

The Rendering Pipeline

*Material about Hardware Graphics Pipeline originally prepared
by*

João L D Comba, Carlos A Dietrich, Christian A Pagot and
Carlos E Scheidegger (SIBGRAPI 2003 Tutorial)

Revised with contributions by
Manuel M. Oliveira (September 2004)



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Rendering Images

Geometric model



rendering
algorithm

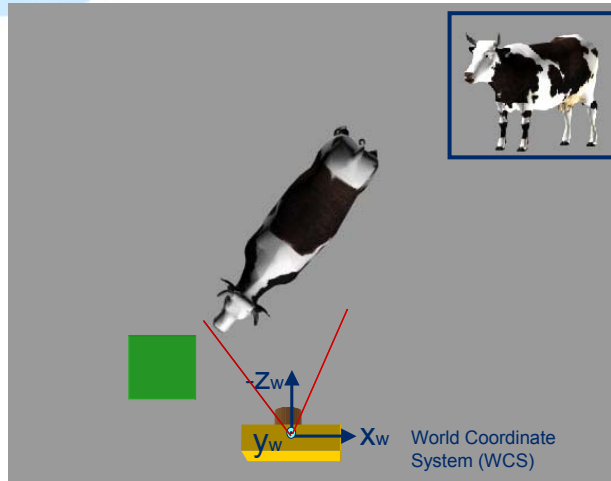


Picture



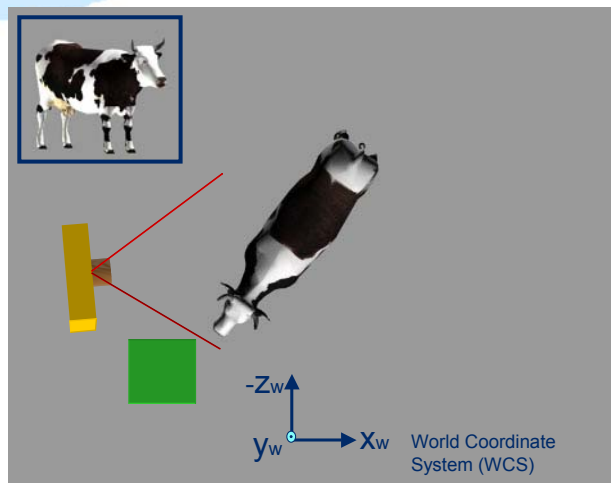
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

The Visualization Problem



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

The Visualization Problem

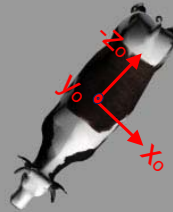
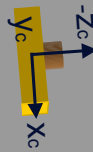


Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

The Visualization Problem

Need to change coordinate systems!

Camera
Coordinate
System (CCS)



Object
Coordinate
System (OCS)



World Coordinate
System (WCS)



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Rendering Pipeline (Overview)

Z-buffer and Gouraud shading



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Rendering Pipeline (Overview)

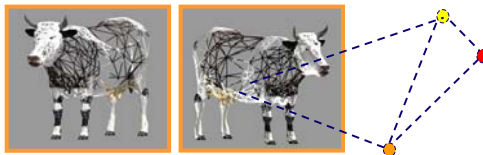


- Map from OCS to CCS
- Clipping in 3D
- Lighting



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Rendering Pipeline (Overview)

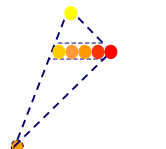
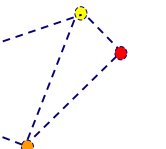
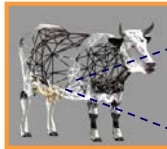


- Map from OCS to CCS
- Clipping in 3D
- Lighting
- Map from CCS to Screen CS
- Persp. division



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Rendering Pipeline (Overview)

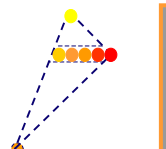
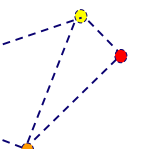
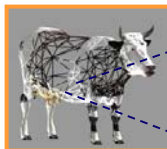


- Map from OCS to CCS
- Clipping in 3D
- Lighting
- Map from CCS to Screen CS
- Persp. division
- Color
- Visibility



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Rendering Pipeline (Overview)

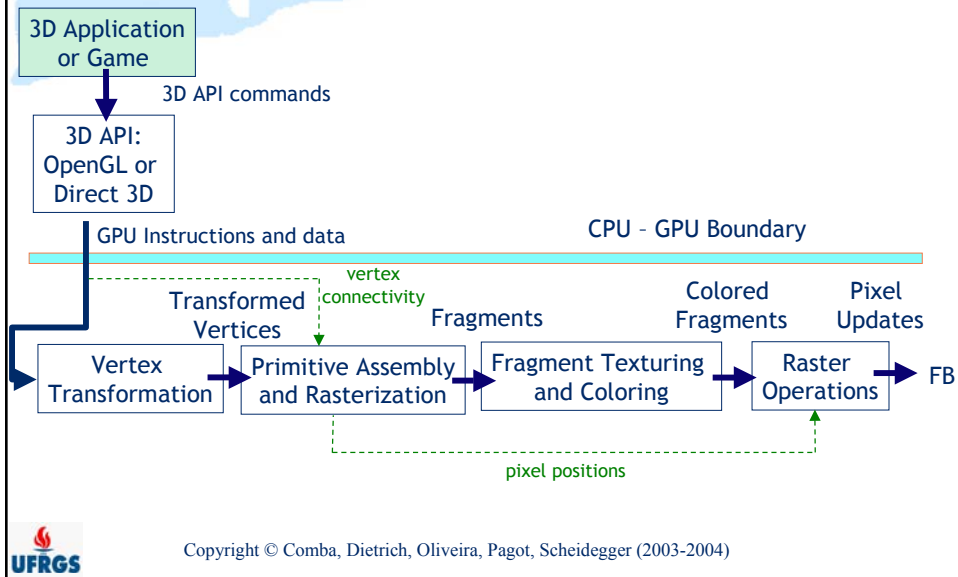


- Map from OCS to CCS
- Clipping in 3D
- Lighting
- Map from CCS to Screen CS
- Persp. division
- Color
- Visibility
- FB display

