Beyond Programmable Shading Retrospective

Mike Houston

AMD

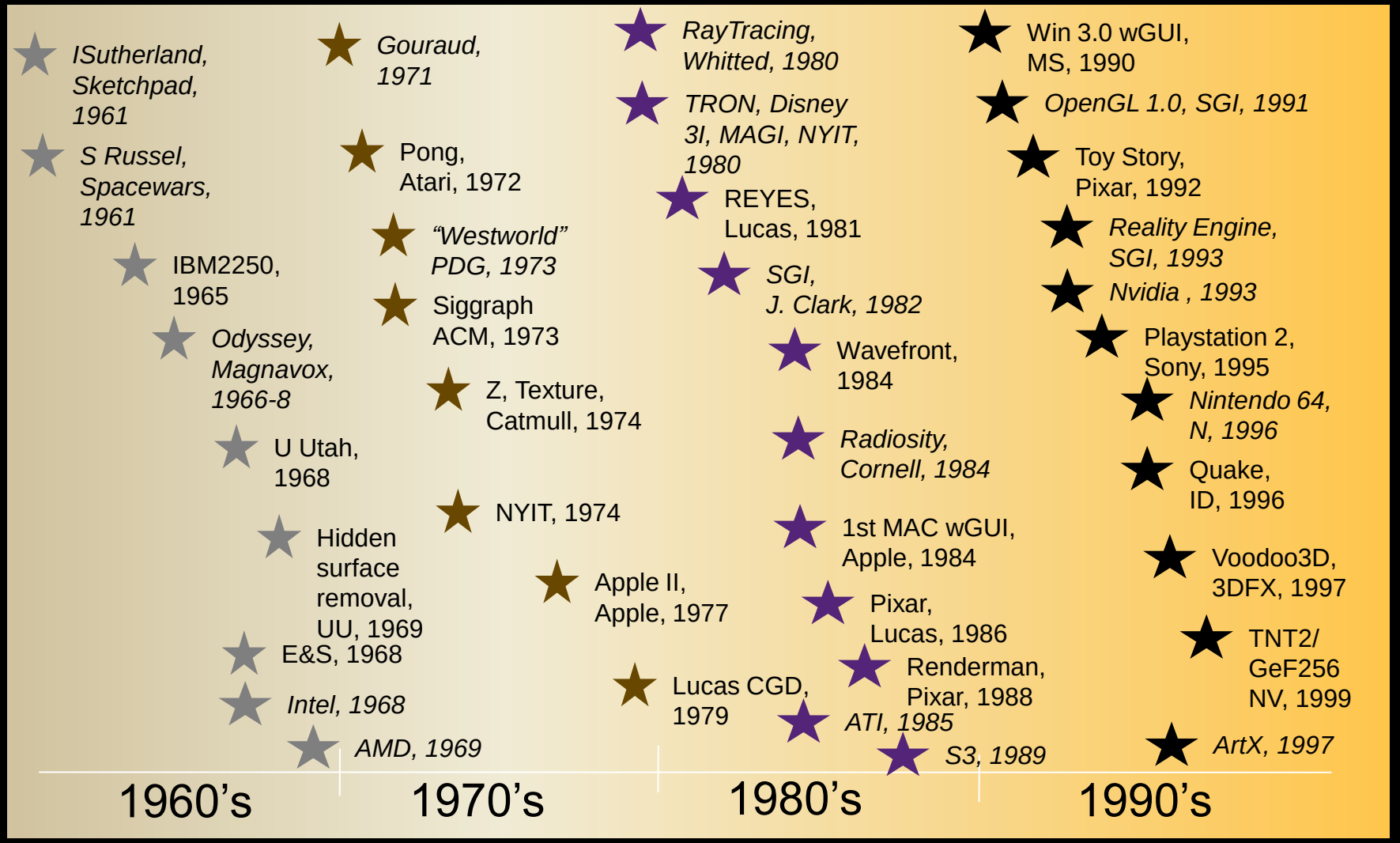
Overview

- Introduction / overview of past, current, and future directions in programming for interactive graphics
- Outline
  - Evolution of interactive rendering programming
  - Parallel programming for GPUs: beyond graphics shaders
  - Wrap-up: Research and industry directions

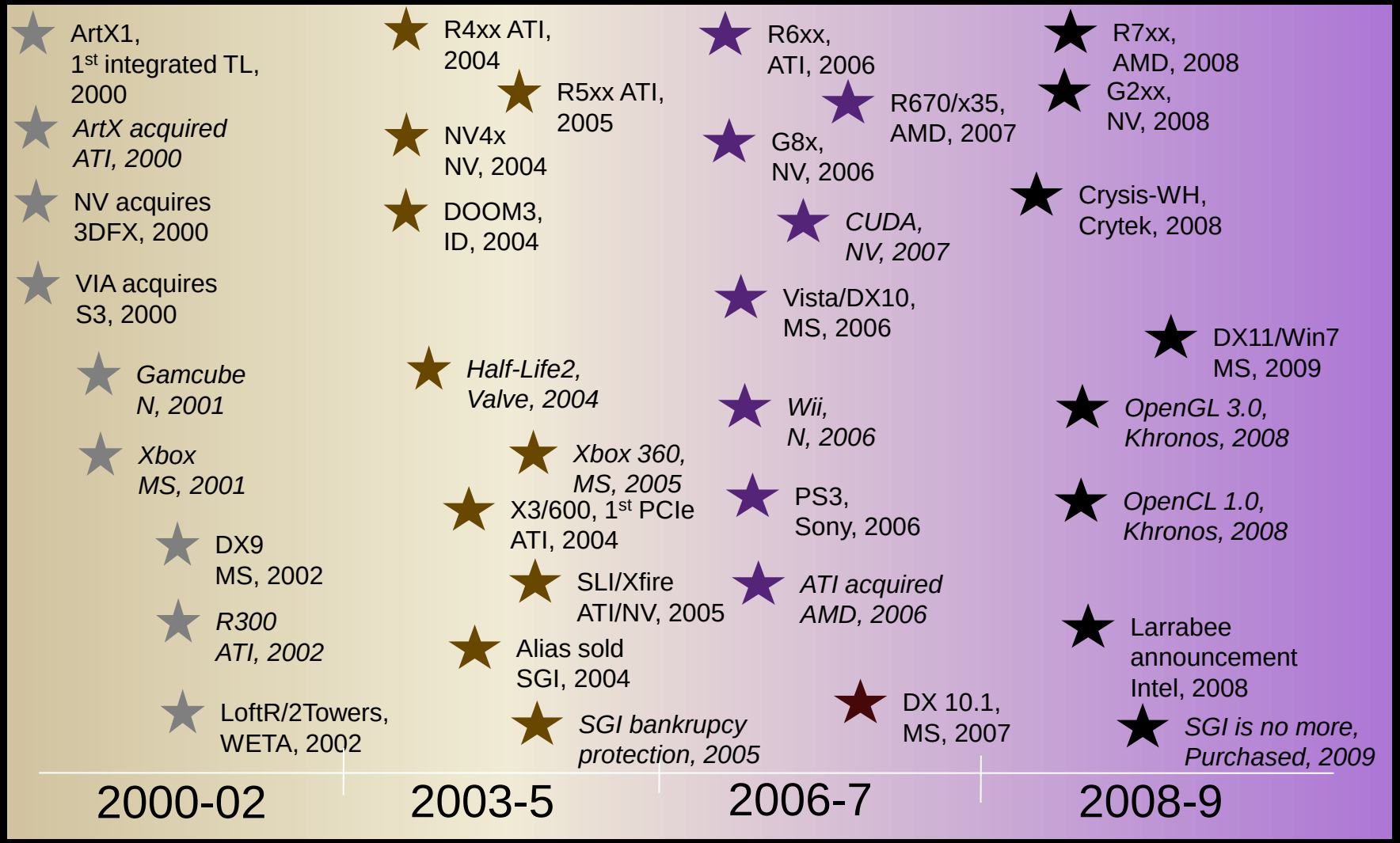
A Brief History of Today's Graphics Pipeline

