Programmable Hardware #2

Focus

Current Programming Languages

Navendu Jain

Outline

- Introduction
- Cg
- Shading and Lighting
- Shading Languages
- Related Issues and Discussion

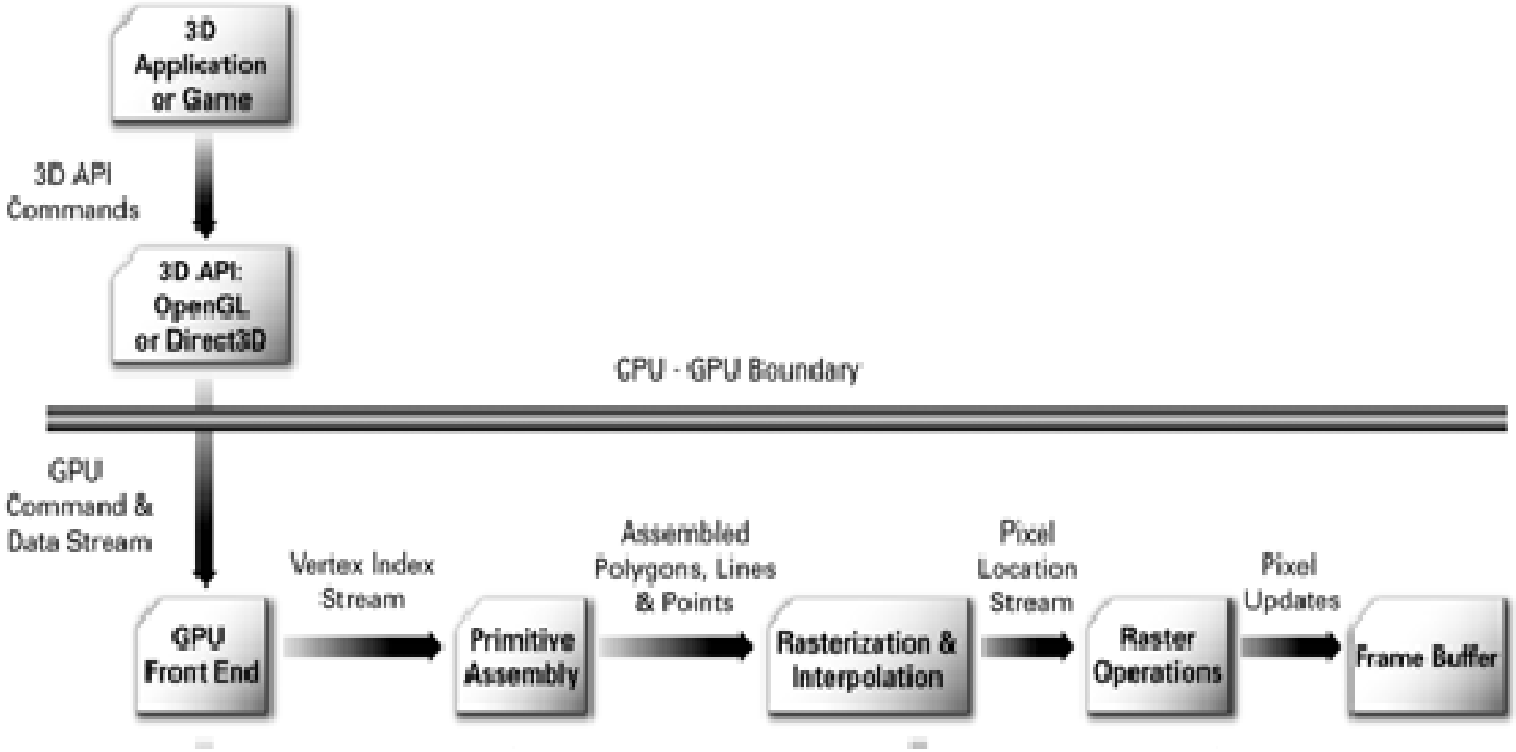
Outline

- **Introduction**
- Cg
- Shading and Lighting
- Shading Languages
- Related Issues and Discussion