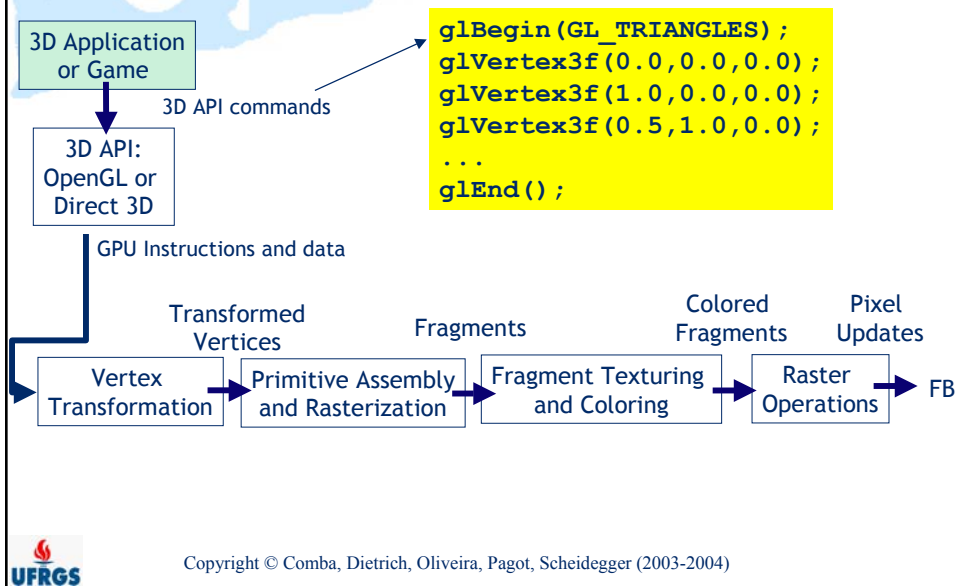


Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

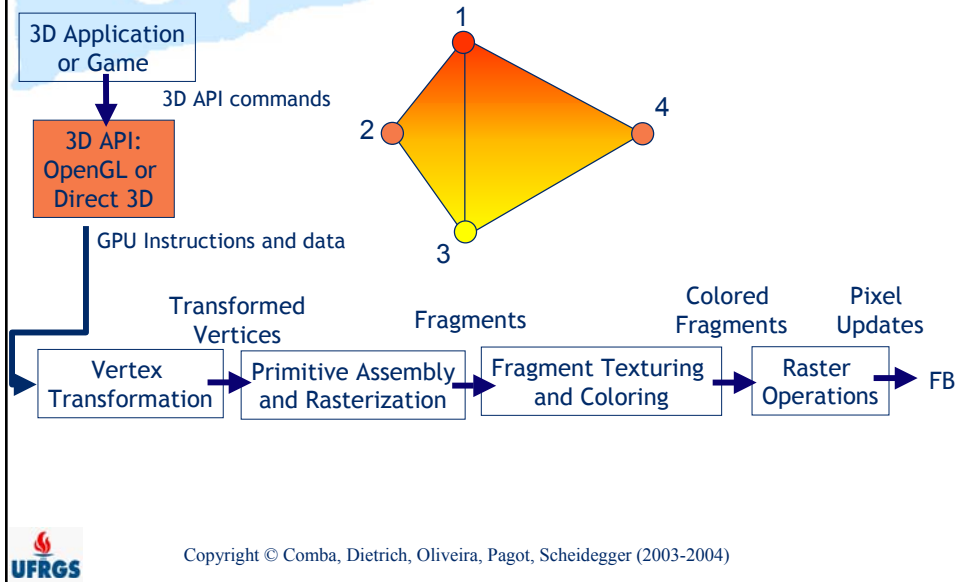
Graphics Hardware Pipeline



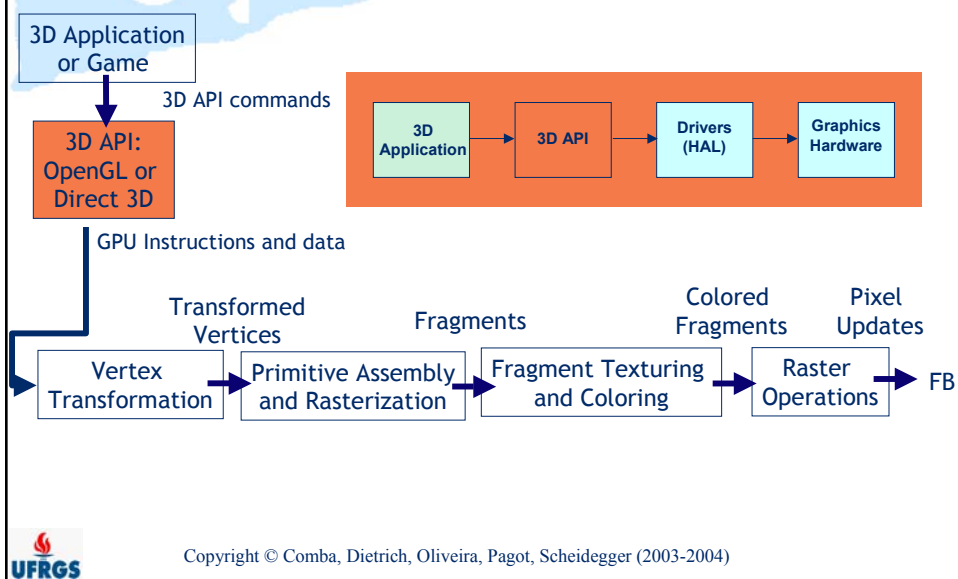
Graphics Hardware Pipeline



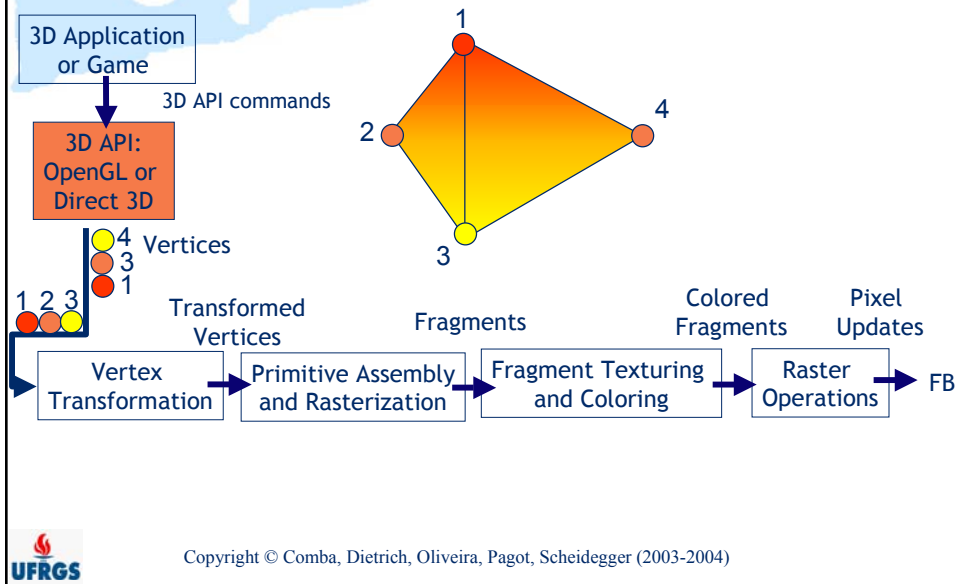
Graphics Hardware Pipeline



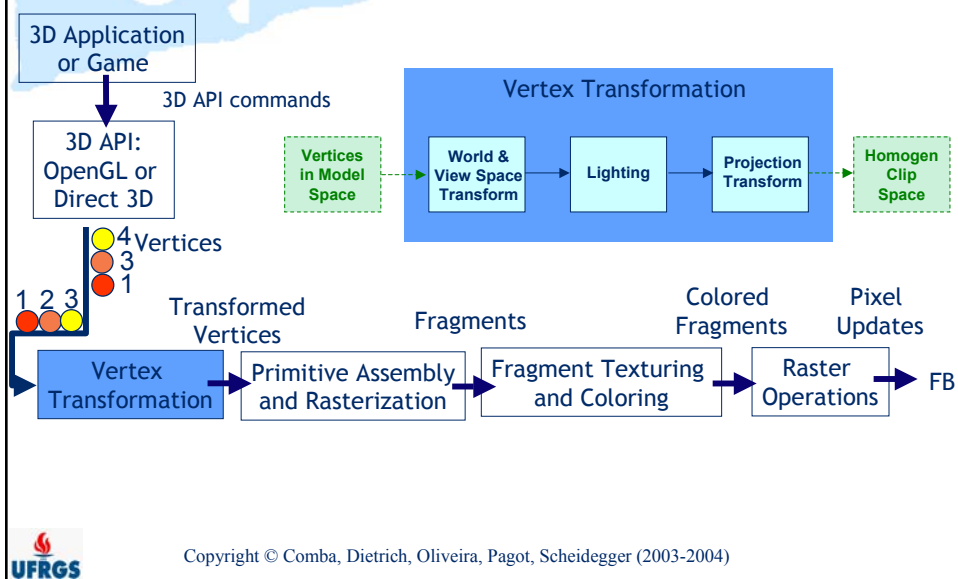
Graphics Hardware Pipeline



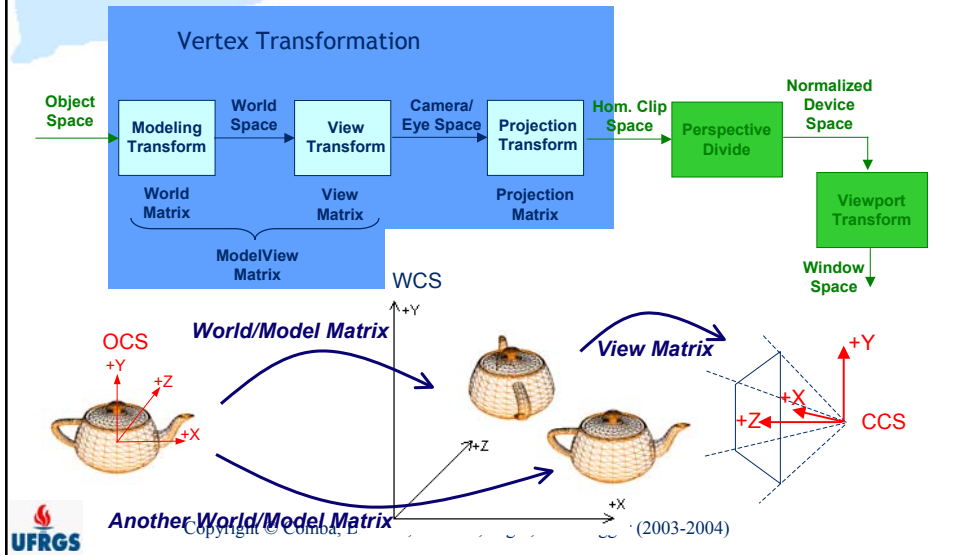
Graphics Hardware Pipeline



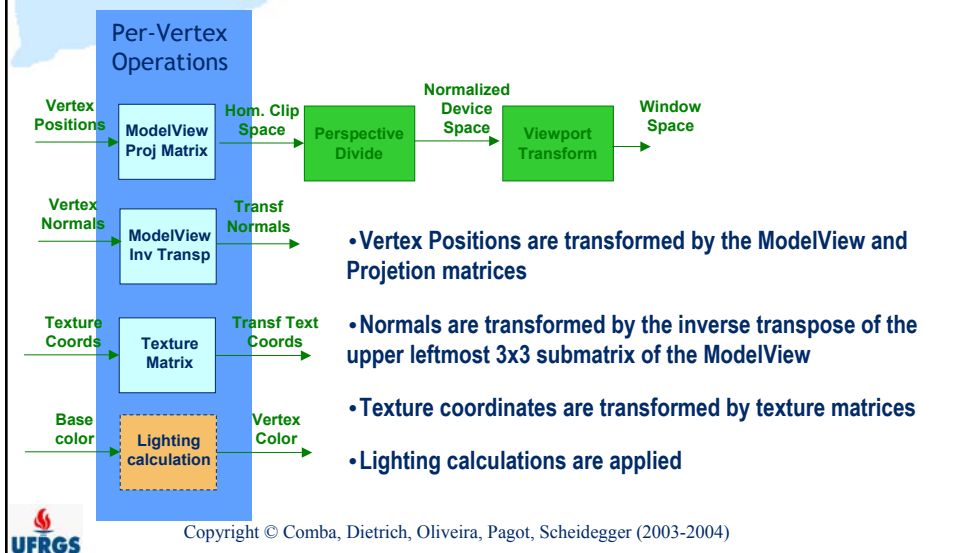
Graphics Hardware Pipeline



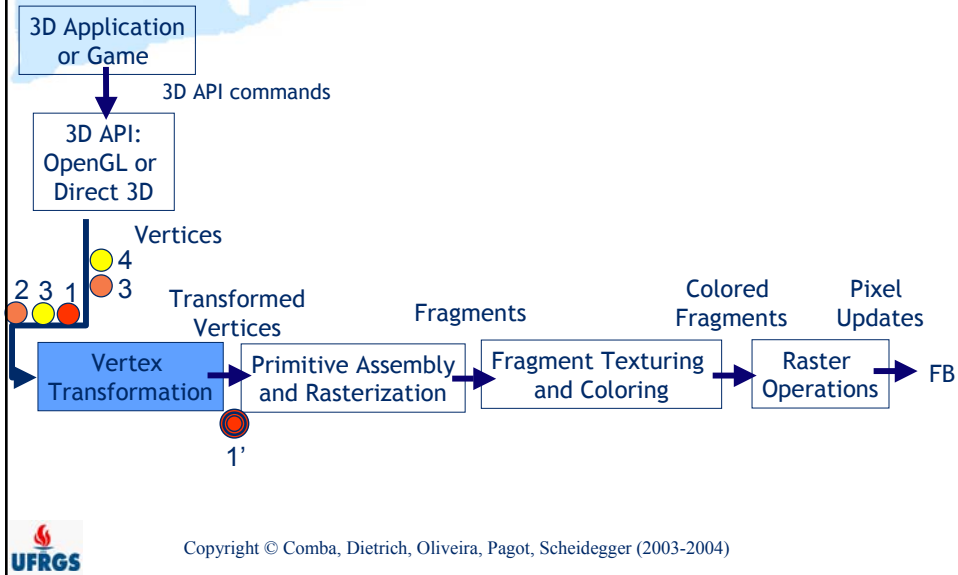
Vertex Transformations



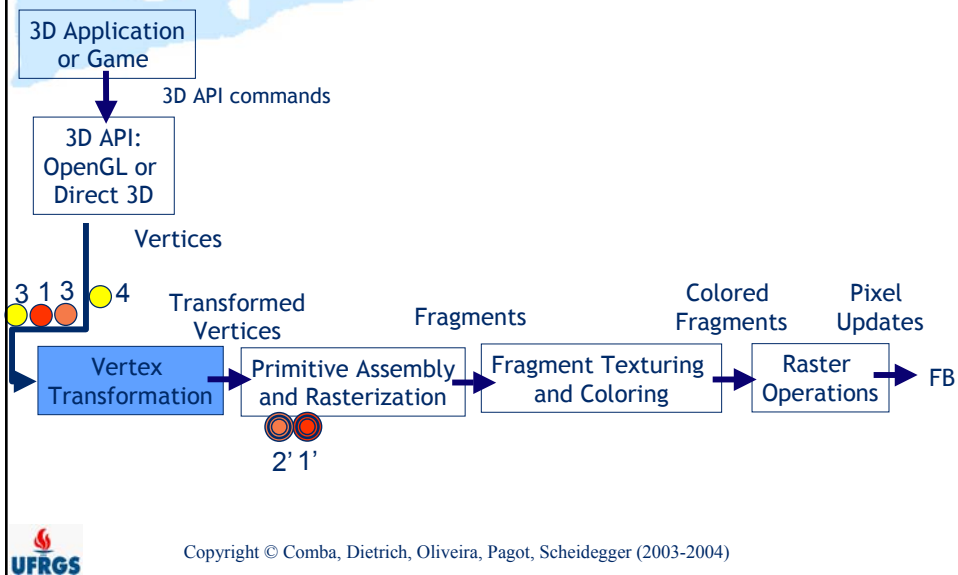
Per-Vertex Operations: Closer Look



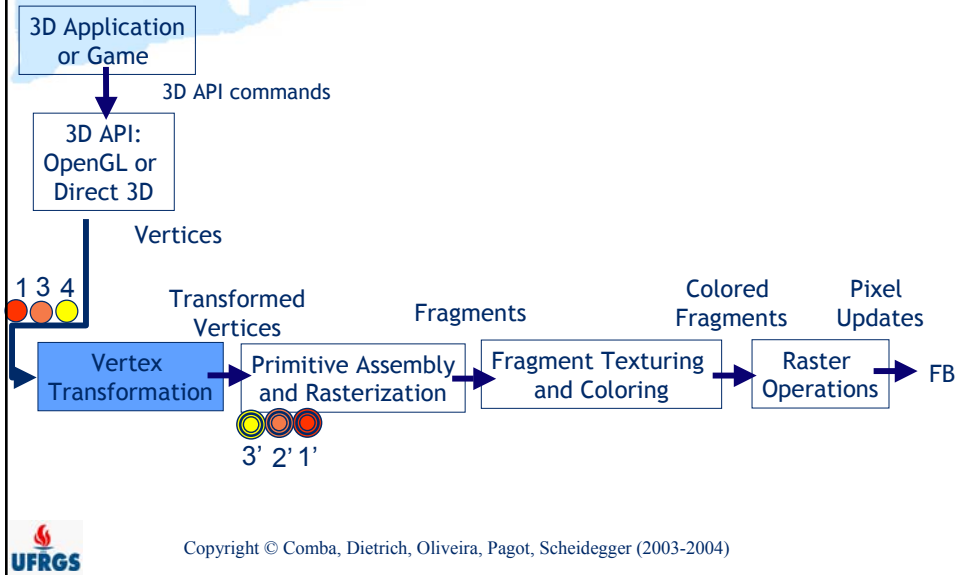
Graphics Hardware Pipeline



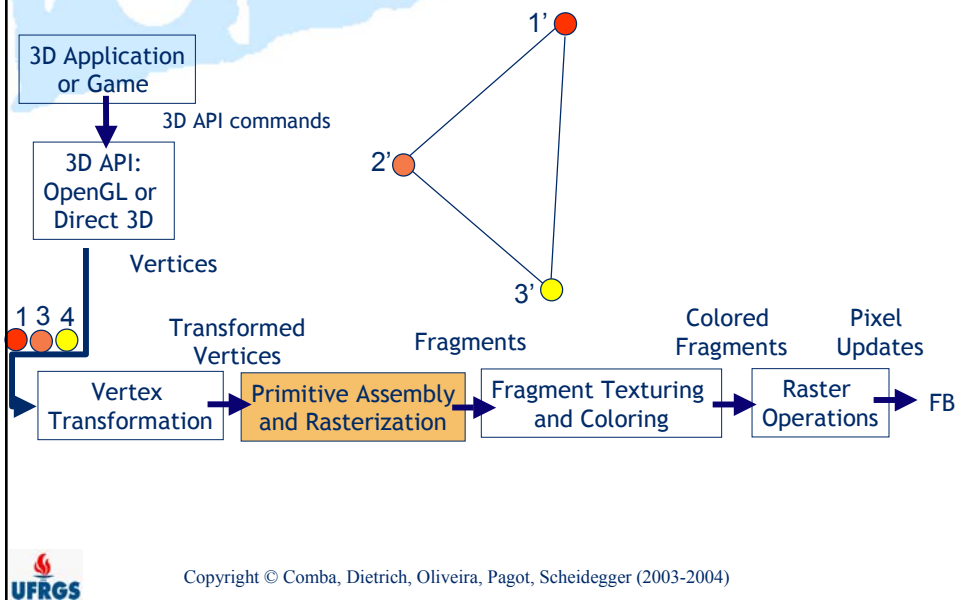
Graphics Hardware Pipeline



Graphics Hardware Pipeline



Graphics Hardware Pipeline



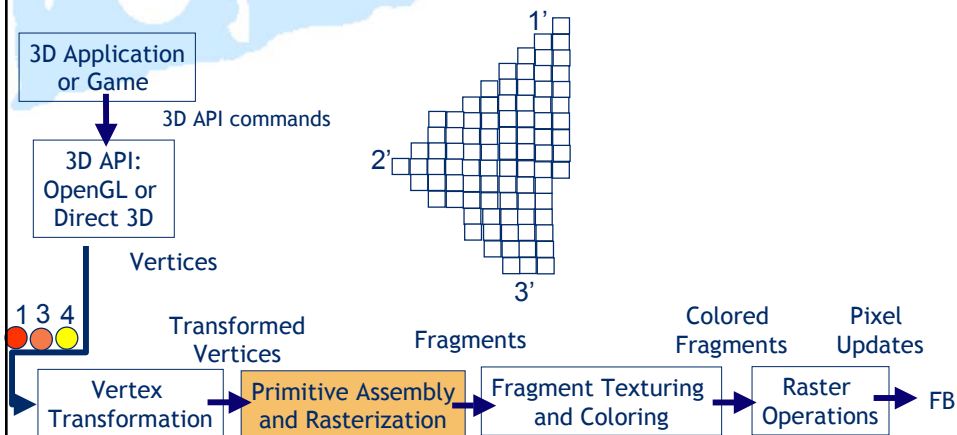
Primitive Assembly and Processing

- Vertex data is collected into complete primitives
