Real-Time Volume Graphics

[02] GPU Programming

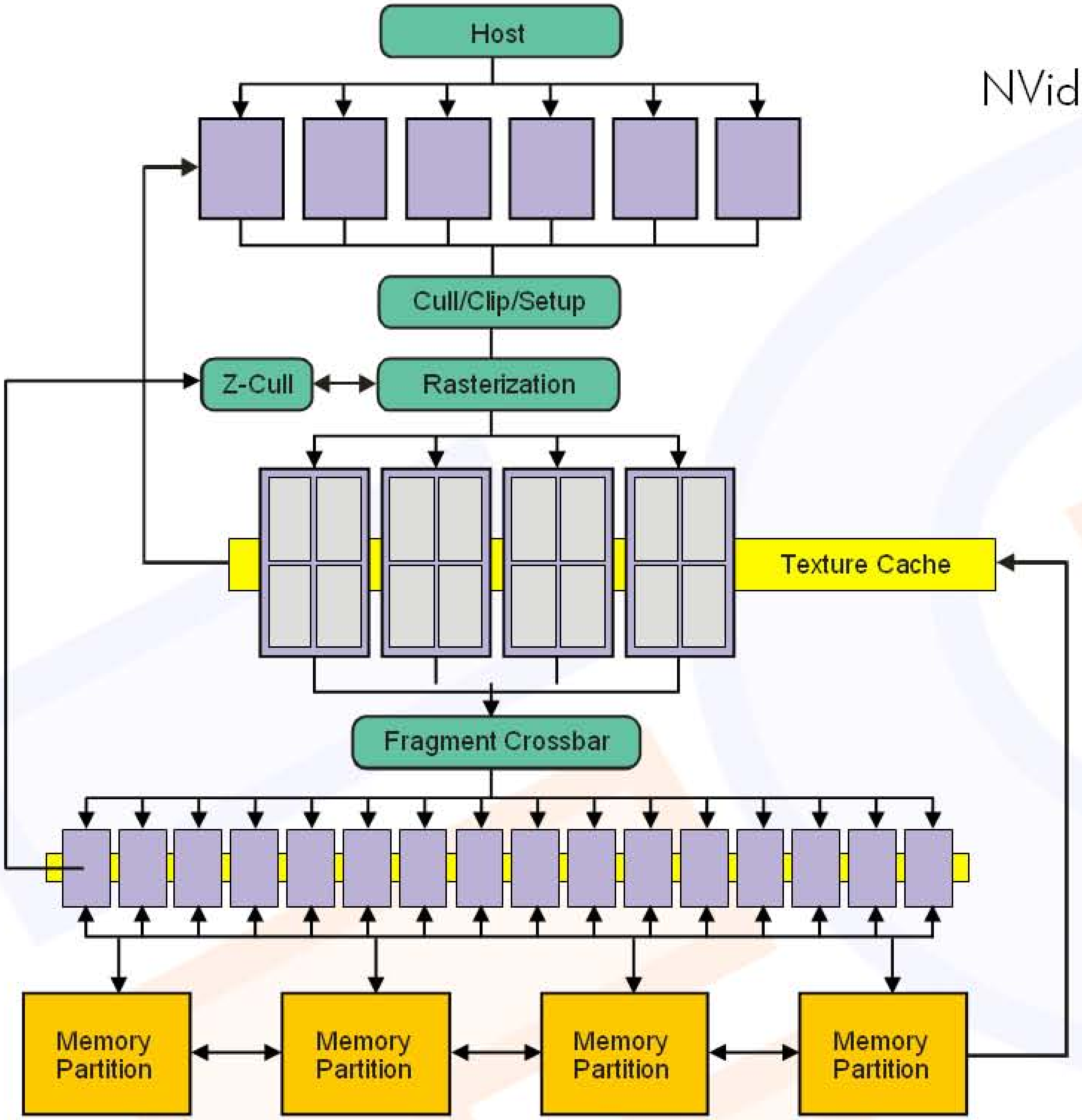
Graphics Processor

Example
NVidia Geforce6

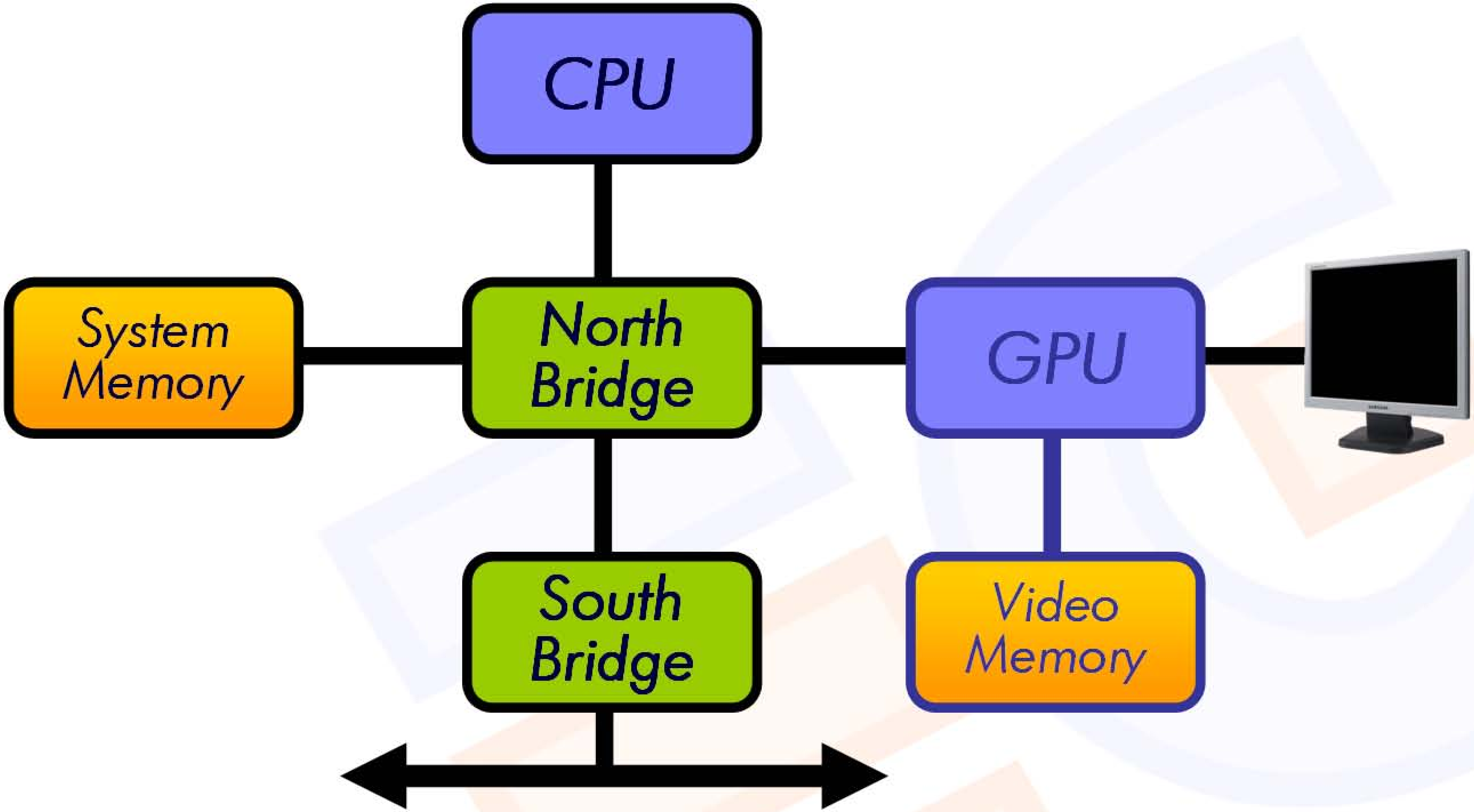
Vertex Processors

Fragment Processors

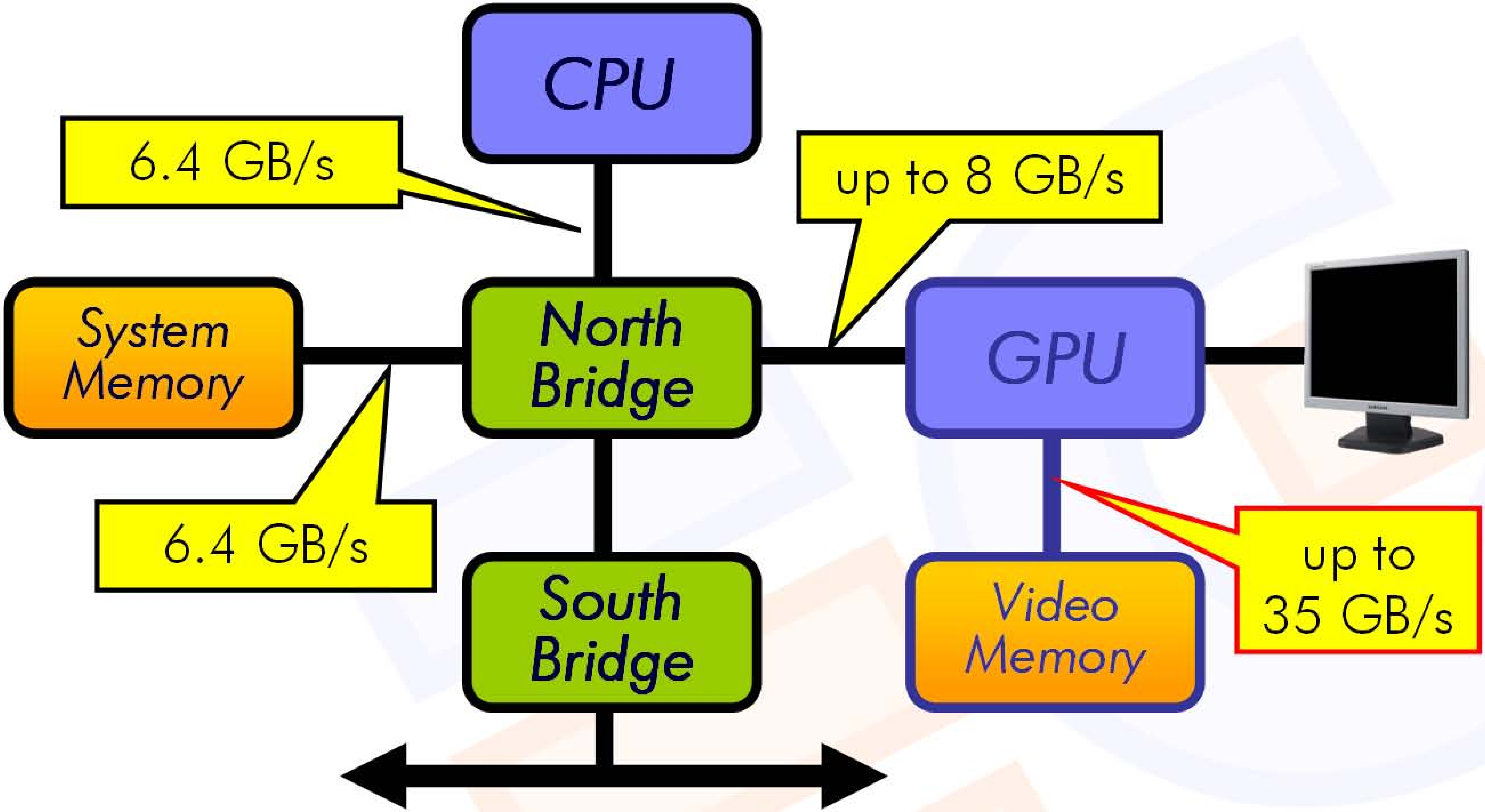
Memory Access
Z-Compare and
Blending



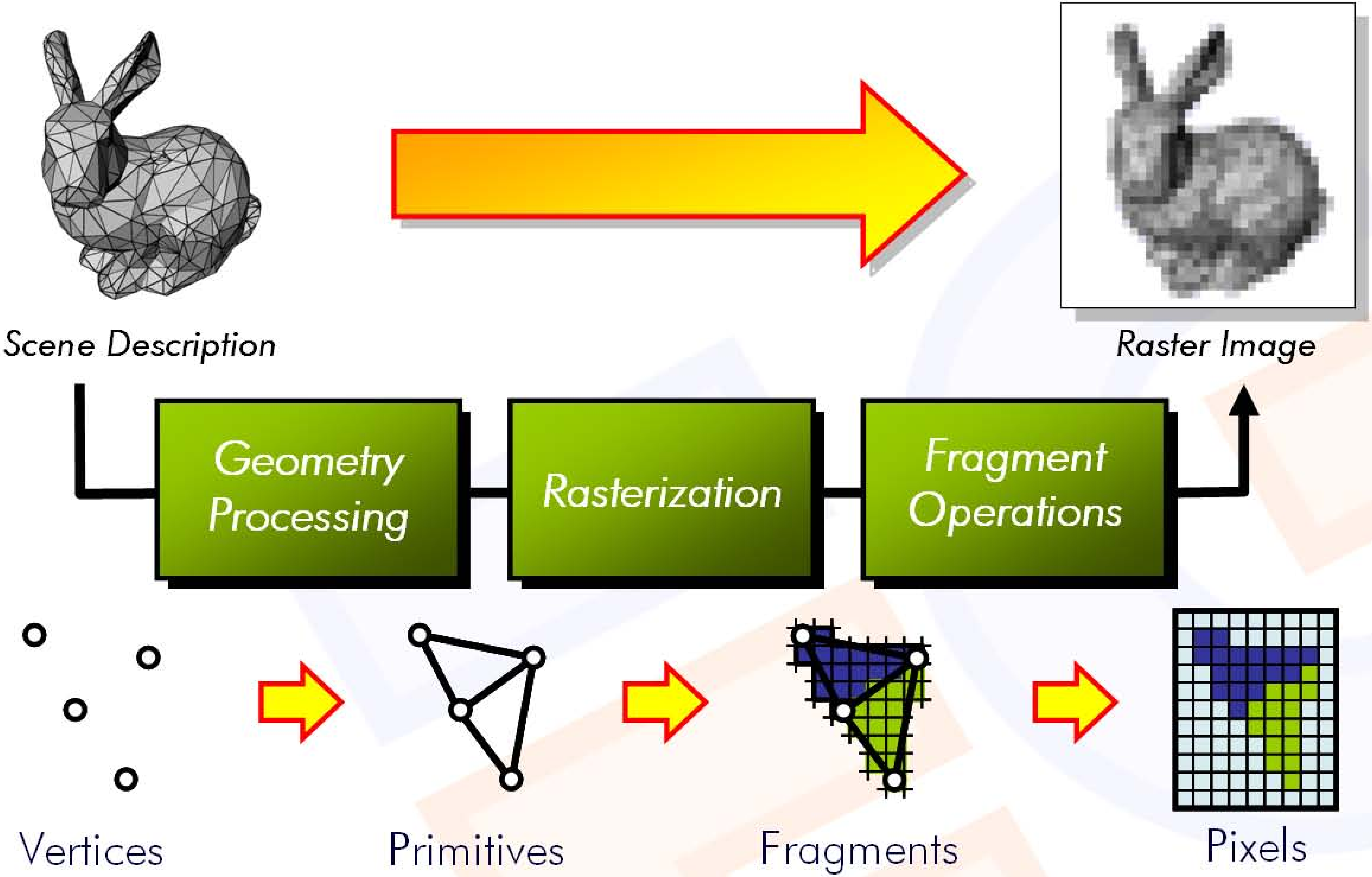