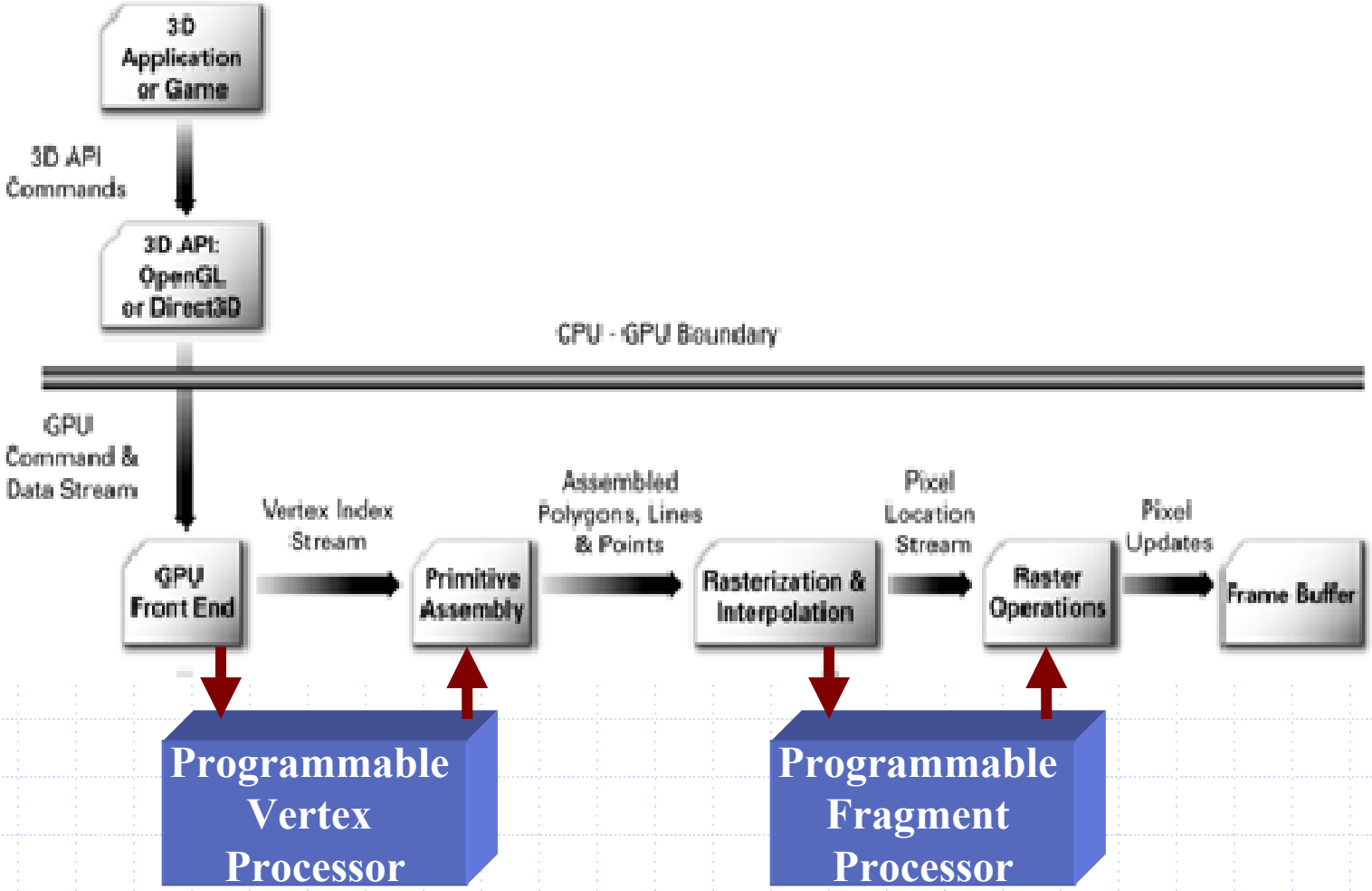
Evolution of the GPU

- Specialized and highly parallelized design
- "Big Iron" **to** "Single Chip"
- Moore's Law cubed
- More Realistic and More Interactive
- Programmability

