Insper

Computação Gráfica

Aula 18: Pipeline Gráfico

Immediate Mode vs Retained Mode

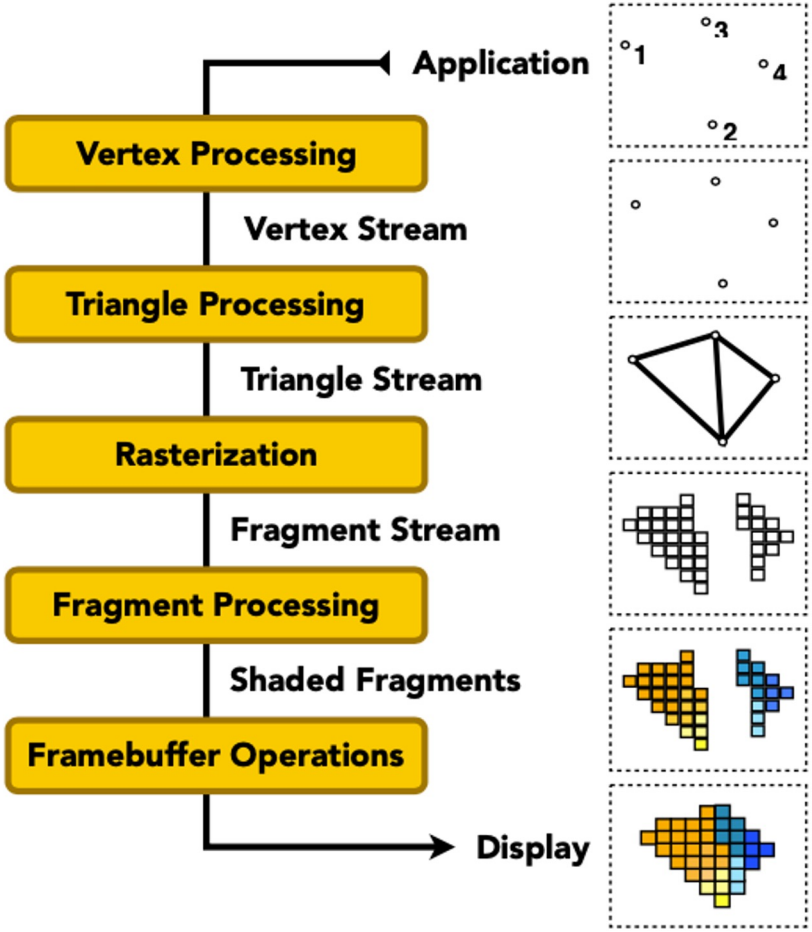
No modo imediato (Immediate Mode) da API gráfica, a aplicação envia todos os comandos de desenho a cada quadro (obsoleto para OpenGL).

```
glBegin(GL_TRIANGLES);
    glVertex3f( 0.0f, 1.0f, 0.0f);
    glVertex3f(-1.0f,-1.0f, 0.0f);
    glVertex3f( 1.0f,-1.0f, 0.0f);
glEnd();
glTranslatef(3.0f,0.0f,0.0f);
```

No modo retido (Retained Mode) da API gráfica, o prepara todos os dados da cena, e para cada vez que um novo quadro é desenhado, a biblioteca gráfica usa esses dados para desenhar a cena.

```
Int bufferIDs[] = new int[2];
gl.glGenBuffers(2, bufferIDs,0);
gl.glBindBuffer(GL_ARRAY_BUFFER, bufferIDs[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(MyVertex)*3, &pvertex[0].x, GL_STATIC_DRAW);
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_SHORT, BUFFER_OFFSET(0));
```

Pipeline de Rasterização



Vértices no espaço 3D

Vértices posicionados em coordenadas da tela

Triângulos posicionados em coordenadas da tela

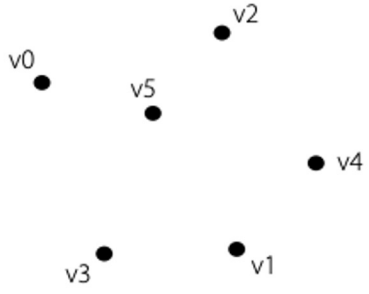
Fragmentos

Fragmentos tonalizados

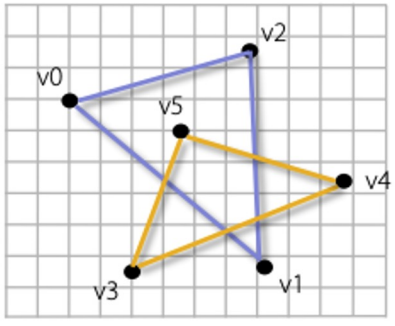
Imagem (pixels)

Insper

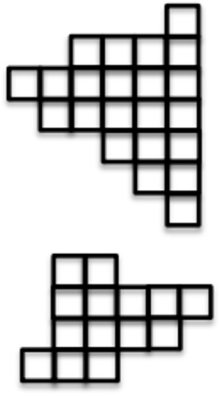
Entidades na Pipeline



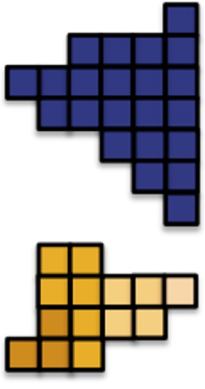
Vertices



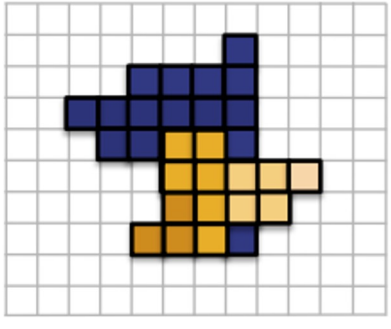
Primitives



Fragments



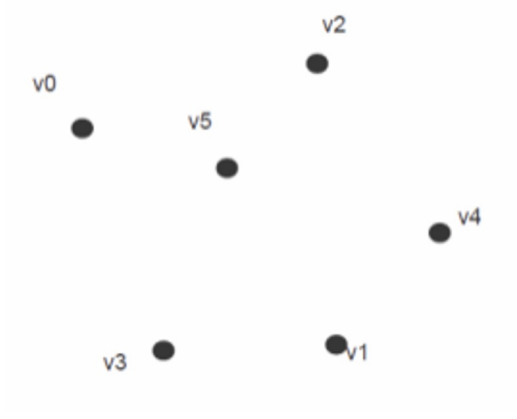
Fragments (shaded)