A quick history 1960's to 2000s

Good reference: http://hem.passagen.se/des/hocg/hocg_1960.htm



A quick history since 2000



Early Software Renderers: Fixed Function

- 1970s through mid-1980s, most graphics pipelines were fixed-function
- Must change core rendering code to change geometry/lighting/surface model

Mid'80s – Early 90s: Programmable Shading

- Most of graphics pipeline is fixed-function, highly optimized architecture, but allow users to customize small portions of pipeline with simple language
  - Shade Trees, Cook 1984
  - An Image Synthesizer, Perlin 1985
  - Renderman Shading Language, Hanrahan/Lawson 1990

Cook Approach

Shader:

- Basic operations: dot products, norms, etc.
- Operations organized into trees

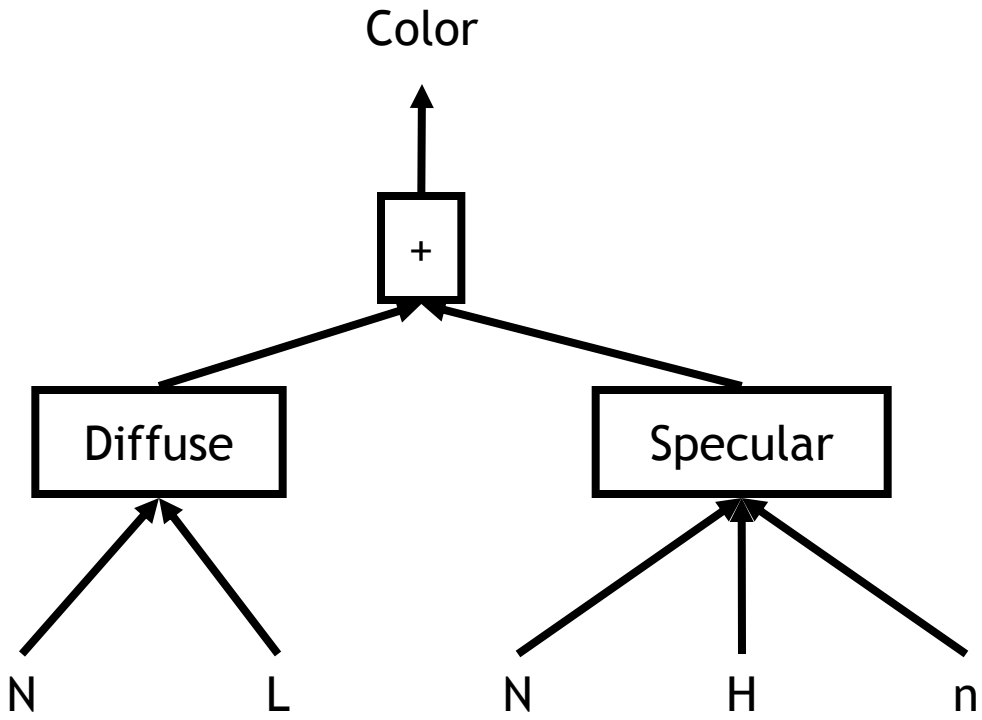
Separated light source specification, surface reflectance, and atmospheric effects

Multiple trees: shade trees, light trees, atmosphere trees, displacement maps

Simple language

Shade Trees

Phong shading model:



