The Hardware Rendering Pipeline

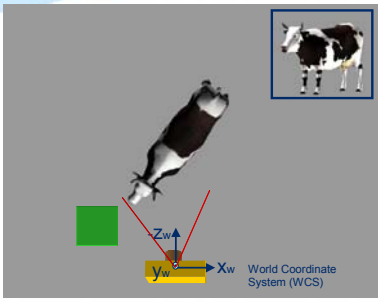
Rendering Images

Geometric model

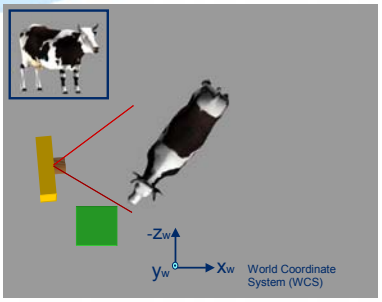
rendering algorithm

Picture

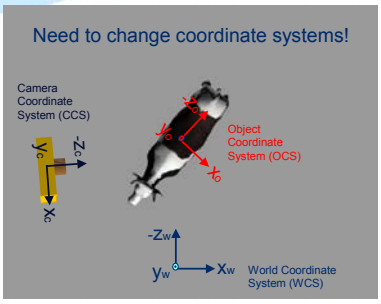
The Visualization Problem



The Visualization Problem

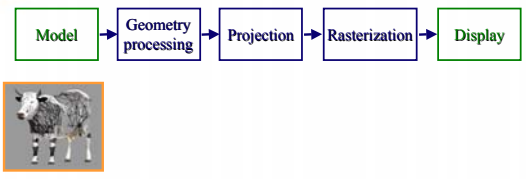


The Visualization Problem

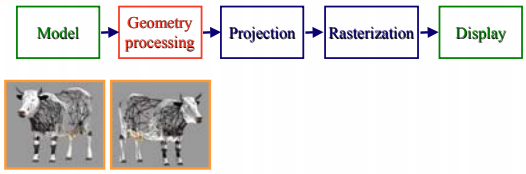


Rendering Pipeline (Overview)

Z-buffer and Gouraud shading

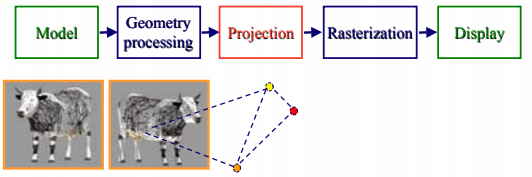


Rendering Pipeline (Overview)



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

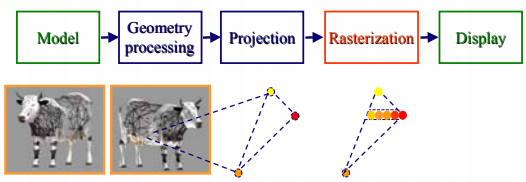
Rendering Pipeline (Overview)



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

- Map from CCS to Screen CS
- Persp. division

Rendering Pipeline (Overview)

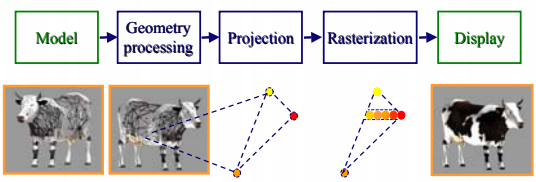


- Map from OCS to CCS
- Clipping in 3D
- Lighting

- Map from CCS to Screen CS
- Persp. division

- Color
- Visibility

Rendering Pipeline (Overview)



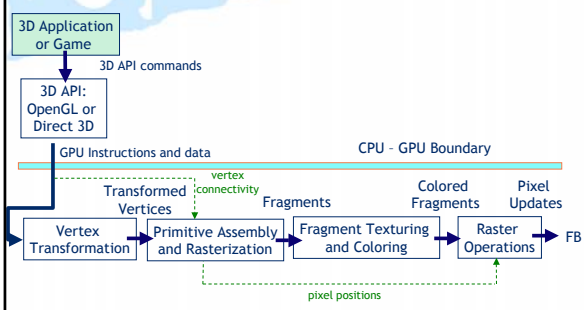
- Map from OCS to CCS
- Clipping in 3D
- Lighting

