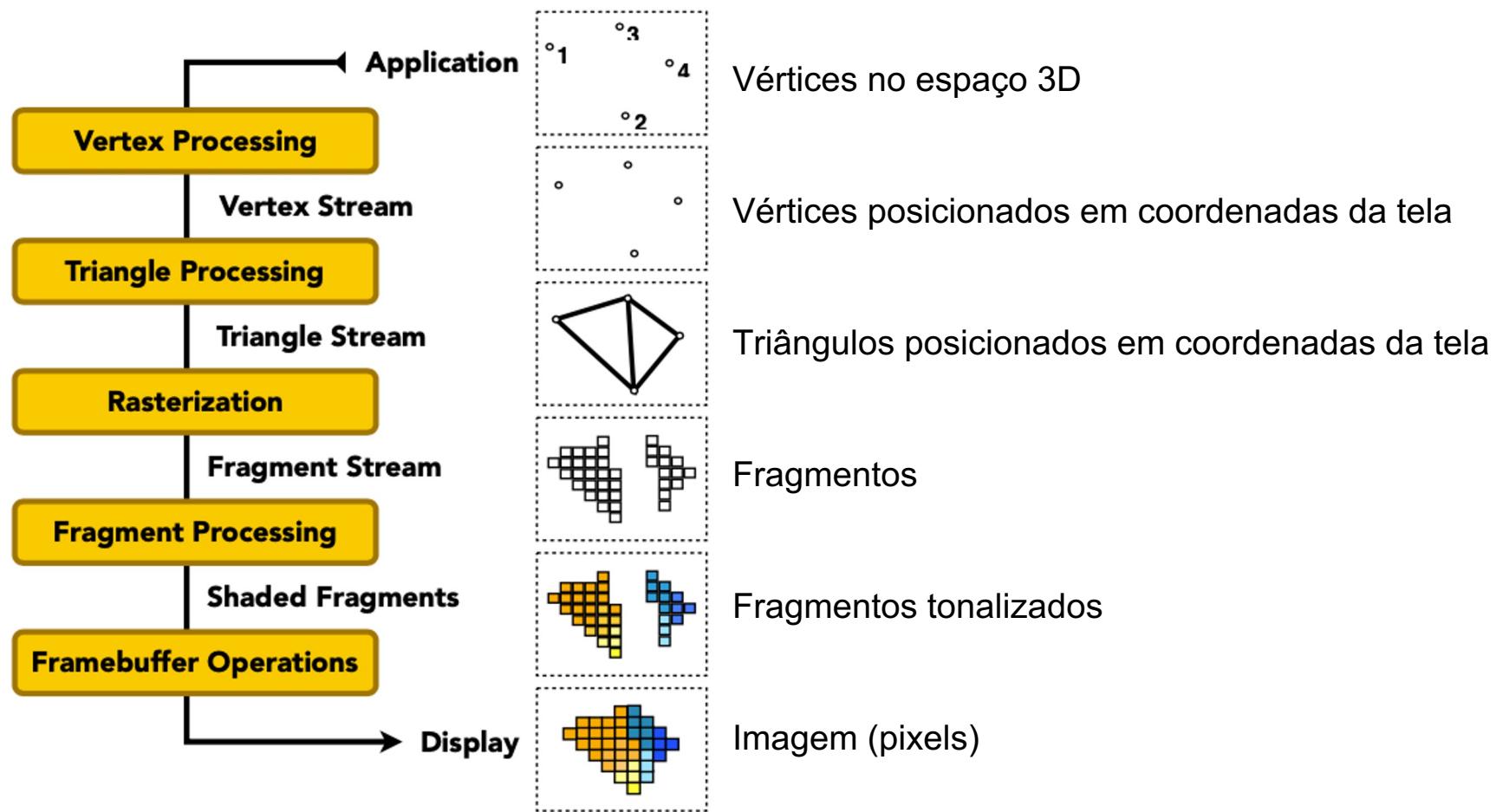


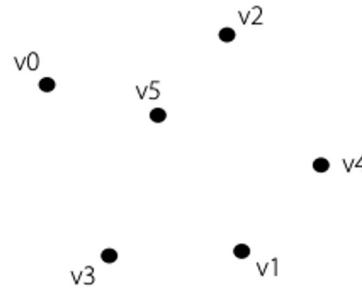
Computação Gráfica

Aula 18: Pipeline Gráfico

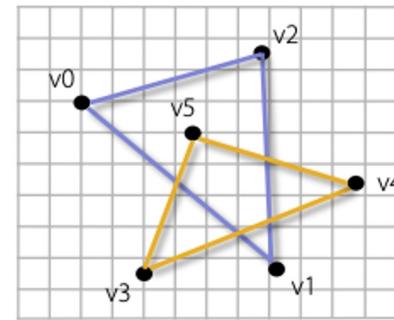
Pipeline de Rasterização



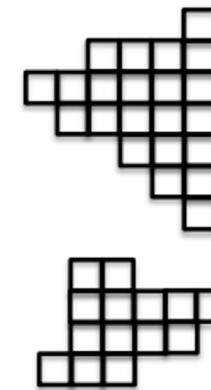
Entidades na Pipeline



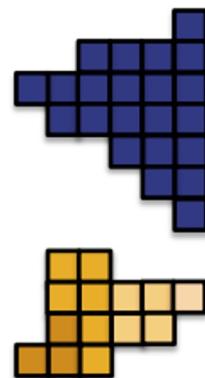
Vertices



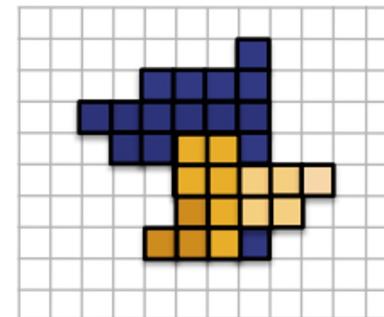
Primitives



Fragments

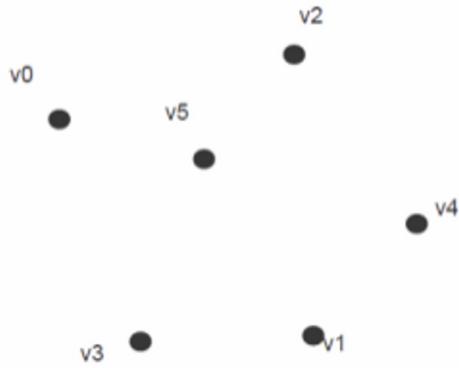


Fragments (shaded)