 - Operations performed next (e.g., clipping) differ between the different types of primitives
- Clipping against user-defined clipping planes and view frustum
- Division by W (*often called perspective projection*)
- Viewport Transformation
 - Generates window coordinates
- Culling is performed at this stage
 - Backface/Frontface Culling



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Graphics Hardware Pipeline



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

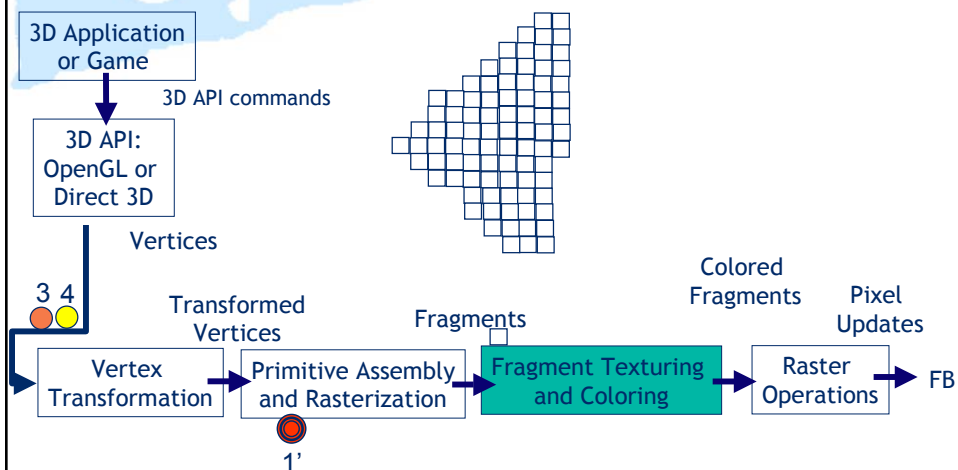
Rasterization

- Primitives are decomposed into smaller units (*fragments*) corresponding to pixels in the frame buffer
- A fragment has several attributes
 - Window coordinates, depth, color, texture coordinates, ...
- The values of each attribute are determined by interpolation from the values specified at the vertices
- The shading model (SMOOTH / FLAT) is important at this stage
- Antialiasing is also performed at this stage



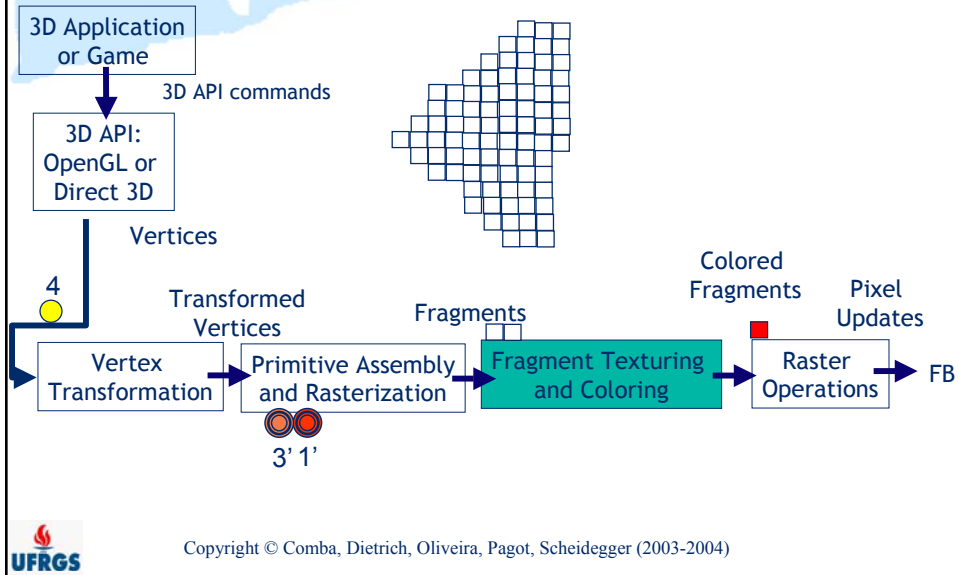
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Graphics Hardware Pipeline