- Map from CCS to Screen CS
- Persp. division

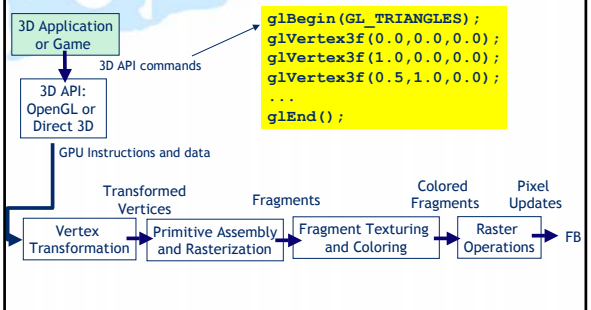
- Color
- Visibility

- FB display

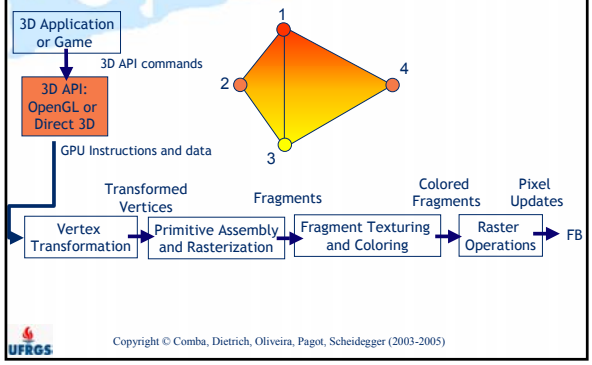
Graphics Hardware *Fixed* Pipeline



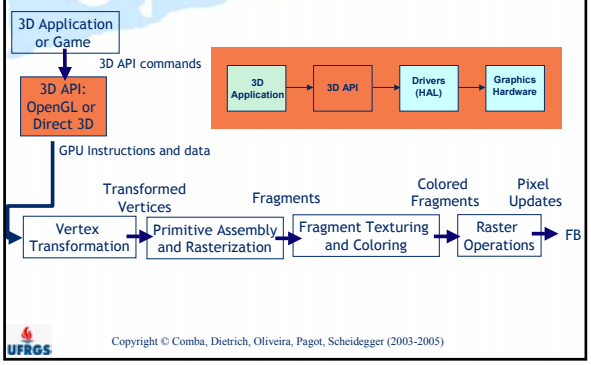
Graphics Hardware Pipeline



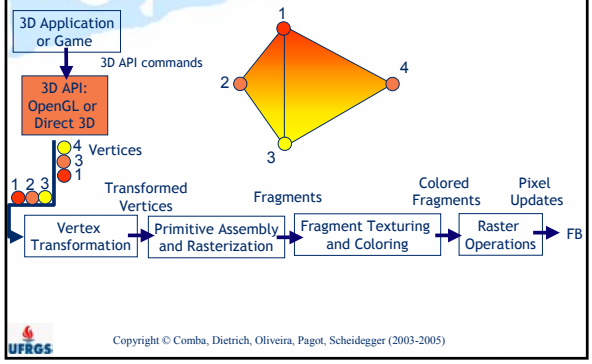
Graphics Hardware Pipeline



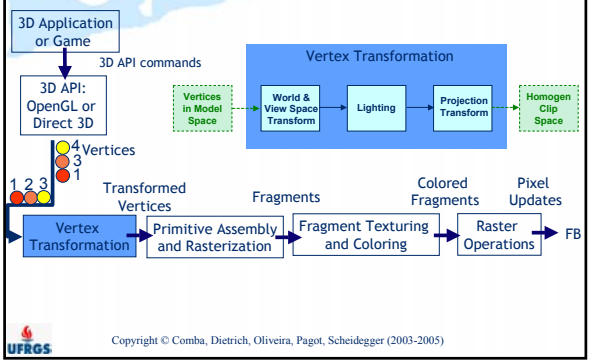
Graphics Hardware Pipeline



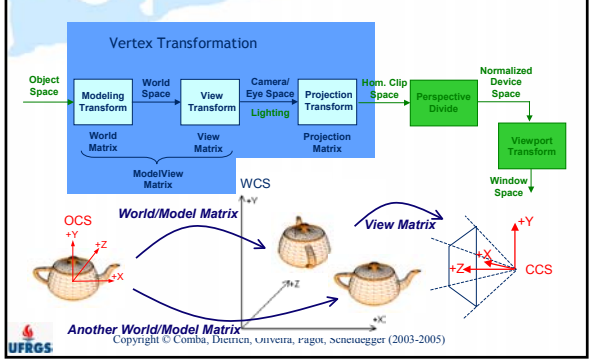
Graphics Hardware Pipeline



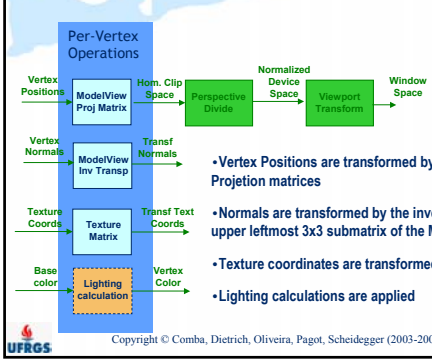
Graphics Hardware Pipeline