Cook: Arbitrary Trees

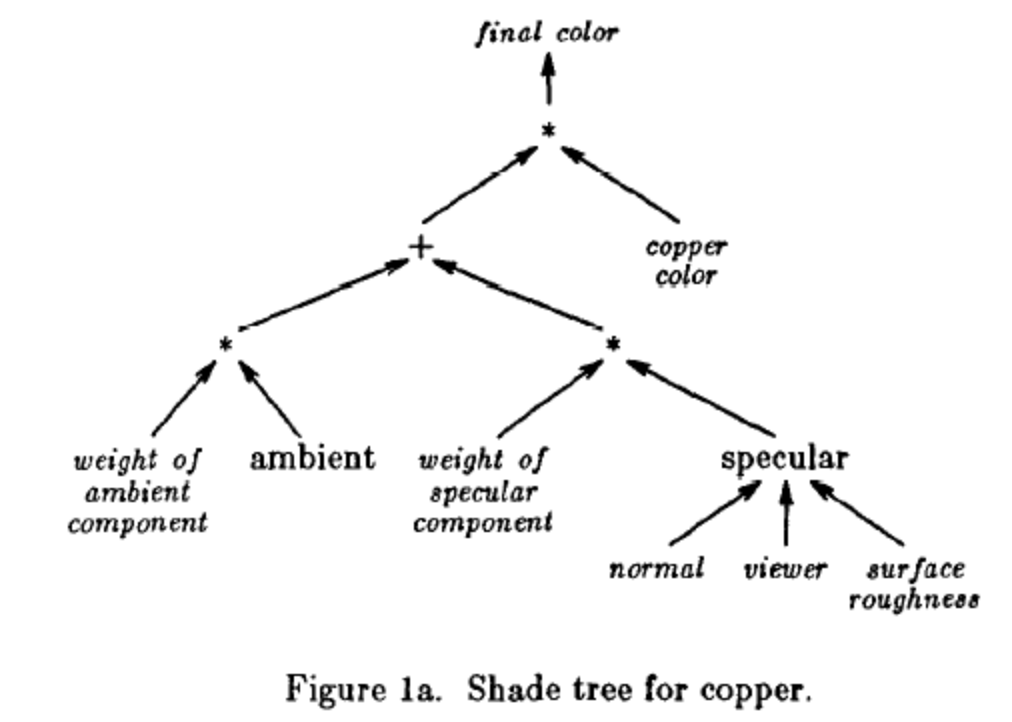
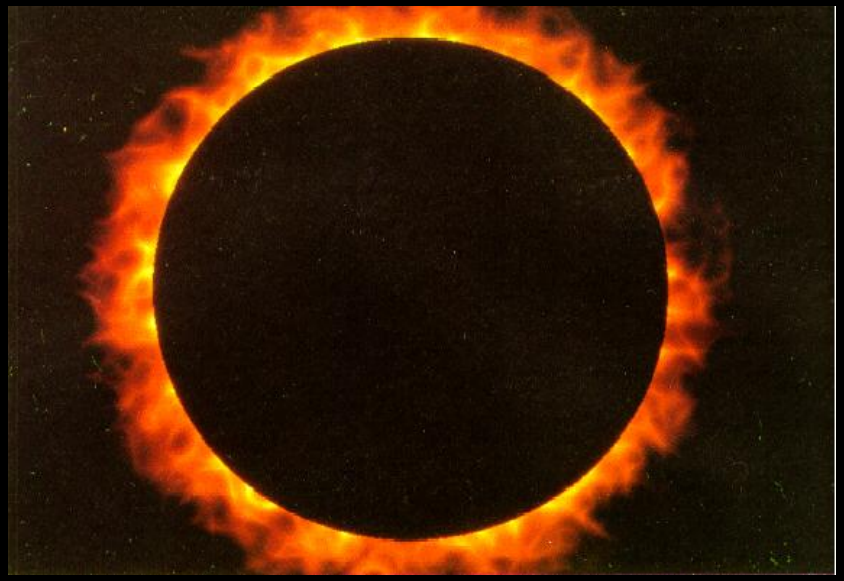
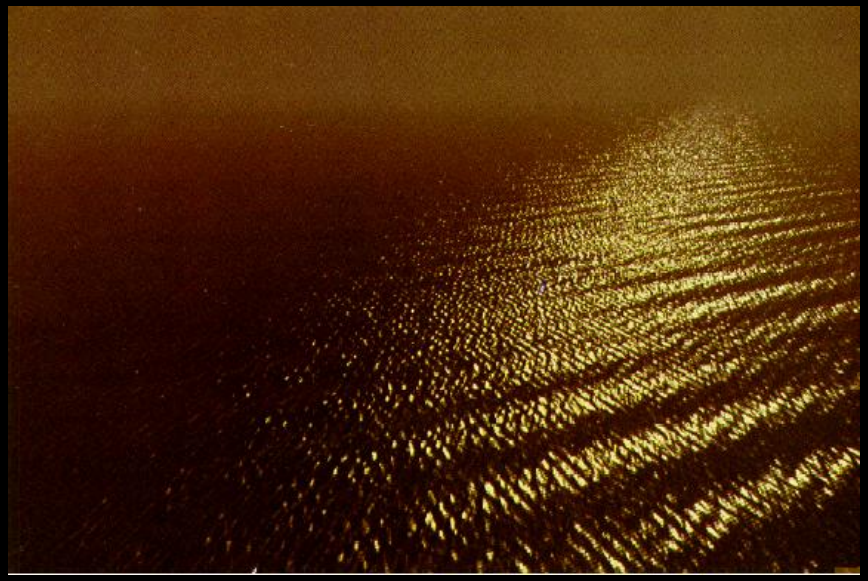


Figure 1a. Shade tree for copper.

Images from Perlin





Contributions

Cook: Separates / modularizes factors that determine shading

- Geometric
- Material
- Environmental

Perlin: Language definition, conditionals, …

Leads to …

RenderMan—Hanrahan/Lawson

1. Abstract shading model based on optics for either global/local illumination

2. Define interface between rendering program and shading modules

3. High-level shading language

Three main kinds of shaders—light source, surface reflectance, volume

Shading Language Summary

- Expert rendering engineers write highly optimized rendering architecture
- Shading languages allow non-expert users to customize renderer
- Consequences
- Renderers specify their own compilers, language runtimes, memory models
- User's shading code runs as inner-most loop---JIT ends up being very important

A Few (but far from all) Seminal Papers

- Depth Buffer: Catmull 1978
- A-Buffer: Carpenter 1984
- Shade Trees: Cook 1984
- Alpha Blending: Porter and Duff 1984
- REYES: Cook, Carpenter, Catmull 1987
- Renderman: Hanrahan, Lawson 1990
- Reality Engine Graphics: Akeley 1993
- …

In Early 90s, Interactive Rendering Started Over

Wolfenstein 3D, 1992

Doom I, 1993

- Interactive software rendering (no GPUs yet)
- NOTE: SGI was building interactive rendering supercomputers, but this was beginning of interactive 3D graphics on PC

By Late 90s, Graphics Hardware Emerging

- "Hey, let's build hardware to make a highly constrained graphics pipeline go really fast"
- 3DFX, NVIDIA, ATI, and countless others
- Eventually replaced SGI's supercomputers with plug-in boards for PC
- OpenGL and Direct3D gained dominance as they provided the *only* programming interface to GPUs

OpenGL and DirectX (90's to early 2000s)

- Fixed function pipeline
  - Configure options via API
  - Fixed per-vertex lighting model
  - Fixed vertex transforms
  - Limited texturing