GPU Model


3D Application or Game
3D API Commands
3D API: OpenGL or Direct3D
CPU - GPU Boundary
GPU Command & Data Stream
GPU Front End
Vertex Index Stream
Primitive Assembly
Assembled Polygons, Lines & Points
Rasterization & Interpolation
Pixel Location Stream
Raster Operations
Pixel Updates
Frame Buffer

GPU Programming Model



Impact

- Result
  - New H/w features, many APIs
  - Assembly Coding
  - Demand for more flexibility, control
- Solution
  - Raise the level of abstraction
  - Programmability
  - **Give me the power – Developer!!**

- Introduction
- **Cg**
- Shading and Lighting
- Shading Languages
- Related Issues and Discussion

Cg

- "C for graphics" ; developed by NVIDIA
  - Ease of Programming (tweak & run)
  - Virtualizes the hardware
  - Library of Shaders
- Run-time compilation - optimization
- Portable Programs

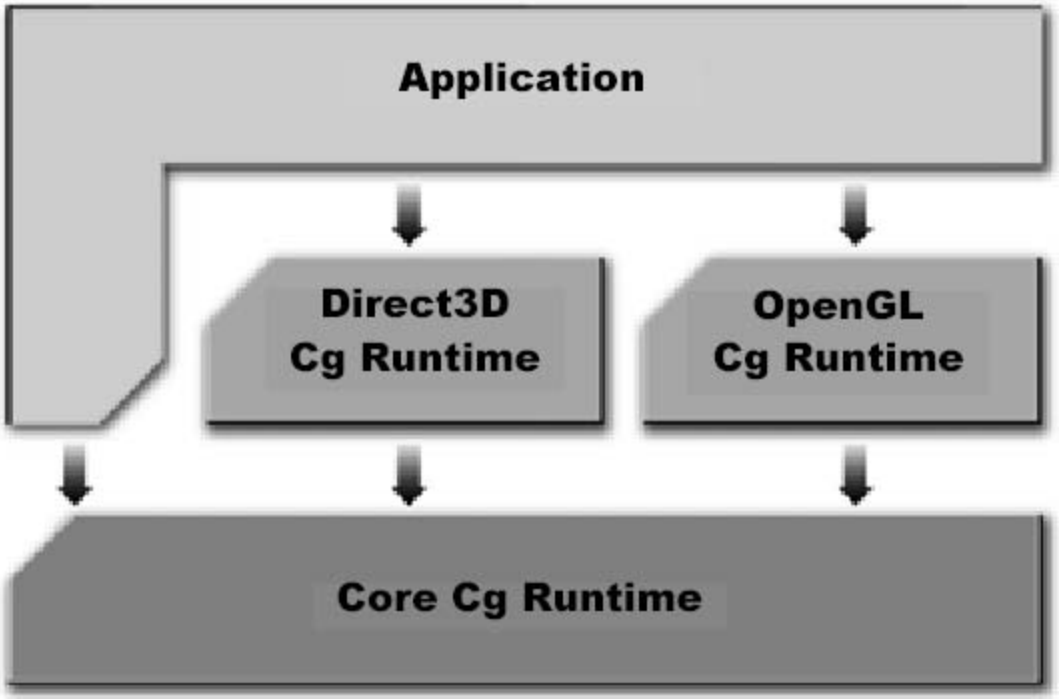
Language Profiles

- GPUs don't support the same capabilities
- Profile defines a subset of the language supported on a particular hardware
- Examples:
  - Vertex Shaders
    - CG_PROFILE_VS_2_X, CG_PROFILE_ARBVP1
  - Fragment Shaders
    - CG_PROFILE_PS_2_X, CG_PROFILE_ARBFP1

Programs

- Operate on streams of data
- Program Inputs and Outputs
  - Varying (per-vertex, per-fragment values)
  - Uniform (transformation matrix)
- Features
  - In-built Vector & Matrix support
  - No support for pointers
  - Modifiable function params by value-result
  - Swizzle *float3(a, b, c).zyx yields float3(c, b, a)*