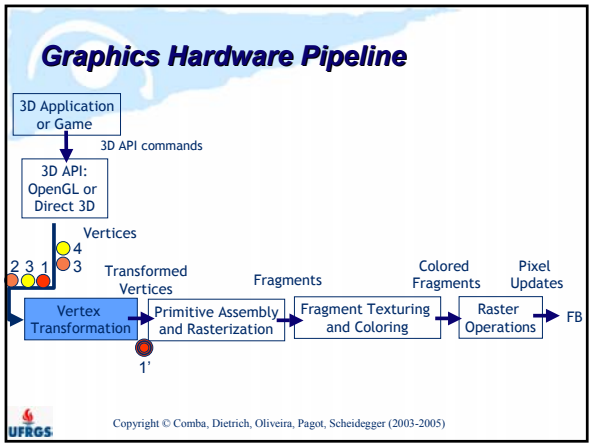
Vertex Transformations

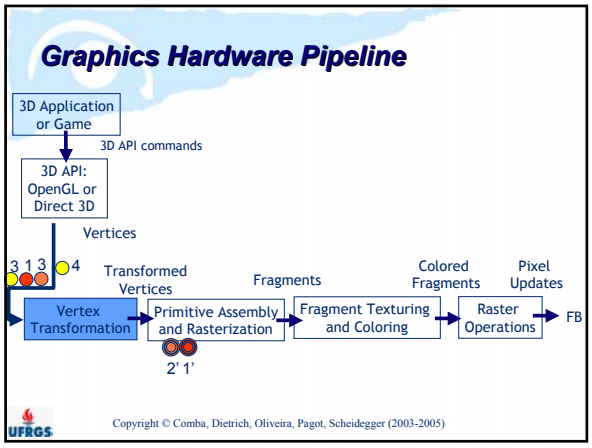


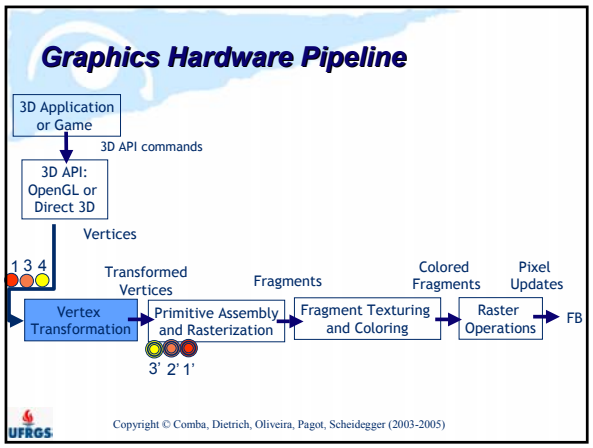
Per-Vertex Operations: Closer Look

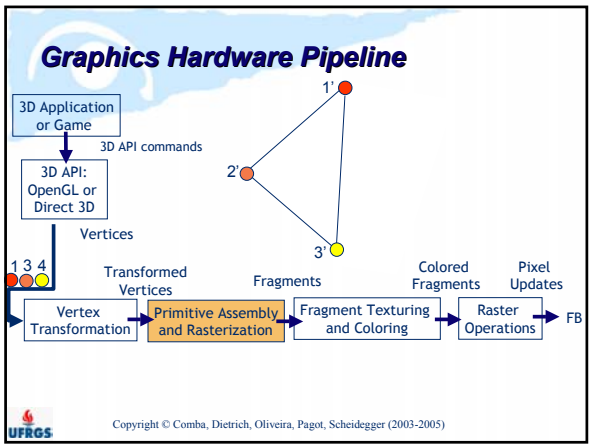


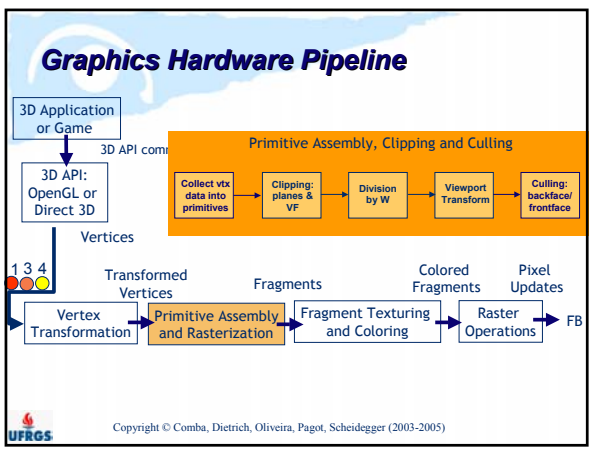
- Vertex Positions are transformed by the ModelView and Projetion matrices
- Normals are transformed by the inverse transpose of the upper leftmost 3x3 submatrix of the ModelView
- Texture coordinates are transformed by texture matrices
- Lighting calculations are applied

