GPU – Graphics Processing Unit

- Mainly based on
  - Rise of the Graphics Processor, by David Blythe, Proceedings of the IEEE, Vol. 96, No. 5, May 2008
  - part of the course reading list
- Some additional references at the end of the presentation

Contents

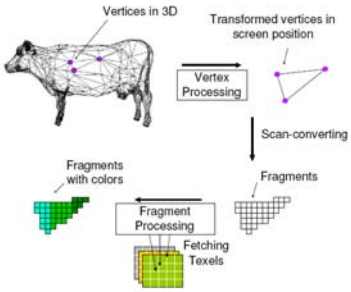
- Introduction
- Evolution, history
- Modern GPU, design ideas
  - Parallelism
  - Exploiting "coherence"
  - Hiding memory latency
  - Programmability
  - Scalability
- New applications
- Example (case study)
- Future, convergence?

Introduction

- approx. 40 years history
  - flight simulators at research labs
  - computer aided design (CAD), expensive workstations (PC)
  - medical devices
  - computer games at PC and consoles
  - cell phones
- Now; GPU more complex than CPU
  - potentially more powerful
  - more difficult to program
  - GPGPU
    - General Purpose programming on GPU

Background

- From model to pixel (Example from [Fan+04])



Intro., graphics pipeline

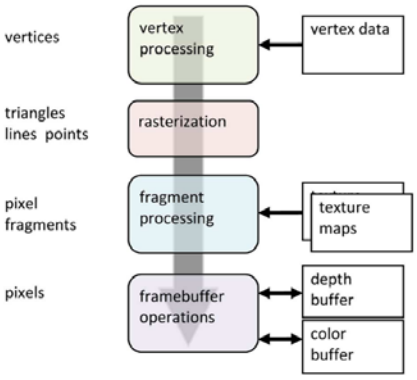


Fig. 2. Three-dimensional processing pipeline.

Stage 1: Vertex processing

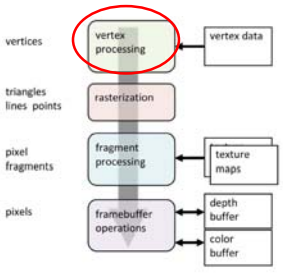


Fig. 2. Three-dimensional processing pipeline.

- (Geometry processing)
- Transform a 3D triangle representation to a 2D projection (screen position)
- Transformations
  - translation, rotation etc.
- Only verticeses are projected
  - surface of triangle is planar
  - projected vertices reconnected by straight lines

Stage 2: Rasterization

- (Scan converting)
- Converts each resulting 2D triangle to pixel fragments
  - discretized to uniform grid (raster)
- Additional parameters (eg. color) can be interpolated at each sample point from the vertice-values

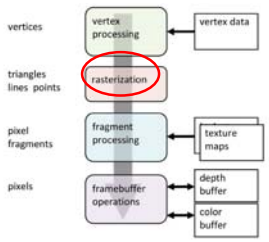


Fig. 2. Three-dimensional processing pipeline.

Stage 3: Fragment processing

- (Shading)
- The final color value of each pixel (fragment) is computed
  - using interpolated value
  - or more complex calculation
  - texture mapping
  - several texture maps (images) may be combined

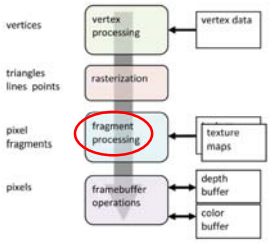


Fig. 2. Three-dimensional processing pipeline.

Stage 4: Framebuffer operations

- The framebuffer is mapped to the screen
- One fragment is stored for each pixel
- Depth buffering
  - resolve fragment visibility
  - what is closest to the viewer?

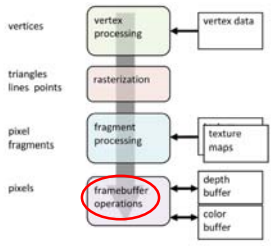


Fig. 2. Three-dimensional processing pipeline.

Evolution of graphics HW

- 1960s - origins
  - Early applications
    - CAD, flight simulation, computer games, content creation for film
  - Analog vector display (like an oscilloscope)
    - Wire-frame repr. of objects
    - Periodic refresh
  - Algorithms for
    - projecting a 3D-object on a 2D-plane
    - hidden line/surface removal
  - First graphics terminals
    - autonomous display processing
    - HW-accelerated line drawing

IDIIOM, an interactive graphics terminal, at our department in 1972.