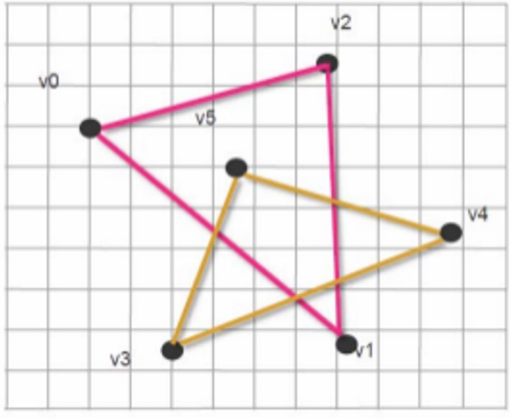
Pixels

Insper

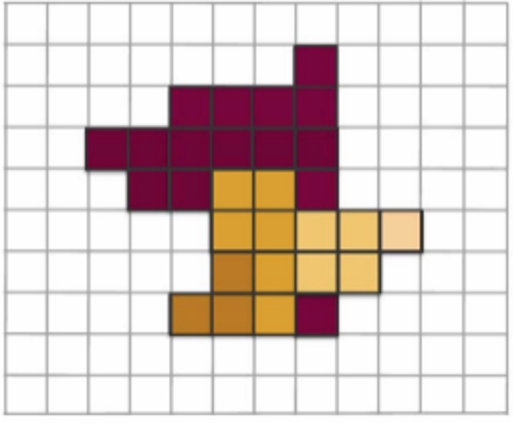
Como os Shaders são Invocados



Vertex Shader
invocado 6 vezes



Geometry Shader
invocado 2 vezes

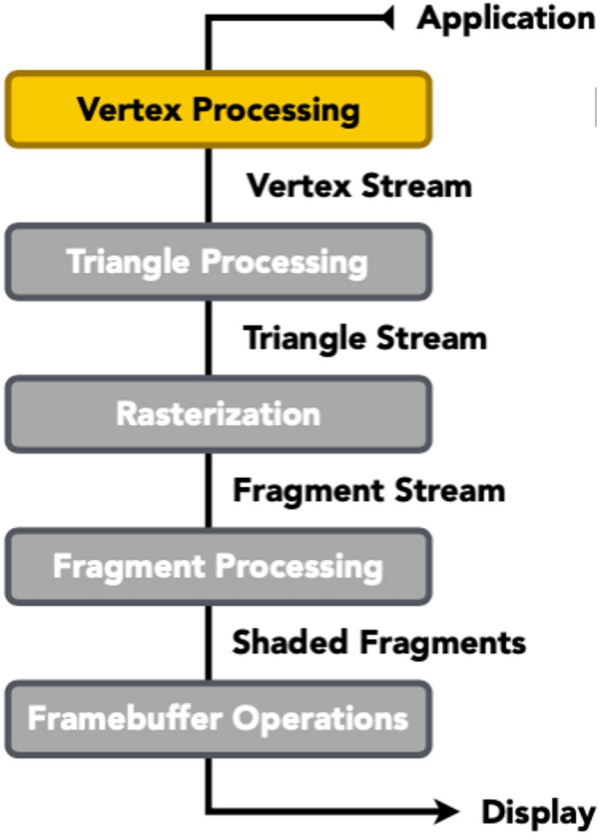


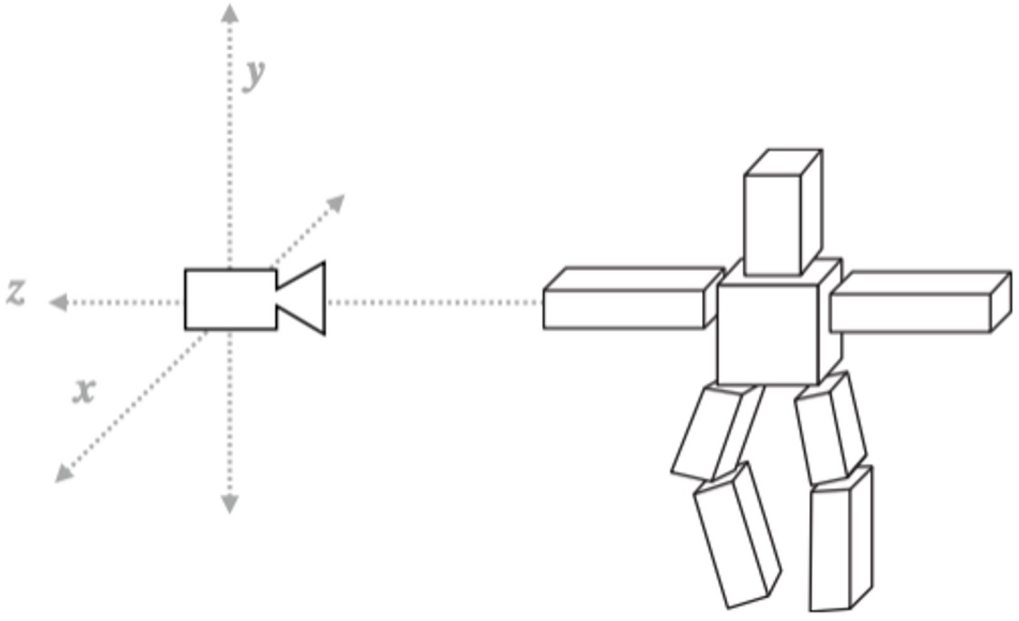
Fragment Shader
invocado 35 vezes

(para os fragmentos
ocultos também)