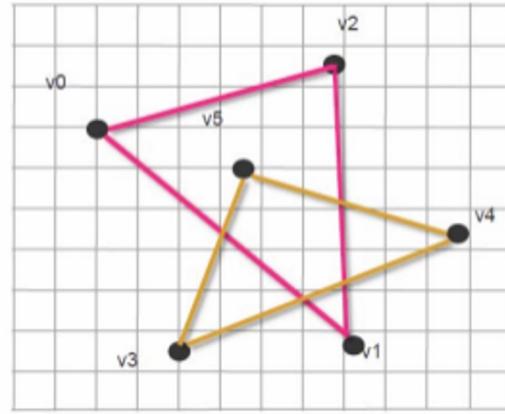


Pixels

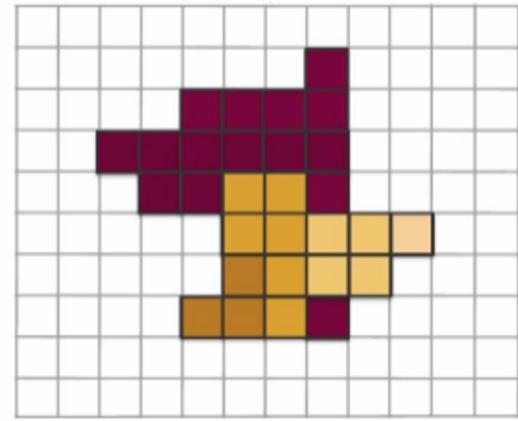
Como os Shaders são Invocados



Vertex Shader
invocado 6 vezes

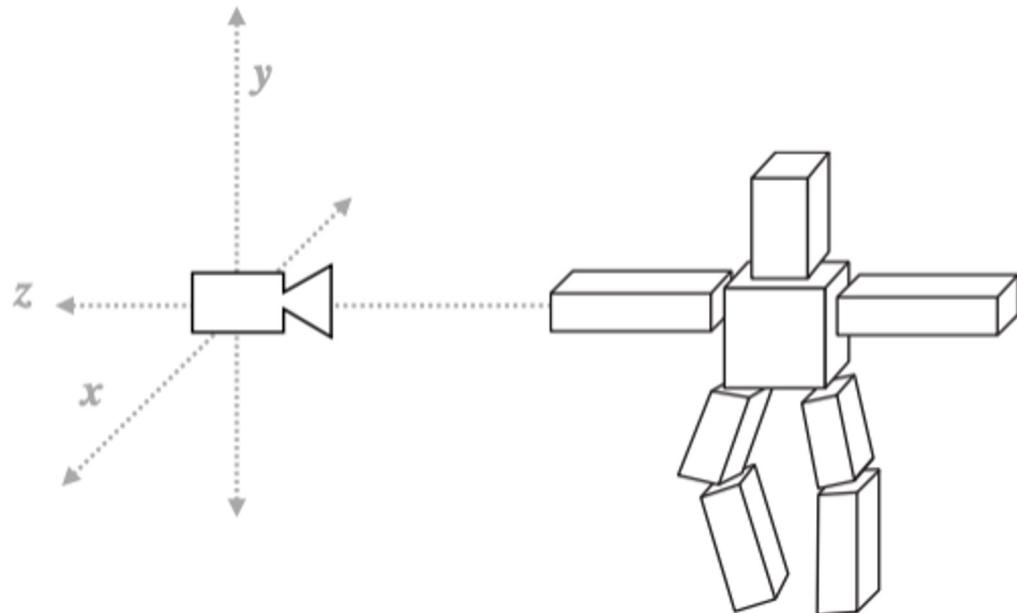
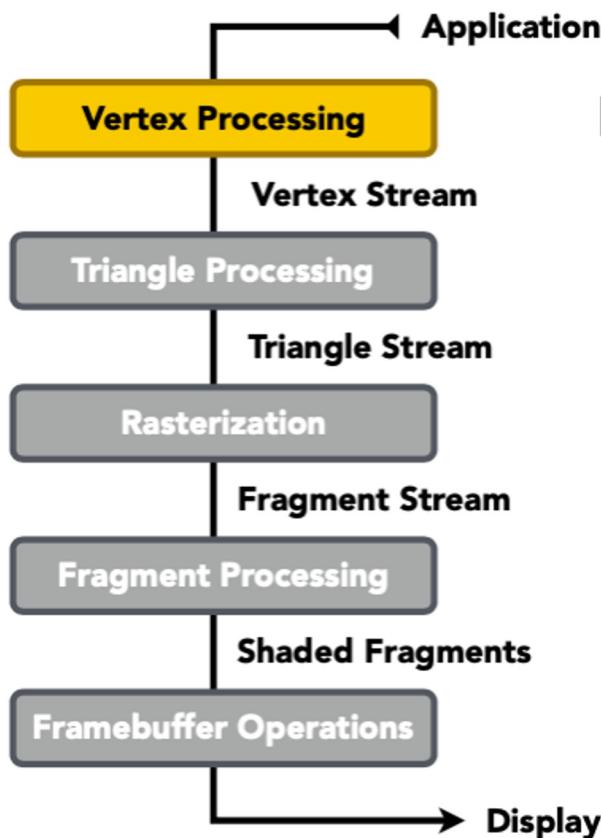


Geometry Shader
invocado 2 vezes

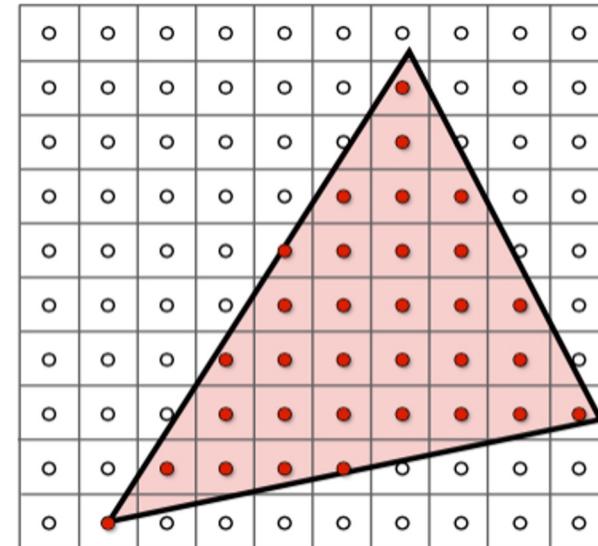
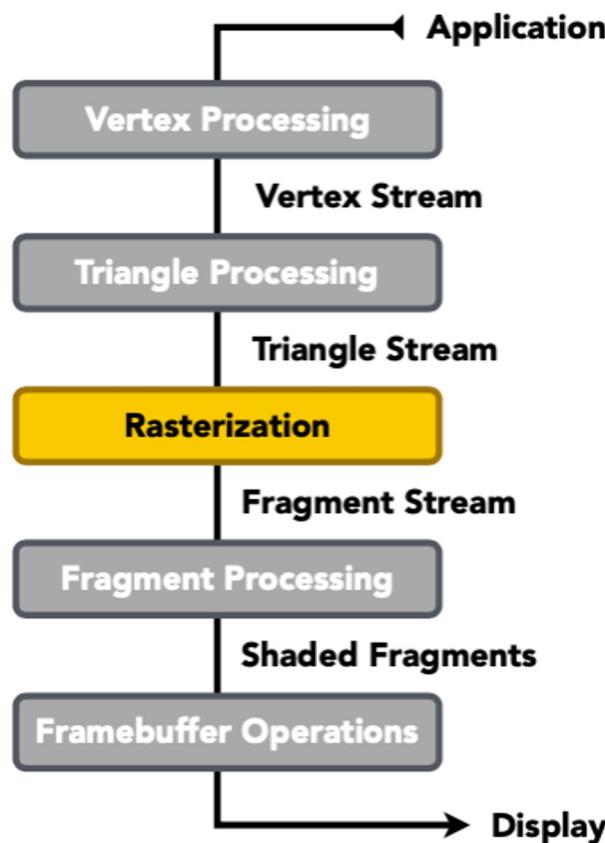