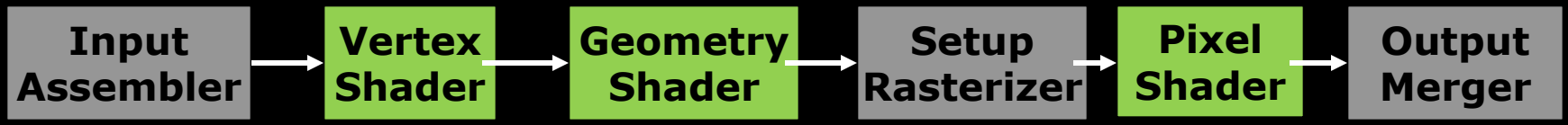
2001+:
GPUs "Rediscover" Programmable Shading

- NVIDIA GeForce 3 and ATI Radeon 8500

- Programmable vertex computations
  - Up to 128 instructions

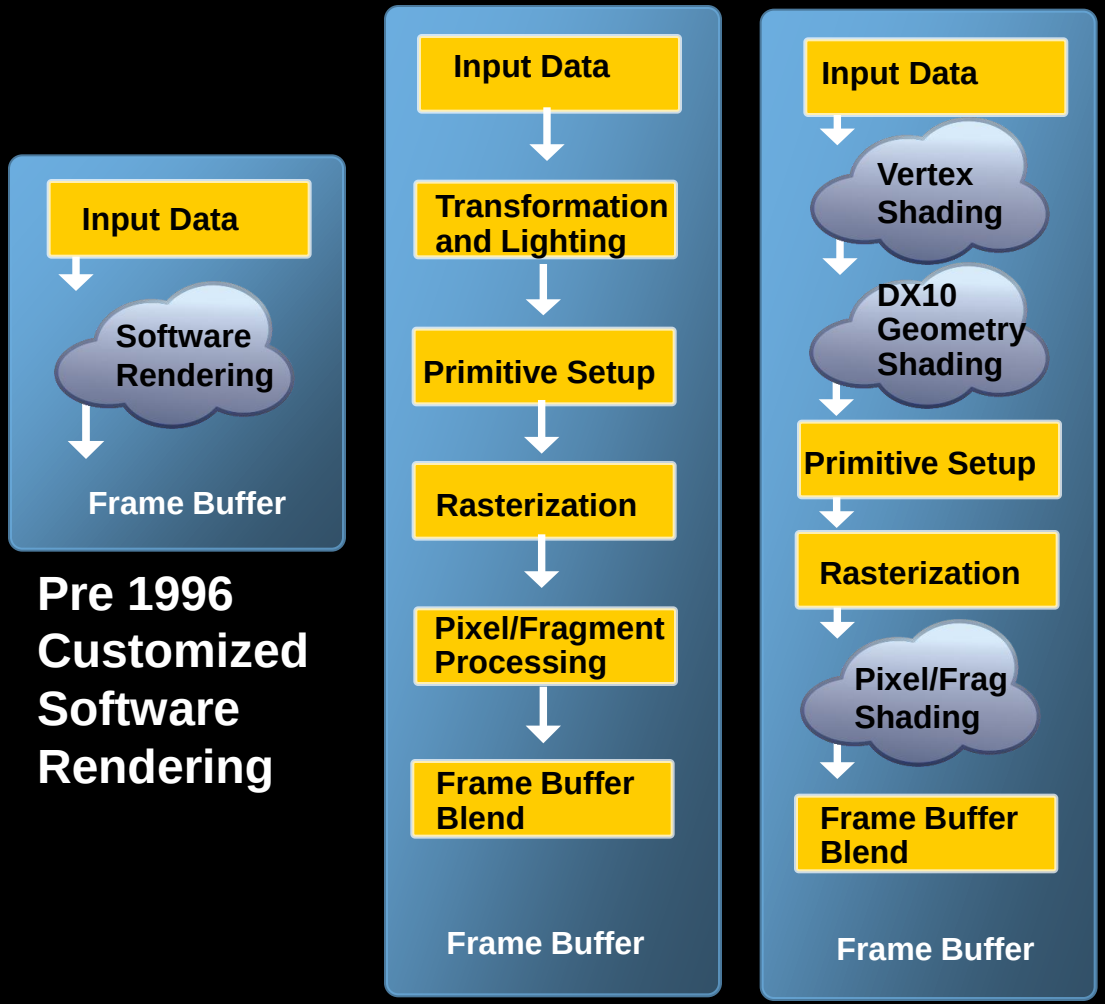
- Limited programmable fragment computations
  - 8-16 instructions

Today's (DirectX 10) Programmable Pipeline



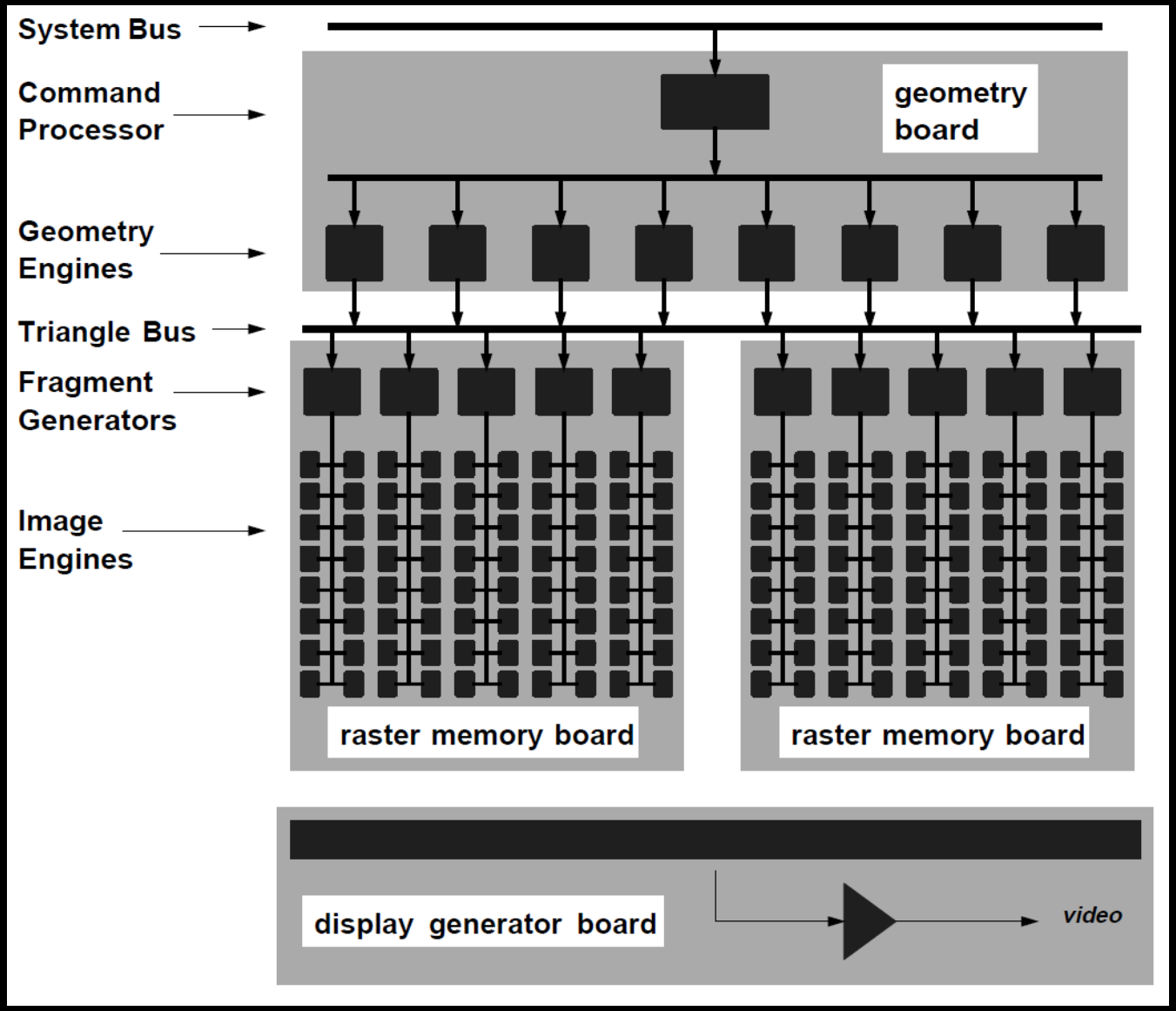
- High-level shading languages
  - GL Shading Language (GLSL) for OpenGL
  - High Level Shading Language (HLSL) for DirectX
- 1000s of instructions permitted per stage
- Flow control, integers, arrays, temporary storage, large number of textures, …