PC Architecture



PC Architecture



Graphics Hardware



Scene Description

Raster Image

Geometry Processing

Rasterization

Fragment Operations

Vertices

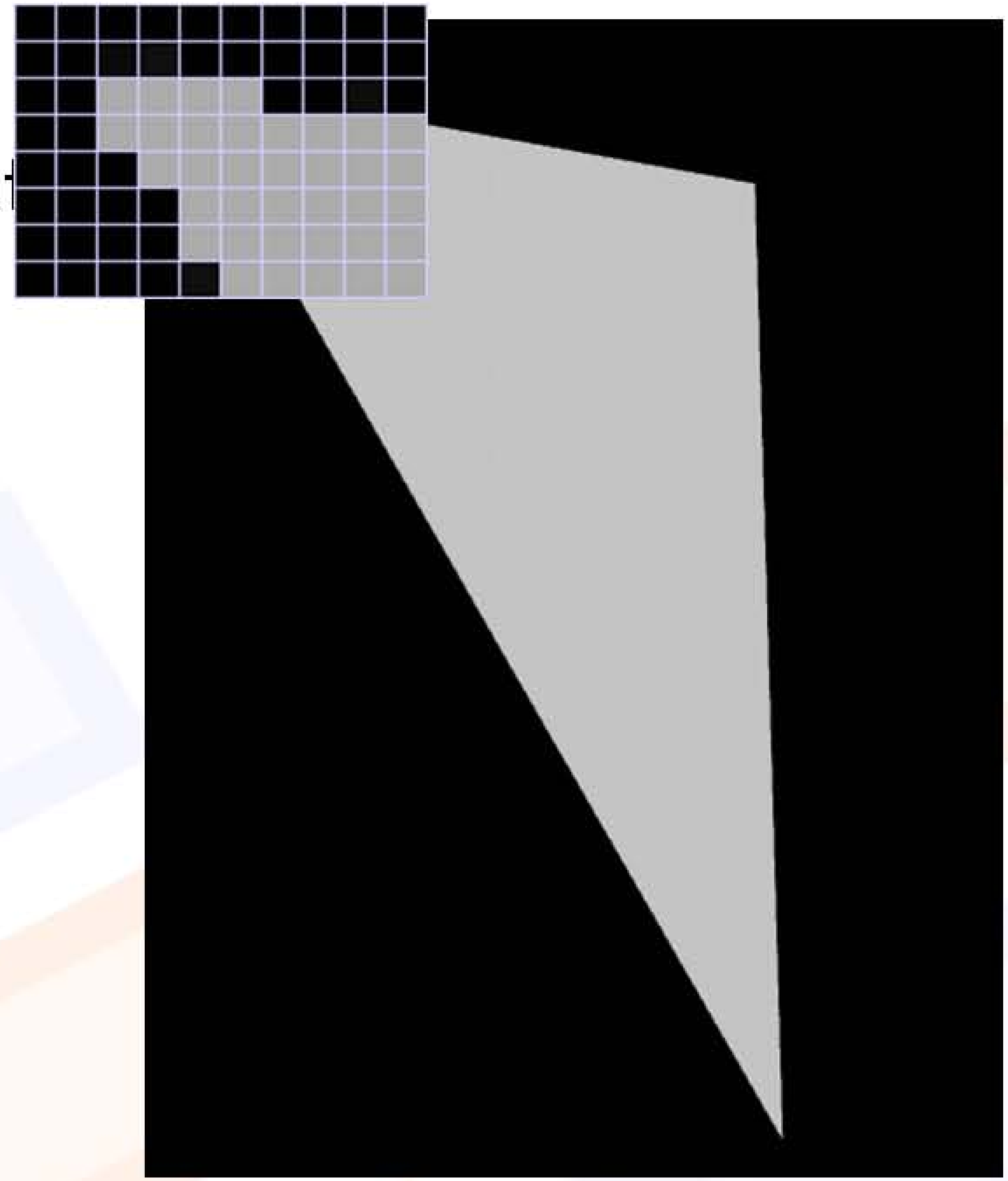
Primitives

Fragments

Pixels

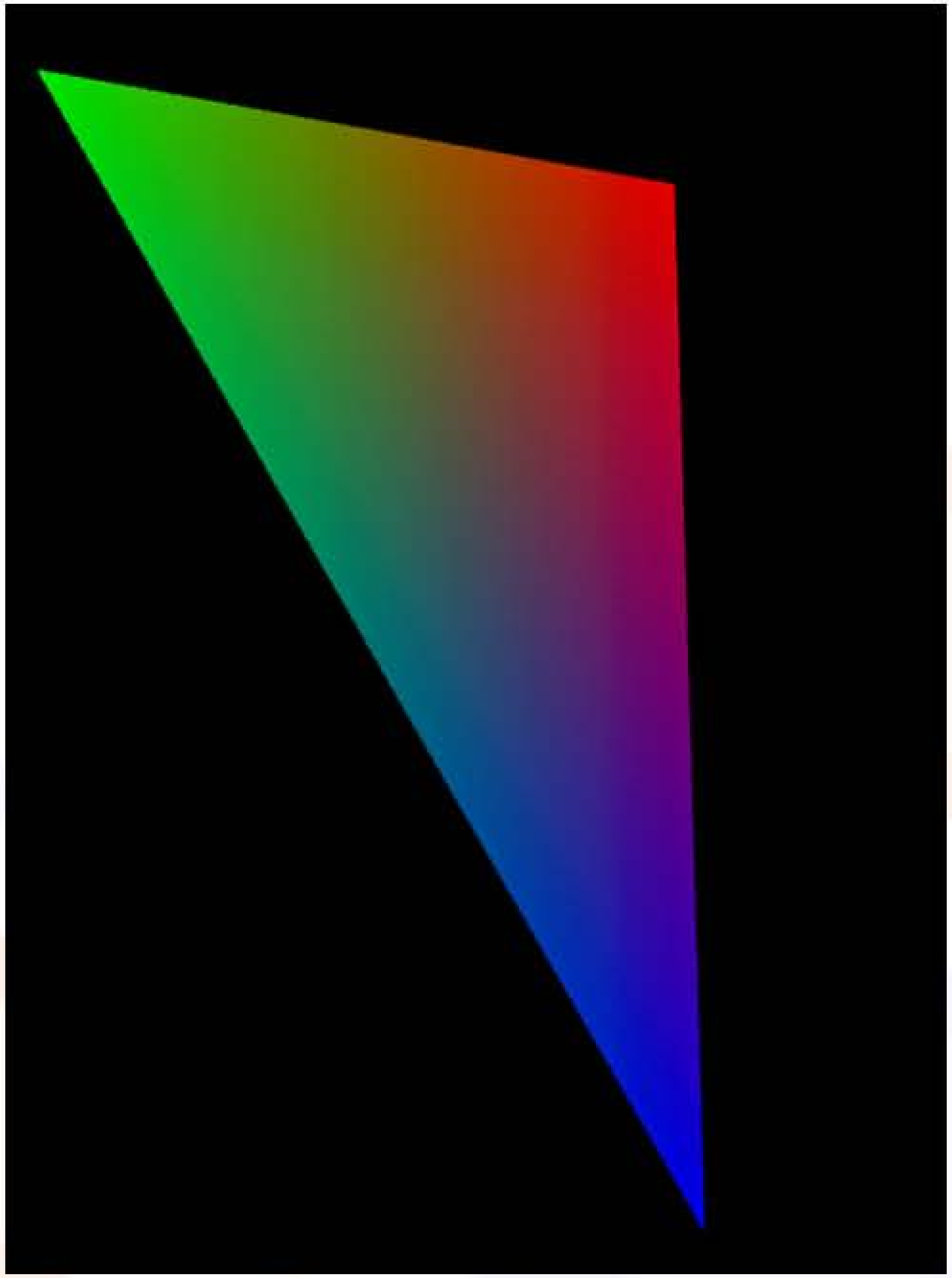
What can the hardware do?