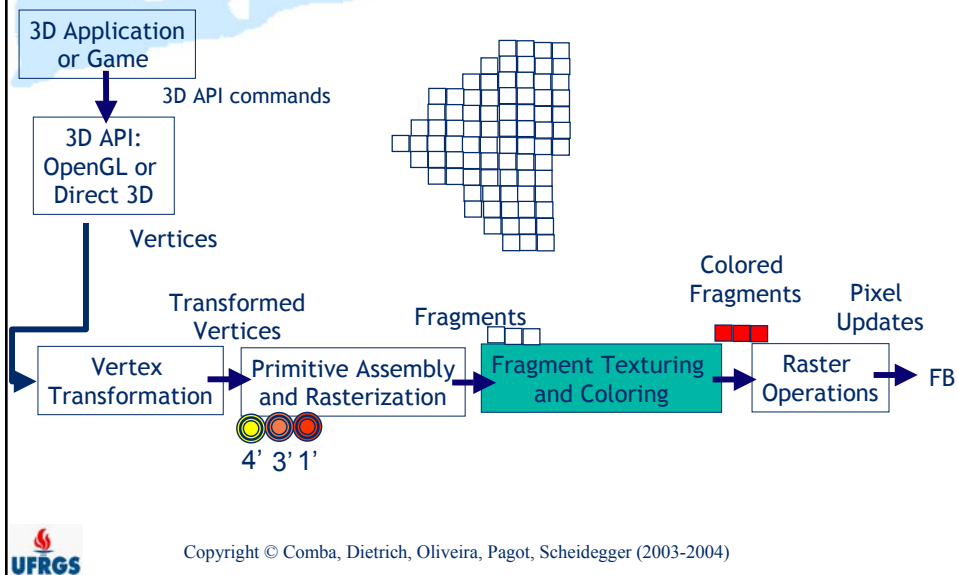


Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

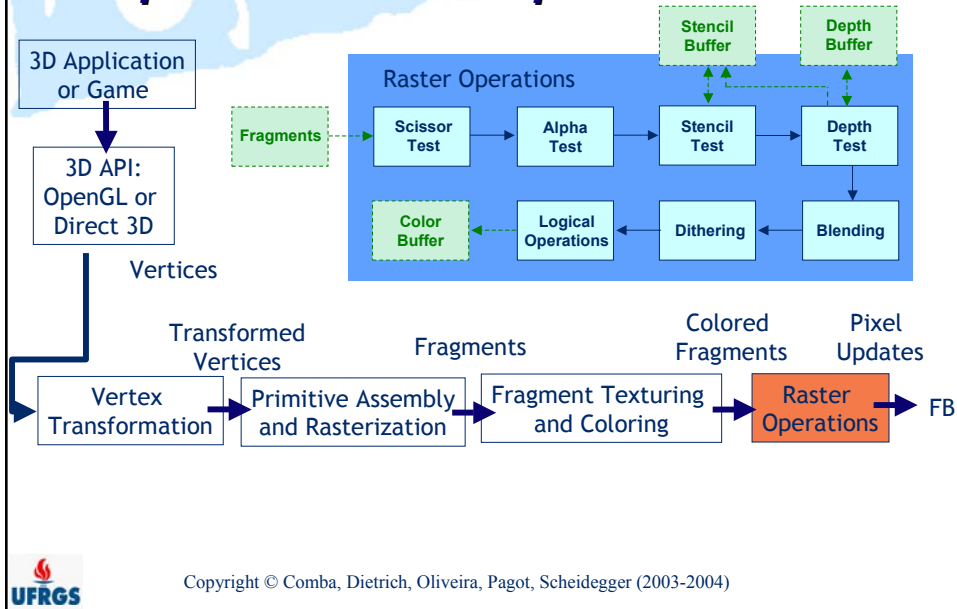
Graphics Hardware Pipeline



Graphics Hardware Pipeline



Graphics Hardware Pipeline



Raster Operations

• Scissor Test

- Check if the fragment is inside a rectangular portion of the window
- A version of the stencil test (easy to design very fast hardware for)

• Alpha Test

- In RGBA mode, accepts or rejects a fragment based on its alpha value
- Requires a reference value and a comparison function (NEVER, ALWAYS, <, ≤, >, ≥, =, ≠)
- **Application:** Transparency algorithms

• Stencil Test

- Requires a stencil buffer (sb), a comparison function and a mask
- Compares a reference value with the value stored at a pixel in the sb
- The value in the sb can be modified depending on the result of the test
- Actions for when: the test fails, the z test succeeds, the z test fails
- **Application:** Mask out an irregularly shaped region of the screen

Raster Operations

- **Depth Test**

- Check if the fragment is closer to the camera than what is in the depth buffer for the pixel

- **Blending**

- Combines the incoming R, G, B, A values with what is in the color buffer
- The result depends on the incoming A and the A value in the color buffer
- **Application:** Transparency algorithms

- **Dithering**

- Improves color resolution on systems with small number of color bitplanes

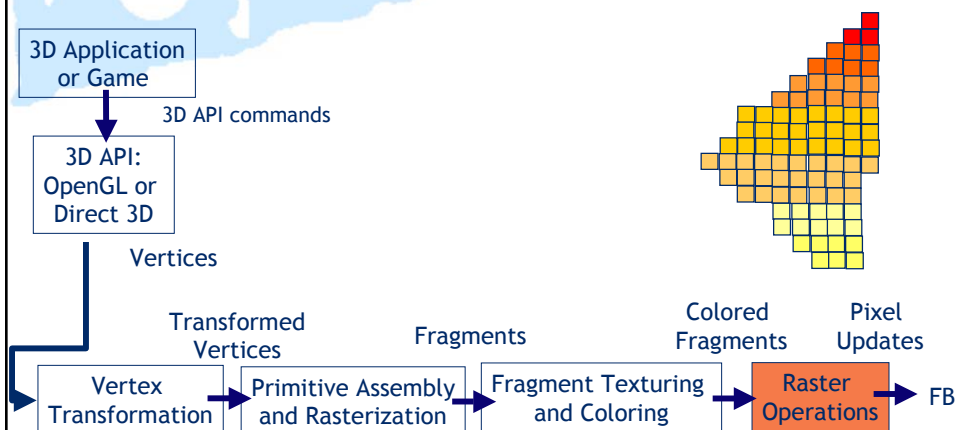
- **Logical Operations**

- Allows logical operations involving the incoming fragment values (s) and the ones in the color buffer (d)
- Ex.: AND ($s \wedge d$), OR ($s \vee d$), NAND ($\neg(s \wedge d)$), NOR, XOR, COPY (s), ...



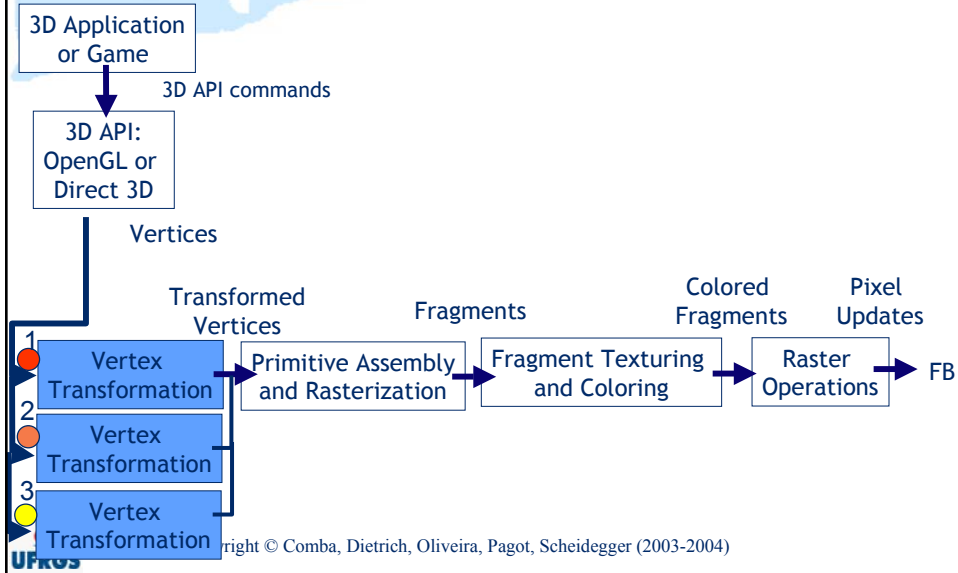
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Graphics Hardware Pipeline

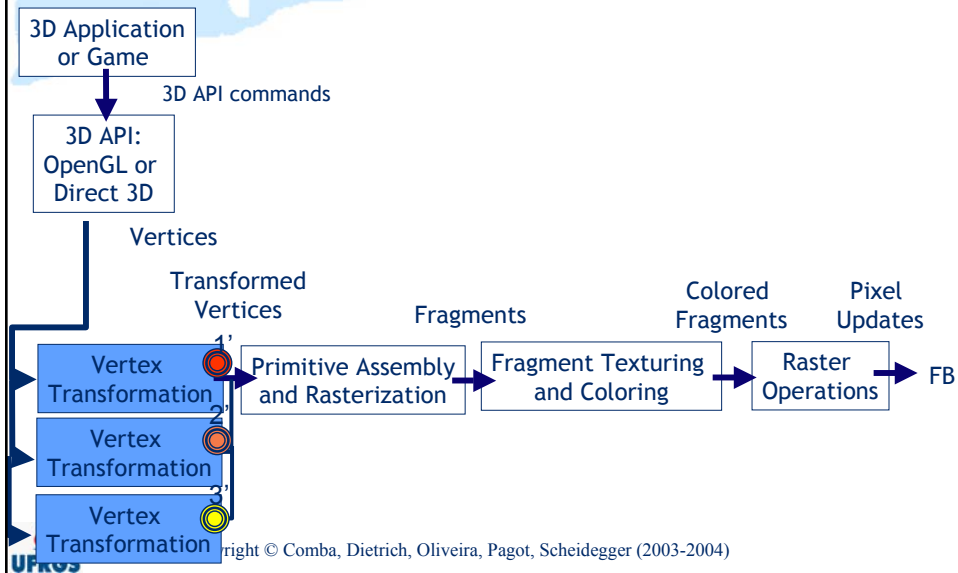


Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

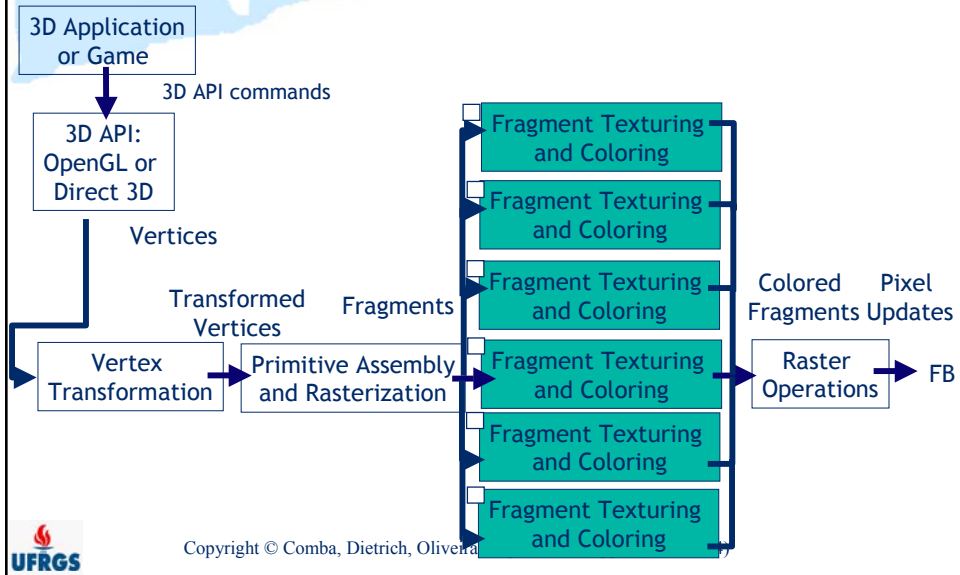
Graphics Hardware Pipeline: Parallelization