Cg runtime library



- Introduction
- Cg
- **Shading and Lighting**
- Shading Languages
- Related Issues and Discussion

Background : Shading and Lighting

- Shading
  - color, texture, optical, anistropic properties, illumination environment
- Aspects
  - Surface reflectance
    - Ambient, Diffuse, Specular, wavelength, polarization
  - Light source distribution
    - Point light sources, Geometric Primitives
    - Intensity = f(x, y, z, λ, **D**)

Models of Shading

- Inherently local processes
- Global Illumination process
- Kajiya's rendering equation
  - General Illumination Process
  - Local/ Global are independent aspects

$$i(x,x') = v(x,x') \left[l(x,x') + \int r(x,x',x'') \, i(x',x'') \, dx'' \right]$$

Types of Shaders

- Light Source **l(x,x')**
  - Color, intensity emitted from a point
- Surface Reflectance **r(x,x',x'')**
  - Integral of bidirectional reflectance with incoming light
- Volume or Atmosphere **v(x,x')**
  - Scattering Effects
  - Other intersections handled by the Renderer

- Introduction
- Cg
- Shading and Lighting
- **Shading Languages**
- Related Issues and Discussion

Shading Languages

- Programming shading computations
- Extend shading and lighting formulae, types of material and light sources
- Offline
  - Hanrahan's, RenderMan
- Real-Time
  - Stanford Shading Language, NVIDIA's Cg

Differences

Real-Time Shading

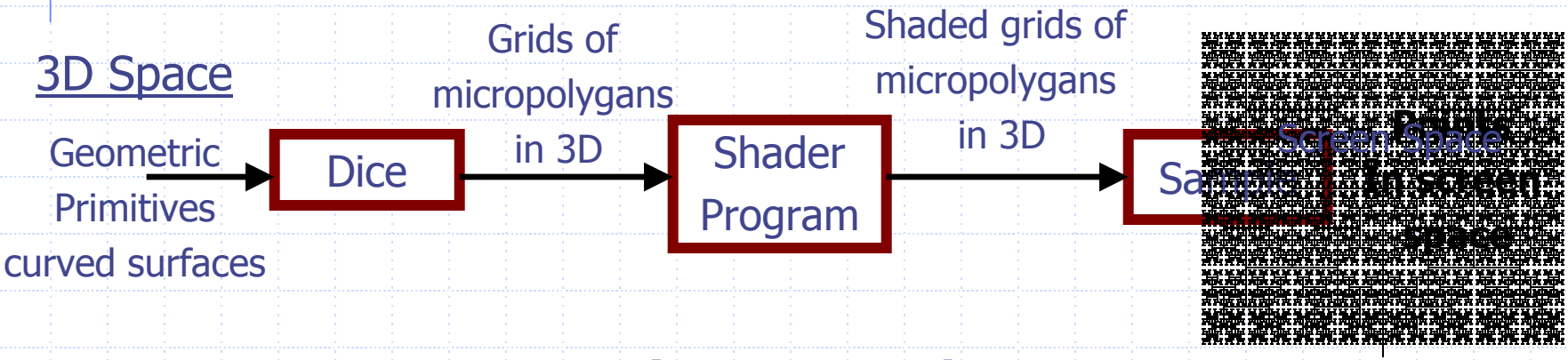
- Interactive apps
- Lighting and Shader Anti-aliasing harder
- Performance critical Frame Rate (FPS)
- Execute on GPU
- Rendering cost per frame is bounded

Offline Shading

- Fixed Viewpoint
- Fine Tuning
- Non real-time
- Execute on CPU
- Differ by orders of magnitude