fonte: https://eclass.aueb.gr/modules/document/file.php/INF143

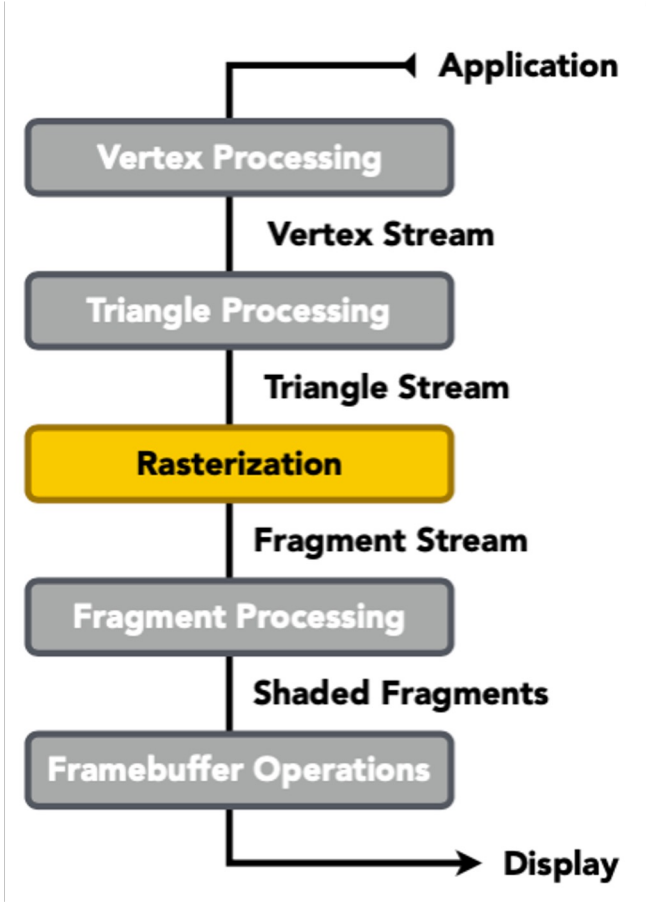
Transformações Geométricas e Visualização