Interactive Rendering Pipelines

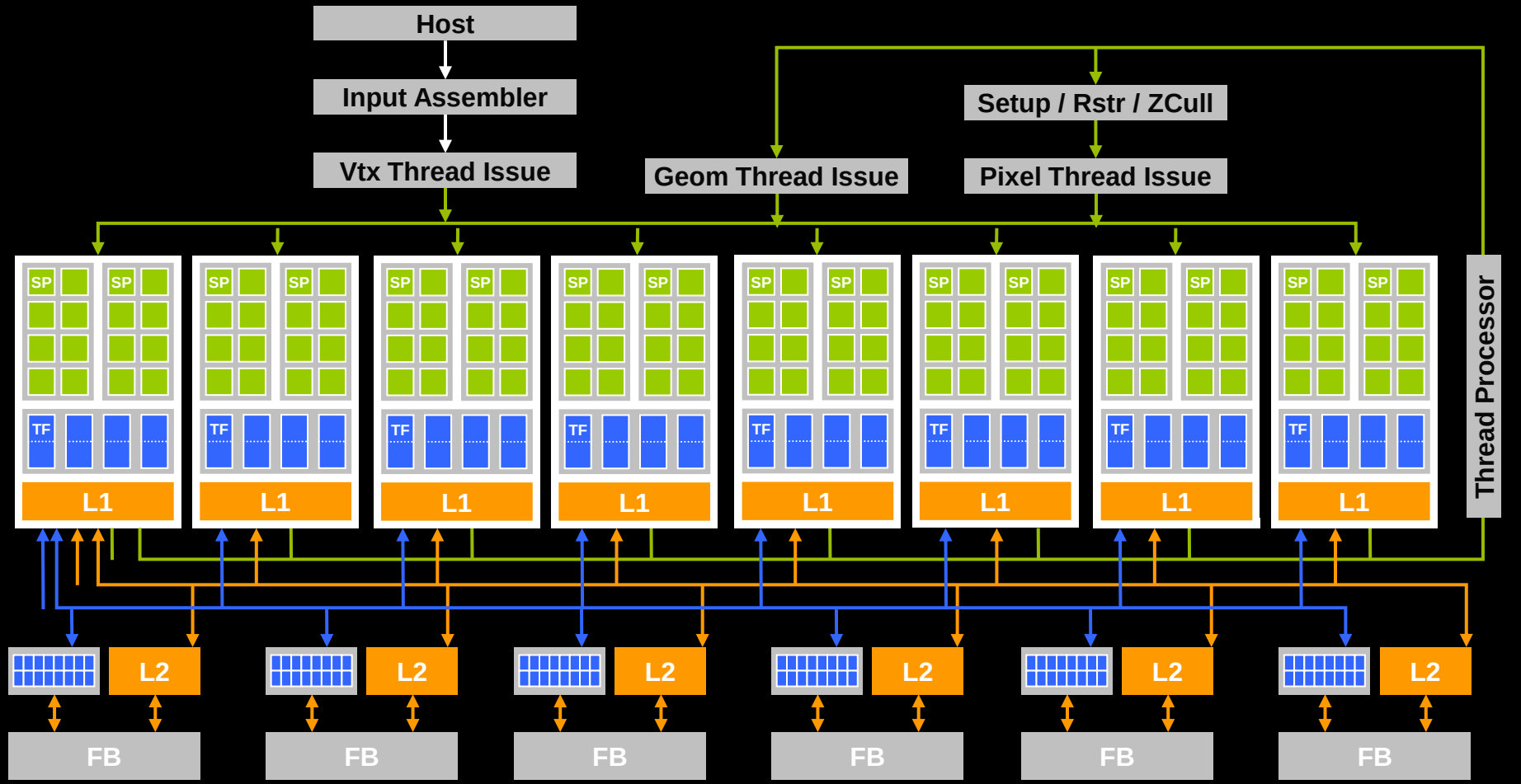


Graphics Hardware Changes

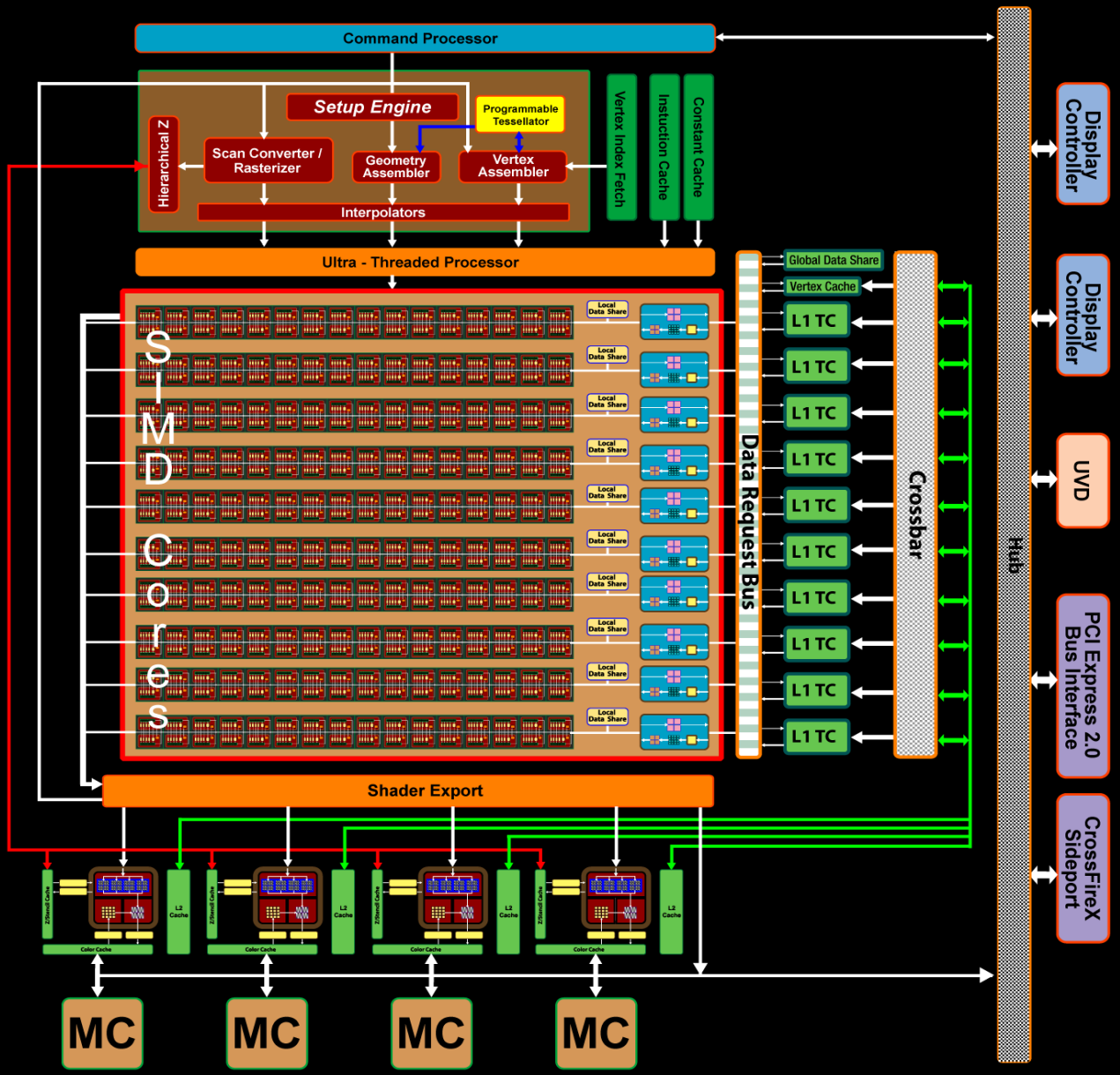
SGI Reality Engine (Akeley, 93)



Nvidia G80


Host
Input Assembler
Vtx Thread Issue
Geom Thread Issue
Setup / Rstr / ZCull
Pixel Thread Issue
SP
TF
L1
L2
FB
Thread Processor

ATI RV770


Command Processor
Setup Engine
Programmable Tessellator
Hierarchical Z
Scan Converter / Rasterizer
Geometry Assembler
Vertex Assembler
Interpolators
Vertex Index Fetch
Instruction Cache
Constant Cache
Ultra - Threaded Processor
SIMD Cores
Local Data Share
Global Data Share
Vertex Cache
L1 TC
Data Request Bus
Crossbar
Shader Export
Z/Stencil Cache
Color Cache
L2 Cache
L3 Cache
MC
Hub
Display Controller
Display Controller
UVD
PCI Express 2.0 Bus Interface
CrossFireX Sideport

Beyond Programmable Shading: Parallel Programming for Graphics

Completing the Circle

- Beginning in 2001/2002, researchers realized that programmable GPUs could do more than graphics
  - GPUs were becoming data-parallel co-processors
  - A research field was born: General Purpose Programming on GPUs (GPGPU)
  - Scores of papers about data-parallel algorithms on GPUs
    - Finance, physical simulation, medical imaging, …

Non-Graphics GPU Programming Models

- From GPGPU, arose parallel programming models that let users program GPUs without using the 3D APIs (OGL / D3D)