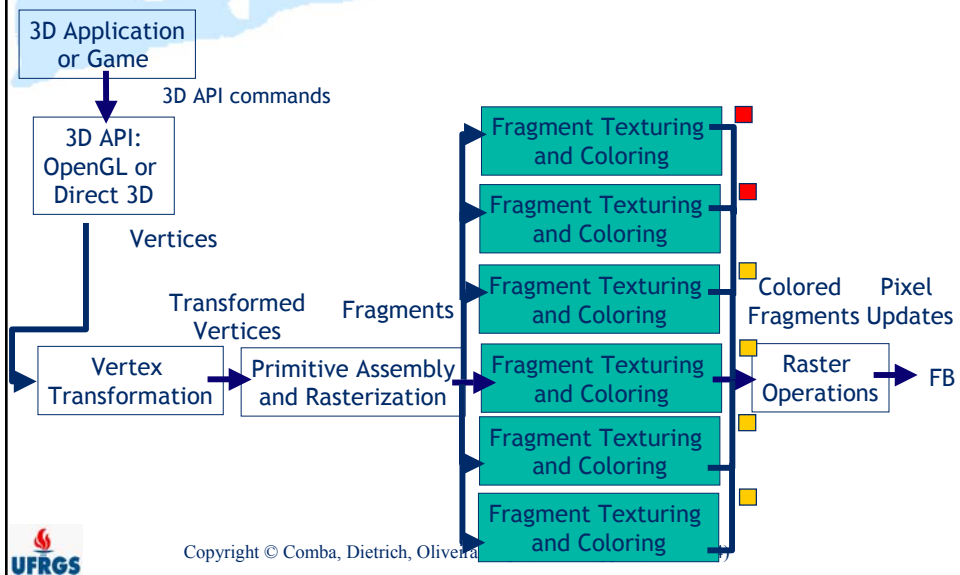
Graphics Hardware Pipeline: Parallelization

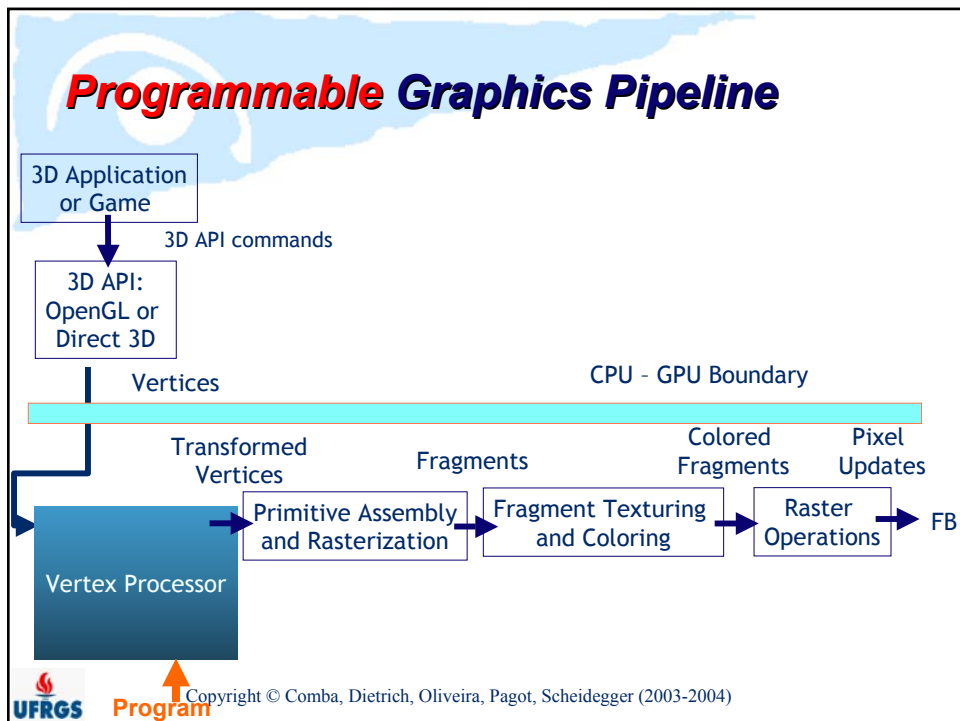
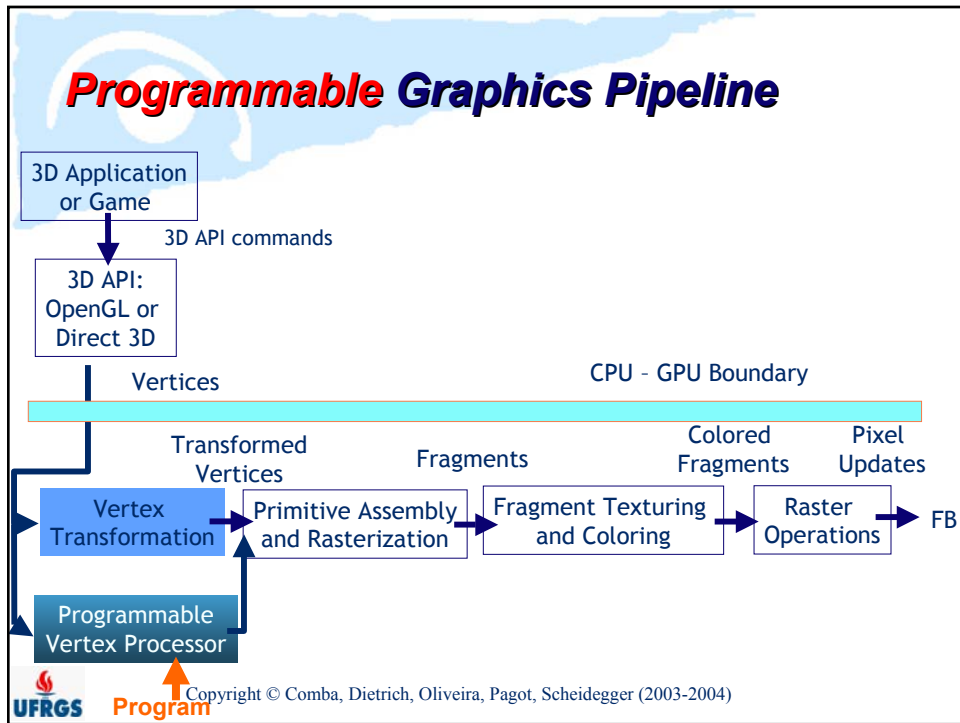


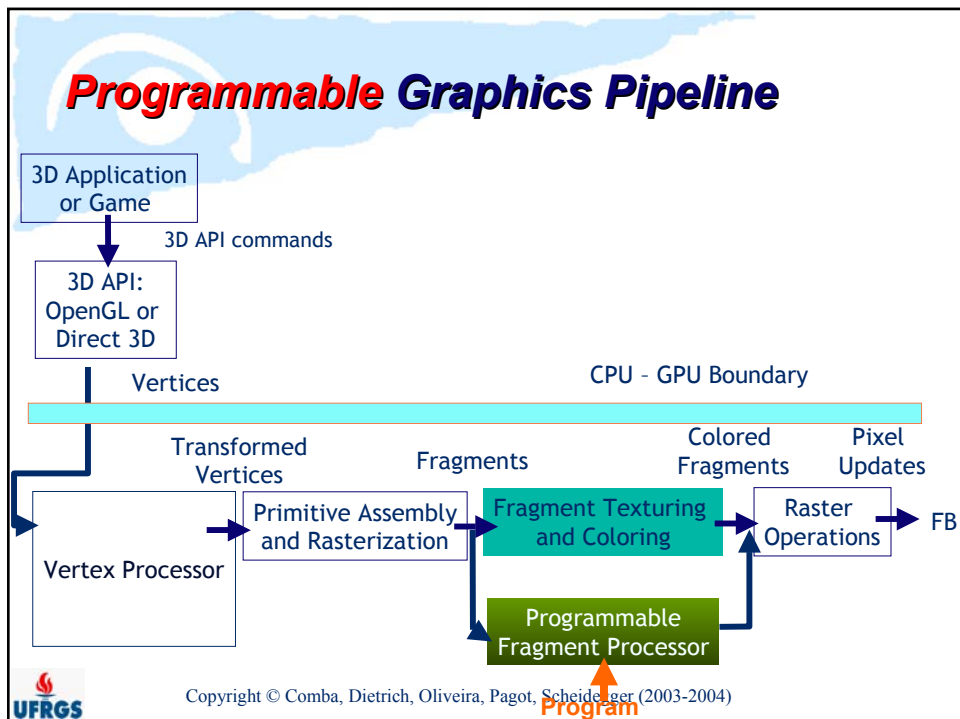
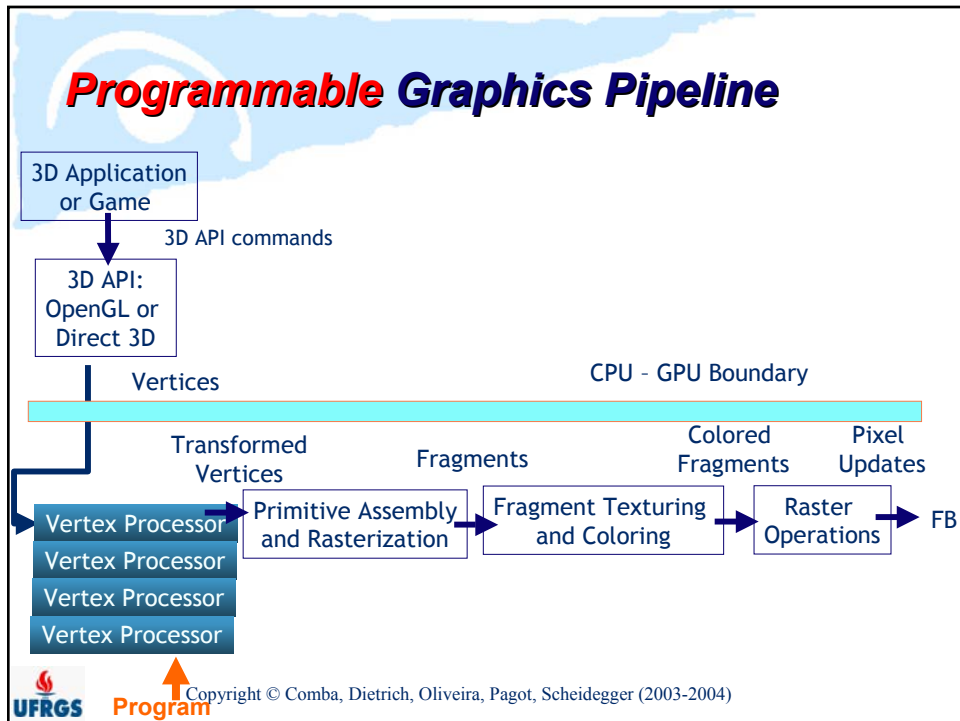
Graphics Hardware Pipeline: Parallelization