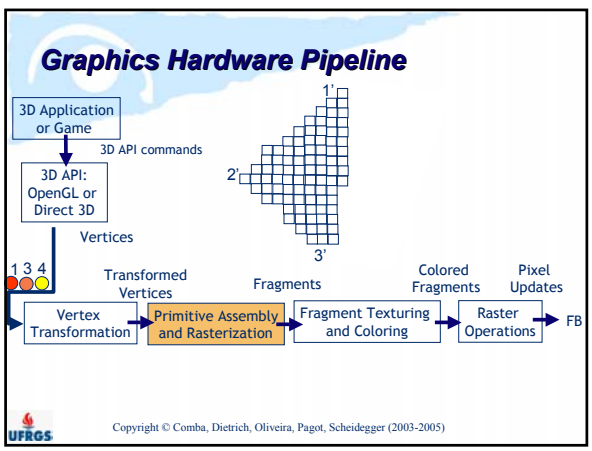

Graphics Hardware Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
2 3 1
4
3
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
1'
UFRGS
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Graphics Hardware Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
3 1 3
4
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
2' 1'
UFRGS
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Graphics Hardware Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
1 3 4
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
3' 2' 1'
UFRGS
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Graphics Hardware Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
1 3 4
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
1'
2'
3'
UFRGS
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)

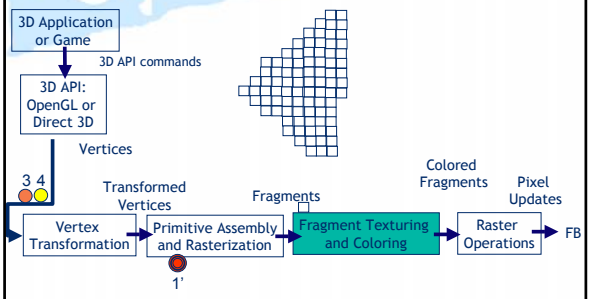

Graphics Hardware Pipeline
3D Application or Game
3D API comm
3D API: OpenGL or Direct 3D
Vertices
1 3 4
Primitive Assembly, Clipping and Culling
Collect vtx data into primitives
Clipping: planes & VF
Division by W
Viewport Transform
Culling: backface/ frontface
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
UFRGS
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Graphics Hardware Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
1 3 4
1'
2'
3'
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
UFRGS
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)

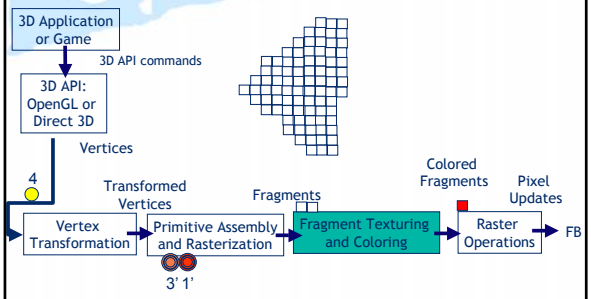
Rasterization

- 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

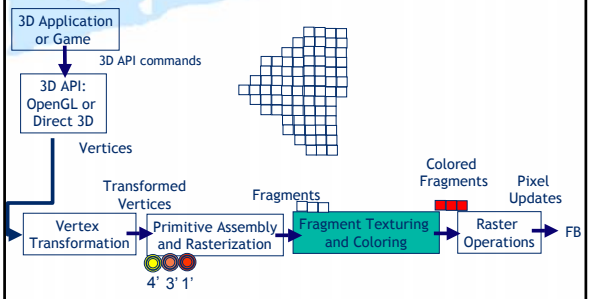
Graphics Hardware Pipeline



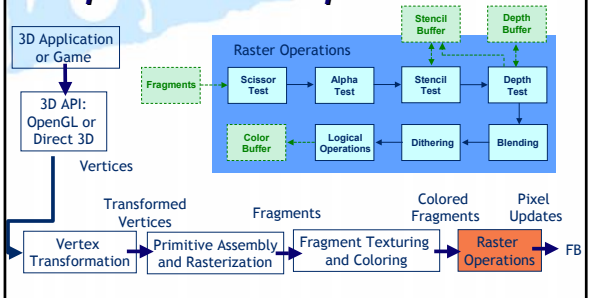
Graphics Hardware Pipeline



Graphics Hardware Pipeline



Graphics Hardware Pipeline



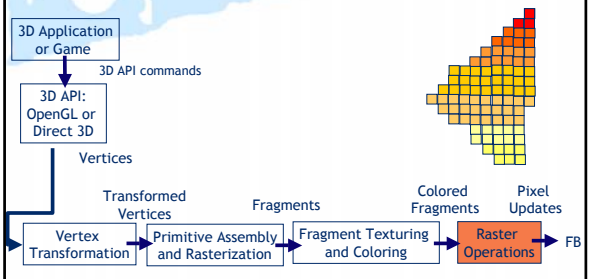
Raster Operations

- **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, $<$, $\leq$, $>$, $\geq$, $=$, $\neq$)
  - **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