Fragment Shader
invocado 35 vezes
(para os fragmentos
ocultos também)

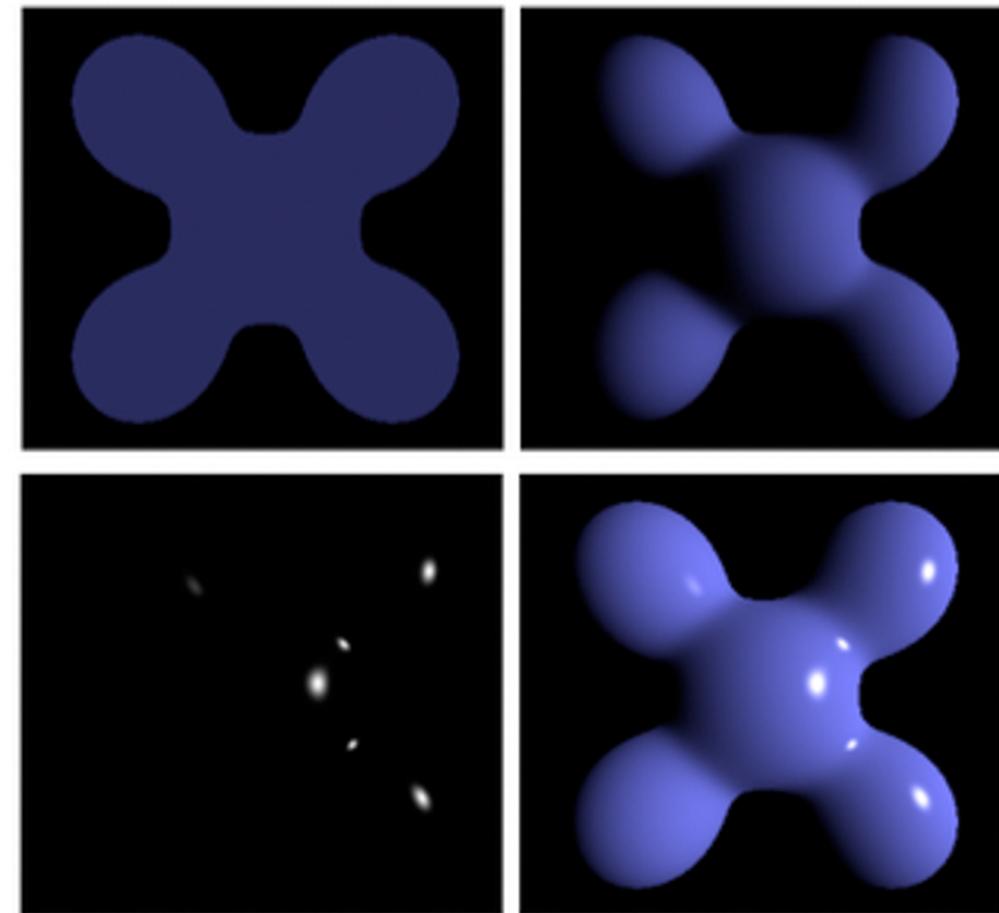
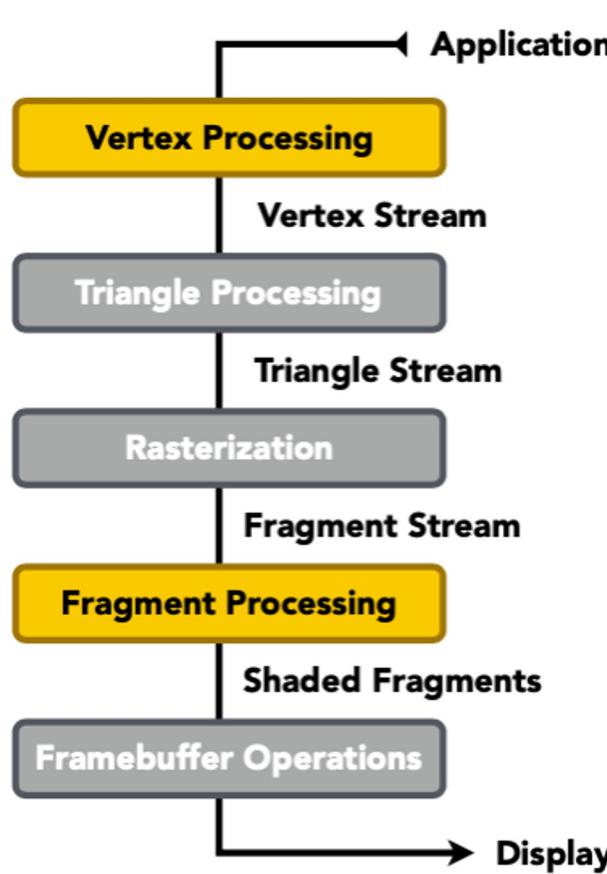
Transformações Geométricas e Visualização



