[1], v[0]; \\  
DP4 o[HPOS].z, c[2], v[0]; \\  
DP4 o[HPOS].w, c[3], v[0]; \\  
MOV o[COLO],v[3]; \\  
END";
```



2001

## Instruction Set: The ops

- 17 instructions total
- ARL
- MOV, MUL, ADD, MAD, DST
- DP3, DP4
- MIN, MAX, SLT, SGE
- RCP, RSQ, LOG, EXP, LIT



2001

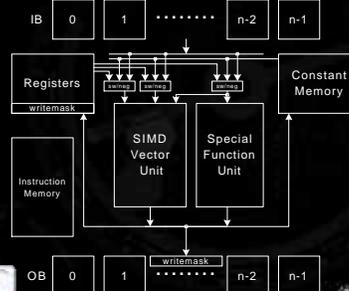
## Hardware Implementation

- Vector SIMD Unit + Special Function Unit
- Multithreaded and pipelined to hide latency
- Any one instruction/cycle
- All instructions equal latency
- Free swizzling/negate/write mask support



2001

## HW Block Diagram



2001

## API Support

- **Designed to fit into OpenGL and D3D API's**
- **Program mode vs. Fixed function mode**
- **Load and bind program**
- **Simple to add to old D3D and OpenGL programs**



## DEMOS



## Conclusion

- **Very simple, efficient implementation**
- **Allows vertex programming continuity**
- **Stanford Imagine Architecture**
- **A work in progress, lots more to come...**
- **We welcome your feedback**