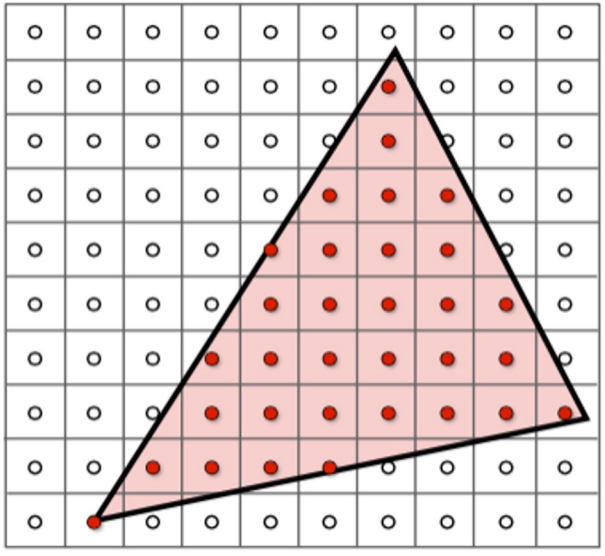




Insper

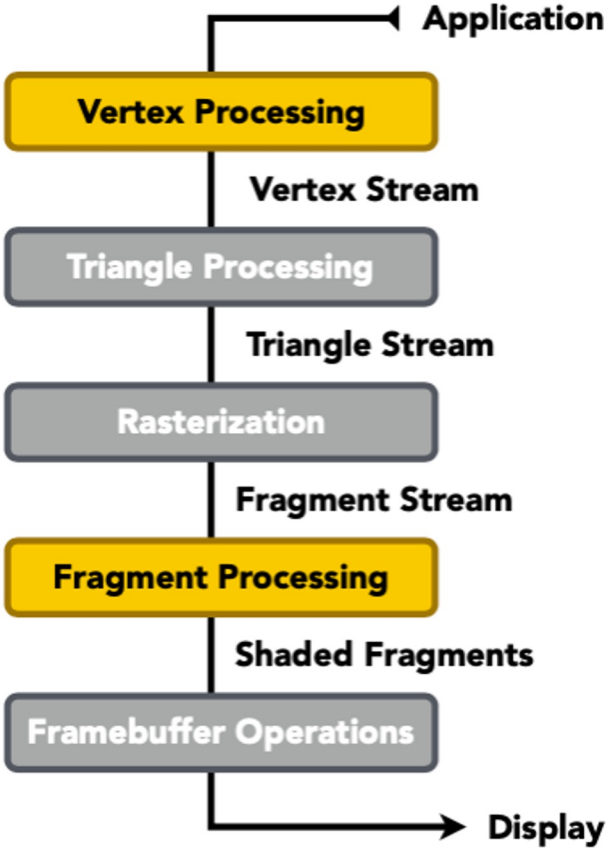
Amostrando os Triângulo

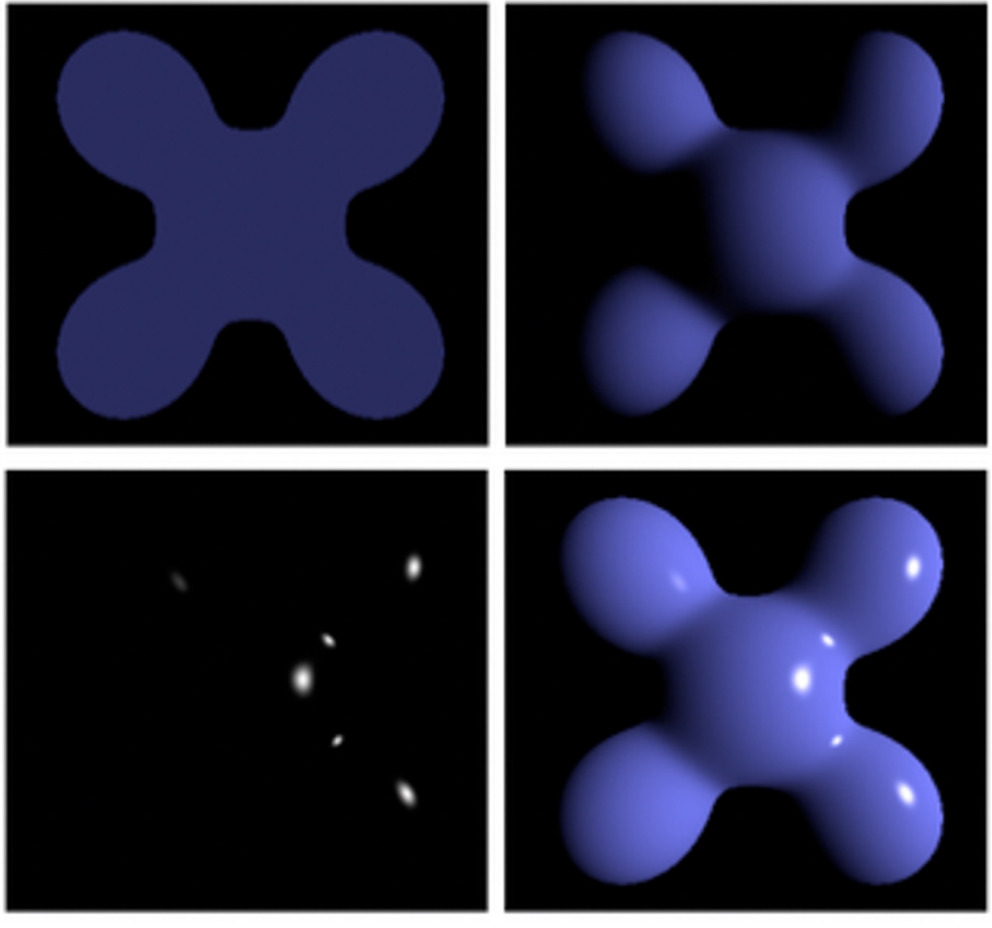




Insper

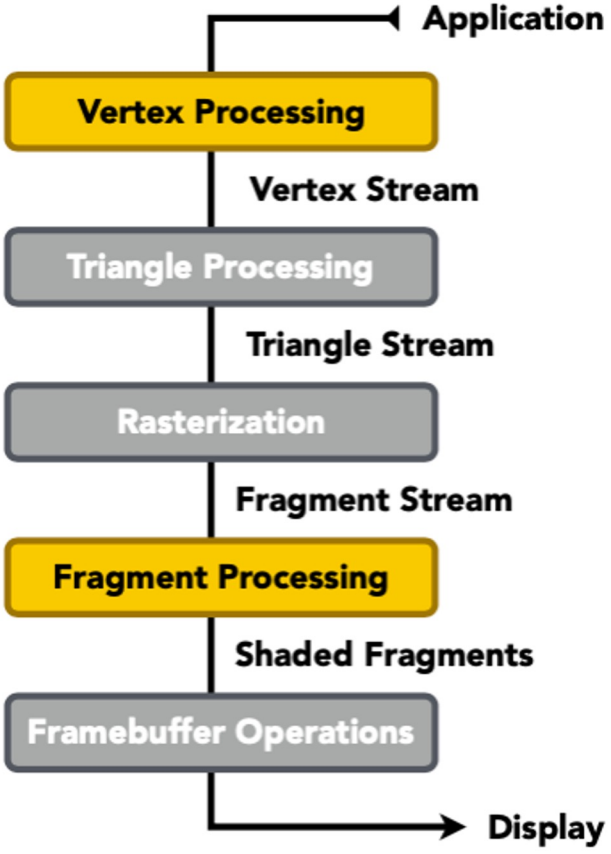
Avaliando a Função de Shading





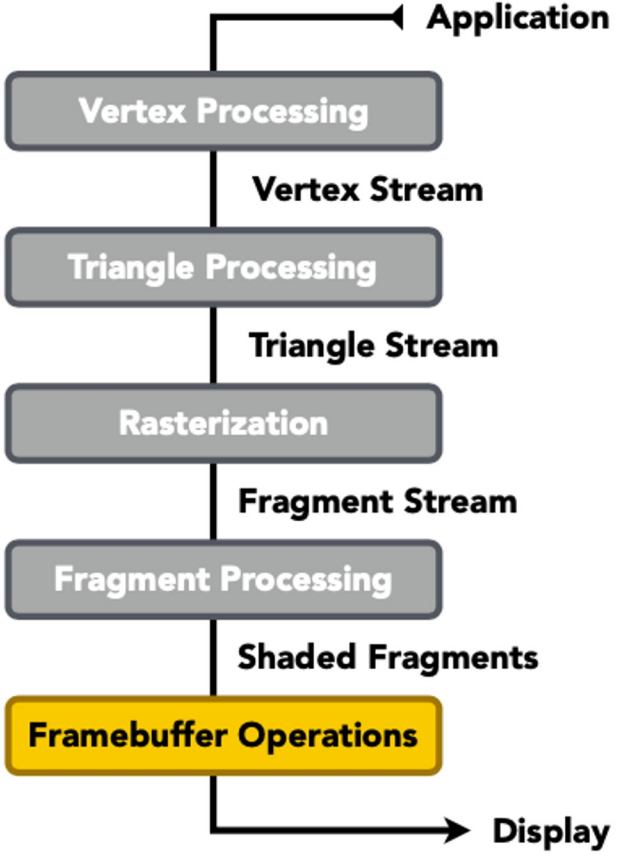
Insper

Mapeamento de Texturas



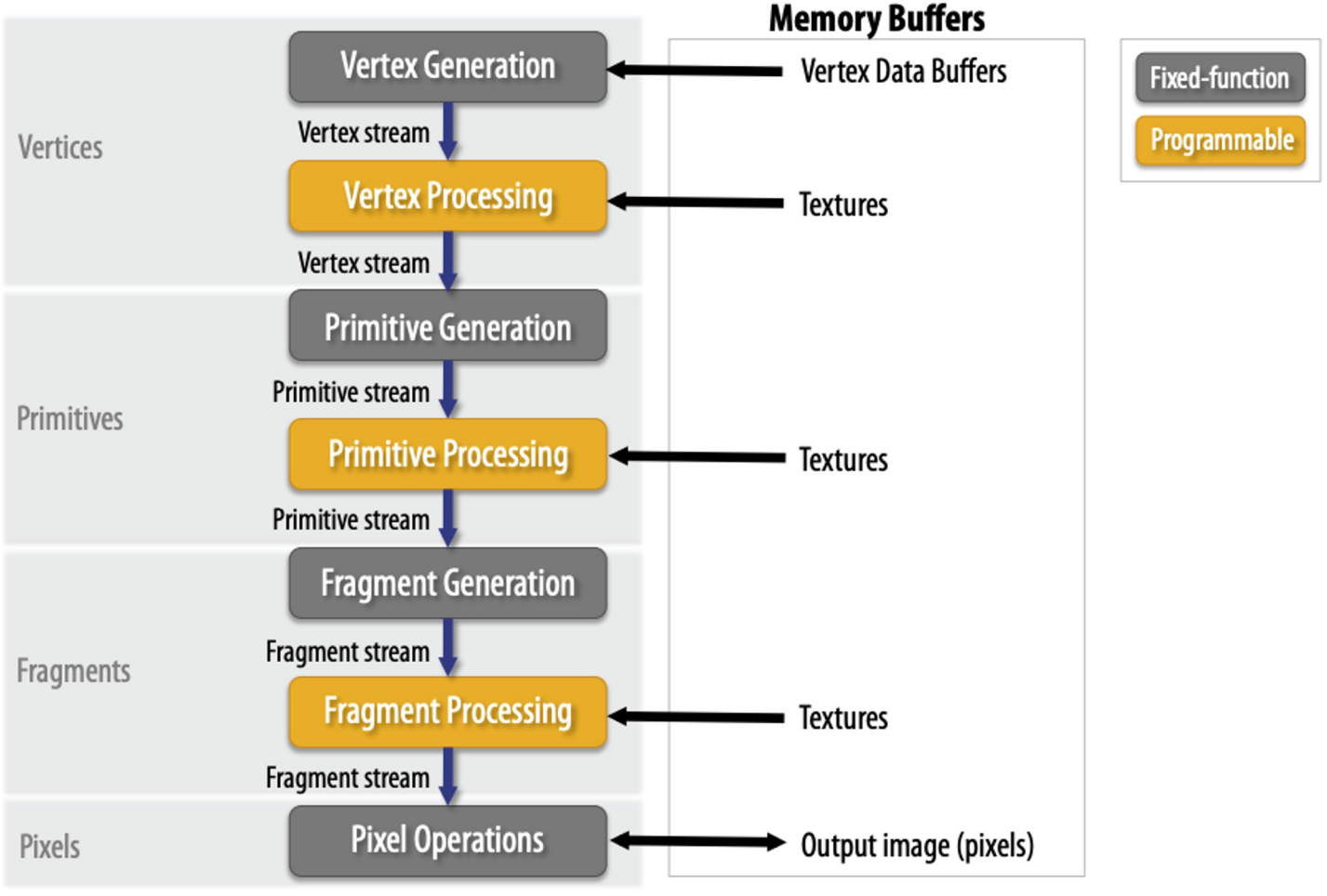
Insper

Teste de Visibilidade do Z-Buffer



Insper

Visão do Pipeline