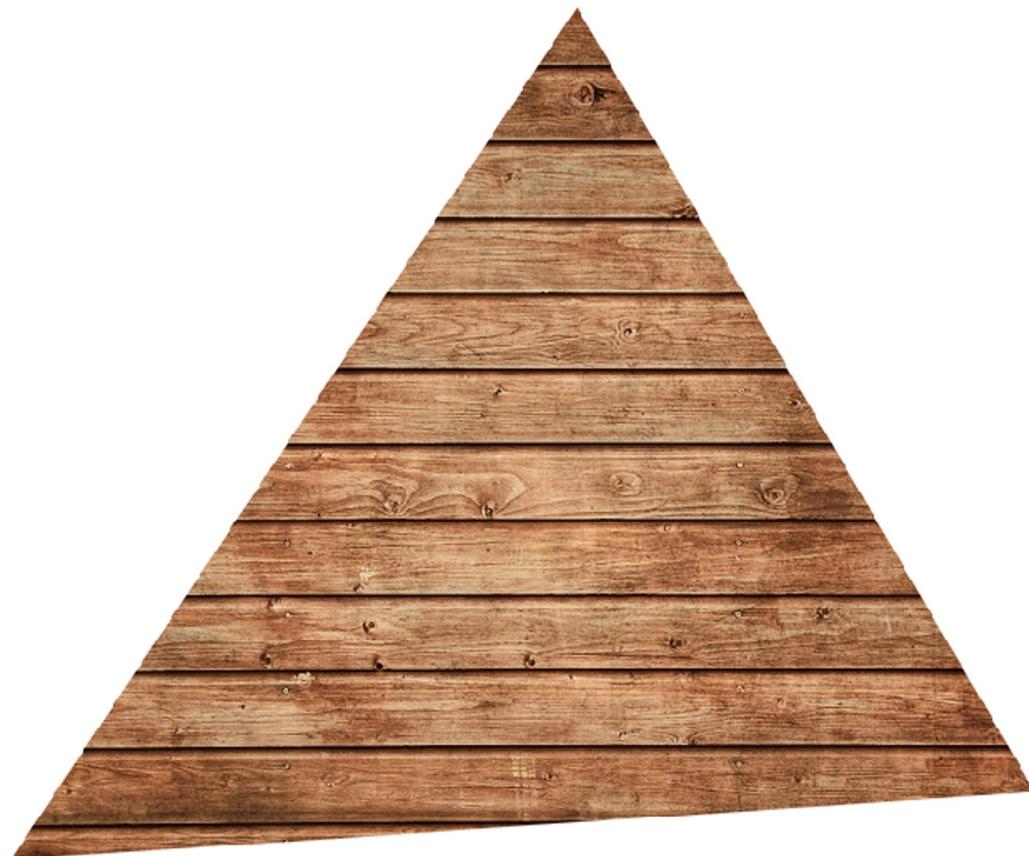
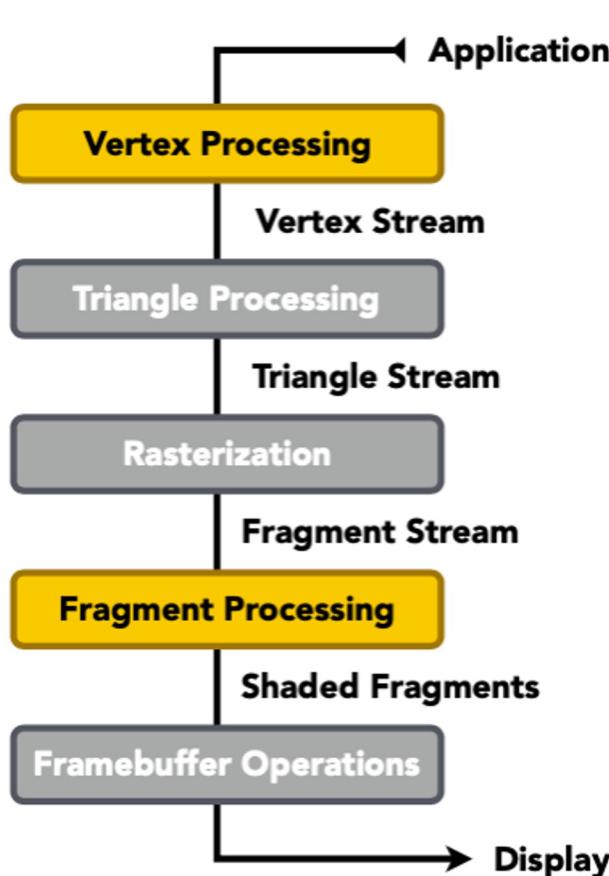
Amostrando os Triângulo