- Rasterization
  - Decomposition into fragment



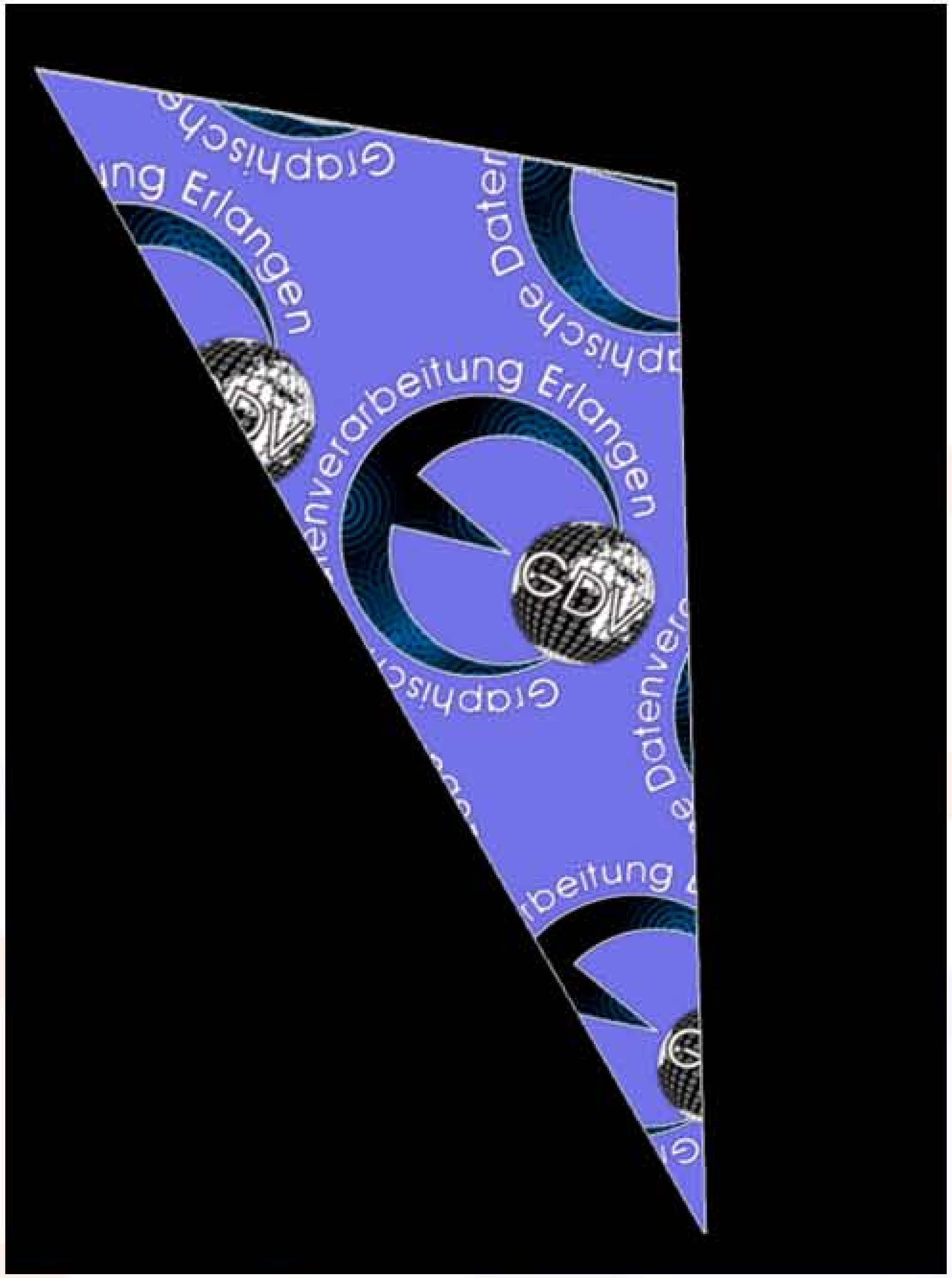
What can the hardware do?

- Rasterization
  - Decomposition into fragments
  - Interpolation of color



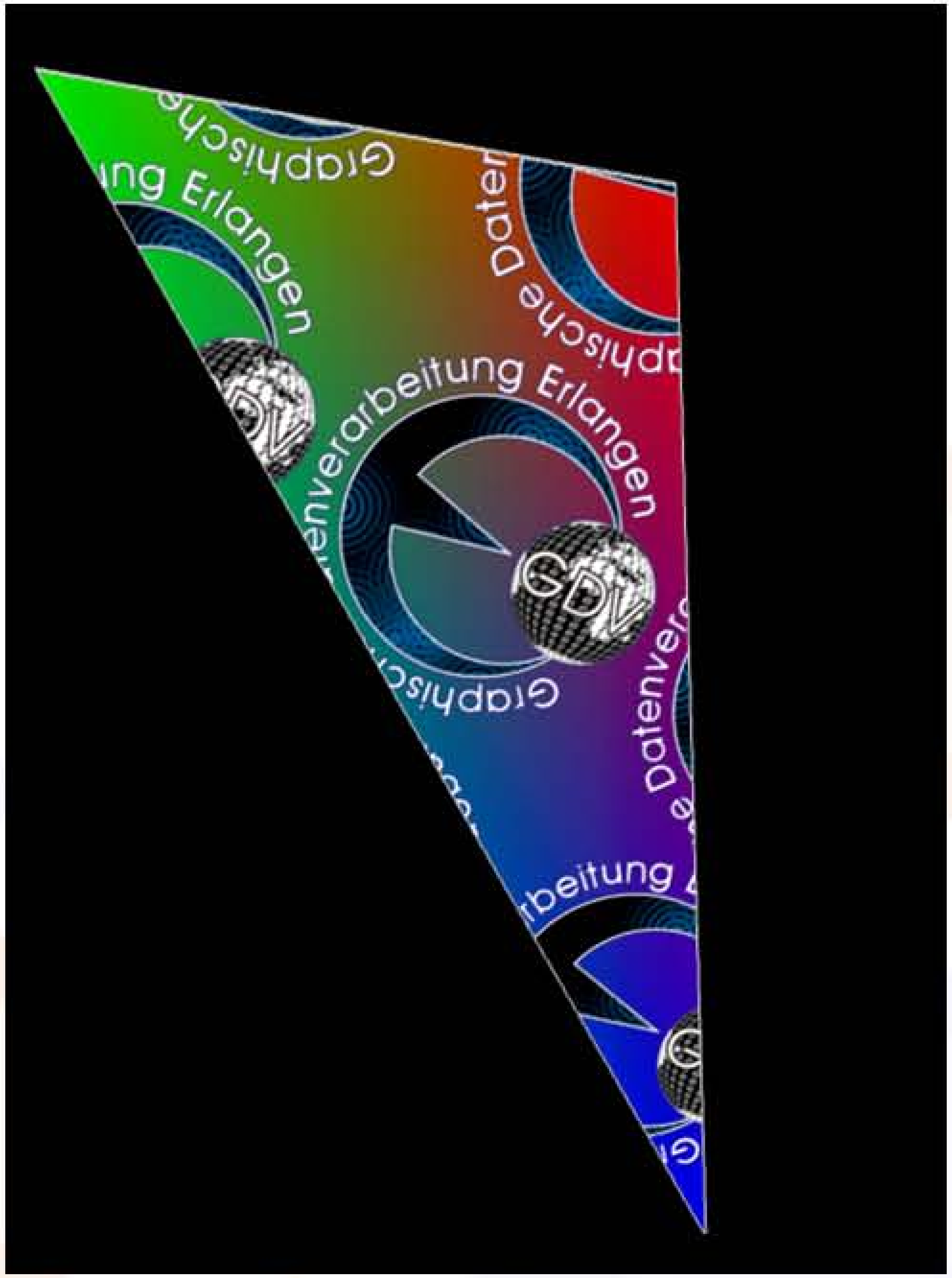
What can the hardware do?

- Rasterization
  - Decomposition into fragments
  - Interpolation of color
  - Texturing
    - Interpolation/Filtering
    - Fragment Shading



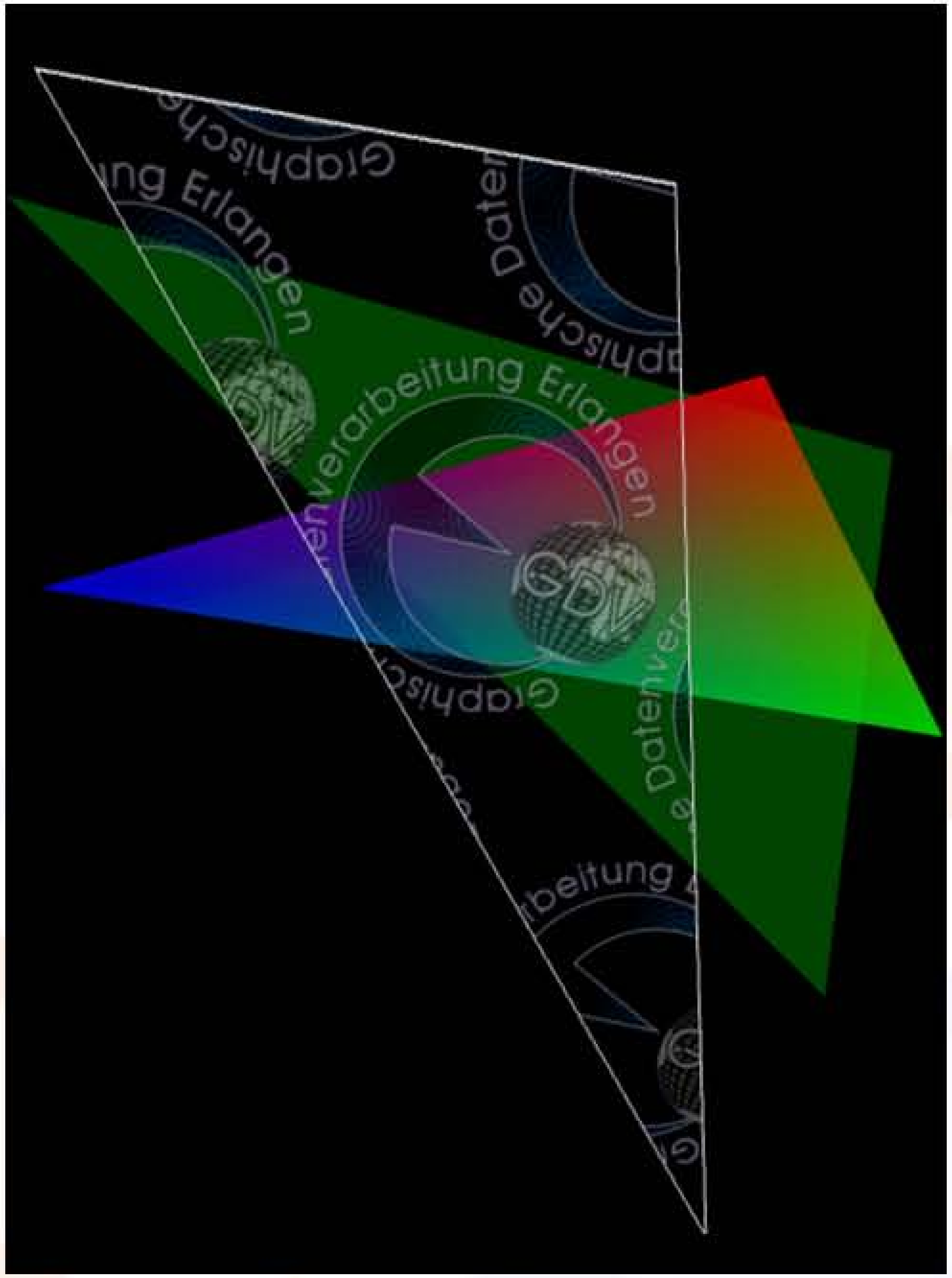
What can the hardware do?