fonte: https://www.cs.cmu.edu/afs/cs/academic/class/15462-f11/www/lec_slides/lec19.pdf

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)

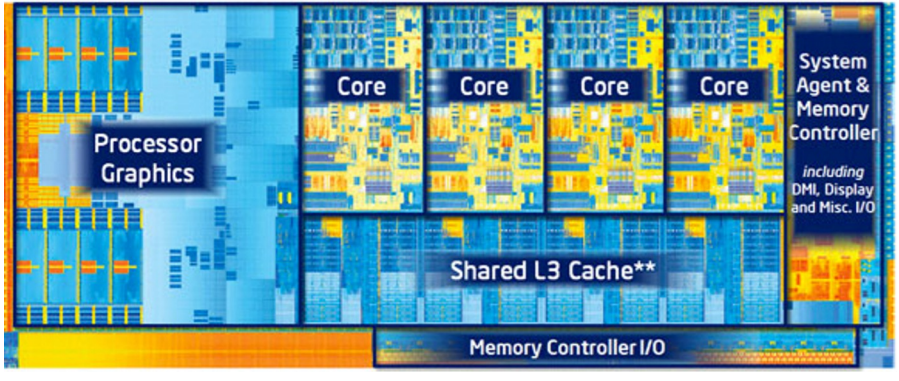
Unrecord - Official Early Gameplay Trailer

Insper

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 = texture(myTexture, uv);
   kd *= clamp(dot(-lightDir, norm), 0.0, 1.0);
   gl_FragColor = vec4(kd, 1.0);
}
```

A função 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);
}
```