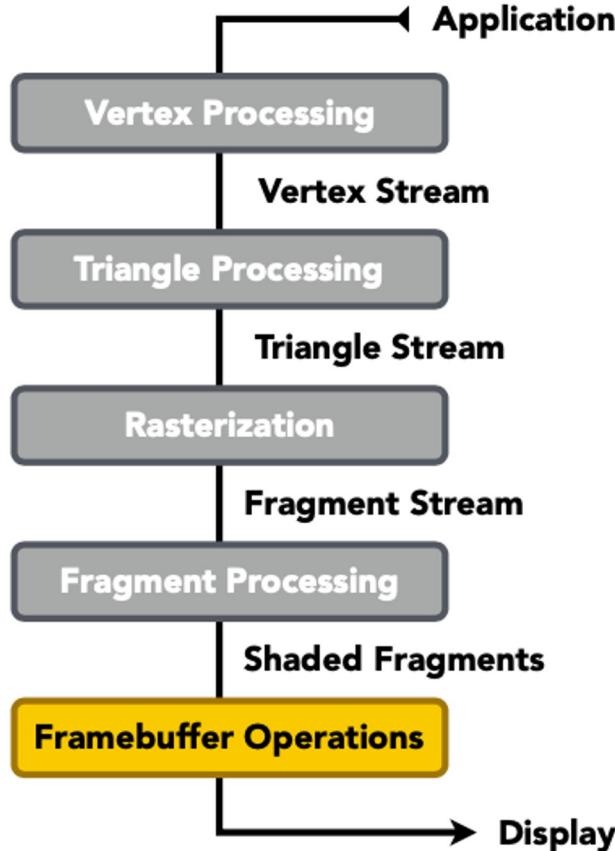
Avaliando a Função de Shading



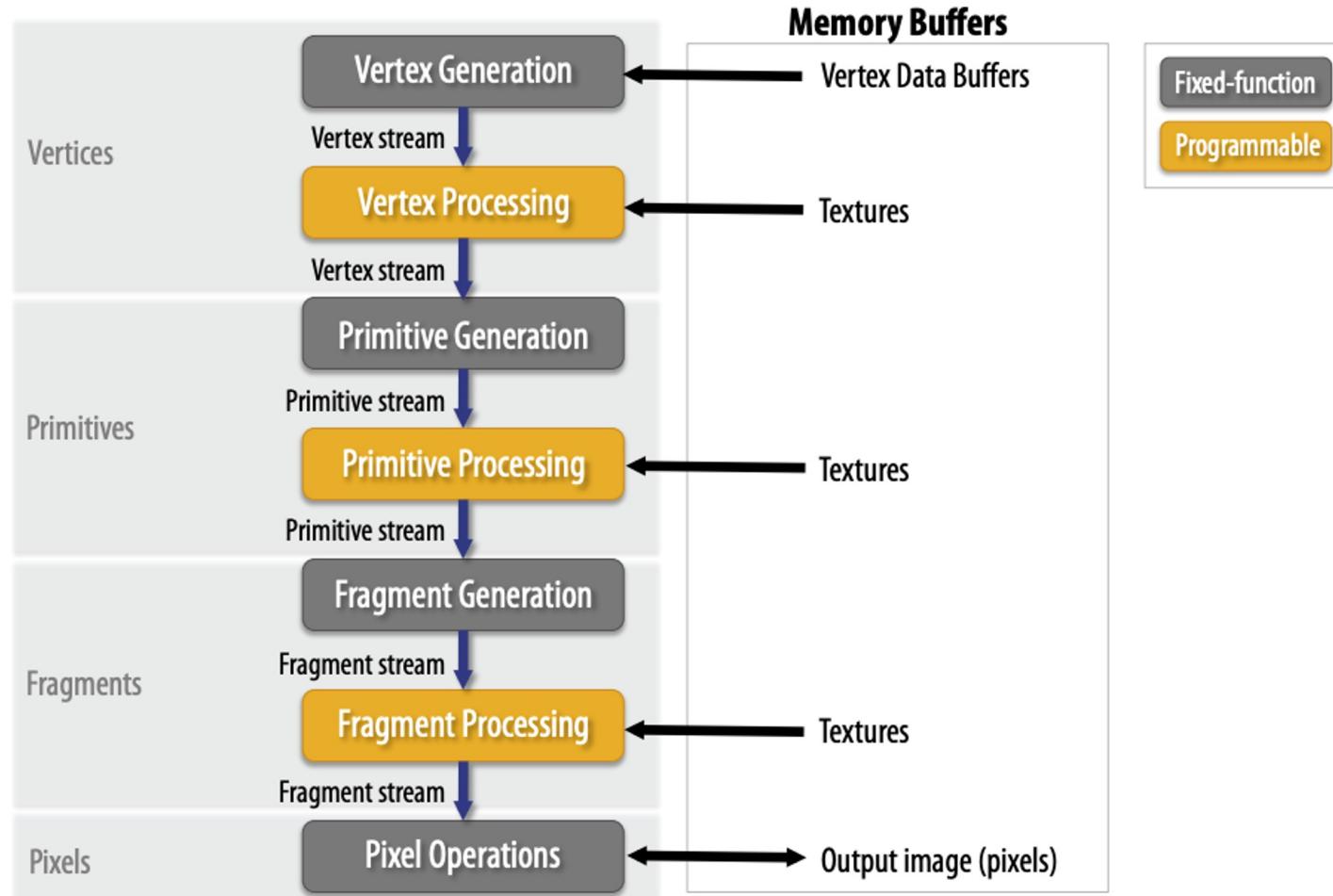
Mapeamento de Texturas



Teste de Visibilidade do Z-Buffer



Visão do Pipeline



Objetivo: Cenas 3D complexas em tempo real

Centenas de milhares a milhões de triângulos em uma cena

Cálculos complexos de vértices e fragmentos nos shaders

Alta resolução (2-4 megapixels + supersampling)

30-60 quadros por segundo (ainda mais alto para VR)

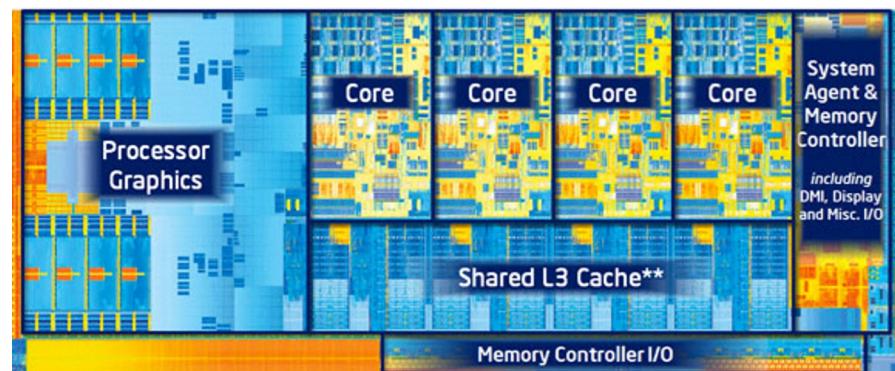


Implementações de Pipelines Gráficos : GPUs

Processadores especializados para executar cálculos de pipeline de gráficos



Placas discretas de GPU
NVIDIA GeForce 3090



GPU Integrada:
(Parte do die da CPU Intel)

Shaders Programáveis

Estágios de processamento de vértice e fragmento do programa
Descrever a operação em um único vértice (ou fragmento)

Exemplo de programa de shader em GLSL

```
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

void diffuseShader() {
    vec3 kd;
    kd = texture2d(myTexture, uv);
    kd *= clamp(dot(-lightDir, norm), 0.0, 1.0);
    gl_FragColor = vec4(kd, 1.0);
}
```

A função Shader é executada uma vez por fragmento.

- Exibe a cor da superfície na posição de amostra da tela do fragmento atual.
- Este shader executa uma pesquisa de textura para obter a cor do material da superfície no ponto e, em seguida, executa um cálculo de iluminação difusa.

Compilação de um Shader

1 fragmento de entrada não processado



```
 sampler mySampler;
Texture2D<float3> myTexture;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
    float3 kd;
    kd = myTexture.Sample(mySampler, uv);
    kd *= clamp ( dot(lightDir, norm), 0.0, 1.0);
    return float4(kd, 1.0);
}
```



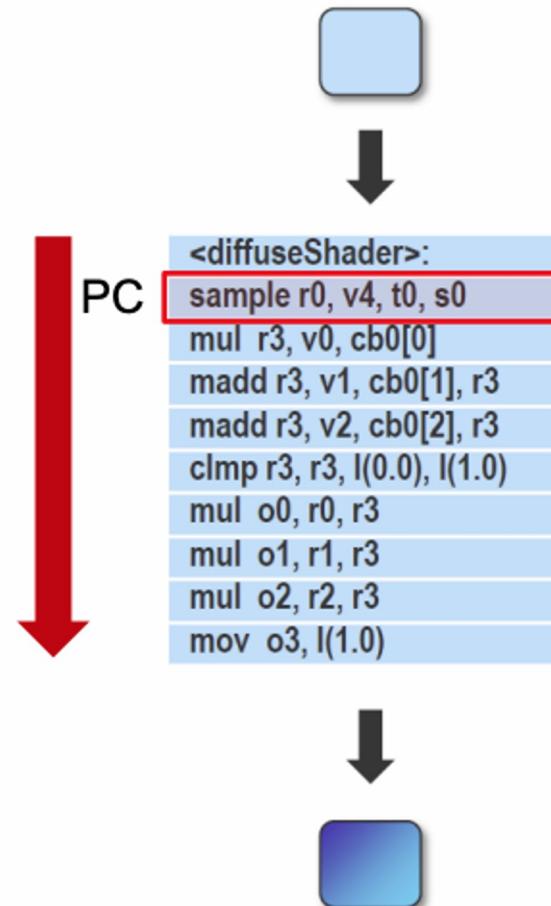
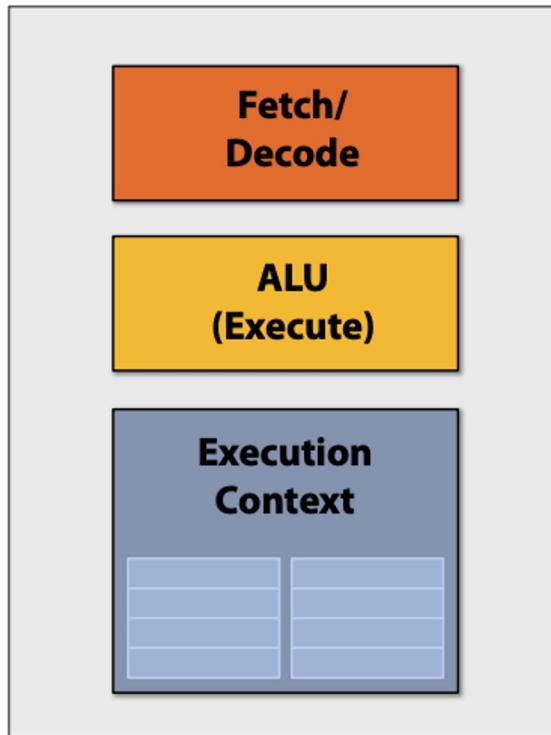
```
<diffuseShader>:
sample r0, v4, t0, s0
mul r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul o0, r0, r3
mul o1, r1, r3
mul o2, r2, r3
mov o3, l(1.0)
```