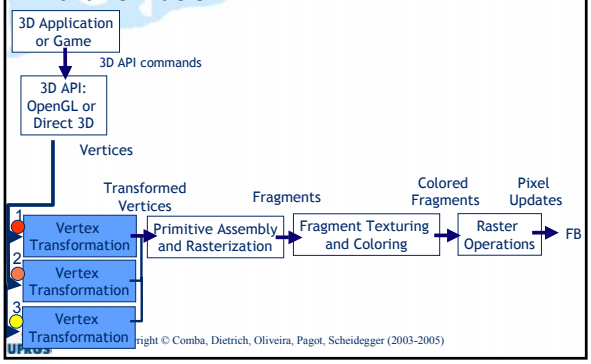
Raster Operations

- **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), ...

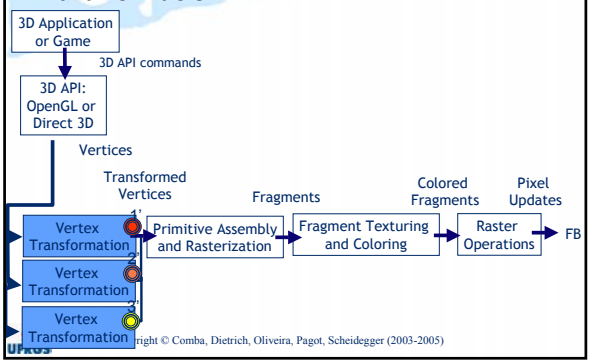
Graphics Hardware Pipeline



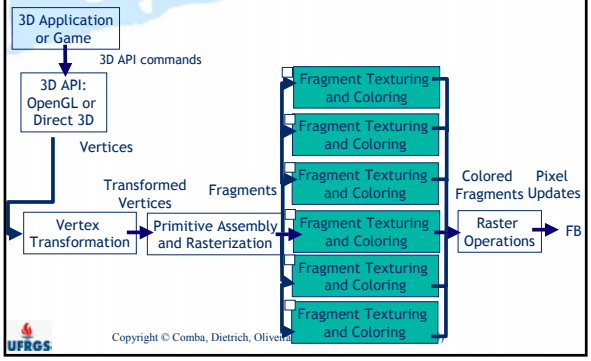
Graphics Hardware Pipeline: Parallelization



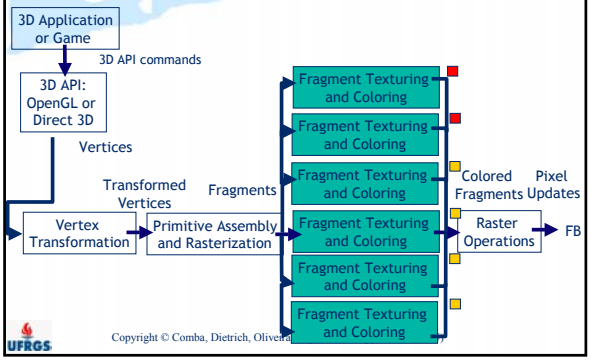
Graphics Hardware Pipeline: Parallelization

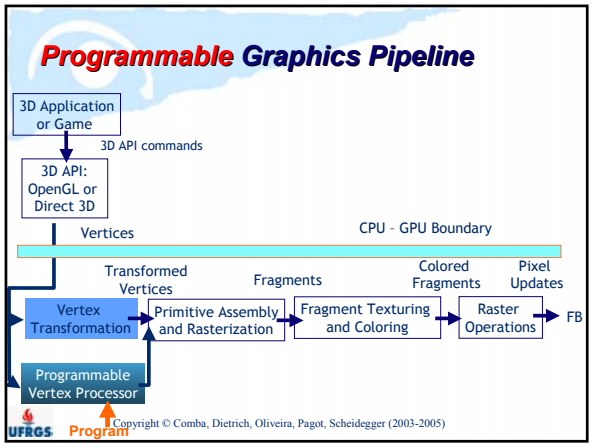


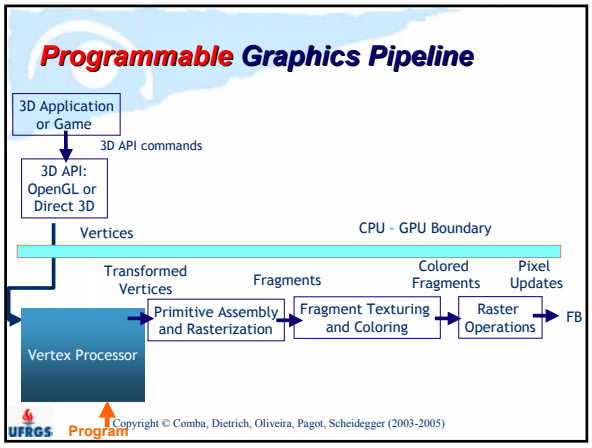
Graphics Hardware Pipeline: Parallelization