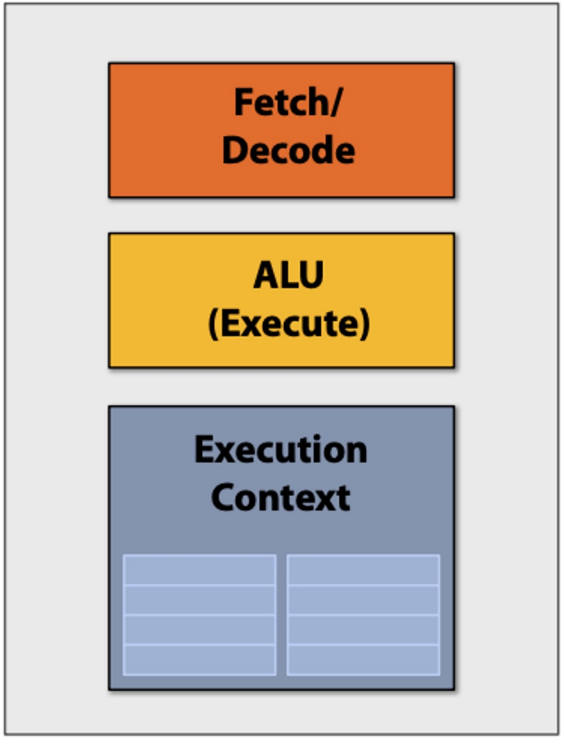
```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

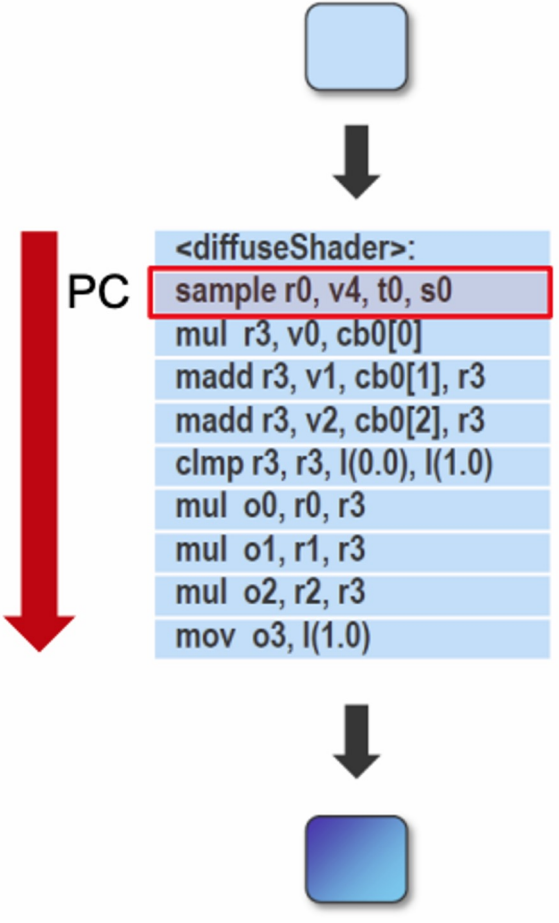
1 fragmento de saída processado

fonte: https://eclass.aueb.gr/modules/document/file.php/INF143

Mapeamento do Shader no HW

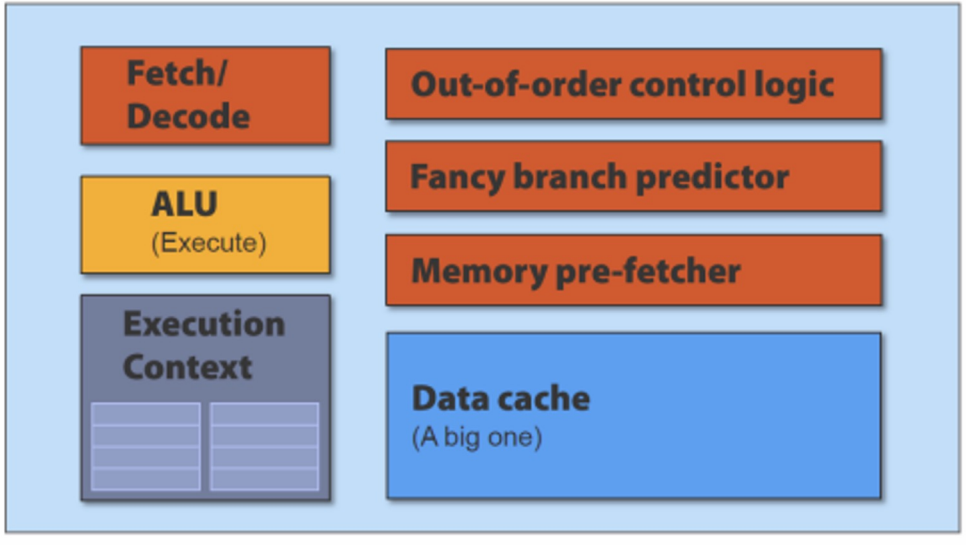
Execute o Shader em um único core:





Mapeamento do Shader no HW

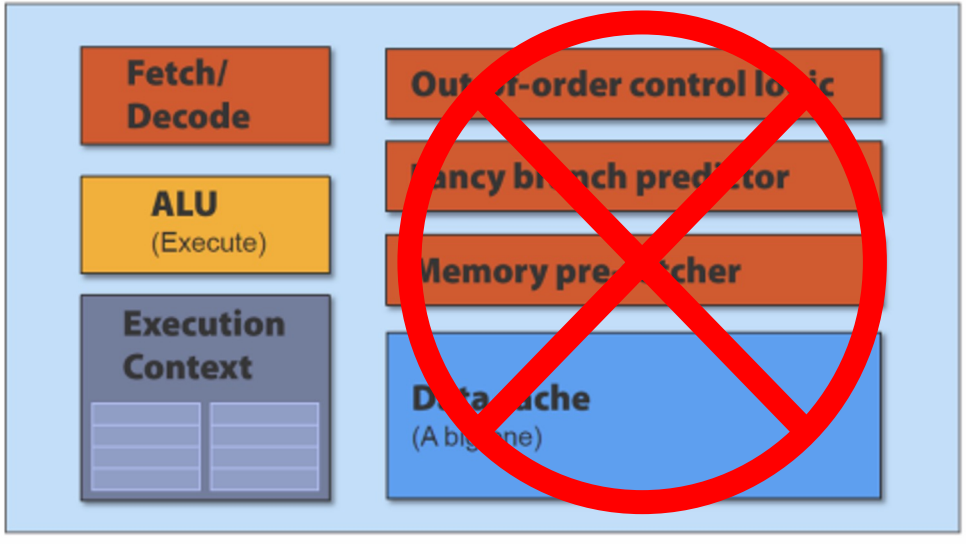
Um *core* de CPU