1 fragmento de saída processado

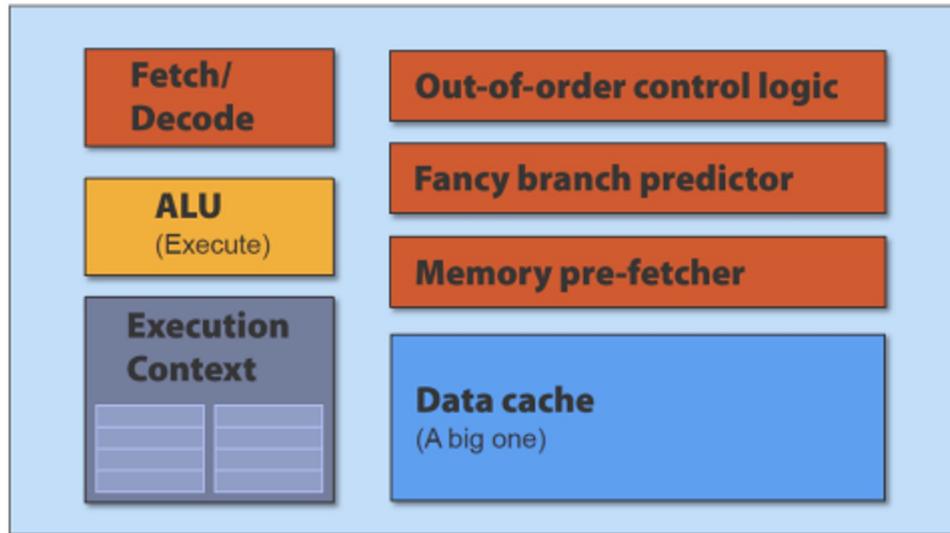
Mapeamento do Shader no HW

Execute o Shader em um único core:



Mapeamento do Shader no HW

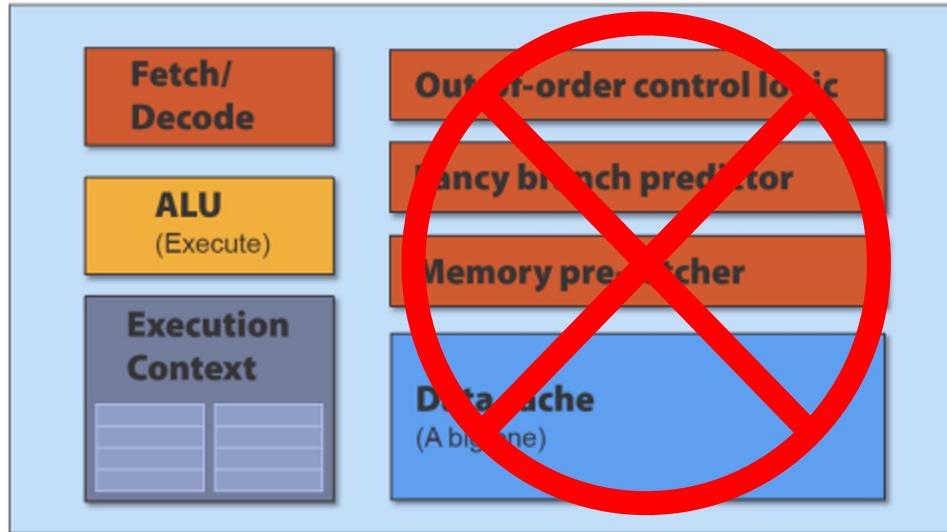
Um core de CPU



- Otimizado para acesso de baixa latência aos dados em cache
- Lógica de controle para execução fora de ordem e especulativa
- Grande cache L2

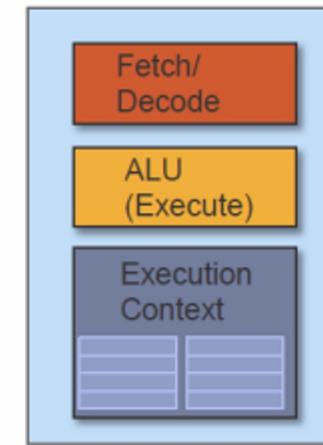
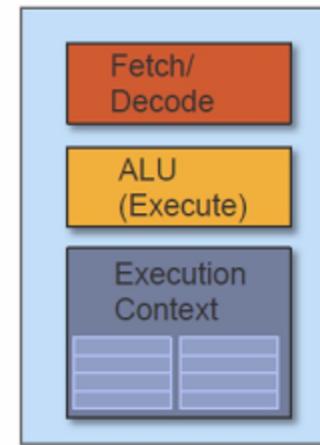
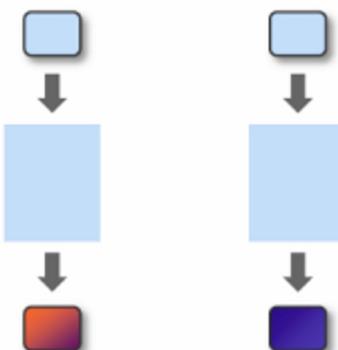
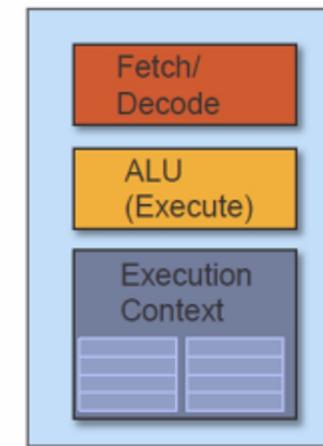
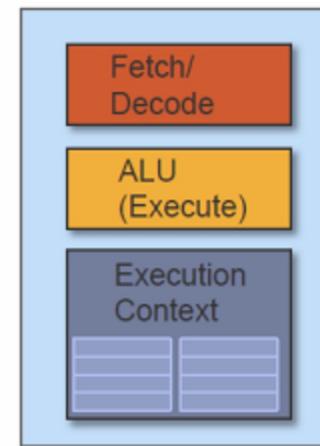
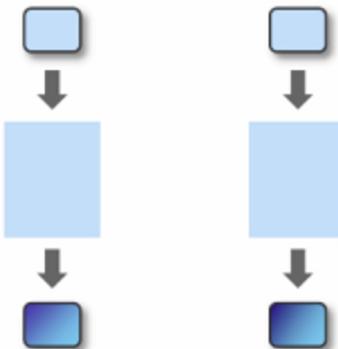
Reduzindo os Cores

Um core de GPU

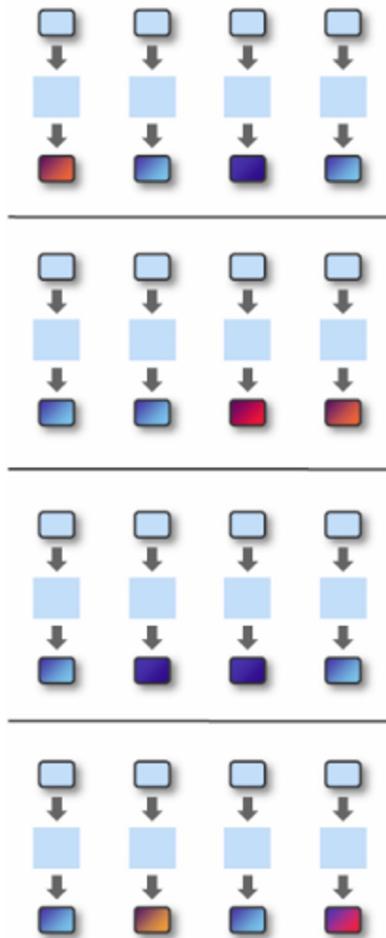


- Otimizado para computação paralela de dados
- Arquitetura tolerante à latência de memória
- Mais cálculos por mais transistores em ALUs
- Redução dos circuitos principais de controle