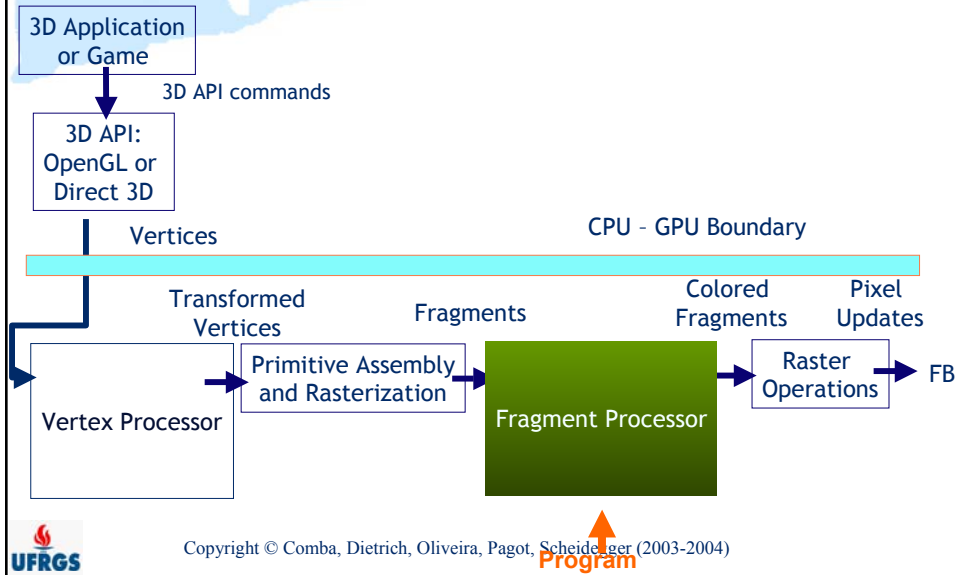
Graphics Hardware Pipeline: Parallelization



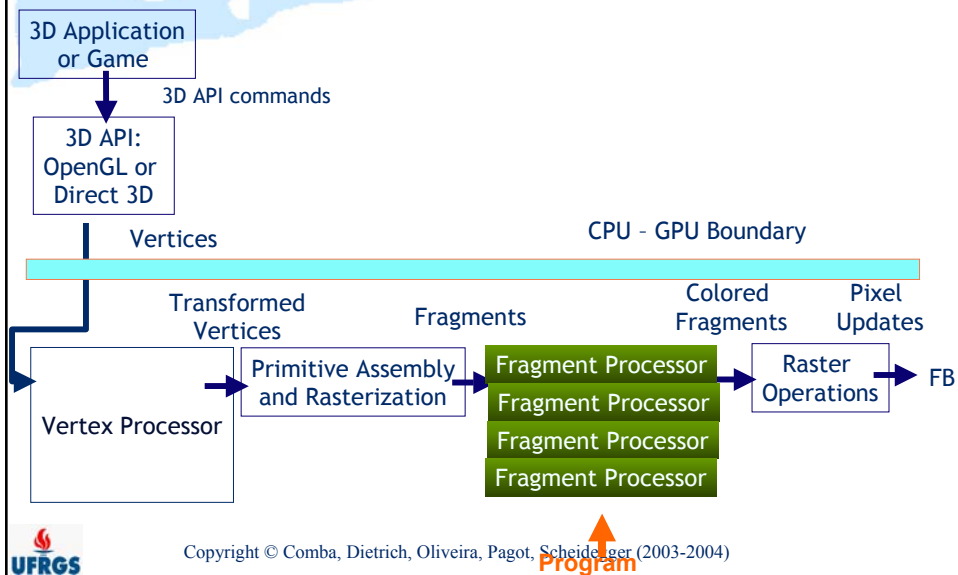




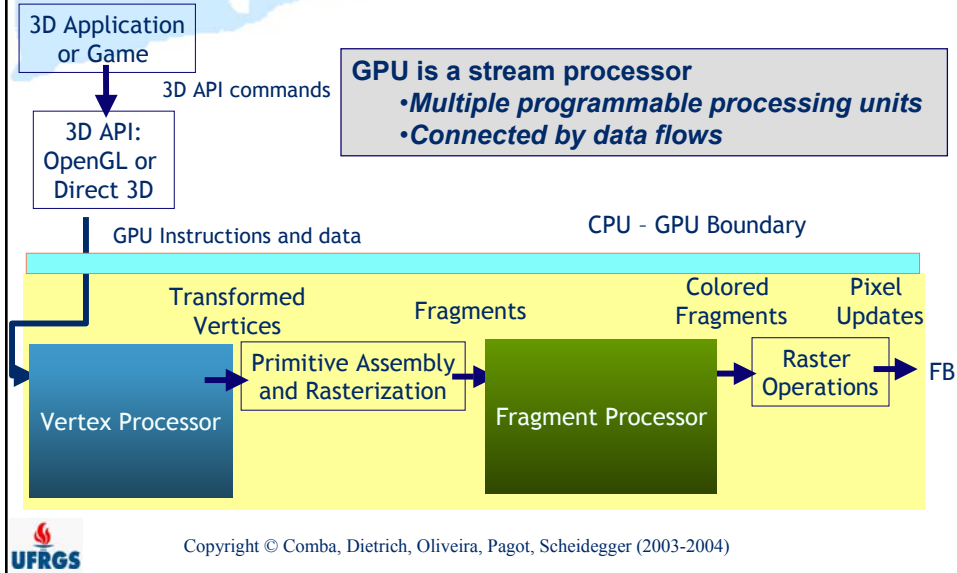
Programmable Graphics Pipeline



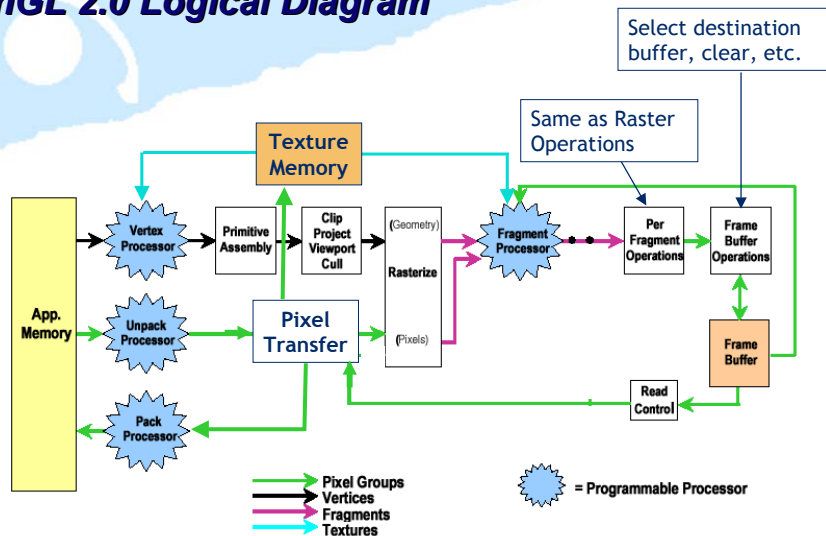
Programmable Graphics Pipeline



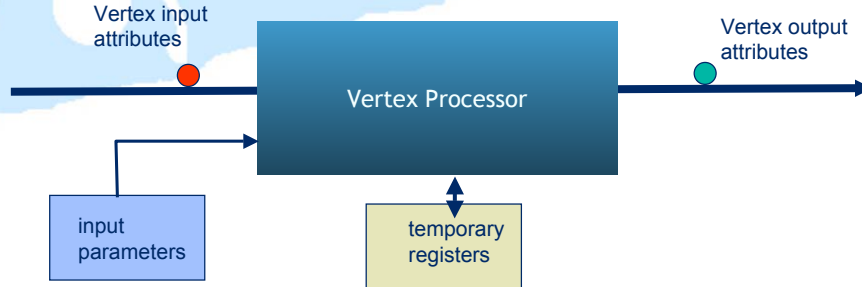
Graphics Hardware Pipeline: The Big Picture



OpenGL 2.0 Logical Diagram



Vertex Processor



- Graphics Processing Unit

- ~ Instruction Set

- ~ Registers

- Does not create or delete vertices

- ~ 1 vertex in and 1 vertex out

- No topological information provided

- ~ No edge, face, nor neighboring vertex info

- Programs can be dynamically loadable

Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)



Vertex Shader Block Diagram

- Performs operations on the data using an ALU

- Vertex data are defined by the application

- Major Output Registers:

- oPos** – Position register

- position in homogeneous clipping space

- oD0** – Diffuse color register

- Interpolated and written to the input color register 0 of the pixel shader

- oD1** – Specular color register

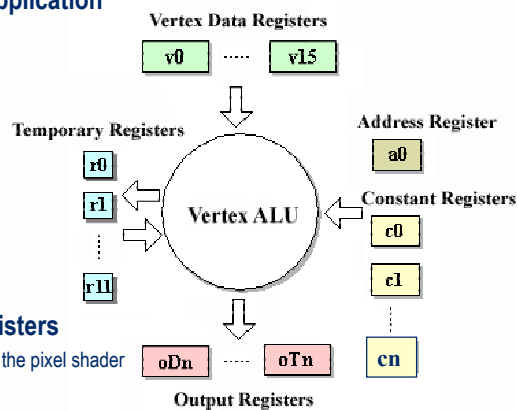
- Interpolated and written to the input color register 1 of the pixel shader

