Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

System Issues

Outline

- Introduce key issues
- A little history
- High-performance application interface
  - More history
- Virtualizing the graphics hardware
  - Windows and window systems
- Reliability

Required reading

- *Display Procedures*, Newman, CACM Oct 1971.
- *On the Design of Display Processors*, Myers and Sutherland, CACM 1968 (Wheel of Reincarnation paper.)

What is a "graphics system"?

GPU?

Graphics board?

Graphics system is

- Graphics hardware (GPU, board, …)
- System software that makes it work
  - Microcode
  - Driver
  - Window system
  - OS extensions and tuning

Graphics API (e.g. OpenGL, X) is the boundary of the graphics system

Key Issues

Performance in an application-accessible way

- Satisfy a wide range of application needs
- Not tuned for the "canonical demo"

Virtualization of the graphics system

- Two mechanisms are employed
  - Graphics context: GPU state, textures, …
  - Window: framebuffer, display resources
- OS and window system together implement virtualization

A Caveat

My experience is with traditional workstations

- Unix
- X Window System, OpenGL
- Vendors were systems companies

I have little experience with PCs

- Microsoft Windows OS
- Direct X, OpenGL
- Vendors typically aren't system companies
  - Apple is a significant exception
  - Others moving in this direction

Were workstations over-engineered?

- Do PCs get by with less system engineering?

SGI = Silicon Graphics Inc.

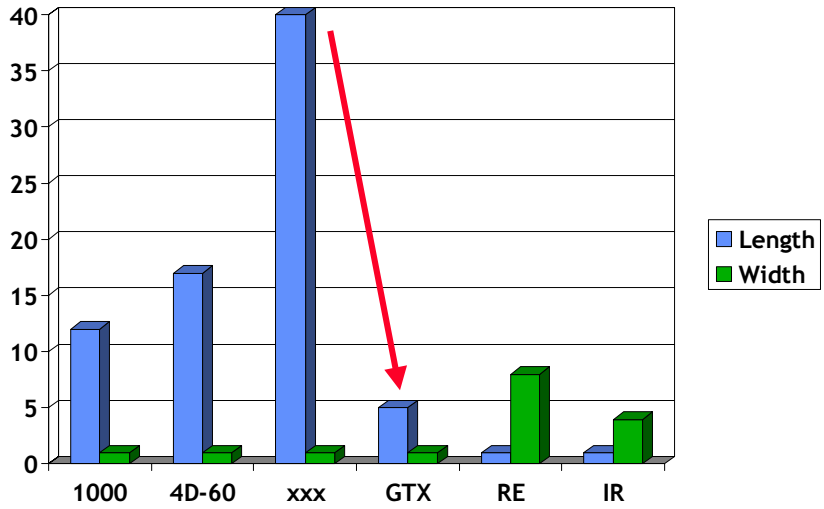
Early 80's

- SGI's development cycle was backward
  - Chips, then boards, then API, then applications
- SGI's "silicon" expertise was limited
  - Mead Conway approach was for dilettantes

Clark Geometry Engine

- Could not support preemptive multitasking
- Had low performance by mid-80's standards

A Change in Course



SGI = *Systems* Graphics Inc.

"Taking the silicon out of Silicon Graphics"