  - Brook, Sh / RapidMind, PeakStream, CUDA, OpenCL, DX ComputeShader, …
- Current focus on 'flat' data-parallelism
- Future focus to include
  - Nested data parallel?
  - Task parallel?
  - Pipelines?

"The Killer App of GPGPU is Graphics!"

- But then researchers starting writing rendering papers that combined data-parallel GPU algorithms with the GPU rendering pipeline
  - Ray tracing and photon mapping, Purcell 2002-2003
  - Summed Area Table Generation, Hensley 2005
  - Resolution Matched Shadow Maps, Lefohn 2007
  - Hair rendering, Sintorn 2009
  - …

Nuts & Bolts: How to Write Interactive Graphics Code

- CPU-only: A lot of multi-core CPUs available and good old C/C++/etc
- NVIDIA-only: CUDA plus Direct3D/OpenGL
- AMD GPU-only: Brook+ plus Direct3D/OpenGL
- Intel Larrabee-only: “Larrabee Native”

- OpenCL plus OpenGL/Direct3D
- DX ComputeShader plus Direct3D

Wrap Up

Research Directions

- Change the graphics pipeline
  - User-configurable pipelines (fewer/new stages)?
  - Add functionality to existing stages
    - Maintain high efficiency or work is useless

- Examples
  - GRAMPS: Sugerman 2009
  - Real-time REYES-style subdivision: Patney 2008
  - RenderAnts: Zhou 2009
  - Real-Time View-Dependent Rendering of Parametric Surfaces: Eisenacher 2009
  - More papers coming…