- Rasterization
  - Decomposition into fragments
  - Interpolation of color
  - Texturing
    - Interpolation/Filtering
    - Fragment Shading

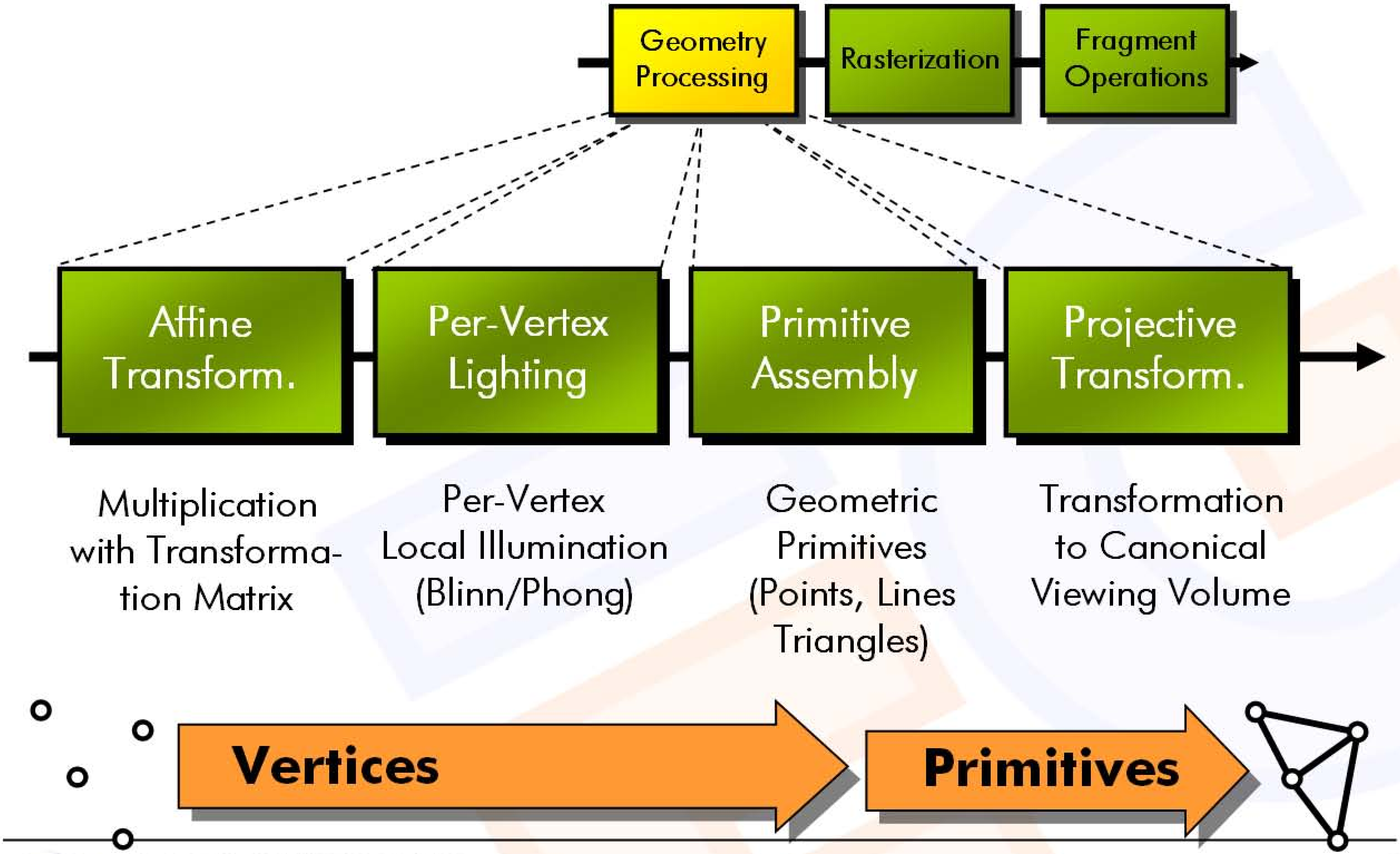


What can the hardware do?

- Rasterization
  - Decomposition into fragments
  - Interpolation of color
  - Texturing
    - Interpolation/Filtering
    - Fragment Shading
- Fragment Operations
  - Depth Test (Z-Test)
  - Alpha Blending (Compositing)



Geometry Processing



Geometry Processing

Rasterization

Fragment Operations

Affine Transform.

Per-Vertex Lighting

Primitive Assembly

Projective Transform.

Multiplication with Transformation Matrix

Per-Vertex Local Illumination (Blinn/Phong)

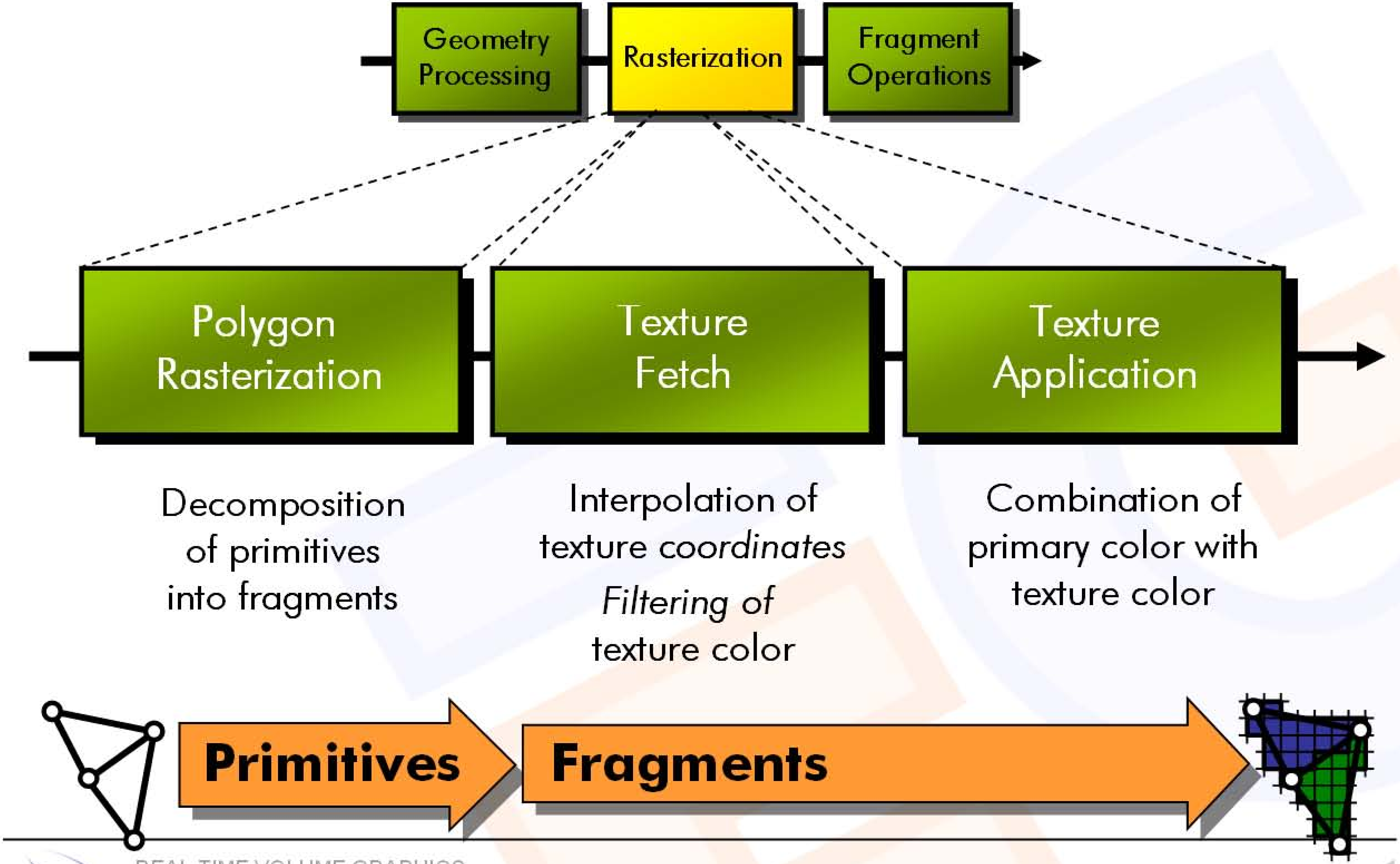
Geometric Primitives (Points, Lines Triangles)

