Proc. of 3rd KSV Conference
SNU, Seoul, Korea, Apr. 23, 2004

# Numerical Computing on Graphics Hardware

임인성

서강대학교 공과대학 컴퓨터학과

최근 일반 범용 PC 에 장착되고 있는 ATI 나 NVIDIA 등의 그래픽스 가속기의 성능은 수년전과 비교할 때 비교가 안 될 정도의 빠른 속도를 자랑하고 있다. 이러한 속도 향상과 함께 급격하게 일어나고 있는 변화 중의 하나는 바로 기존의 고정된 기능의 그래픽스 파이프라인(fixed-function graphics pipeline)과는 달리 프로그래머가 가속기의 기능을 자유자재로 프로그래밍할 수 있도록 해주는 프로그래밍이 가능한 파이프라인(programmable graphics pipeline)의 출현이라 할 수 있다. 이러한 가속기에 장착되고 있는 GPU (Graphics Processing Unit)는 간단한 형태의 SIMD 프로세서라 할 수 있는데, 특히 GPU 의 한 부분인 픽셀 쉐이더는 그 처리 속도가 매우 높기 때문에 이를 통하여 기존의 수치 알고리즘을 병렬화 하려는 시도가 활발히 일어나고 있다. 본 강연에서는 다양한 수치 계산을 그래픽스 가속기를 사용하여 해결하려는 시도에 대하여 간단히 살펴본다.

강연자 전자메일
ihm@sogang.ac.kr

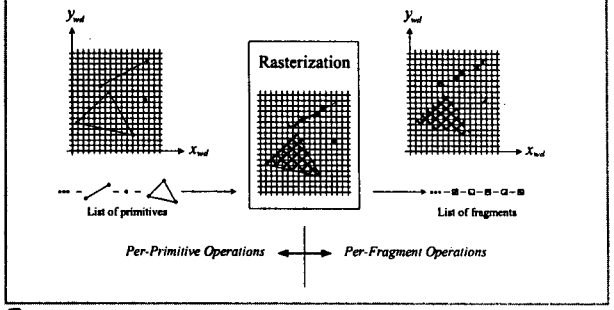
# Numerical Computing on Graphics Hardware