1970s - uncovering fundamentals

- Semiconductor memory ➔ raster techniques
  - image as a matrix of pixels (picture elements)
- Television like display
- Limited by cost of semiconductor memory
  - no of pixels (space)
  - color resolution
- New image synthesis algorithms
  - texture mapping
  - reflection (light)
  - bump mapping
  - illumination models
- Dedicated acceleration hardware
- Unification of text and graphics processing
  - XEROX PARC, Alto (image)

From http://en.wikipedia.org/wiki/Xerox_Alto

1970s cnt'd

- Early text displays stored text as text data and not as pixels in the bitmap; they converted from text to a raster image on-the-fly as part of the display process
  - "Alphanumeric controllers"
  - Fast but restricted, only one or a few fonts
- Intermixing of text and graphics
  - Windows
- Interactive processing
  - 60 images pr. second
- Offline processing (noninteractive)

GPU evolution pattern

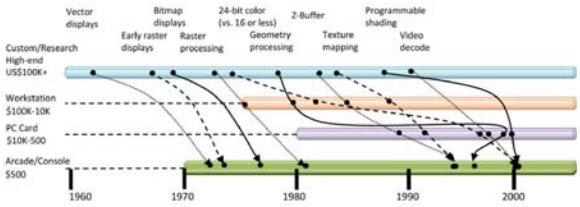
- "… early use of fixed-function hardware acceleration to produce a cost-effective solution, followed by replacement with a more flexible interface to broaden applicability and improve quality. To avoid an unmanageable explosion of operating "modes", the interface frequently evolves to a programmable implementation." (Page 763)

1980s – Hardware acceleration

- Operation with commonality, simplicity and frequency ➔ HW acceleration
  - Bit-block transfer (BITBLT)
    - copy rectangular area from source to destination
      - combined with logical op (AND, OR, XOR etc...)
- Add-in 2D raster graphics cards
- Matrix, vector operations, four-element floating point vectors
  - HW acceleration in dedicated logic
- New applications
  - Scientific and medical visualization
- Arcade and home-entertainment based on raster graphics

PERQ, computer science dept. 1982

Graphics accelerator evolution



- *"… recurring pattern in the evolution of graphics hardware, where technology improvements allow low-cost hardware to meet the performance and quality requirements of formerly high-end applications, moving the market to lower cost platforms*"

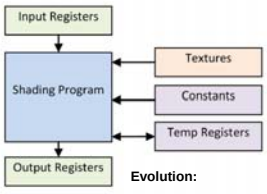
1990s – standardization, consolidation

- Dedicated game consoles moved from traditional 2D raster pipelines to crude texture-mapping 3D pipelines
  - Playstation, Nintendo64
- 3D acceleration add-in cards for PC
- Standardization of processing pipeline to allow portable applications
  - Open GL API
  - Direct 3D API
- Internet and WWW ➔ increased color and spatial resolution for low-cost PC systems

2000s - Programmability

- Cost reductions in HW acceleration
- PC-add-in accelerators
  - NVIDIA GeForce
  - ATI Radeon
- Game consoles using the same technology
- New term: GPU = graphics processing unit
- Shading: increased complexity
  - multipass
  - demand for flexibility
  - application-developer-accessible programmability
  - ➔ shader programming
    - A small custom program (kernel, subroutine) invoked on each vertex and another on each pixel
  - Need for portable applications ➔machine independent shader representation

Shading program, "shader"



Evolution:

• increase in range and precision of datatypes, longer shading programs, dynamic flow control, additional resources (larger no of textures)

•shading languages

•HLSL, Cg, GLSL, ...

Modern GPU

- PC add-in card
- Same techn. scaled down in mobile phones etc.
  - IDI & Electronic students ➔ Falanx ➔ ARM
- 3D graphics pipeline
  - similar structure in 20 years
  - recent change to ”unified shader architecture”

The Falanx story (ARM Trondheim)

Traditional GPU

Vertex processors and fragment processors

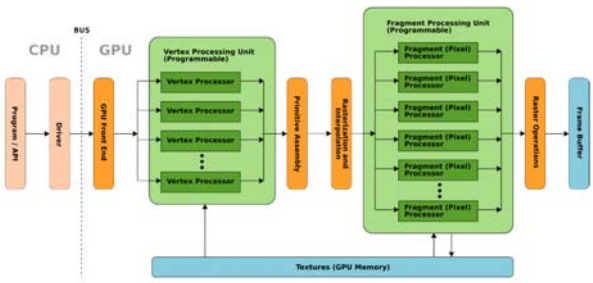
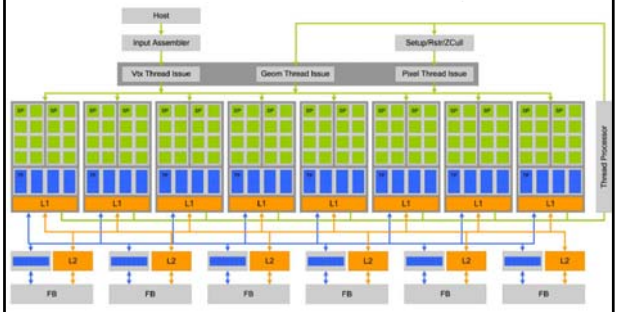