Transformation to Canonical Viewing Volume

Vertices

Primitives

Rasterization



Geometry Processing

Rasterization

Fragment Operations

Polygon Rasterization

Texture Fetch

Texture Application

Decomposition of primitives into fragments

Interpolation of texture coordinates

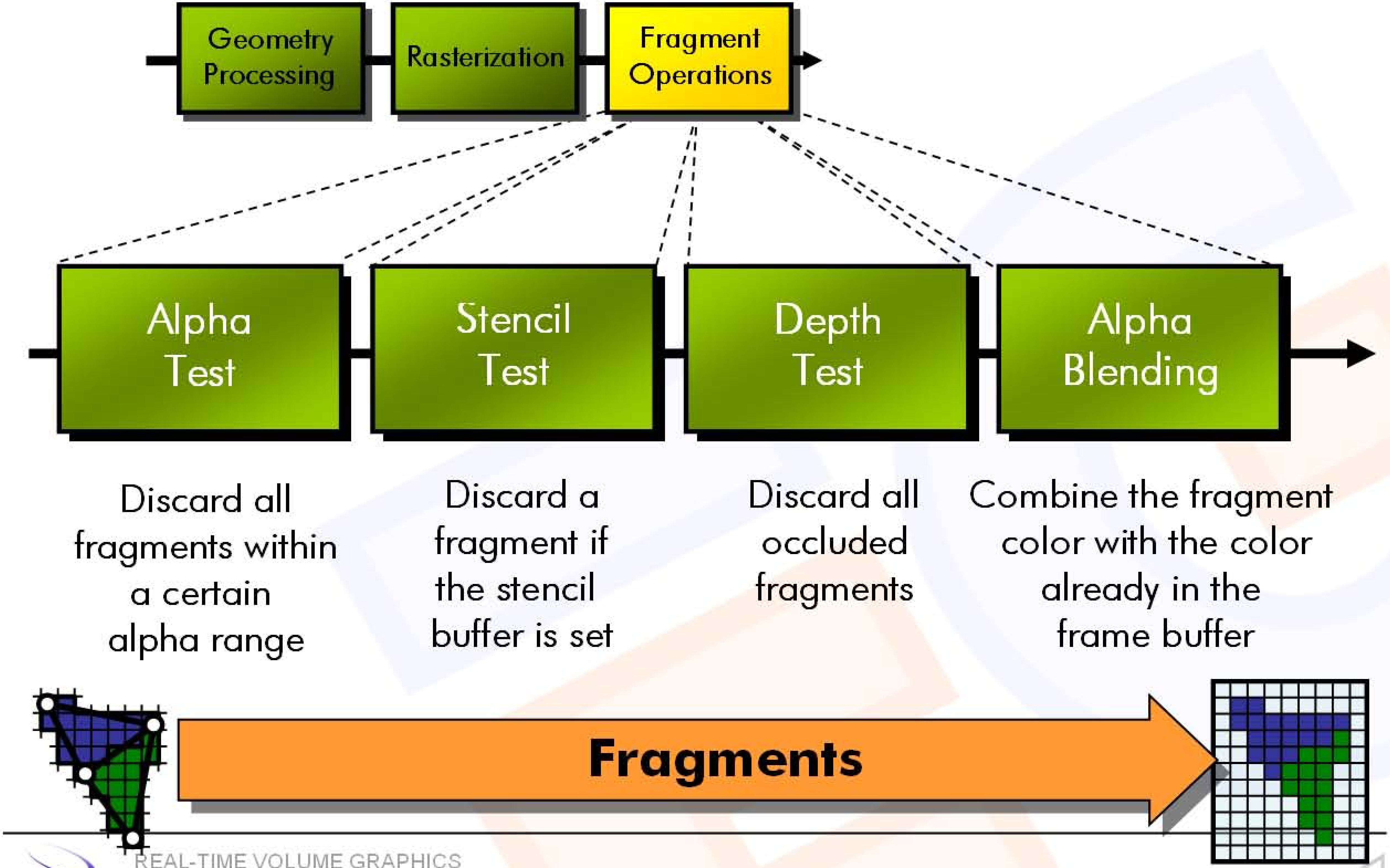
Filtering of texture color

Combination of primary color with texture color

Primitives

Fragments

Fragment Operations



Geometry Processing

Rasterization

Fragment Operations

Alpha Test

Stencil Test

Depth Test

Alpha Blending

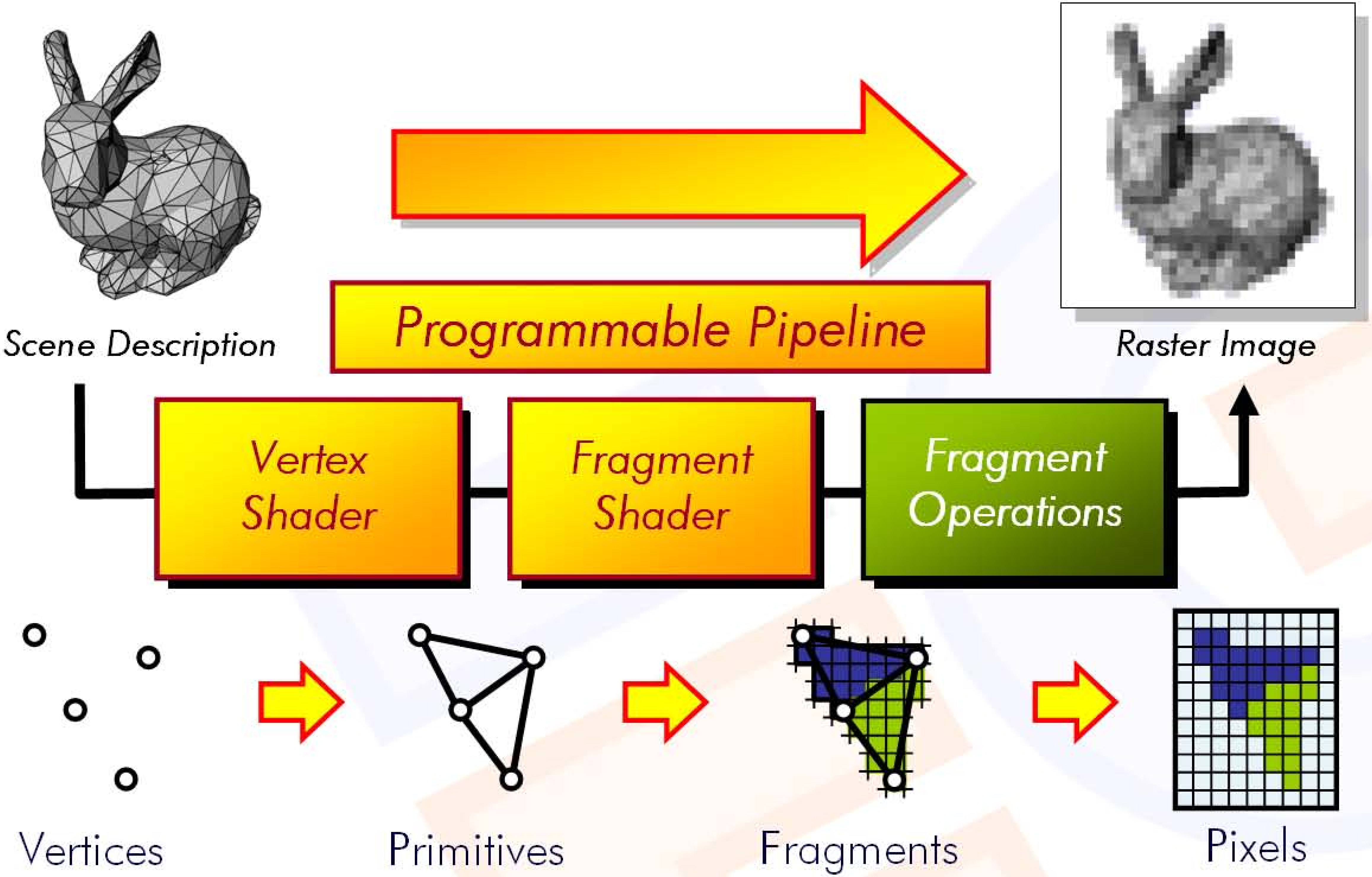
Discard all fragments within a certain alpha range

Discard a fragment if the stencil buffer is set

Discard all occluded fragments

Combine the fragment color with the color already in the frame buffer

Graphics Hardware



Scene Description

Programmable Pipeline

Raster Image

Vertex Shader

Fragment Shader

Fragment Operations

Vertices

Primitives

Fragments

Pixels

Vertex Shader

Important Features:

- Vertex Shader has information about *one single* Vertex only (no topological information)!
- For each set of vertex-attributes, the vertex shader generates *exactly one* vertex
  - The vertex shader cannot create additional vertices!
  - The vertex shader cannot discard vertices from the stream!
- The term „shader“ is somehow misleading, since the vertex shader can change the geometry!

Vertex Shader Instructions

- Assembly-Language, such as

ABS	absolute value
ADD	addition
DP3	scalar product (dot product)
DP4	scalar product 4-components
DST	distance vector
LIT	illumination terms
MUL	multiplication
MAD	multiply and add
SUB	subtraction
XPD	cross product

- Most commands are vector commands (4 components)

High-Level Shading Languages

Who wants to write assembler code?

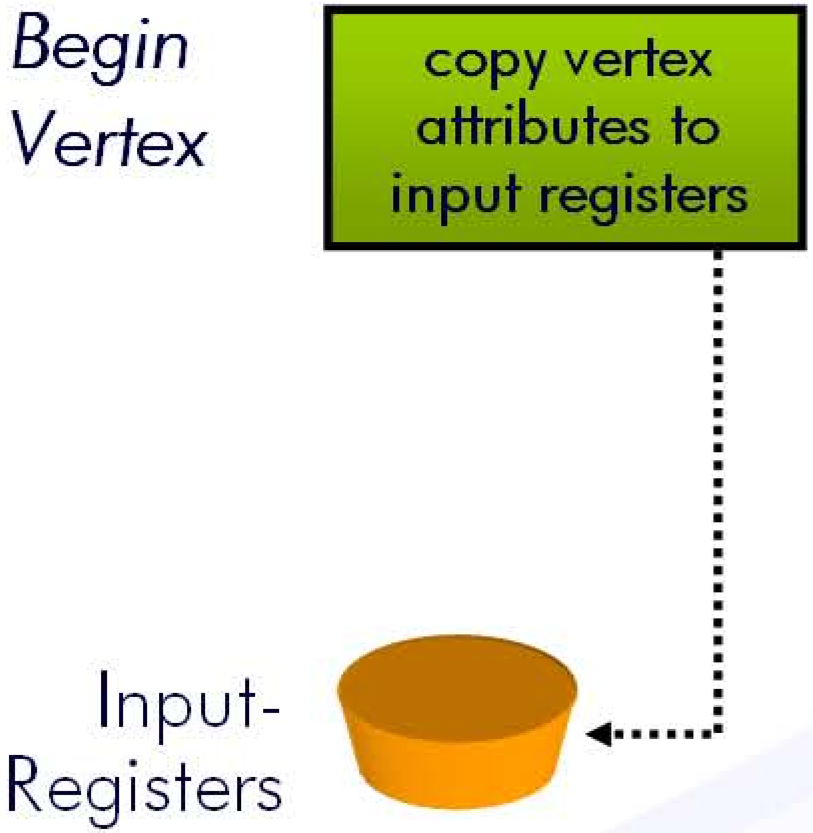
- *Stanford Shading Language*
  Cg (developed by Nvidia) for OpenGL *and* DirectX
  DirectX 9.0 HLSL (DirectX only, Syntax similar to Cg)
  GLSL (OpenGL shading language)

- Syntax similar to C
- plus vector variables und vector instructions:

```
float4 v1; // same as float v1[4] in C
int3 v2; // same as int v2[3] in C
```