2004. 4. 23

서강대학교 공과대학 컴퓨터학과
임 인 성

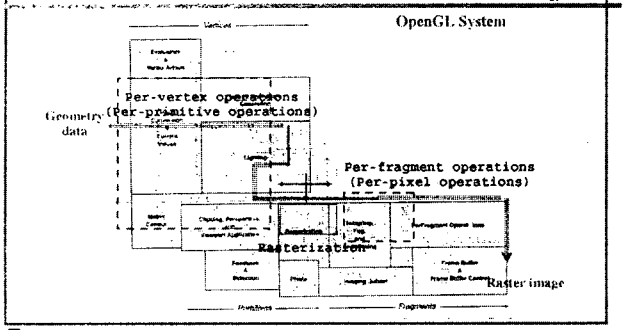


## Real-Time Rendering Pipeline on GPU



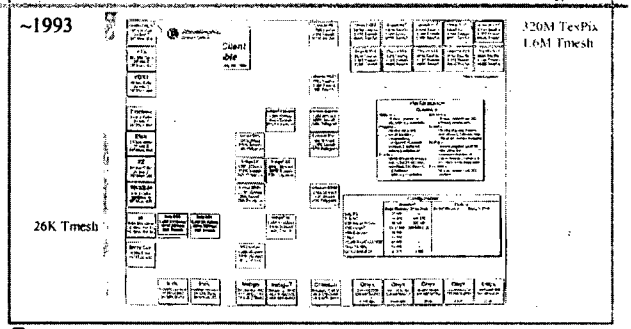



## Fixed-Function Graphics Pipeline






## Advances in Graphics Processors






2000 ~ 2001

| | Xbox | Sony PlayStation 2 | Nintendo Game Cube |
|---|---|---|---|
| Graphics Processor | 250 MHz Custom-designed chip by MS and NVIDIA | 147.456 MHz | 202.5 MHz Custom chip "Flipper" |
| Total memory | 64 MB | 32 MB | 43 MB |
| Memory Bandwidth | 6.4 GB/s | 3.2 GB/s | 3.2 GB/s |
| Polygon Performance | 125 M/s | 66 M/s | 6-12M /s |
| Simultaneous Textures | 4 | 1 | N/A |
| Pixel Fill Rate - No Texture | 4.0 G/s | 2.4 G/s | N/A |
| Pixel Fill Rate - 1 Texture | 4.0 G/s | 1.2 G/s | N/A |
| Pixel Fill Rate - 2 Texture | 4.0 G/s | 0.6 G/s | N/A |



2002 ~ 2003

| | ATI Radeon 9700 PRO | NVIDIA GeForce4 Ti4800 | NVIDIA GeForceFX 5800 | NVIDIA GeForceFX 5800 Ultra |
|---|---|---|---|---|
| Chip Technology | 256-bit | 256-bit | 256-bit | 256-bit |
| Transistors | ~107 Million | 63 Million | 125 Million | 125 Million |
| Memory Bus | 256-bit DDR | 128-bit DDR | 128-bit DDR2 | 128-bit DDR2 |
| Memory Bandwidth | 19.8GB/s | 10.4GB/s | 12.8GB/s | 16GB/s |
| Pixel Fill Rate | 2.6 Gigapixels/s | 1.24 Gigapixels/s | ~3.2Gigapixels/s | ~4 Gigapixels/s |
| Anti-Aliased Fill Rate | 15.6 Billion AA Samples/s | 4.8 Billion AA Samples/s | ~12.8 Billion AA Samples/s | ~16 Billion AA Samples/s |
| Triangle Transform Rate | 325M Triangles/s | 69M Triangles/s | 280M Triangles/s | 350M Triangles/s |
| Graphics Memory | 128/256MB | 128MB | 128/256MB | 128/256MB |
| GPU Clock | 325 MHz | 300 MHz | ~400 MHz | ~500 MHz |
| Memory Clock | 310 MHz (620 DDR) | 325 MHz (650 DDR) | 400 MHz (800 DDR2) | 500 MHz (1000 DDR2) |
| Textures per Texture Unit | 8 | 4 | 16 | 16 |

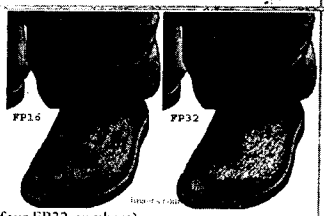
## Programmable GPU(Graphics Processing Unit)

- GPUs have become programmable!
  - NVIDIA GeForce FX and ATI Radeon 9700
- The recent GPUs support *4-wide full floating-point SIMD* operations.
  - 4-component vector in FP32 → 128bit processing
- In particular, fragment(pixel) shader hardware supports high pixel rates through its flexible texturing and blending units.



## Some Words on Latest Programmable GPUs

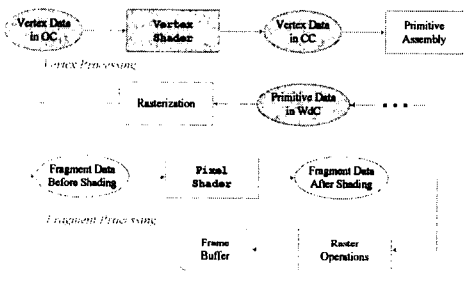
- Speed: faster than ever
- Graphics memory: larger than ever
- Bandwidth: wider than ever




- 4-wide SIMD processing
- 128-bit vertex processing pipeline (four FP32 numbers)
  128-bit pixel processing pipeline (four FP32 numbers)
- 4 parallel geometry pipelines
- 8 parallel pixel pipelines
- Programmability