- oT0-oT7** – Texture coordinate registers

- Interpolated and used as texture coordinates in the pixel shader

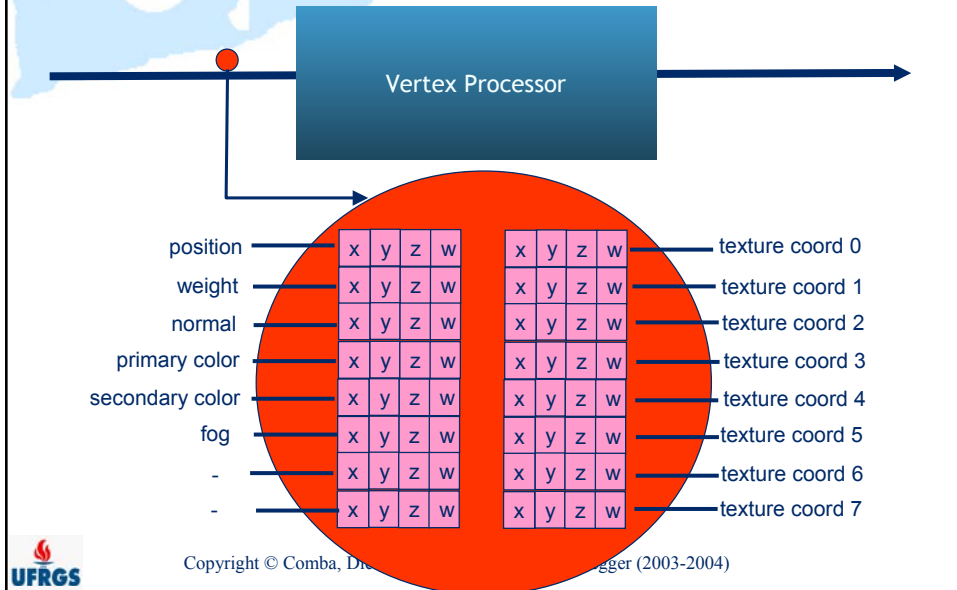
- oFog** – Fog value register

- oPts** – Point size register

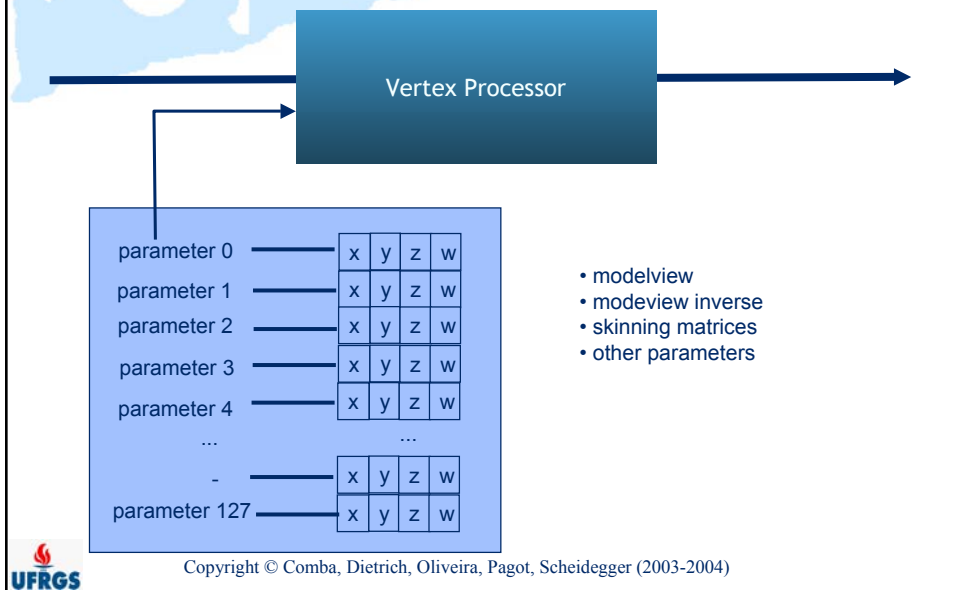


Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

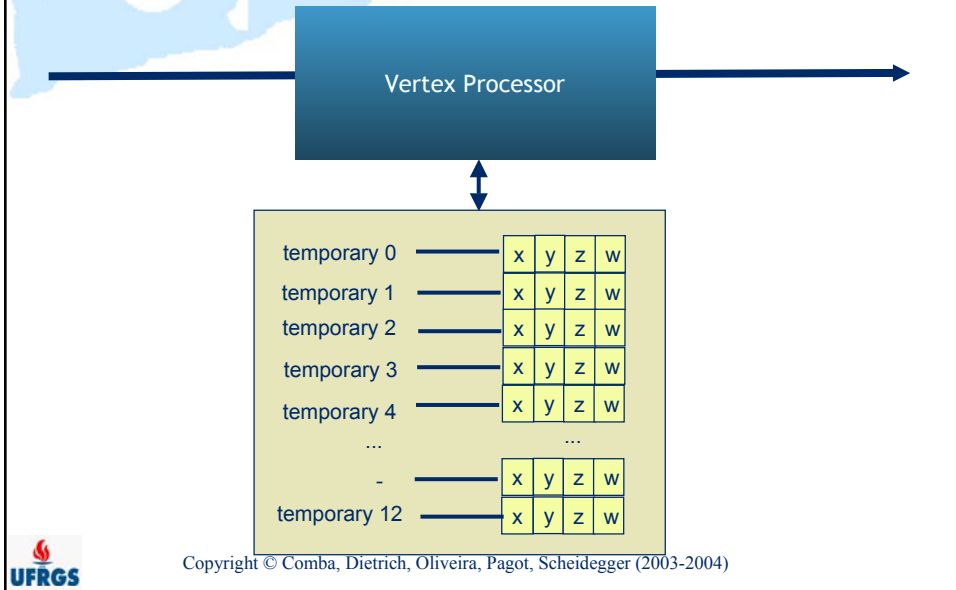
Vertex Processor Inputs: Vertex Attributes



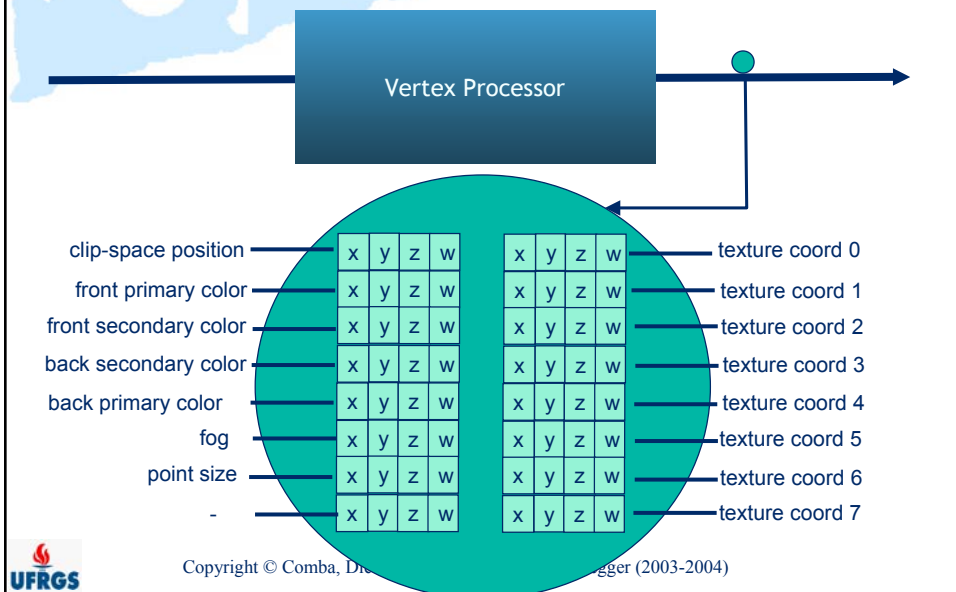
Vertex Processor Inputs: Constant Registers

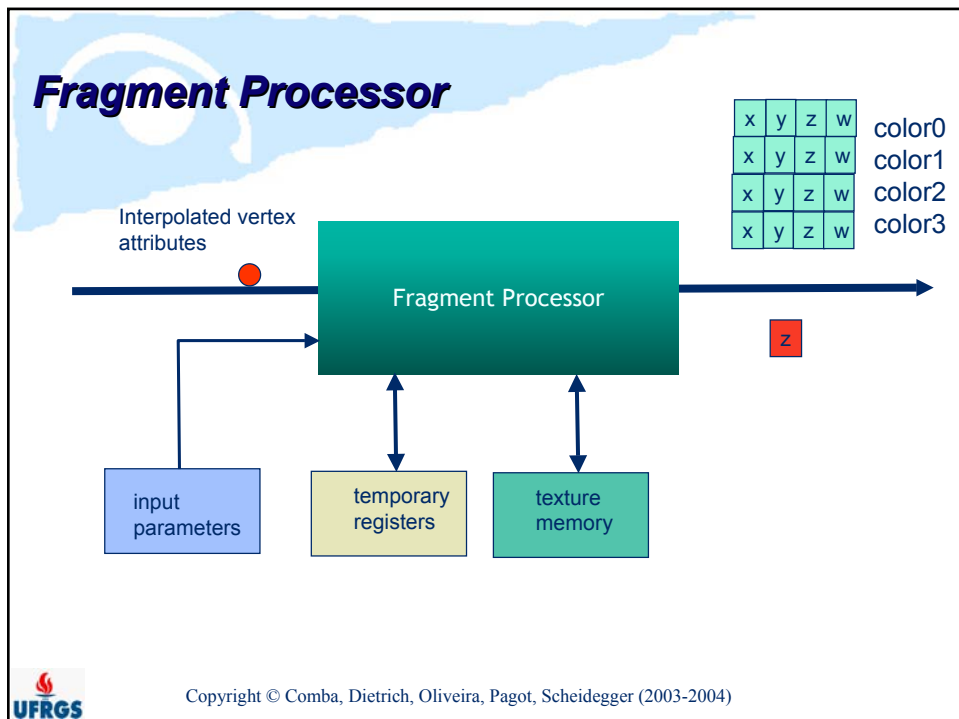
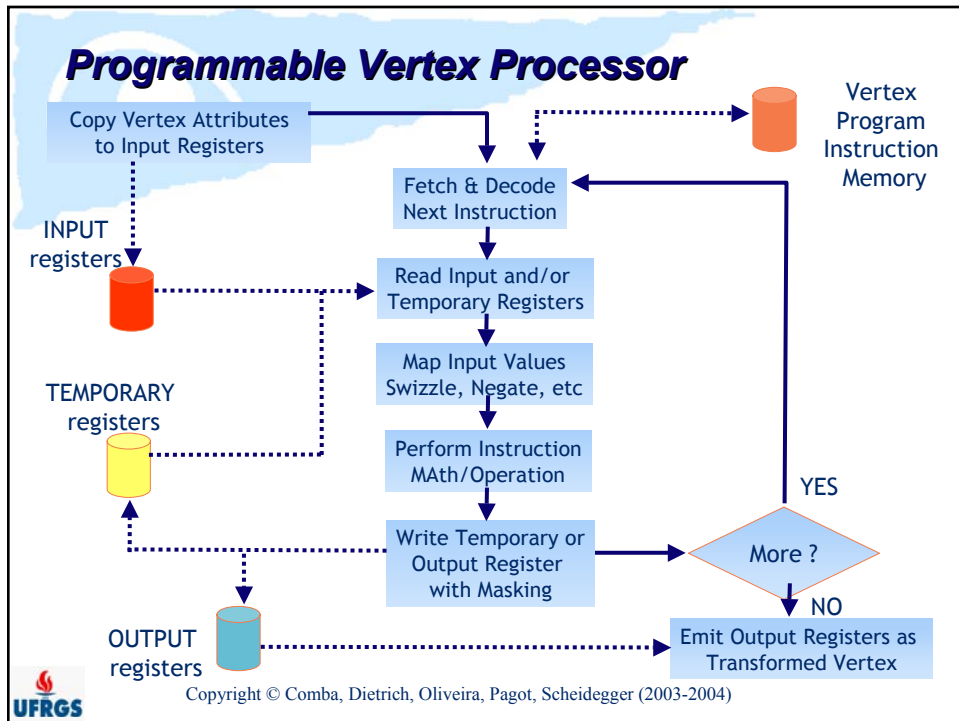