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

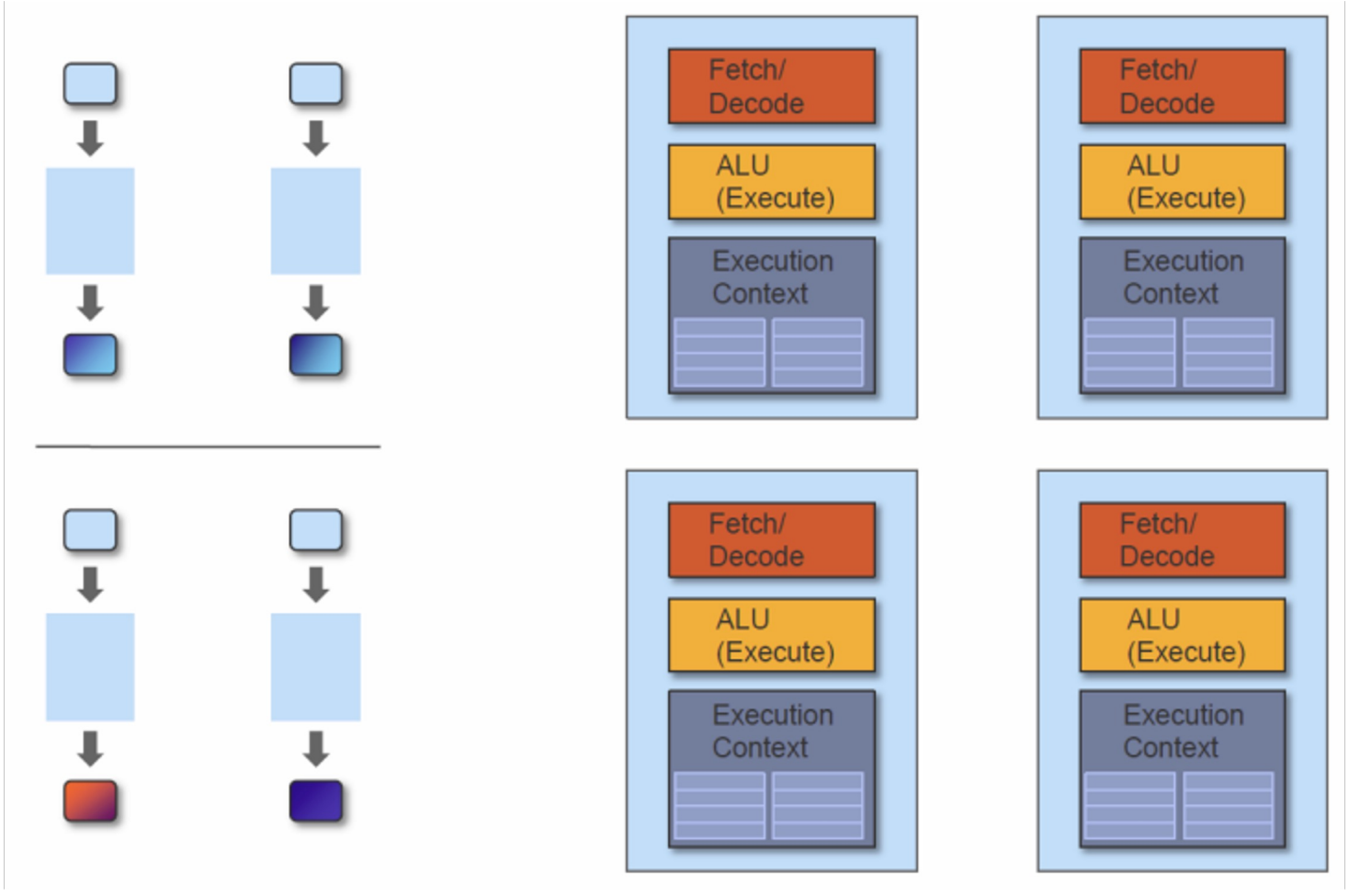
Reduzindo os Cores

Um *core* de GPU



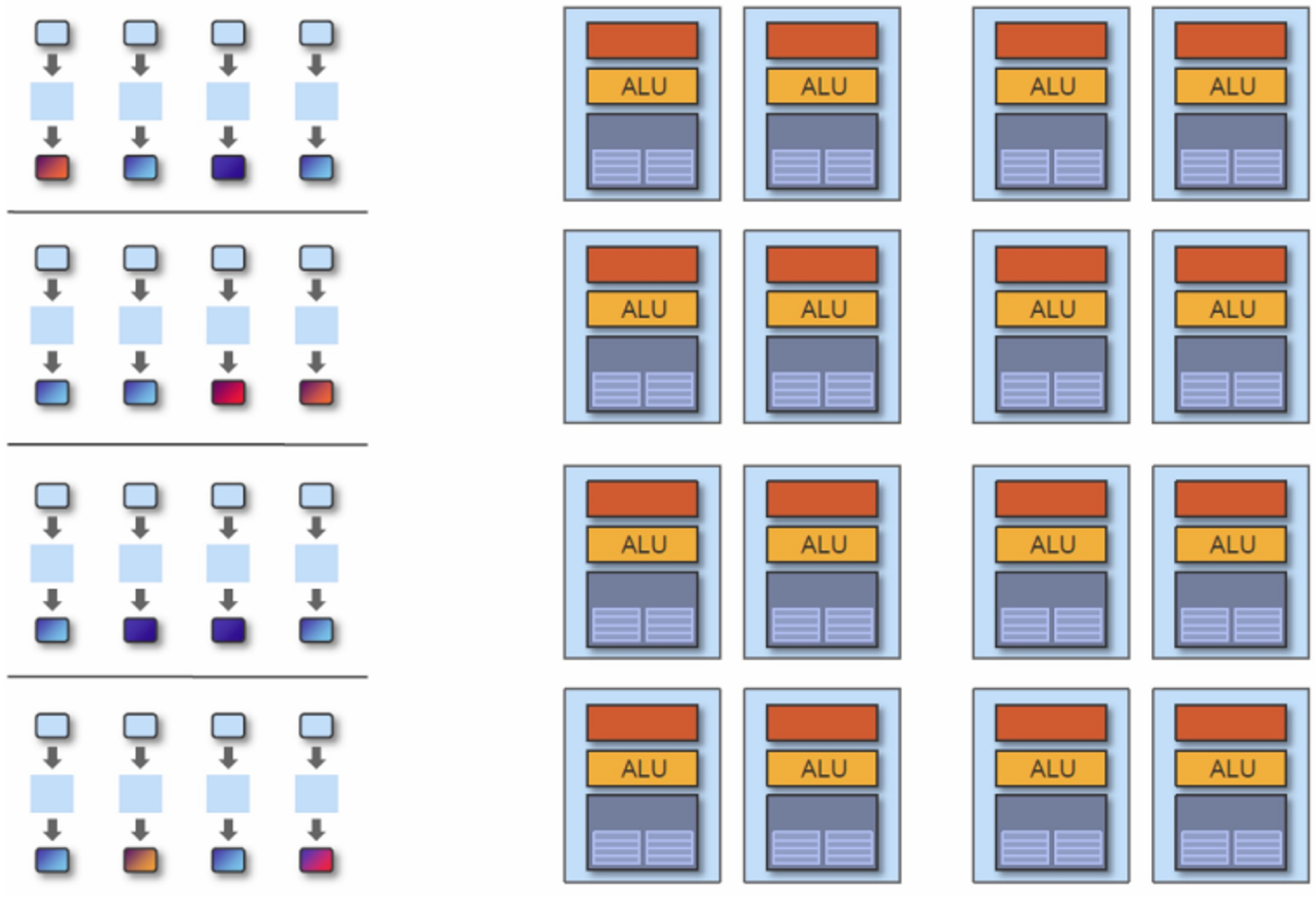
- Otimizado para computação paralela de dados
- Arquitetura tolerante à latência de memória
- Mais cálculos por mais transistores em ALUs
- Redução dos circuitos principais de controle

Múltiplas Threads



Insper

GPUs com muitos *Cores*

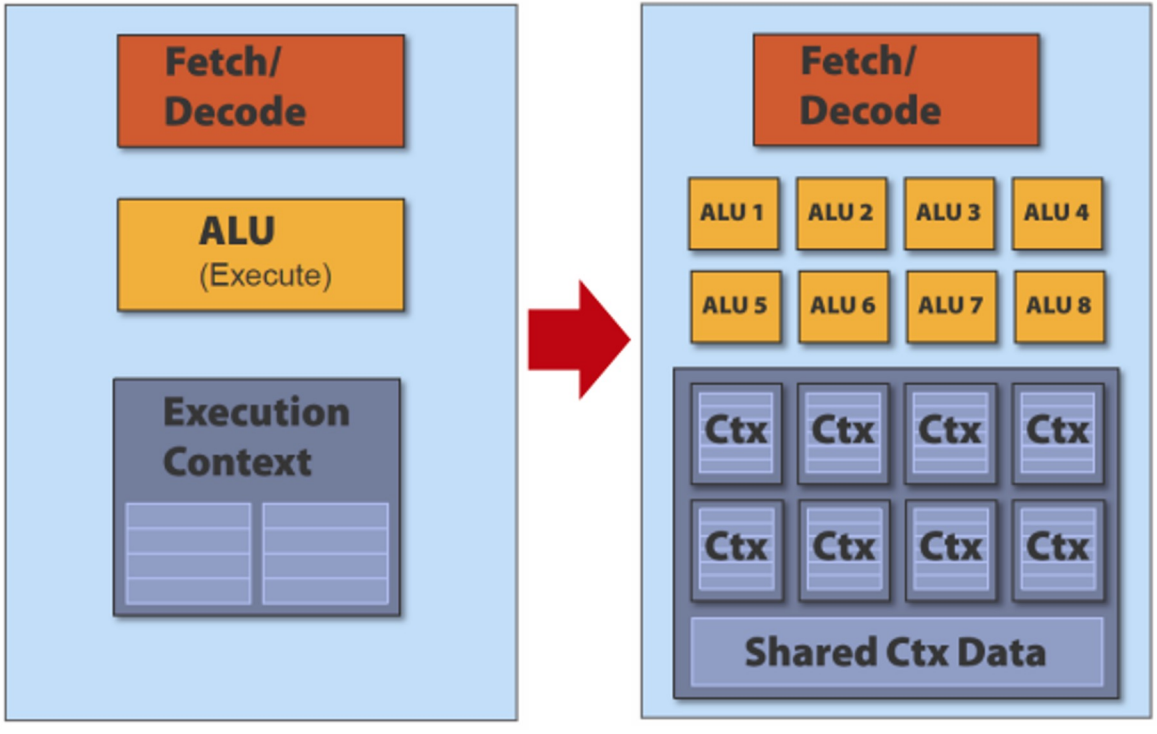