Research Directions

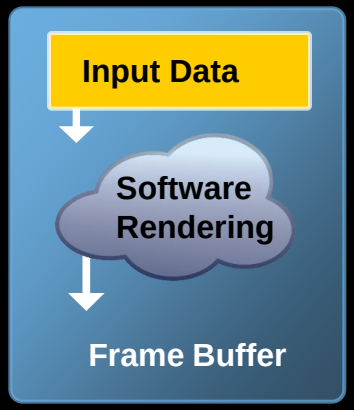
- New interactive rendering algorithms that mix task-, data-, and pipeline parallelism with graphics pipeline
  - Build dynamic data structures
  - Intra-frame scene and/or image analysis
  - Pixel-level or geometry-level acceleration structures
  - See following talks from Cass and Johan

- Examples
  - Hair and shadow rendering: Sintorn/Assarson 2008-2009
  - Quadtree shadow maps: Lefohn 2007
  - Global illumination: Pellacini 2006, Purcell 2003, …
  - Lots more papers out there and more coming…

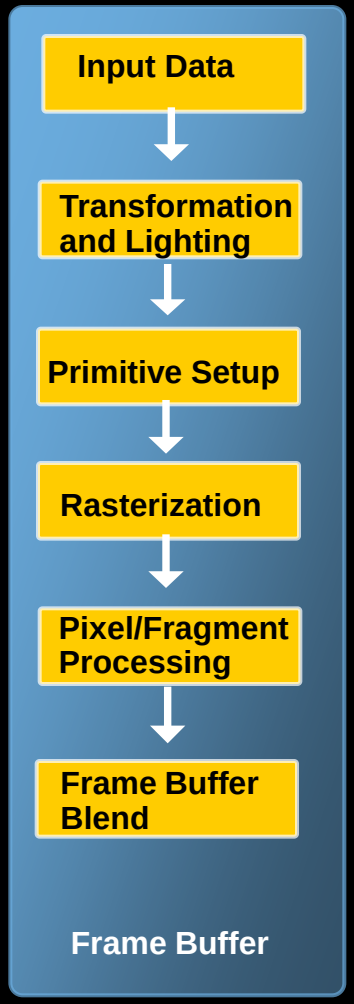
Graphics Programming Model Research Directions

- Current graphics APIs are "easy parallelism"
  - Graphics very successful parallel computing
  - Implicitly parallel, automatic dependency tracking, no deadlocks
- Future graphics programming include more types of parallelism
  - Task parallelism, user-defined pipelines, data-parallelism, user-defined data structures, …
- How to give graphics programmers this flexibility without cratering productivity?
  - Multiple levels of expression / parallelism?
  - New languages?
  - What is the role of shading languages in the fully programmable world and if it exists, what does it look like?

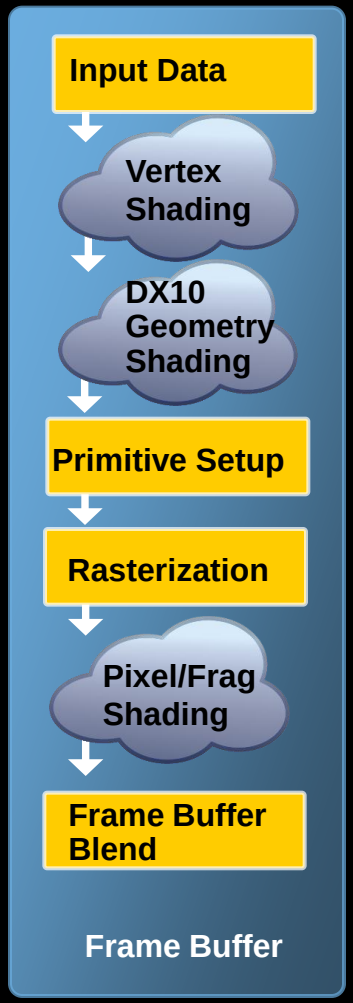
Interactive Rendering Returns to Software?



Pre 1996 Customized Software Rendering



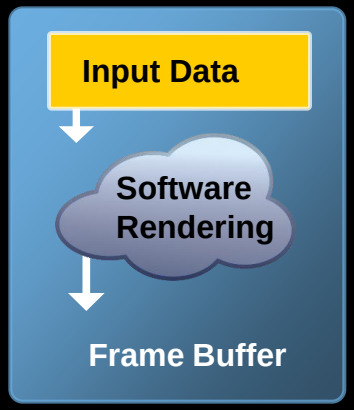
Pre 2001



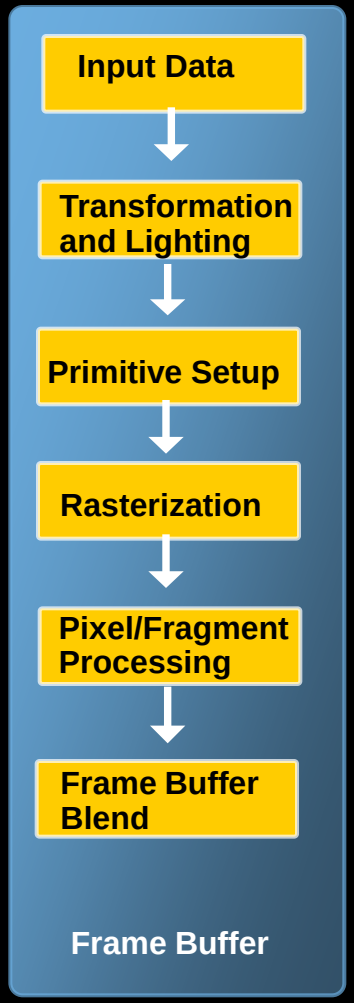
DX10

?

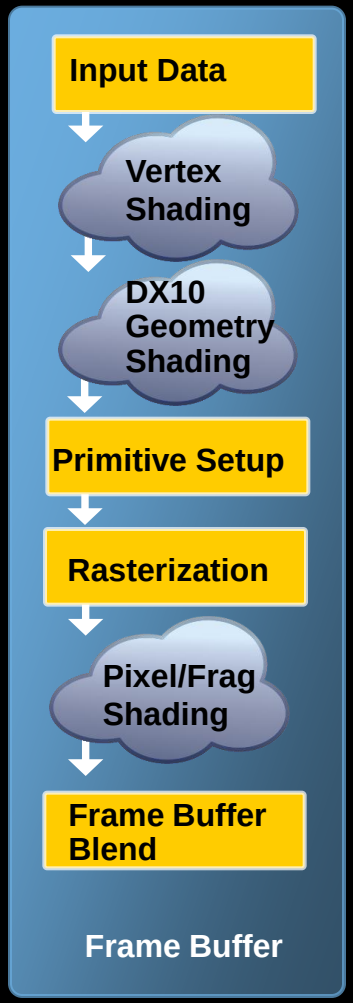
Interactive Rendering Returns to Software?