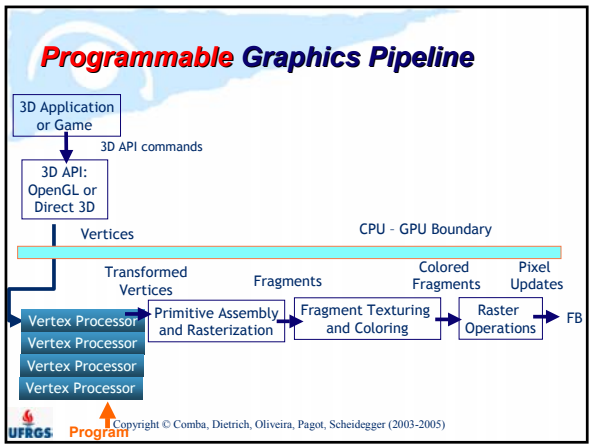


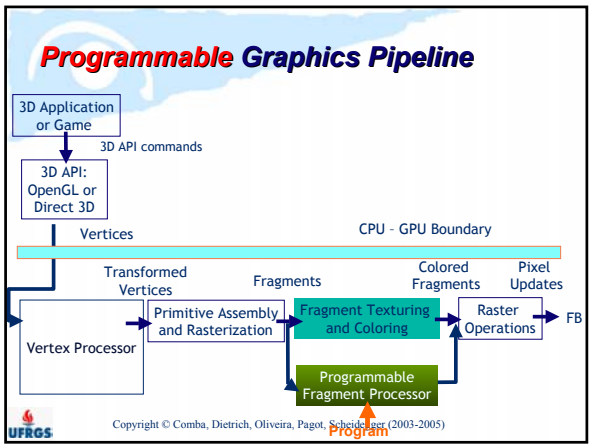
Graphics Hardware Pipeline: Parallelization

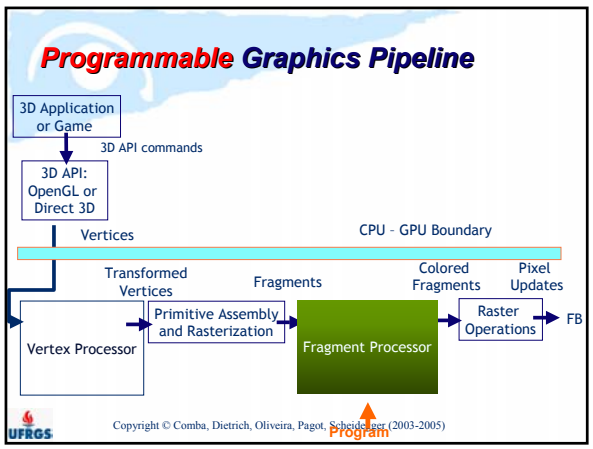


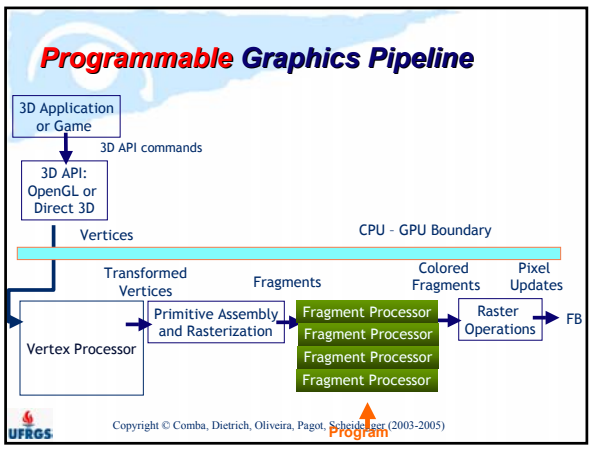

Programmable Graphics Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
CPU - GPU Boundary
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Transformation
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
Programmable Vertex Processor
Program
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Programmable Graphics Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
CPU - GPU Boundary
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Processor
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
Program
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)

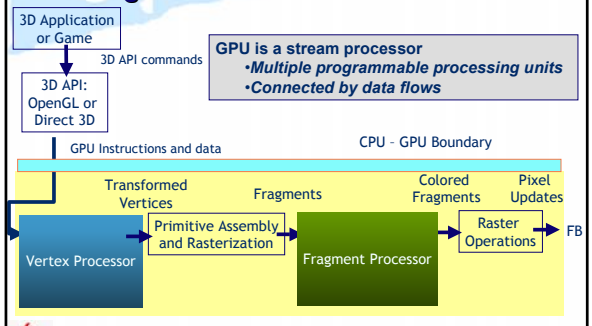

Programmable Graphics Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
CPU - GPU Boundary
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Processor
Vertex Processor
Vertex Processor
Vertex Processor
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
Program
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Programmable Graphics Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
CPU - GPU Boundary
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Processor
Primitive Assembly and Rasterization
Fragment Texturing and Coloring
Raster Operations
FB
Programmable Fragment Processor
Program
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)