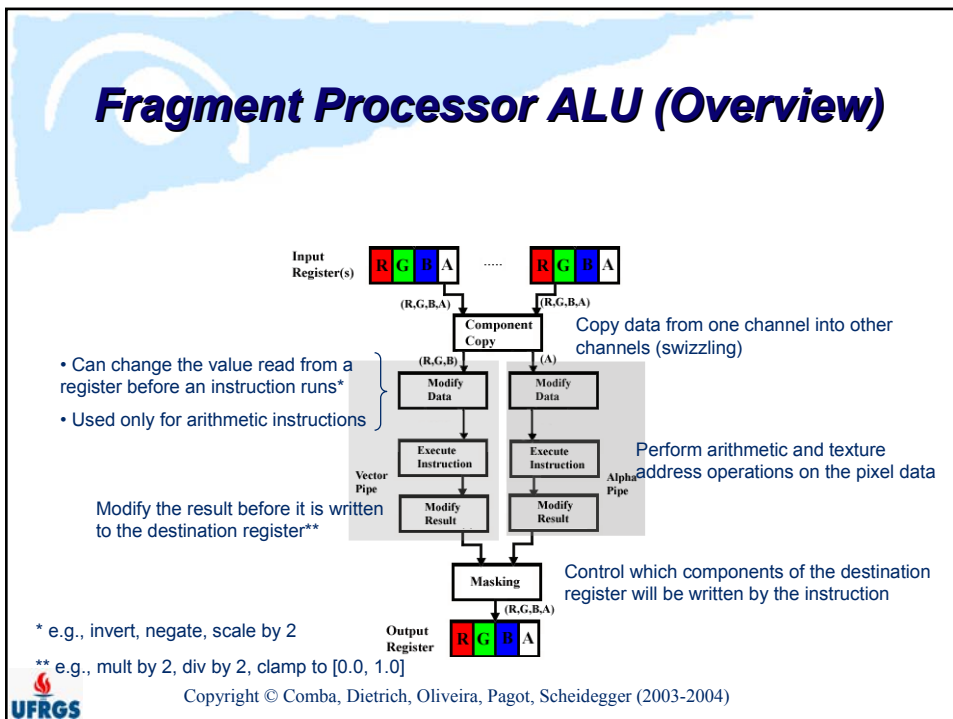
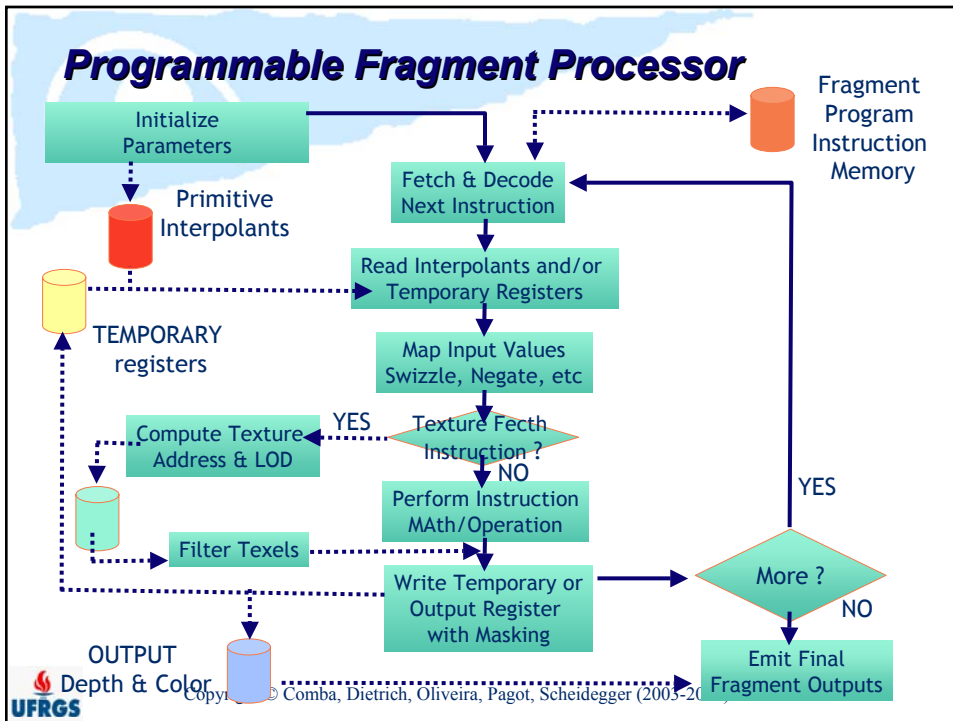
Vertex Processor Inputs: Temporary Registers



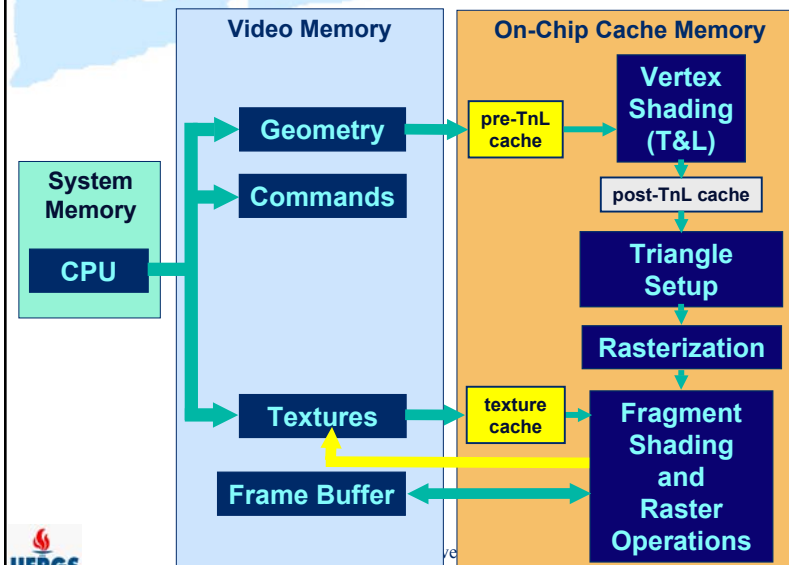
Vertex Processor Output: Vertex Attributes





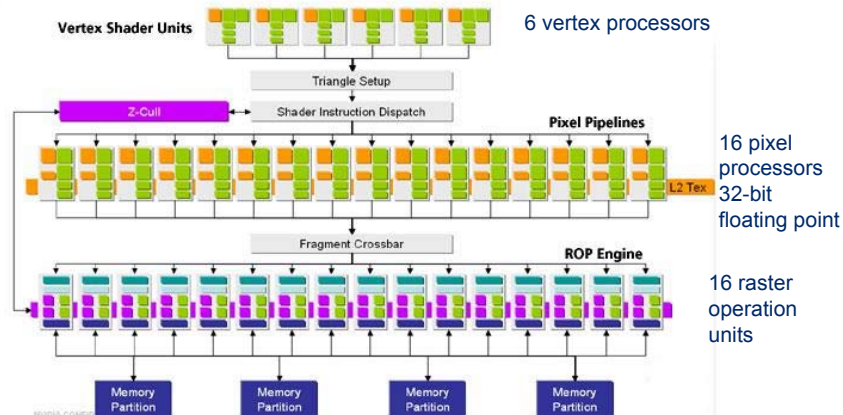


Graphics Hardware Block Diagram



Example: GeForce 6800 Ultra

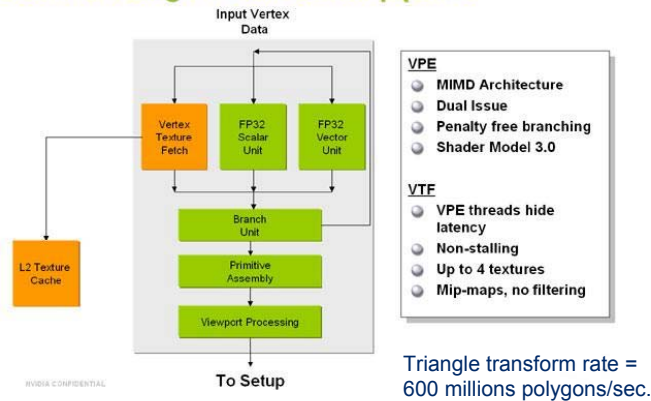
GeForce 6800 series 3D Pipeline



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Example: GeForce 6800 Ultra

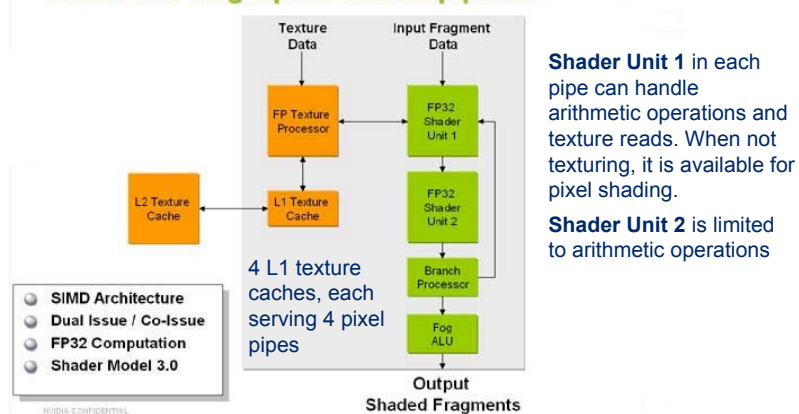
Detail of a single vertex shader pipeline



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Example: GeForce 6800 Ultra

Detail of a single pixel shader pipeline



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Example: GeForce 6800 Ultra

new features of the pixel shader engine

- Full Support for shader model 3.0
- 65,535 length pixel shader programs
- Dynamic Flow control - Loops & Branching, Call & Return, Subroutines
- Highest precision pixel shading - Native/optimized FP32 processing
- Flexible data type support - FP32, FP16 operand & texture formats
- Multiple Render Target Support



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)

Shader Model 3.0

Vertex shader feature	Shader 2.0	Shader 3.0	Description
Shader length	256 Instructions	65535 instructions	More instructions allow more detailed character lighting and animation
Dynamic branching	No	Yes	Saves performance by skipping animation and calculations on irrelevant vertices
Vertex texture	No	Any number of lookups from up to 4 textures	Allows displacement mapping, particle effects
Instancing support	No	Required	Allows many varied objects to be drawn with only a single command



Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2004)