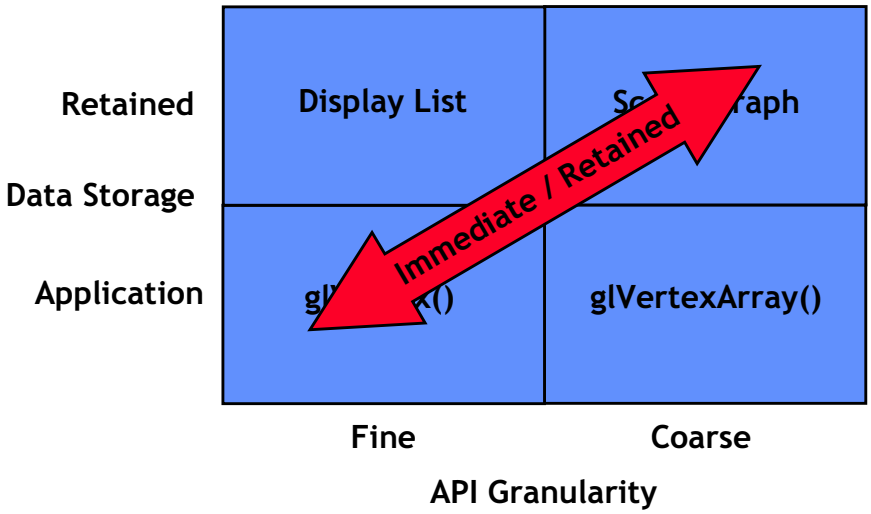
- Emphasize system integration of graphics
- De-emphasize custom IC development
  - Utilize commodity ALUs
  - Reduce device design schedule (Gate Arrays)
  - Re-order tasks → API, then system, then devices

Great strategy for the late 80's and early 90's

- Did it set up eventual failure?

Performance

Interface Choices



Immediate Mode

Advantages

- Good impedance match to application
  - Application chooses data format and arrangement
  - Application defines data traversal (Proceduralism)
- Minimized transfer of data
  - Modal interface
  - Small data types

Performance Issues

- Many subroutine calls
- Small data packets
- Complex and unpredictable input sequence

Retained Mode

Advantages

- Optimized traversal (not application limited)
- Command sequence is regular and/or predictable
  - Large arrays, or
  - Precompiled display lists
- Large atoms mean
  - Few subroutine calls
  - Large blocks of data

Issues

- Specific traversal
- Application must conform to graphics API
- Excess data may be transferred (e.g. facet info)

Trend

Moving from fine grain immediate to coarse grain

- PCs use indexed vertex arrays in AGP memory

Why was fine-grain interface used?

- Read *Display Procedures*, Newman
- Low geometric complexity emphasized efficiency of modal interface
- Interface matched natural quanta at the time
  - Objects have much more complexity now

Interface History

IRIS 1000 terminal

IRIS 2000 workstation

4D-60 MIPS-based workstation

GTX MP workstation

RealityEngine Challenge-MP workstation

InfiniteReality Origin-ccNUMA system

Modern PC

IRIS 1000

Terminal

- Connected to VAX via Ethernet
- Ran custom OS on modified SUN 68000 CPU board

Iris GL display lists critical

- Pre-convert floating point to custom GE format
- Transfer data over Ethernet only once

Great demos

- We ran applications in immediate mode on terminal
- Customers couldn't do this → big mess!

IRIS 2000

Direct mapped hardware

- Command-in-address (ala SUN graphics card)
- "Geometry Accelerator" (floating point convert)
- Query-based flow control

No pre-emptive context switching

- Clark GE could not be interrupted
- Unix OS had to query repeatedly ;-)

4D-60

"Magic FIFO"

- Enabled pre-emptive context switch with Clark GE
- Direct mapped
- Designed as a portable plug-in

Flow-control

- No software queries
- Hardware interrupts process when FIFO near full
- Spin loop waits for FIFO low-water mark
  - Typically no wait is needed

GTX

Introduced "vector" commands

- V4f(float* v);
- Became glVertex4f(glFloat* v); in OpenGL
- Implemented with 3-way transfer (next slide)

Flow control

- MP system routes interrupt to "gfx" CPU

Pinned memory pull model

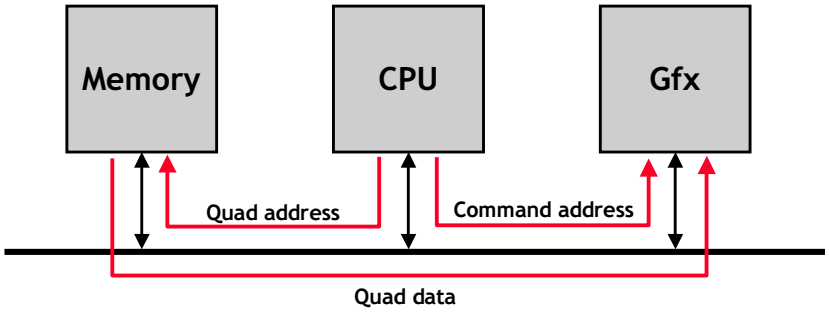
- Pixel transfers only

GTX 3-Way Transfer

Use CPU to convert virtual to physical address

Avoid CPU cache, dual data transfer

What about page boundaries?