Figure 2.1: Programmable pipeline

The Modern GPU

"Unified processor/ Unified shader"



Exploiting parallelism

- primitives, vertices, pixel fragments and pixels are largely independent
  - ➔ (data) parallelism
    - example: scene = 1 million triangles with 25 pixel fragments per triangle
- different operations concurrently executed in different pipeline stages
  - task parallelism
- constraint: the net result must be the same as one where primitives are rendered in the order they are submitted
  - order-dependent algorithms

Exploiting coherence

- SIMD
  - Achieving maximum efficiency requires processing *n* entities with the identical instruction stream.
- ➔ maximum efficiency is achieved by executing groups of vertices or groups of pixel fragments that have the identical shader program
  - grouping of similar/equal/like processing called “computational coherence”
    - more challenging with conditional flow control
  - techniques
    - smart data structures/layout
    - rearranging pixel fragments to increase coherency

Hiding memory latency

- memory stall (latency typically 100 cycles)
- GPUs have small caches
- hiding by multithreading/hyperthreading

Programmability & scalability

- Trend towards increased programmability
- Sigificant parts still in fixed-function HW
  - texture filtering
  - rasterization
- Scalability
  - large degree of parallelism makes it easy to scale hardware implementation to different sizes
    - high-end (enthusiast)
    - low-end (entry)
    - ...
  - scalable application programming
    - change richness of content, varying resolution

New applications

- GPGPU
  - organize data in *n x m* domains (pixels)
  - operation on each domain point is done by a "pixel shading program"
  - some data parallel operations require a multipass implementation
  - other data types and irregular data structures can be cumbersome to implement
  - parallel languages
    - CTM, CUDA, PeakStream, ...
- New applications
- Image and signal-processing on 1-, 2- or 3-D data
- Linear algebra
- Engineering analysis, physical simulation
- Database management
- Molecular biology etc.
- **Programmer productivity will continue to be the limiting factor!**

Example

Figure 11: The simulation area shown on the Manhattan map, enclosed by the blue contour. This area extends North from 38th Street to 59th Street, and East from the 8th Avenue to Park Avenue. It covers an area of about 1.66 km × 1.13 km, consisting of 91 blocks and roughly 850 buildings.

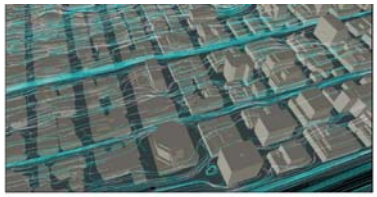
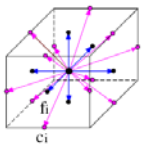
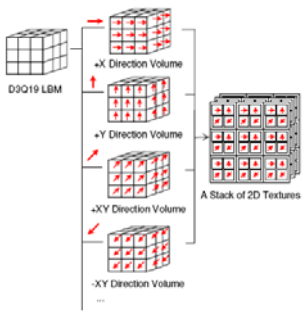


Figure 12: A snapshot of the simulation of air flow in the Times Square area of New York City at time step 1000, visualized by streamlines. The blue color indicates that the direction of velocity is approximately horizontal, while the white color indicates a vertical component in the velocity as the flow passes over buildings. Red points indicate streamline origins. Simulation lattice size is 480 × 400 × 80. (Only a portion of the simulation volume is shown in this image.)

Example: datastructuring for GPU





Future, hybrid systems, convergence

- GPUs
  - unified processor architecture
  - increased programmability
- CPUs
  - multicore/manycore
  - many with vector units
- Hybrids
  - integrate existing CPU and GPU designs on a single die
  - Intel Larrabee ... → convergence"

References

- GPGPU.org
- [Fan+04] = GPU Cluster for High Performance Computing, …,
  DOI Bookmark http://dx.doi.org/10.1109/SC.2004.26
- GPU-activity at IDI, mainly:
  http://www.idi.ntnu.no/~elster/hpc-lab/