Múltiplas Threads



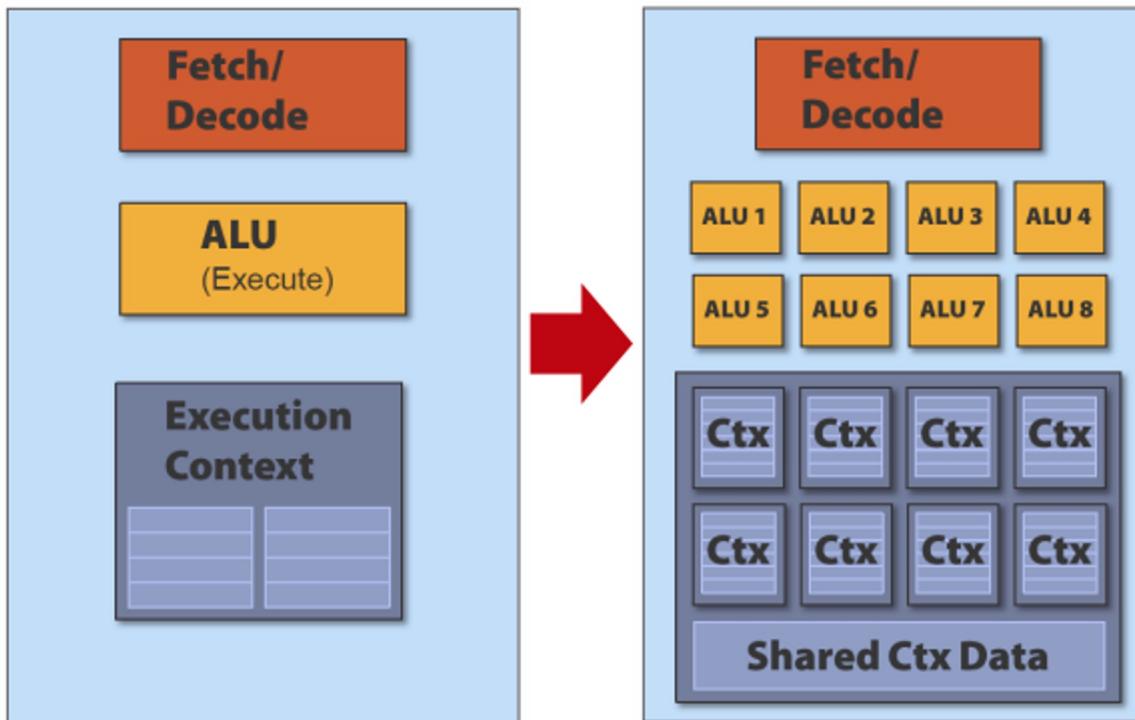
GPUs com muitos Cores



16 cores = 16 fluxos de instruções simultâneos

Múltiplos dados (SIMD)

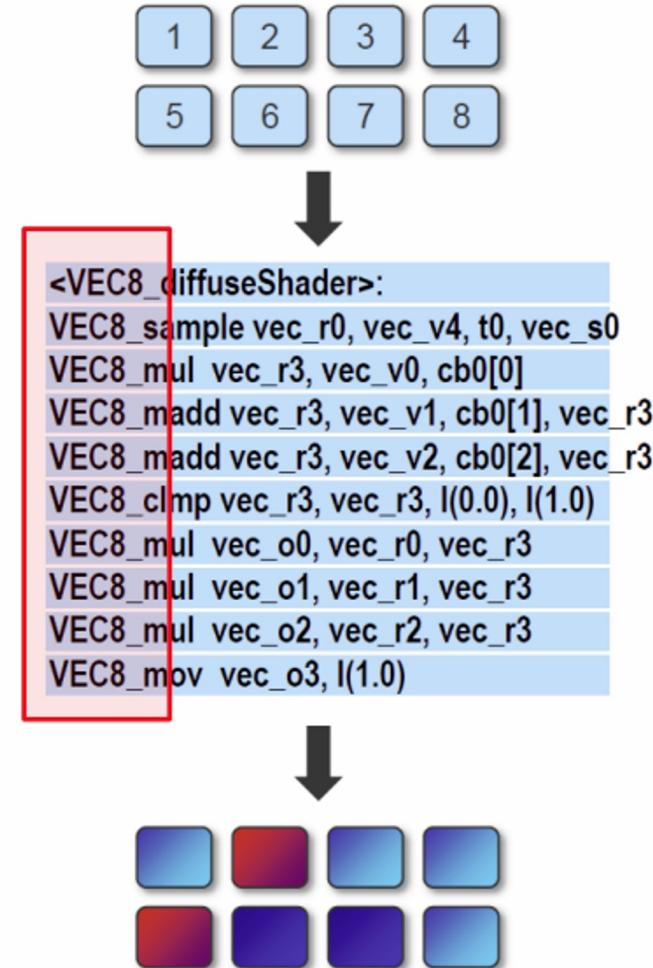
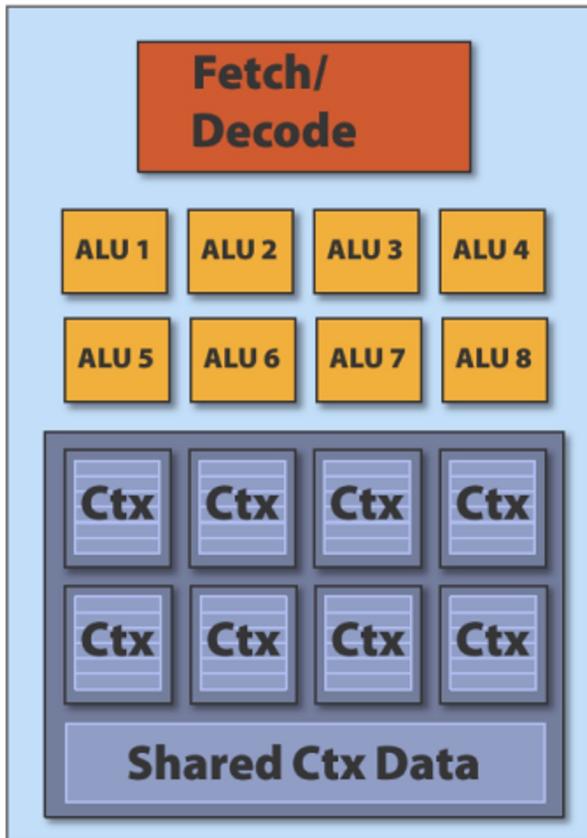
Shaders são inherentemente executados muitas vezes, repetidas vezes em vários registros de seus fluxos de dados de entrada.