- Swizzling: `float4 v3 = v1.xzzy;`

Programmable Vertex Processor

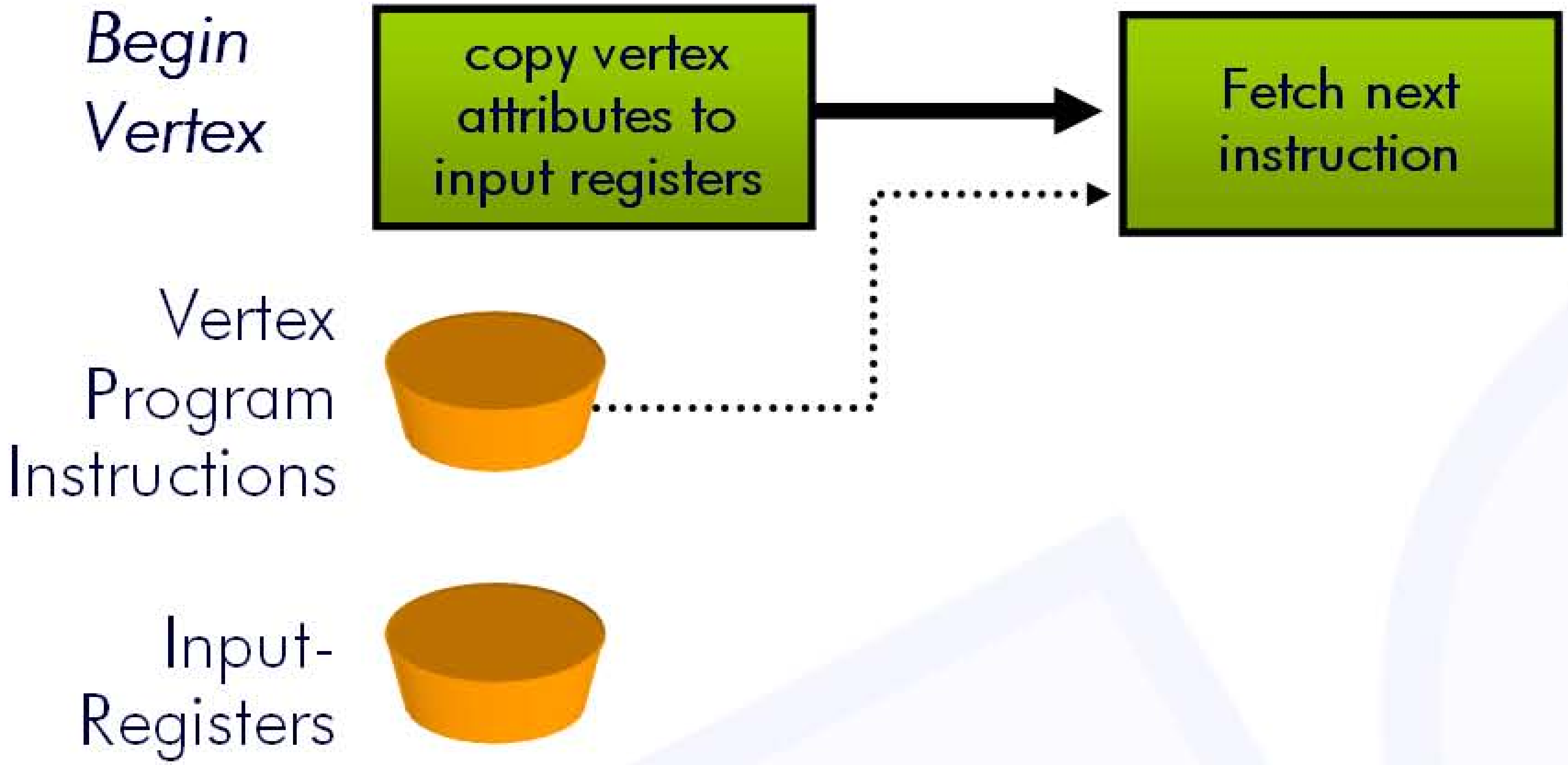


Begin Vertex

copy vertex attributes to input registers

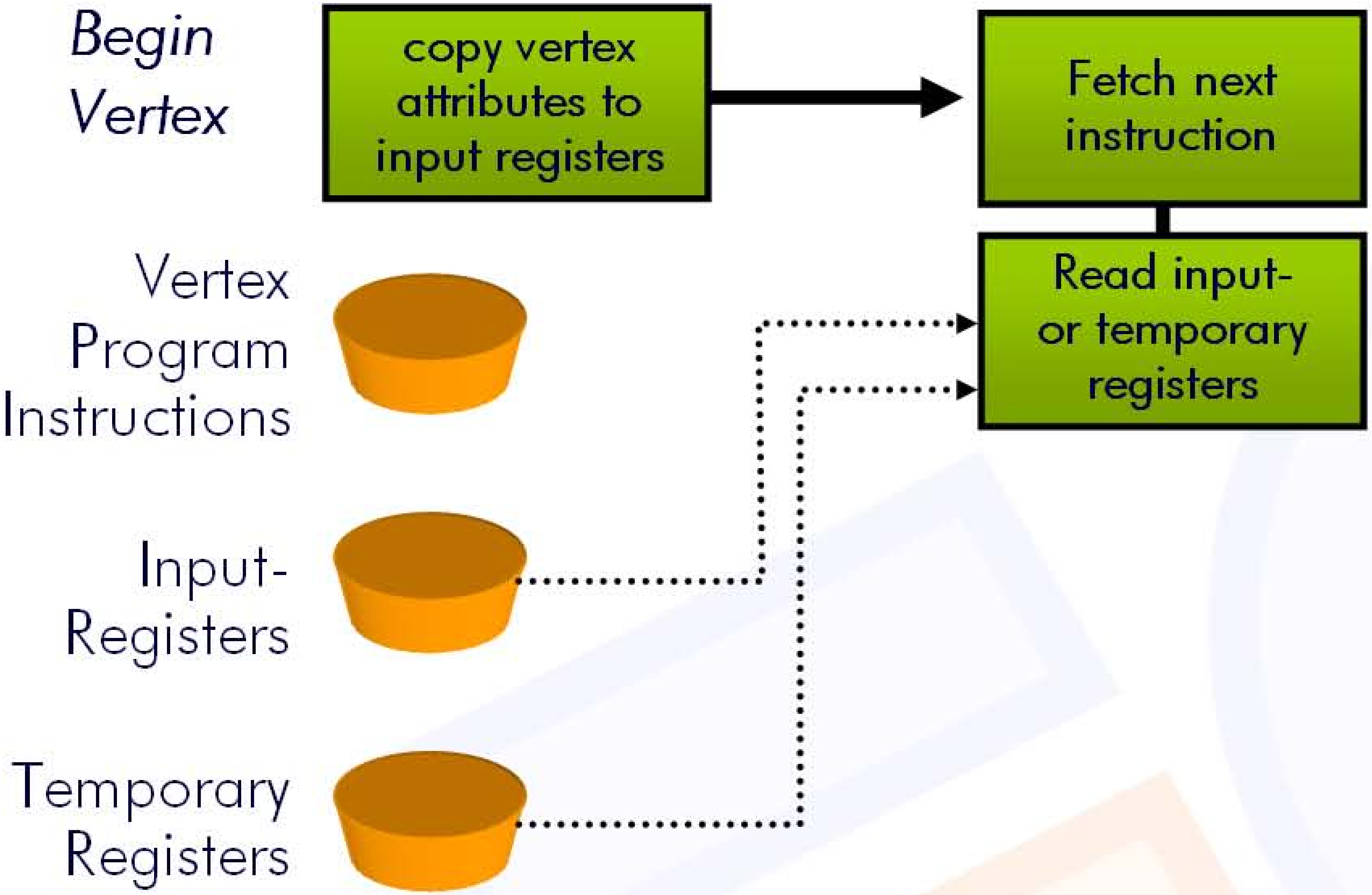
Input-Registers

Programmable Vertex Processor

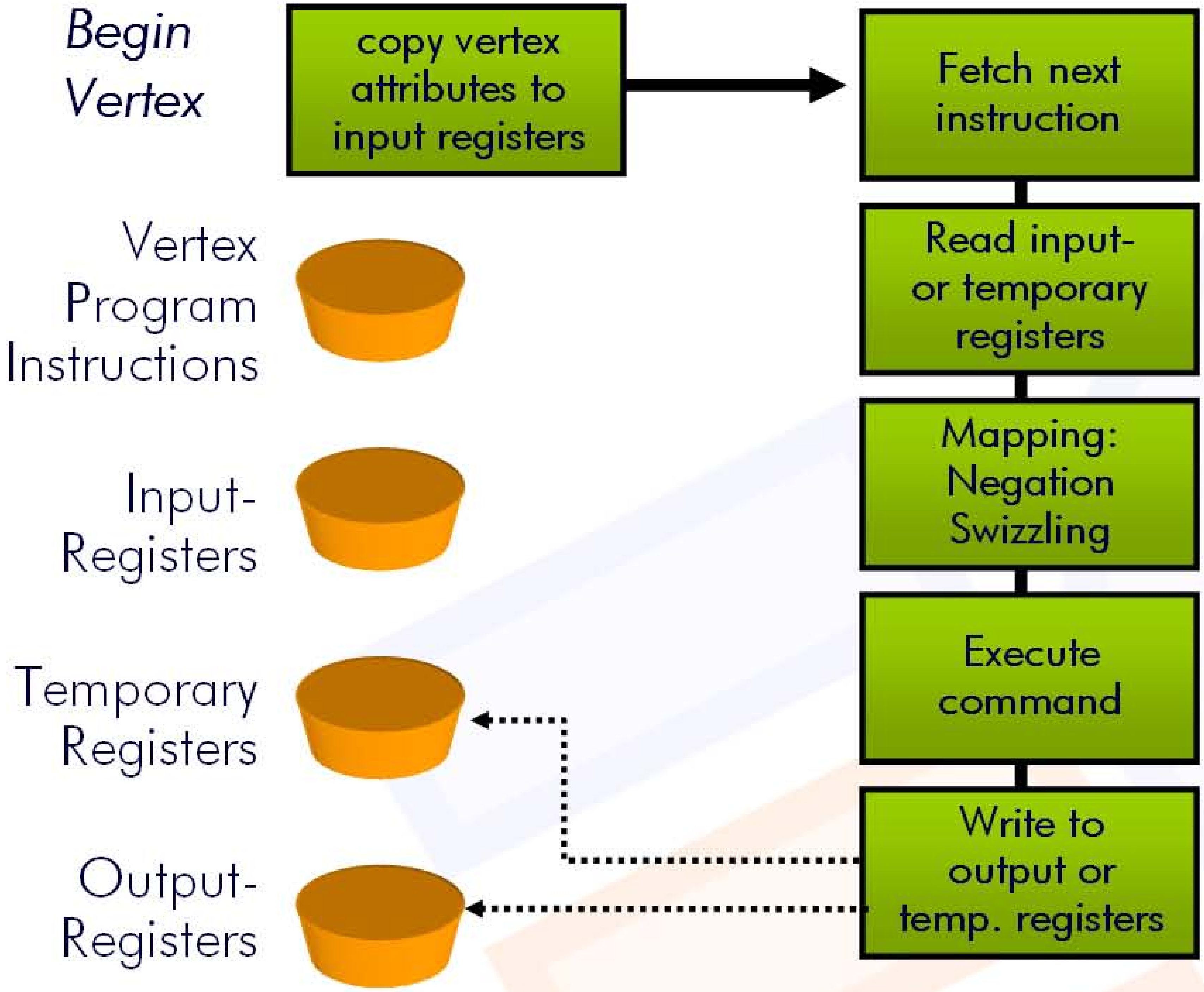


Begin Vertex
copy vertex attributes to input registers
Fetch next instruction
Vertex Program Instructions
Input-Registers

Programmable Vertex Processor


Begin Vertex
copy vertex attributes to input registers
Fetch next instruction
Vertex Program Instructions
Read input- or temporary registers
Input-Registers
Temporary Registers