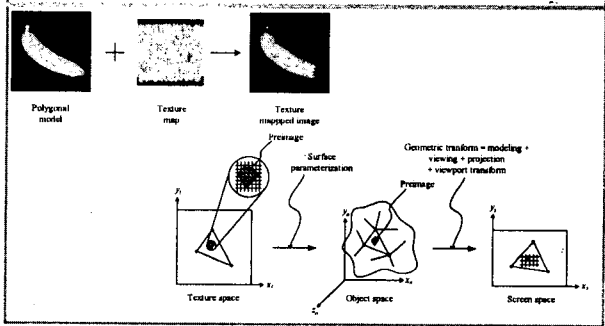


## (Current) Programmable Graphics Pipeline



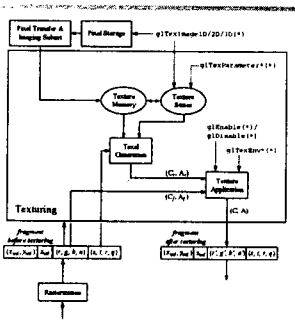



## Pixel Shader and Texture Mapping






## Conventional Texture Mapping in OpenGL






## Numerical Computing on GPU

- Parallelization of numerical algorithms on multi-processor architectures has been an active research area in applied numerical analysis.
- Many physically-based computer graphics (and CFD) applications require high-intensity numerical solvers.
- Various numerical algorithms are being actively mapped onto GPUs by graphics people.

## Recent GPU Mappings of Numerical Algorithm

- SIMD optimization of linear expressions for programmable GPU [Bajaj, Ihm, Oh, Min??]
- A sparse matrix conjugate gradient solver [Bolz03, Kruger03]
- A regular-grid multigrid solver [Bolz03]
- A Gauss-Seidel solver [Kruger03]
- A level set solver with curvature flow [Lefohn03]
- A dense matrix multiplier [Thompson02]
- A non-linear diffusion solver [Strzodka01]
- Etc.

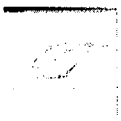
*Interactive real-time visual simulation of natural phenomena on commodity PCs!*



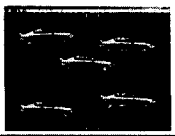
## Physical Simulation Examples

- Fluid animation

- Wave animation



- Vehicle simulation





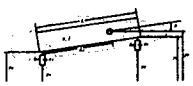
## Inside of the Simulations

- Fluid animation

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla\cdot(\nabla\mathbf{u}) - \frac{1}{\rho}\nabla p + \mathbf{f}$$

- Vehicle simulation



$$\begin{aligned} y_0' &= y_1 \\ y_1' &= \frac{1}{m}\{-(k_0+k_1)y_0 - (c_0+c_1)y_1 \\ &\quad + (k_1L_0 - k_1(L-L_0))y_2 + (c_1L_0 - c_1(L-L_0))y_3 + d_0\} \\ y_2' &= y_3 \\ y_3' &= \frac{1}{I}\{(L_0k_0 - (L-L_0)k_1)y_0 + (L_0c_0 - (L-L_0)c_1)y_1 \\ &\quad - (L_0^2k_0 + (L-L_0)^2k_1)y_2 - (L_0^2c_0 + (L-L_0)^2c_1)y_3 + d_1\} \end{aligned}$$

- Wave animation

$$\frac{\partial^2 z}{\partial t^2} = \alpha^2\left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2}\right)$$



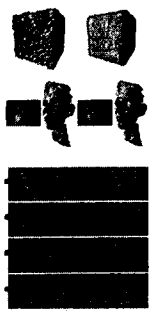
## A Pixel Shader Application

```
!!FP1.0
DECLARE Value;
#Frag00
TEX     R0, f[TEX0].xyxx, TEX2, RECT;
TEX     R1, f[TEX0].zyxx, TEX8, RECT;
TEX     R3, f[TEX1].xyxx, TEX0, RECT;
TEX     R3, f[TEX2].xyxx, TEX0, RECT;
TEX     R4, f[TEX3].xyxx, TEX0, RECT;
TEX     R5, f[TEX4].xyxx, TEX0, RECT;
TEX     R10, f[TEX5].xyxx, TEX10, RECT;
MULR    R10, Value.wwww, R10;
MADR    R10, Value.yyyy, R0.xzyw, R10;
MADR    R10, Value.yyyy, R1.xwyz, R10;
MADR    R10, Value.yyyy, R2, R10;
MADR    R10, Value.yyyy, R3, R10;
MADR    R10, Value.yyyy, R4, R10;
MADR    o[COLR], Value.yyyy, R5, R10;
END
```