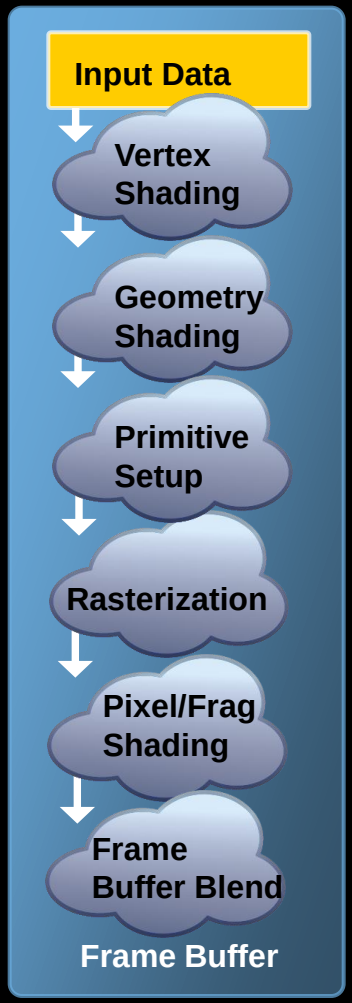
Pre 1996 Customized Software Rendering



Pre 2001



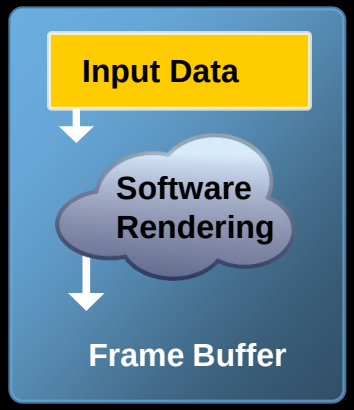
DX10



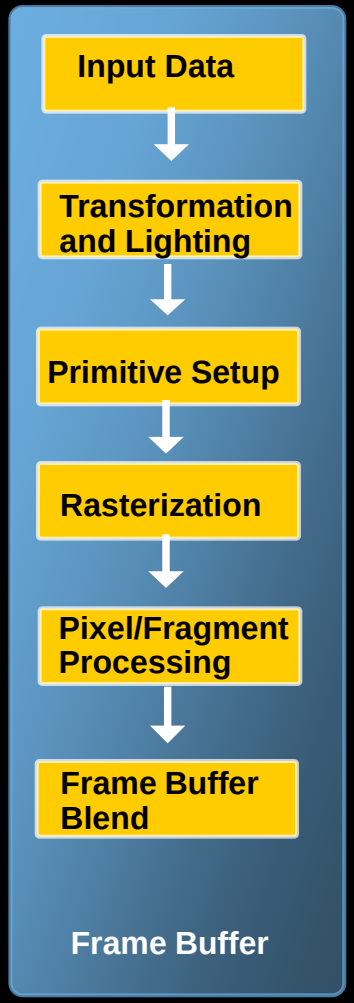
No fixed function?

?

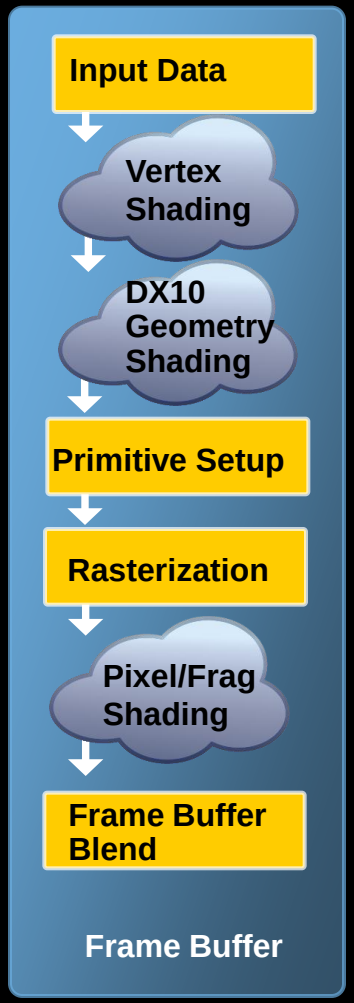
Interactive Rendering Returns to Software?



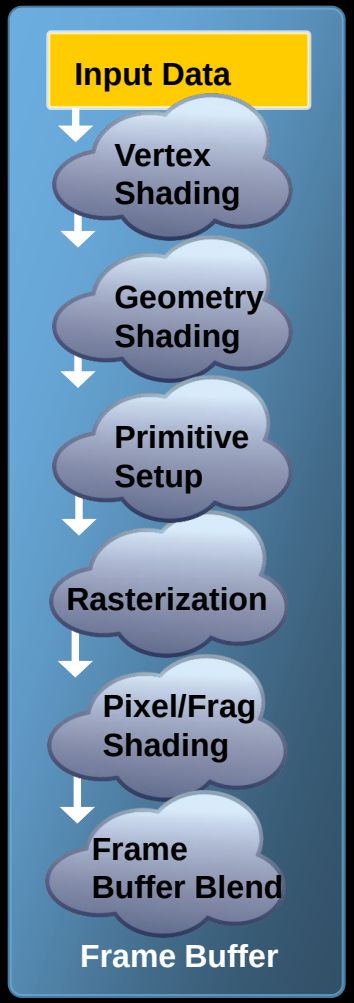
Pre 1996 Customized Software Rendering



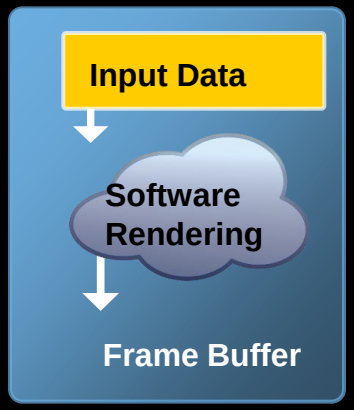
Pre 2001



DX10



No fixed function?



Software Rendering?

Acknowledgements

- Intel
  - Aaron Lefohn, Tim Foley
- Stanford
  - Kayvon Fatahalian
- UC Davis
  - John Owens
- NVIDIA
  - Mark Harris, David Luebke