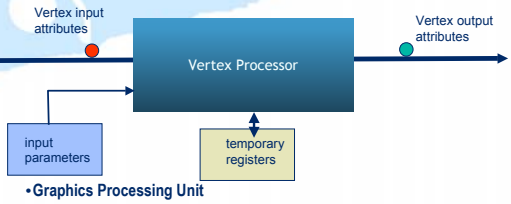

Programmable Graphics Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
CPU - GPU Boundary
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Processor
Primitive Assembly and Rasterization
Fragment Processor
Raster Operations
FB
Program
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)


Programmable Graphics Pipeline
3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D
Vertices
CPU - GPU Boundary
Transformed Vertices
Fragments
Colored Fragments
Pixel Updates
Vertex Processor
Primitive Assembly and Rasterization
Fragment Processor
Fragment Processor
Fragment Processor
Fragment Processor
Raster Operations
FB
Program
Copyright © Comba, Dietrich, Oliveira, Pagot, Scheidegger (2003-2005)

Graphics Hardware Pipeline: The Big Picture

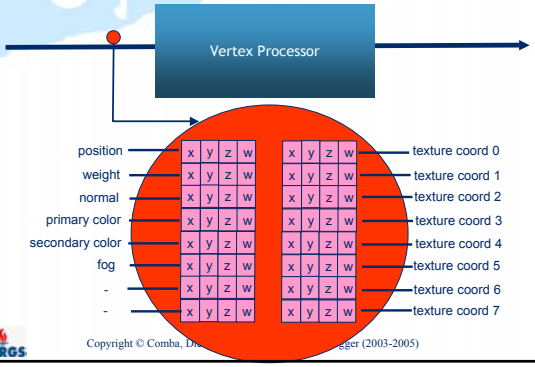


Vertex Processor

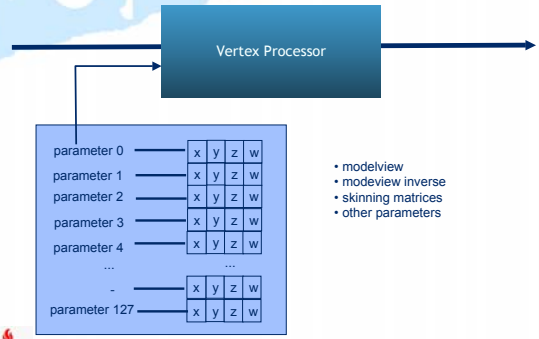


- Graphics Processing Unit
  - ~ *Instruction Set*
  - ~ *Registers*
- Does not create or delete vertices
  - ~ *1 vertex in and 1 vertex out*
- No topological information provided
  - ~ *No edge, face, nor neighboring vertex info*
- Programs can be dynamically loadable

Vertex Processor Inputs: Vertex Attributes

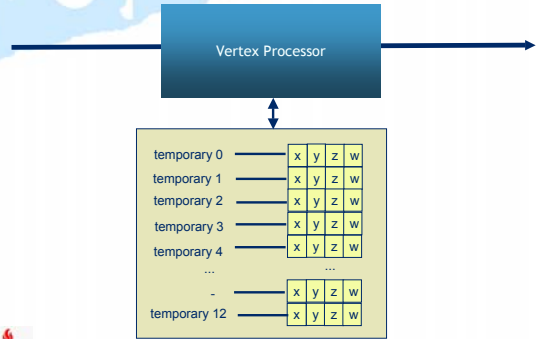


Vertex Processor Inputs: Constant Registers

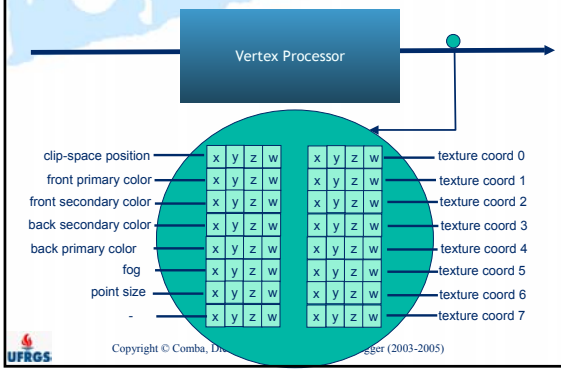


- modelview
- modeview inverse
- skinning matrices
- other parameters

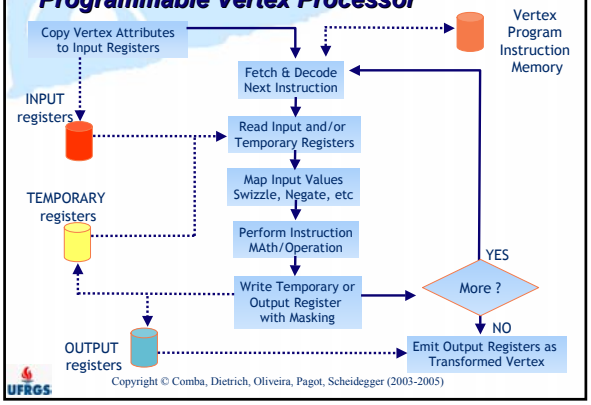
Vertex Processor Inputs: Temporary Registers