Programmable Vertex Processor



Begin Vertex

copy vertex attributes to input registers

Fetch next instruction

Read input- or temporary registers

Mapping: Negation Swizzling

Execute command

Write to output or temp. registers

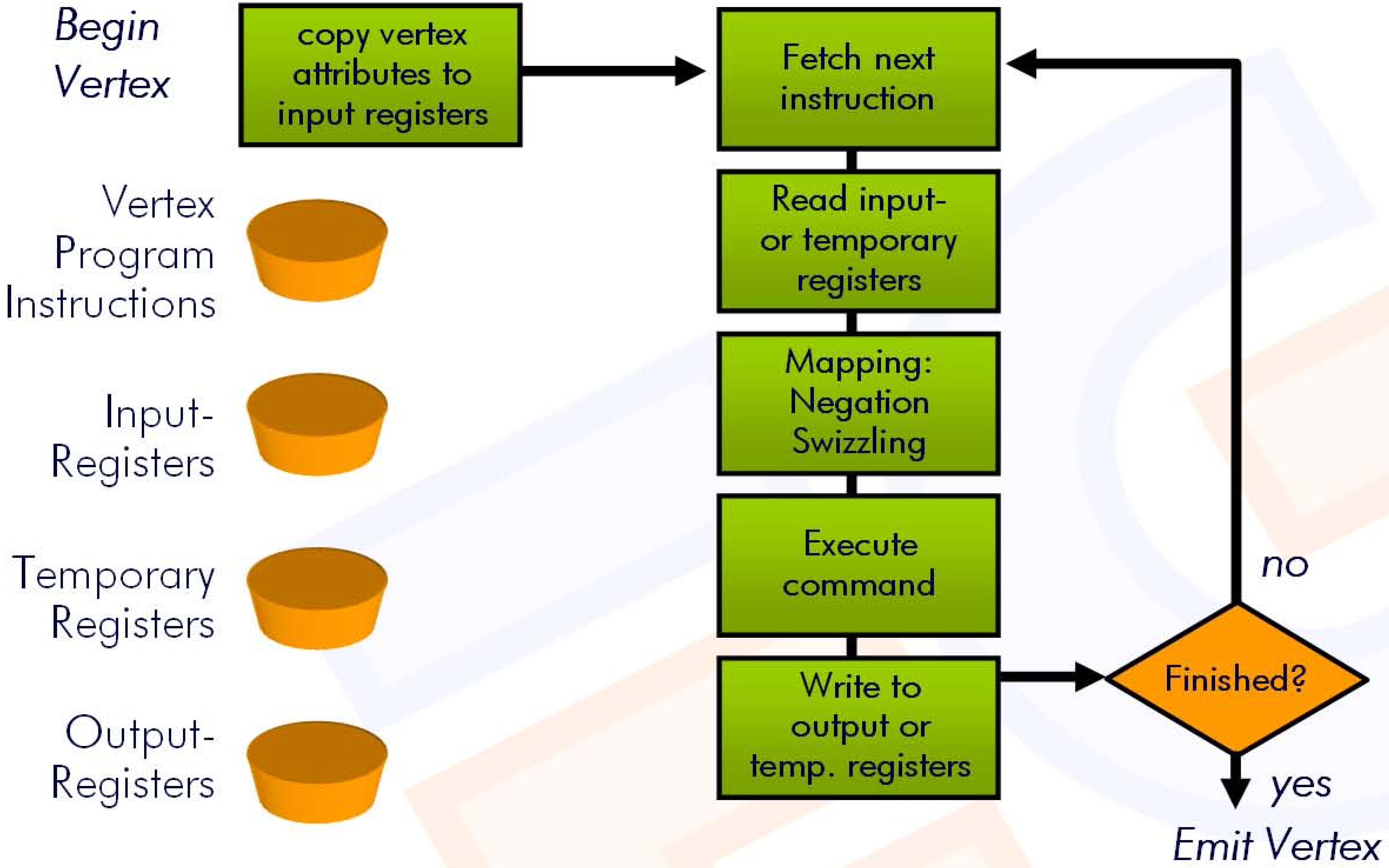
Vertex Program Instructions

Input-Registers

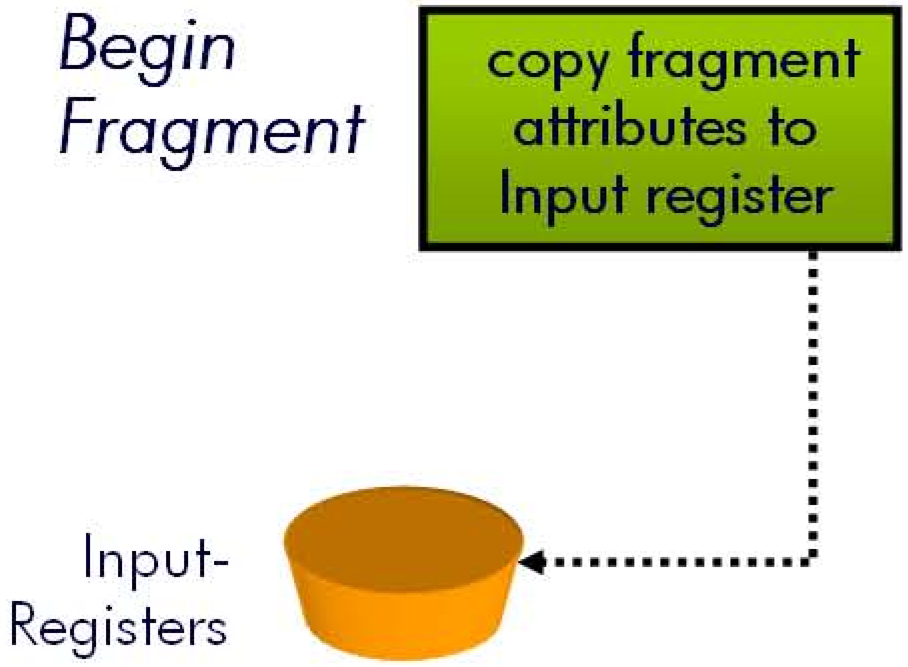
Temporary Registers

Output-Registers

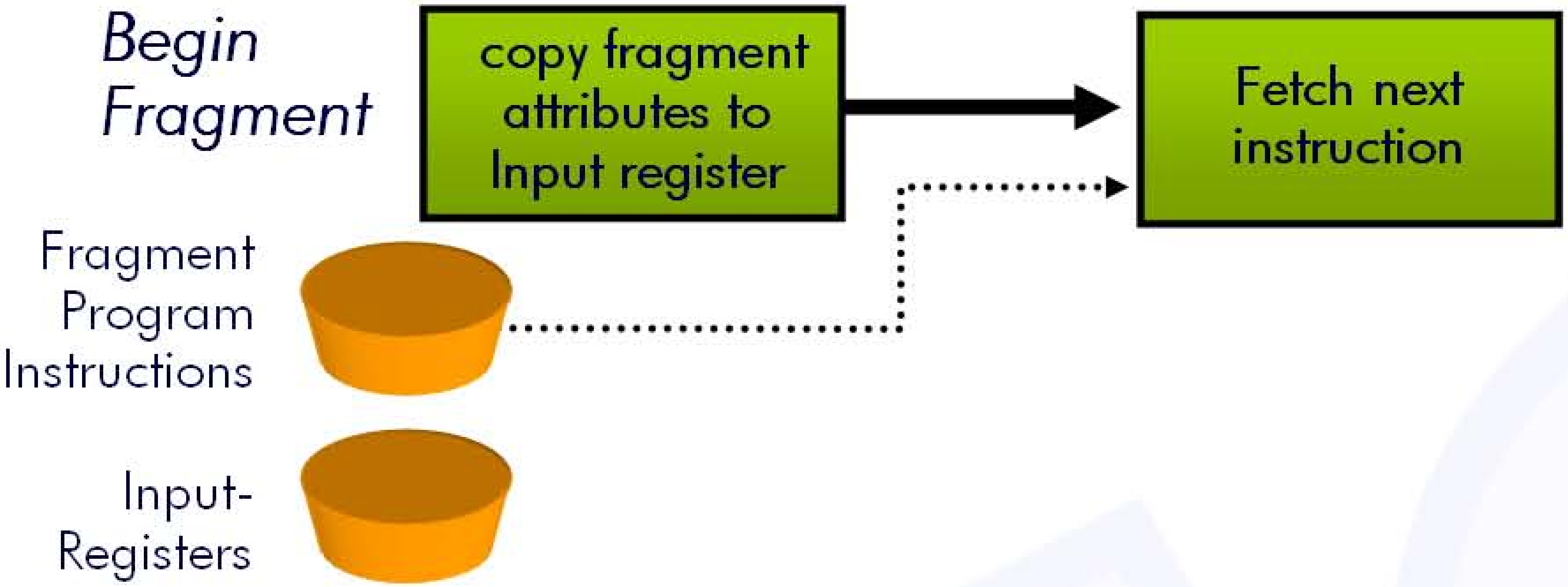
Programmable Vertex Processor


Begin Vertex
copy vertex attributes to input registers
Fetch next instruction
Vertex Program Instructions
Read input- or temporary registers
Input-Registers
Mapping: Negation Swizzling
Temporary Registers
Execute command
no
Finished?
Output-Registers
Write to output or temp. registers
yes
Emit Vertex

Fragment Processor



Fragment Processor



Begin Fragment

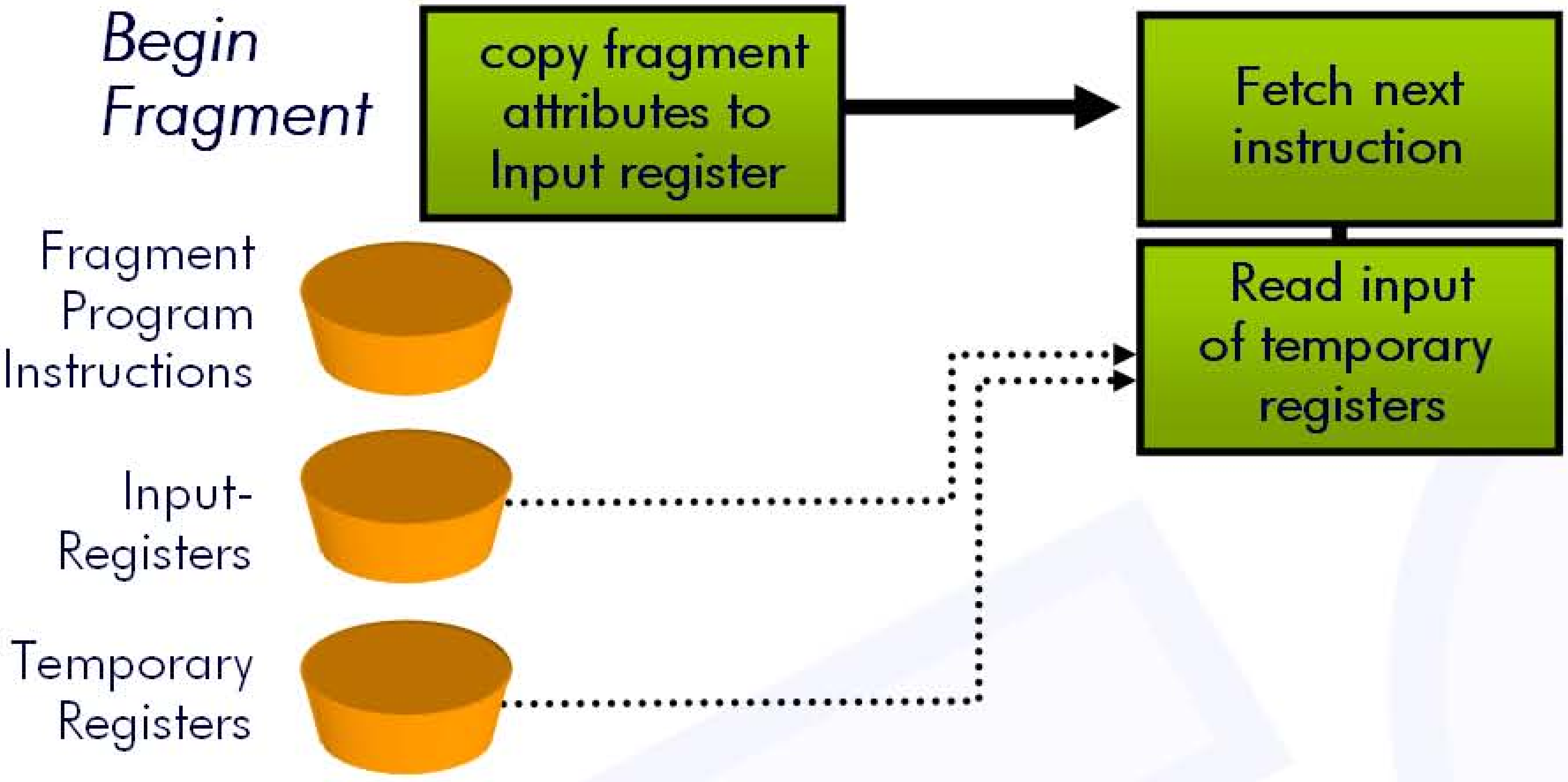
copy fragment attributes to Input register