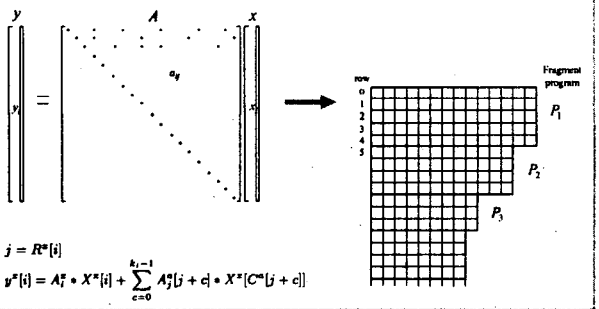
## Sparse Matrix Vector Multiplication [Bolz03]

- Computing $y = Ax$ on pixel shader
- Three features of GeForce FX pixel shader hardware
  - Random texture memory access fetch
  - Single fragment stream
  - SIMD execution model





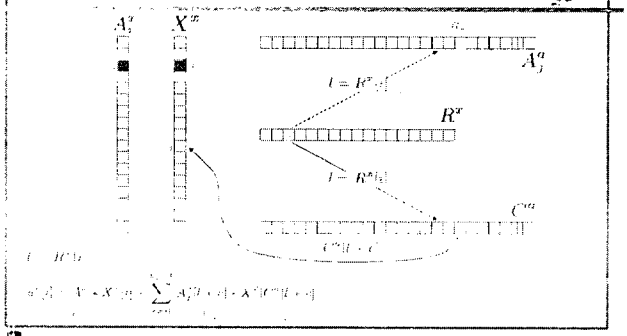
## Packed Representation for Sparse Matrix




$$j = R^a[i]$$

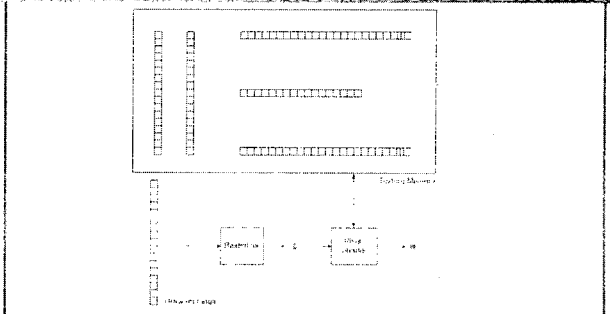
$$y^a[i] = A_j^a * X^a[i] + \sum_{c=0}^{k_i-1} A_j^a[j+c] * X^a[C^a[j+c]]$$

## Texture Layout and Indirection





## Mapping onto Pixel Shader





## SIMD Optimization of Linear Expressions [Bajaj, Ihm, Oh, Min??]

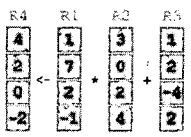
- Minimization of 4-wide SIMD instructions for computing $y = Ax + b$ through matrix transformation
- Abstract SIMD shader model
  1. The shader supports four-wide SIMD parallelism.
  2. Its instruction set includes the instructions shown in the table.
  3. Any component of the source registers may swizzle and/or replicate into any other component. Furthermore, destination registers may be masked. These register modifiers do not harm shader performance.
  4. Every instruction executes in a single clock cycle. Hence, the number of instructions in a vertex shader program is the major factor affecting shader performance.