Shading approaches

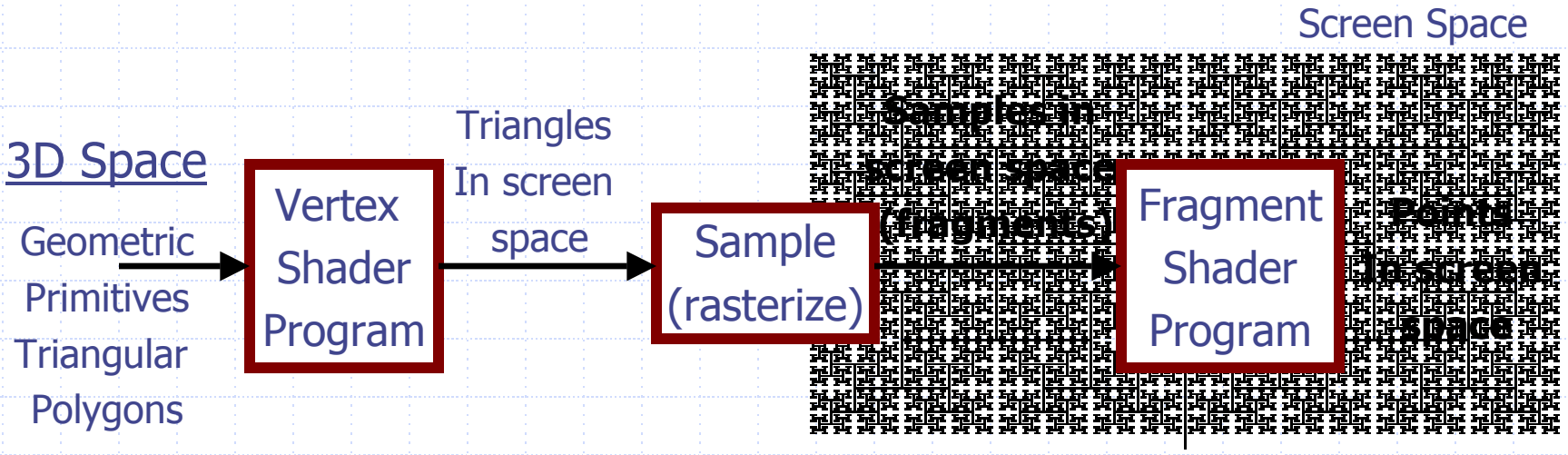
- Object Space (at vertices)
  - REYES (Pixar's PRMan)
    - Change position (displacement map, procedural)



- Screen Space (at pixels)

Graphics hardware approach

- Hybrid vertex/pixel shading
  - Triangle Vertices
  - Fragment Shading
    - Can only change screen space depth



Pros and Cons

Object Space

- Computations per polygon
- Inexpensive motion-blur, depth of field effects
- Anti-aliasing complex Geometric Normal

Hybrid Model

- Computations per pixel
- Expensive
- Simple derivative computations
- Only shading normal

- Introduction
- Cg
- Shading and Lighting
- Shading Languages
- **Related Issues and Discussion**

Related Issues

- Parallelism
  - "Single" vertex model
  - Memory access
  - Multiple rendering passes
- Computation model
  - SPMD (vertex)
  - SIMD (fragment)

Related Issues (contd.)

- Data Types (Low, High Precision)
- Memory/Register resource limits
- Host-to-GPU Bandwidth

Discussion

- Surface and Light Shaders
  - Separability
  - Z-buffer – surface renderer
  - Expressibility power
  - User's flexibility
  - Binding Model
    - Early-binding (expensive)
    - Late-binding (optimize)

Discussion (contd.)

- Sharing Computations
  - FP executed more than vertex programs
- Move computation from FP to VP
  - When : Result is
    - constant over all fragments
    - Linear across a triangle
    - Nearly linear across a triangle
- Move from VP to CPU (uniform params)

Discussion (contd.)

- Optimizations
  - Vector operations, swizzle
  - Library of shaders
  - Low precision data types
  - Minimize if/then/else
  - H/W – S/W approach

Thanks

References

- "Real-Time Programmable Shading", excerpt from "Texturing and Modeling: A Procedural Approach", Ebert et al, pp. 97-121.
- Cg toolkit user's manual", NVIDIA
- Language for Shading and Lighting Calculations" Hanrahan and Lawson, SIGGRAPH 90