Fetch next instruction

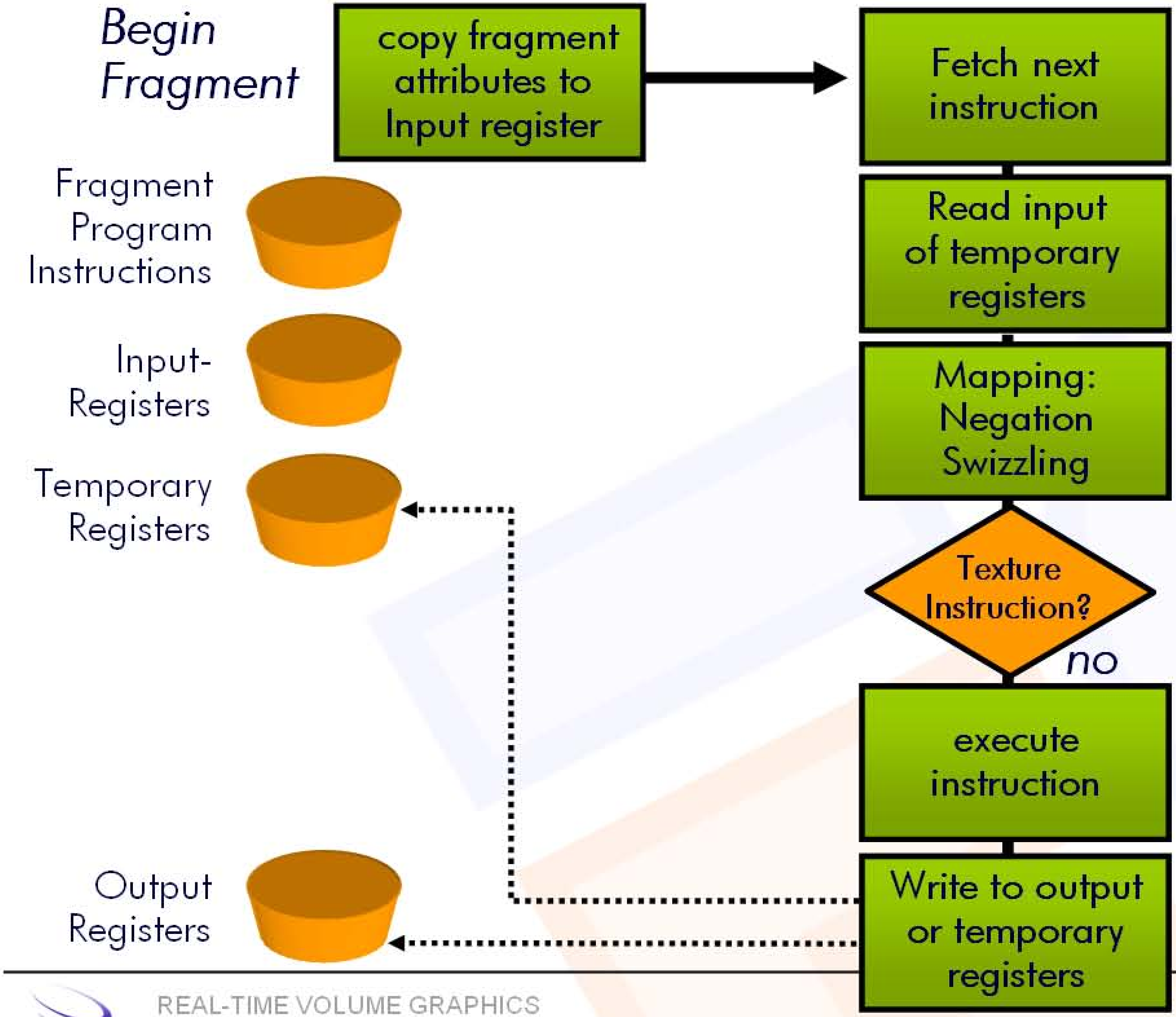
Fragment Program Instructions

Input-Registers

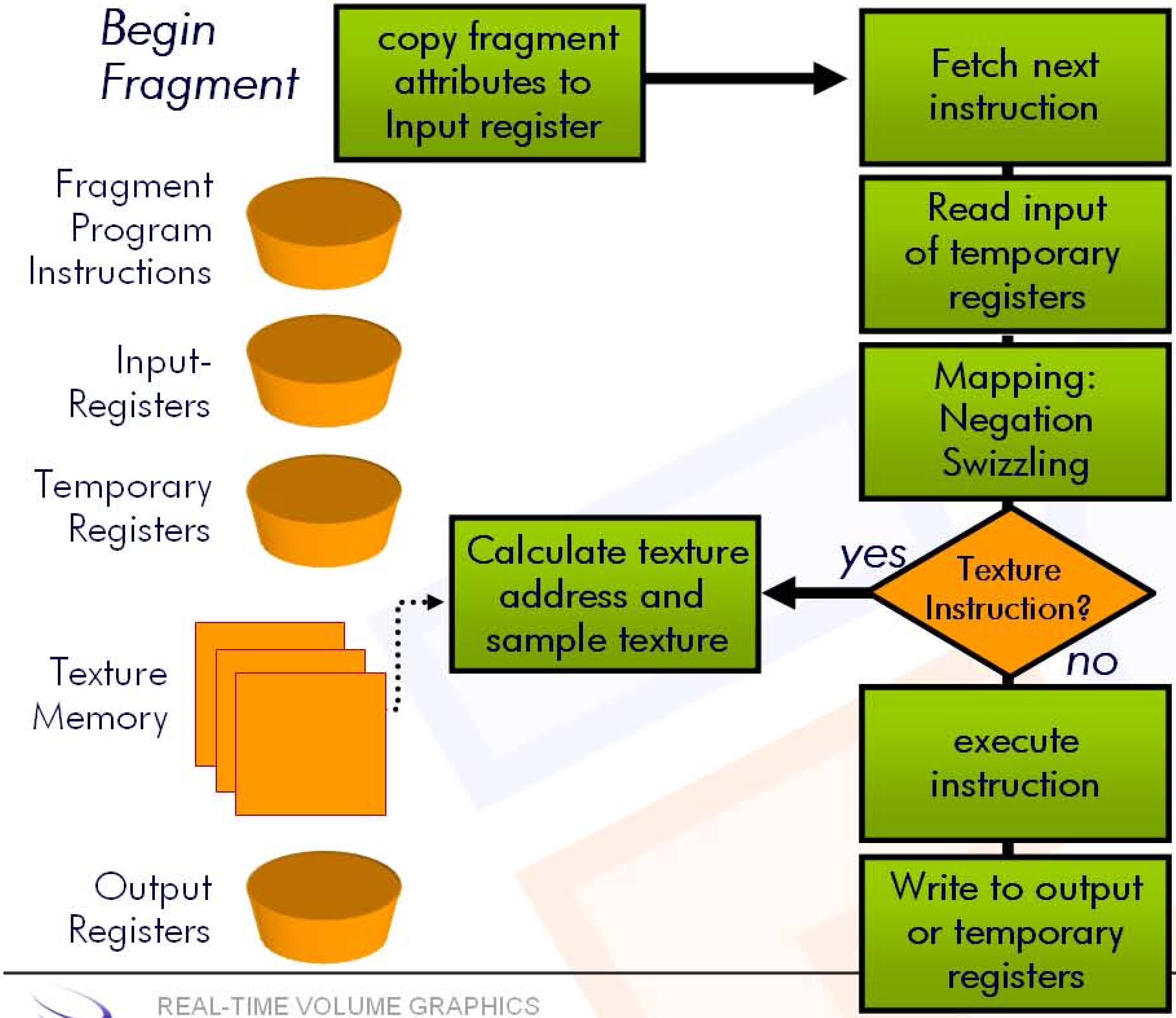
Fragment Processor


Begin Fragment
copy fragment attributes to Input register
Fetch next instruction
Read input of temporary registers
Fragment Program Instructions
Input-Registers
Temporary Registers

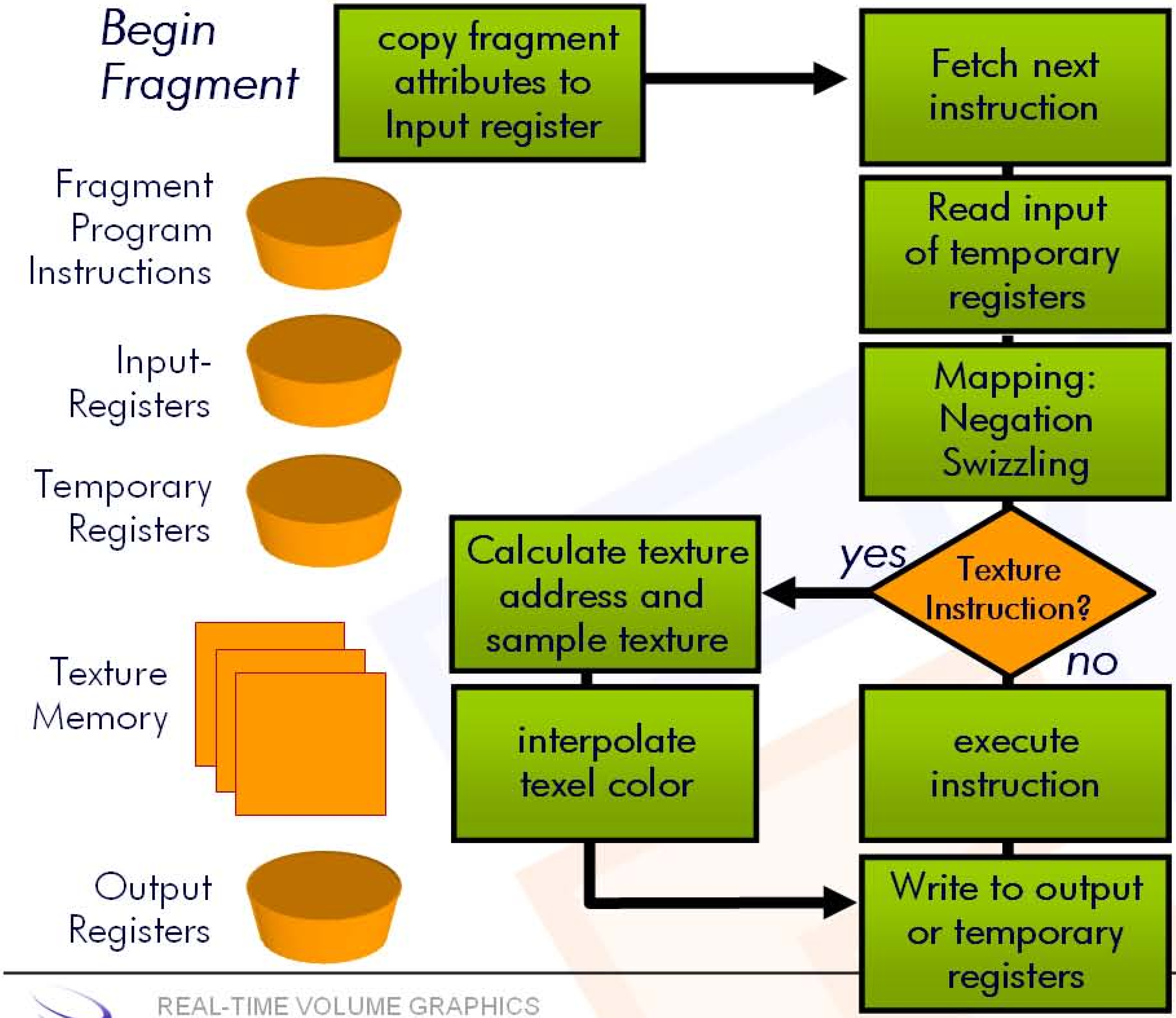
Fragment Processor


Begin Fragment
copy fragment attributes to Input register
Fetch next instruction
Fragment Program Instructions
Read input of temporary registers
Input-Registers
Mapping: Negation Swizzling
Temporary Registers
Texture Instruction?
no
execute instruction
Output Registers
Write to output or temporary registers

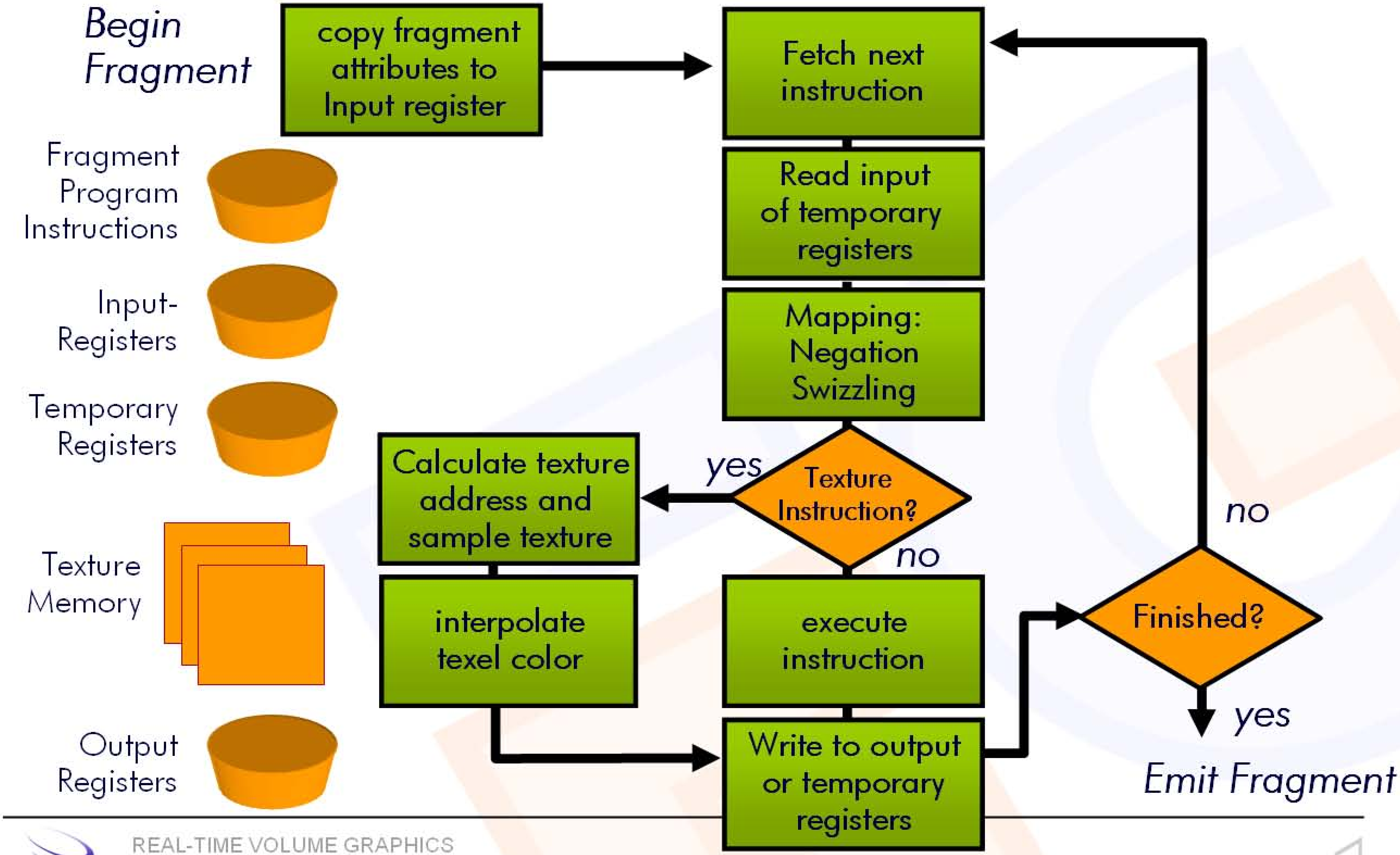
Fragment Processor


Begin Fragment
copy fragment attributes to Input register
Fetch next instruction
Fragment Program Instructions
Read input of temporary registers
Input-Registers
Mapping: Negation Swizzling
Temporary Registers
Calculate texture address and sample texture
yes
Texture Instruction?
no
Texture Memory
execute instruction
Output Registers
Write to output or temporary registers

Fragment Processor


Begin Fragment
copy fragment attributes to Input register
Fetch next instruction
Fragment Program Instructions
Read input of temporary registers
Input-Registers
Mapping: Negation Swizzling
Temporary Registers
Calculate texture address and sample texture
yes
Texture Instruction?
no
Texture Memory
interpolate texel color
execute instruction
Output Registers
Write to output or temporary registers

Fragment Processor


Begin Fragment
copy fragment attributes to Input register
Fetch next instruction
Fragment Program Instructions
Read input of temporary registers
Input-Registers
Mapping: Negation Swizzling
Temporary Registers
Calculate texture address and sample texture
yes
Texture Instruction?
no
no
Texture Memory
interpolate texel color
execute instruction
Finished?
yes
Output Registers
Write to output or temporary registers
Emit Fragment

Phong Shading

- *Per-Pixel Lighting:* Local illumination in a fragement shader

```
void main(float4 position  : TEXCOORD0,
          float3 normal    : TEXCOORD1,

      out float4 oColor     : COLOR,

  uniform float3 ambientCol,
  uniform float3 lightCol,
  uniform float3 lightPos,
  uniform float3 eyePos,
  uniform float3 Ka,
  uniform float3 Kd,
  uniform float3 Ks,
  uniform float  shiny)
{
```

Phong Shading

- *Per-Pixel Lighting:* Local illumination in a fragement shader

```
  float3 P = position.xyz;
  float3 N = normal;
  float3 V = normalize(eyePosition - P);
  float3 H = normalize(L + V);

  float3 ambient = Ka * ambientCol;

  float3 L         = normalize(lightPos - P);
  float  diffLight = max(dot(L, N), 0);
  float3 diffuse   = Kd * lightCol * diffLight;

  float specLight = pow(max(dot(H, N), 0), shiny);
  float3 specular = Ks * lightCol * specLight;

  oColor.xyz = ambient + diffuse + specular;
  oColor.w = 1;
}
```