## Instruction Set of the SIMD Model

| Operation | Usage | Description |
|---|---|---|
| ADD | ADD D, S0, S1 | D ← S0 + S1 |
| MUL | MUL D, S0, S1 | D ← S0 * S1 |
| MAD | MAD D, S0, S1, S2 | D ← S0 * S1 + S2 |
| DP4 | DP4 D, S0, S1 | D ← S0 · S1 |
| MOV | MOV D, S | D ← S |

```
MAD R4, R1, R2, R3;
```






## Partition of a Linear Expression

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{m-2} \\ a_{m-1} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} A_0^0 & A_0^1 & A_0^2 & A_0^3 & A_0^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ A_0^0 & A_1^1 & A_1^2 & A_1^3 & A_1^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ A_2^0 & A_2^1 & A_2^2 & A_2^3 & A_2^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ A_3^0 & A_3^1 & A_3^2 & A_3^3 & A_3^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ A_4^0 & A_4^1 & A_4^2 & A_4^3 & A_4^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ A_{m-2}^0 & A_{m-2}^1 & A_{m-2}^2 & A_{m-2}^3 & A_{m-2}^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ A_{m-1}^0 & A_{m-1}^1 & A_{m-1}^2 & A_{m-1}^3 & A_{m-1}^4 & \cdots & A_0^{m-2} & A_0^{m-2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} V^0 \\ V^1 \\ V^2 \\ V^3 \\ V^4 \\ \vdots \\ V^{m-2} \\ V^{m-1} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} A_0^m \\ A_1^m \\ A_2^m \\ A_3^m \\ A_4^m \\ \vdots \\ A_{m-2}^m \\ A_{m-1}^m \\ 0 \\ 0 \end{bmatrix}$$



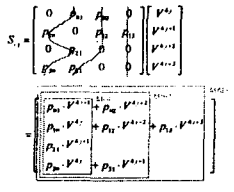
## Decomposition

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \end{bmatrix} = M_{00} \cdot \begin{bmatrix} V^0 \\ V^1 \\ V^2 \\ V^3 \\ \vdots \end{bmatrix} + M_{01} \cdot \begin{bmatrix} V^4 \\ \vdots \\ V^7 \\ \vdots \end{bmatrix} + \cdots + M_{0\,n-1} \cdot \begin{bmatrix} V^{m-2} \\ V^{m-1} \\ 0 \\ 0 \\ \vdots \end{bmatrix} + \begin{bmatrix} A_0^m \\ A_1^m \\ A_2^m \\ A_3^m \\ \vdots \end{bmatrix}$$

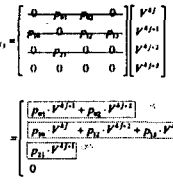
$$\begin{bmatrix} a_{m-2} \\ a_{m-1} \\ 0 \\ 0 \end{bmatrix} = M_{n-1\,0} \cdot \begin{bmatrix} V^0 \\ V^1 \\ V^2 \\ V^3 \end{bmatrix} + M_{n-1\,1} \cdot \begin{bmatrix} V^4 \\ \vdots \\ V^7 \end{bmatrix} + \cdots + M_{n-1\,n-1} \cdot \begin{bmatrix} V^{m-2} \\ V^{m-1} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} A_{m-2}^m \\ A_{m-1}^m \\ 0 \\ 0 \end{bmatrix}$$

## Evaluation Cost

- C-type



- R-type



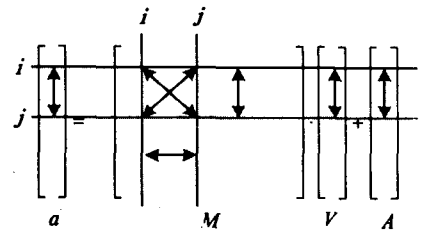




$$C_{i,j} = \min(k_1, k_2) \qquad \beta_j = n(R) - 1 \longrightarrow C(M) = \sum_{i,j=0}^{n-1} C_{i,j} + \sum_{i=0}^{n-1} \beta_i$$