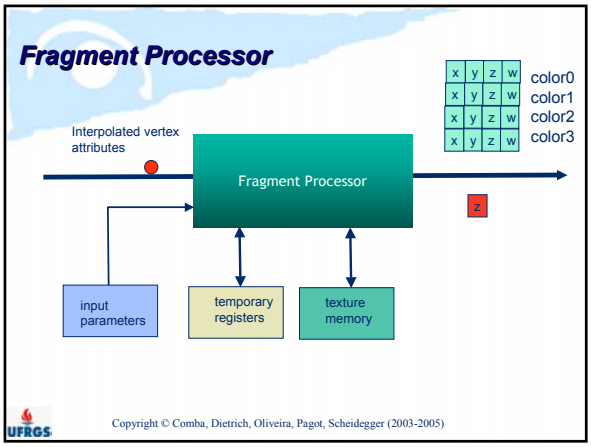
Vertex Processor Output: Vertex Attributes



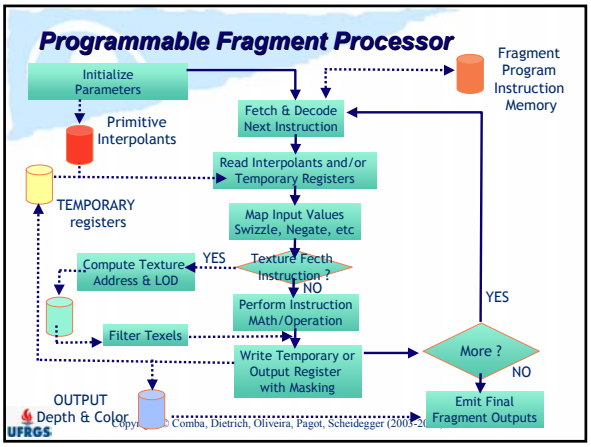
Programmable Vertex Processor



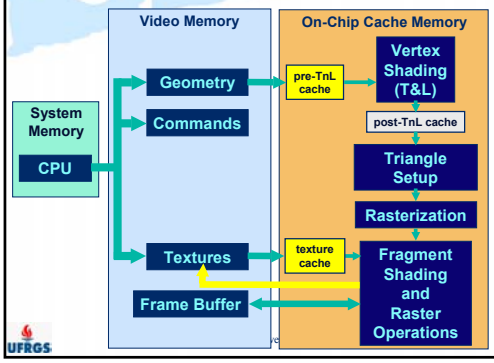
Fragment Processor



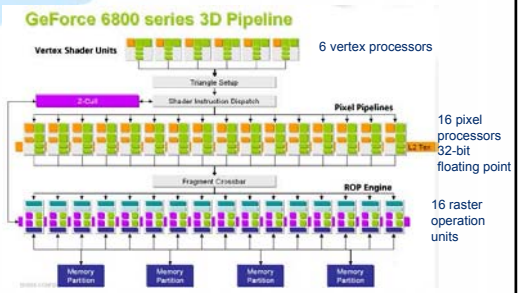
Programmable Fragment Processor



Graphics Hardware Block Diagram

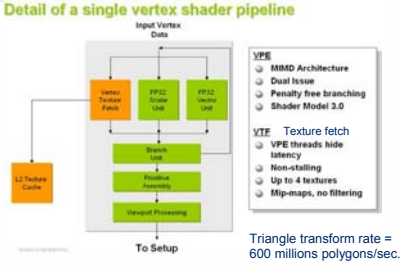


Example: GeForce 6800 Ultra



Example: GeForce 6800 Ultra

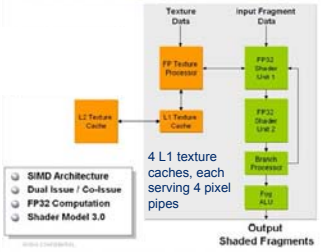
Detail of a single vertex shader pipeline



Triangle transform rate = 600 millions polygons/sec.

Example: GeForce 6800 Ultra

Detail of a single pixel shader pipeline



Shader Unit 1 in each pipe can handle arithmetic operations and texture reads. When not texturing, it is available for pixel shading.

Shader Unit 2 is limited to arithmetic operations

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

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