RealityEngine

Challenge multi-processor system

- Packet bus quanta is 128-byte cache line
- 3-way transfer not possible
- Bus interface chips
  - Queue graphics commands in special buffers
  - Convert address bits to command token data
  - Transfer buffers automatically when full

CPU process migration

- Complicates interrupt flow control
- Requires flushing of hardware buffers

InfiniteReality

ccNUMA distributed memory system

- Cannot support command-in-address
  - Command tokens generated under software control
- Cache-line size buffers remain

Display list optimizations

- DMA pull model
  - Supports nested display lists
  - Requires pinned memory
- On-board display list storage
  - Doubled available bandwidth
  - Stores only leaf-node display lists

Modern PC

No direct map

- API calls write to buffers
- Buffers pulled by hardware DMA front-end

Vertex data in indexed arrays

- AGP memory
- Efficient pull of data
- Cache eliminates redundant vertex transfers

System Programming

Callee-save compiler

- Identify leaf-node routines (e.g. glVertex)
- Do not save or restore registers
- (Broken by n32 compiler!)

Overload branch tables

- Must have for shared library / DSL
- Utilize branch table to implement
  - Display list mode
  - Diagnostic mode
  - Feedback mode
  - …
- Creates leaf-node routines!

System Programming (cont.)

Assembly language

- Jump-table tricks
- Local optimization
- Utilize sparingly

Base register pointer

- Allocate a CPU register permanently

Virtualization

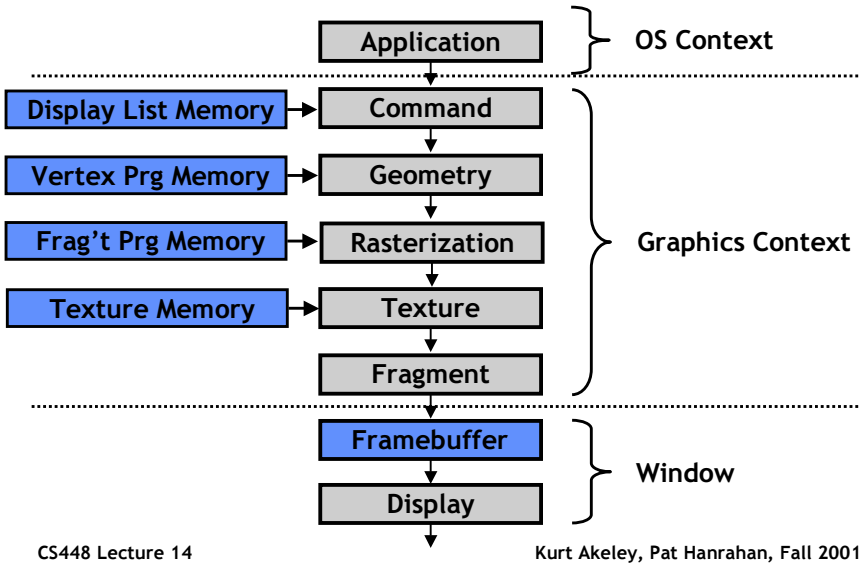
Virtualization

One graphics system appears to be many

Clients

- Application processes
- Viewers

Virtualization domains



Why Separate Domains?