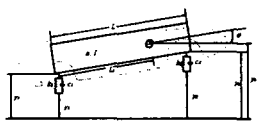


## Matrix Transformation





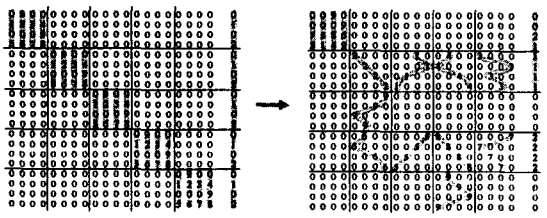

## Computation of 4th Order RK on Shader



$$\begin{aligned}
y_0' &= y_1 \\
y_1' &= \frac{1}{m}\{-(k_0+k_1)y_0 - (c_0+c_1)y_1 \\
&+ (k_0L_0 - k_1(L-L_0))y_2 + (c_0L_0 - c_1(L-L_0))y_3 + d_0\} \\
y_2' &= y_3 \\
y_3' &= \frac{1}{I}\{((L_0k_0 - (L-L_0)k_1)y_0 + (L_0c_0 - (L-L_0)c_1)y_1 \\
&- (L_0^2k_0 + (L-L_0)^2k_1)y_2 - (L_0^2c_0 + (L-L_0)^2c_1)y_3 + d_1\}
\end{aligned}$$

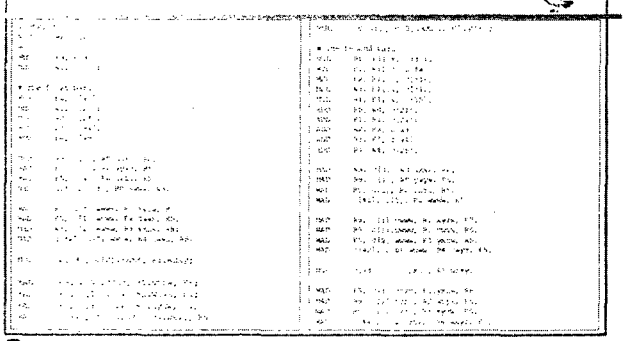


## Before and After Transformation

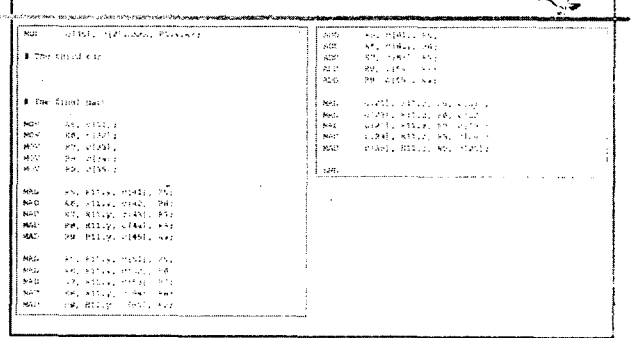




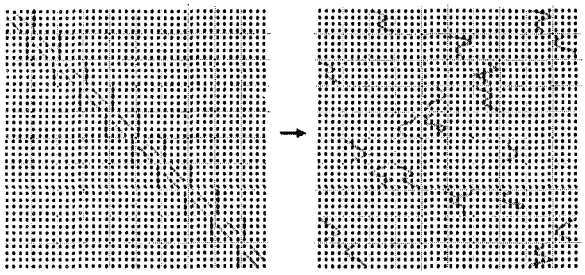
## Vertex State Program for 4th Order RK







## Solving 260,000 X 260,000 Poisson Equation on GPU



서강대학교 컴퓨터 그래픽스 연구실

## Timing Performance

| | | tolerance | | |
|---|---|---|---|---|
| | | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ |
| GS on CPU | time | 101.6 | 364.8 | 1178.6 |
| | iter | 16 | 61 | 200 |
| EBJ on GPU | time | 62.8 | 142.4 | 419.1 |
| | iter | 20 | 84 | 291 |
| EBJ on GPU (optimized) | time | 49.1 (9.8) | 111.6 (68.3) | 315.9 (272.6) |
| | iter | 19 | 77 | 253 |

서강대학교 컴퓨터 그래픽스 연구실