Amortize o custo / complexidade do gerenciamento de instruções para várias ALUs.

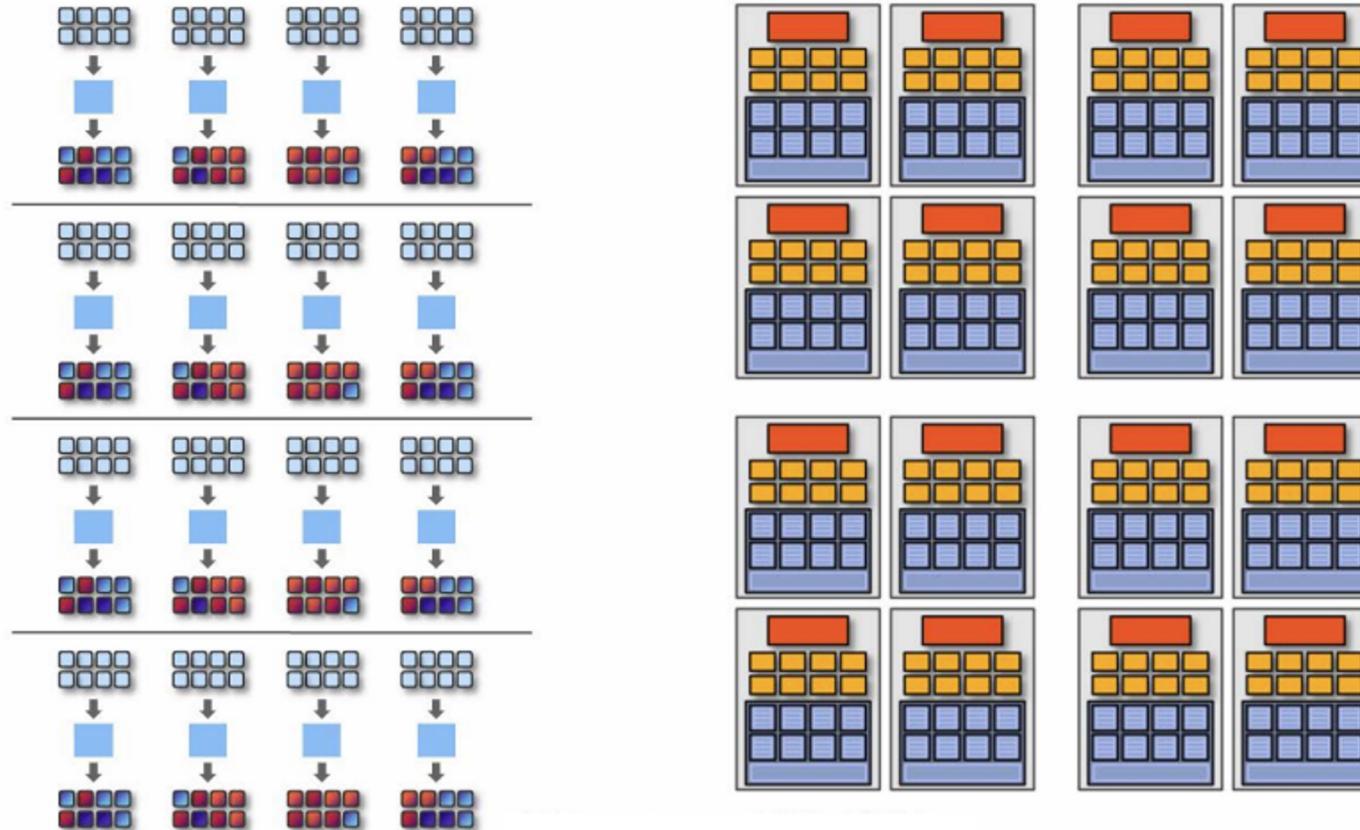
Compartilhe a unidade de instrução.

SIMD Cores: Vectorized Instruction Set



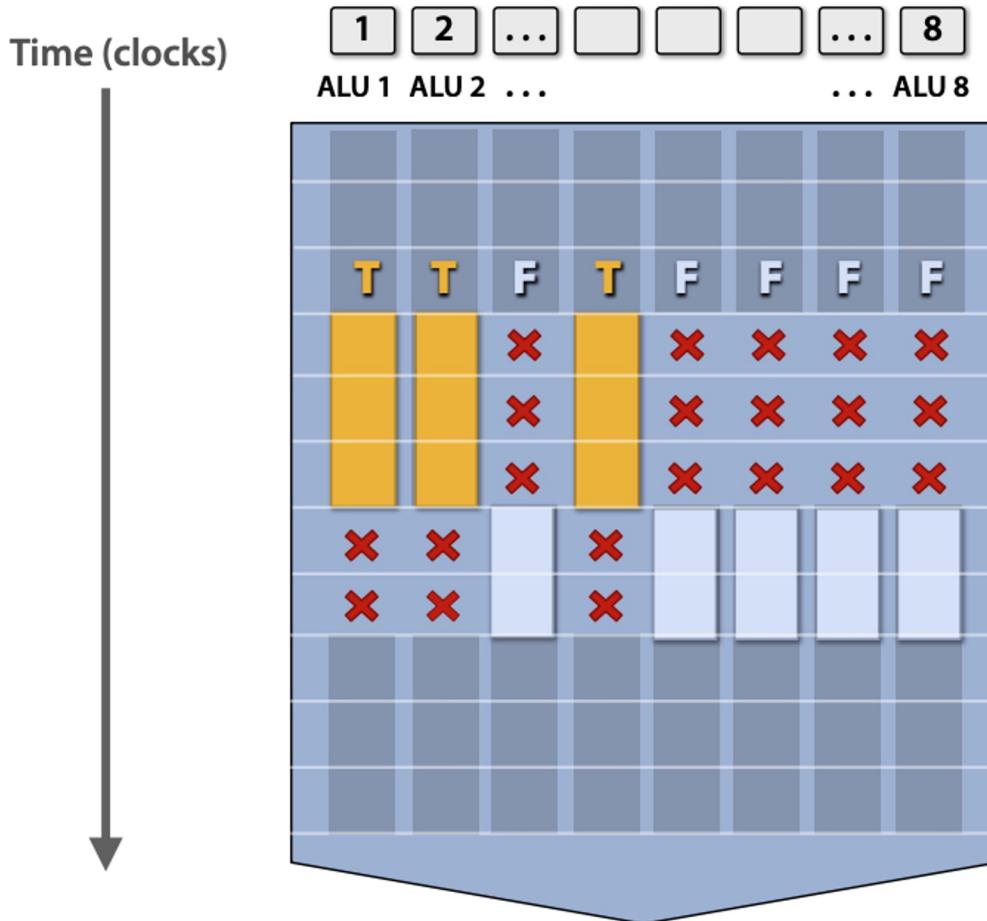
Adicionando tudo: vários núcleos SIMD

Neste exemplo: 128 dados processados simultaneamente



16 cores = 128 ALUs = 16 fluxos de instruções simultâneos

Condicionais / Branches



```
<unconditional shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional shader code>
```

Computação Gráfica

Luciano Soares
[<lpssoares@insper.edu.br>](mailto:lpssoares@insper.edu.br)