Rendering hardware can be time shared

- Like CPU hardware
- But separate management to allow multiple graphics contexts per CPU context

Display hardware is "real time"

- Cannot be time shared
- Must be managed differently

Framebuffer

- Is display related by definition (pixel allocated)
- Same physical memory treated differently
  - Framebuffer
  - Texture memory, display lists, …

Graphics Context Switch

Hardware requirements

- Can take interrupt at any time
- All state can be queried and restored

Architecture limitations

- Avoid huge state build-up
  - E.g. Tile bins, scan line rendering, …
- Avoid long uninterruptible operations
  - E.g. Patch rendering

Speed of switch is important

- For the window system rendering
- For other applications

Graphics Context Switch (cont.)

Decouple from application context switch

- Some applications don't do graphics
- Others may utilize multiple graphics contexts
- Demand swap is best
  - Direct map - use CPU page table to set trap
  - Buffered - incorporate into driver logic

Memory optimization

- Avoid transfer if possible
  - Share until full
- Overlay to minimize data transfer

Graphics Context Switch (cont.)

Hardware expense is minimal

Primary issues are

- Architectural limitations
- Implementation complexity
- Testing complexity
  - Lots of cases
  - Especially if both light (GPU storage) and heavy (CPU storage) switches are supported

OS Tuning

Manage physical memory for graphics contexts

Implement graphics context swap on demand

Tune application context swap to

- Avoid thrashing the graphics hardware
- Smoothly support multiple active graphics applications

What is a Window?

Allocation of visible pixels on a display

- Whole pixels only
  - Leads to OpenGL's 1x1 notion of a pixel
  - Pixel is area, not sample (frustum, AA, etc.)
- Format
  - Color (RGB, RGBA, index; single/double buffer; …)
  - Depth, stencil
  - Multisample
- Necessary display hardware
  - Lookup table for color indexes (In X, all formats)
  - Gamma correction

Window Overlap

Architectural choice

- Provide "backing store" for obscured pixels
- Do not, and issue expose events as needed

Backing store options

- All except color buffers
  - Typically front buffer only (copy swap)
  - Context switch non-visible buffers (depth, stencil, …)
  - Convenient to disallow front-buffer rendering ☺
- All buffers
  - Requires scatter-gather display hardware
  - This gets re-invented repeatedly

Window Overlap (cont.)

Backing store issues

- Can require lots of memory
  - E.g. 100x or 1000x
- Consumes rendering time for invisible pixels
  - Expensive way to speed window dragging
  - Better to put the resources into faster rendering!
- Scatter-gather display is difficult to implement
  - Imaging 1024 1-pixel-wide window slivers
  - Window managers not tolerant of arbitrary constraints

No backing store is typical workstation choice.

- PCs?

Window IDs

Small integer field in each pixel

Controls display fetch

- Color format (RGB, RGBA, index, …)
- Buffer selection (front, back, stereo, …)
- Overlay and underlay

Window clipping

- Test Window ID, or (better)
- Test rectangle list or separate Clip ID (1-bit)

Cadillac algorithm

- Test rectangle list or Clip ID
- Update Window ID during render only

Color Table Virtualization

All used mapping must be present – can't time share

Assign table entries

- In contiguous blocks (OpenGL requirement)
- One at a time (Windows 2D rendering)
  - Can't interpolate easily!

X Window System gotcha:

- Rendering and table change commands are ordered!
- Allows table-driven double buffering
- Greatly complicates implementations

Do as rendering operation

- Virtualizes nicely
- Works like multisample resolve

Reliability

Application

An application must not be able to

- Crash the hardware
  - By issuing undefined or illegal commands
  - By providing invalid data:

    IEEE Nan, Denormalized, negative zero, etc.
- Modify or access
  - State owned by another graphics context
  - Pixel or display data owned by another window

These high standards are strived for, but not often met!

Memory

Graphics memory should be reliable and secure

- Errors corrected
- Or at least detected

Fact

- No SGI graphics system implements framebuffer error detection or correction.
- Not even O2, which shared a single memory for CPU and framebuffer
  - Each page was allocated as CPU or graphics memory
  - Graphics memory supported byte writes, no ECC

Current GPUs?

Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Notes

Virtual graphics process = context

Physical graphics process = GPU

Virtual framebuffer = Window

Physical framebuffer = memory