16 cores = 16 fluxos de instruções simultâneos

Insper

Múltiplos dados (SIMD)

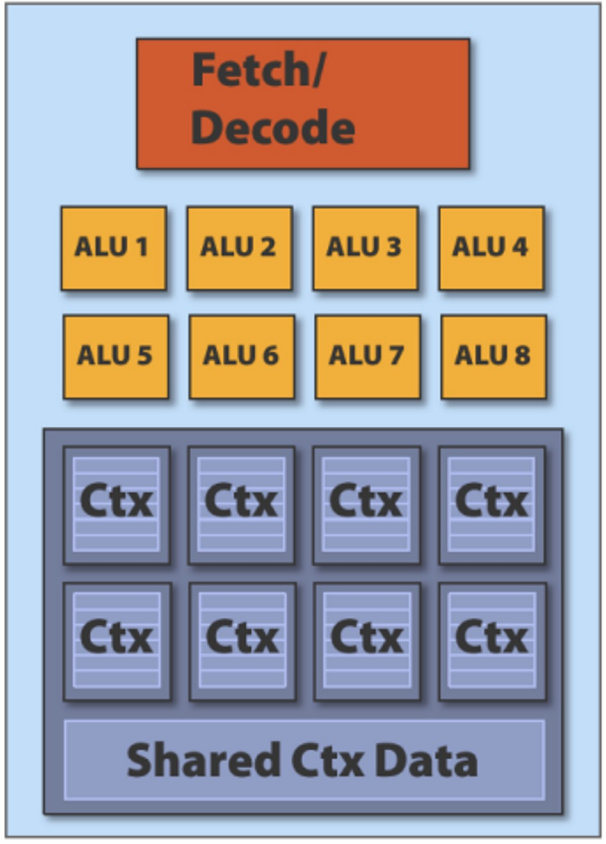
Shaders são inerentemente executados muitas vezes, repetidas vezes em vários registros de seus fluxos de dados de entrada.

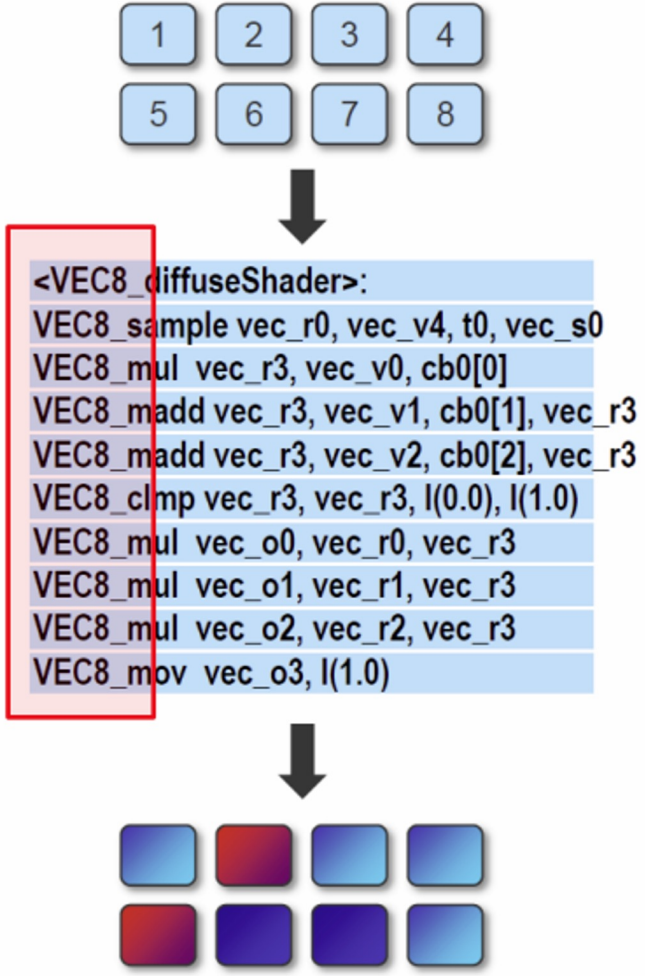


Amortize o custo / complexidade do gerenciamento de instruções para várias ALUs.

Compartilhe a unidade de instrução.

SIMD Cores: Vectorized Instruction Set

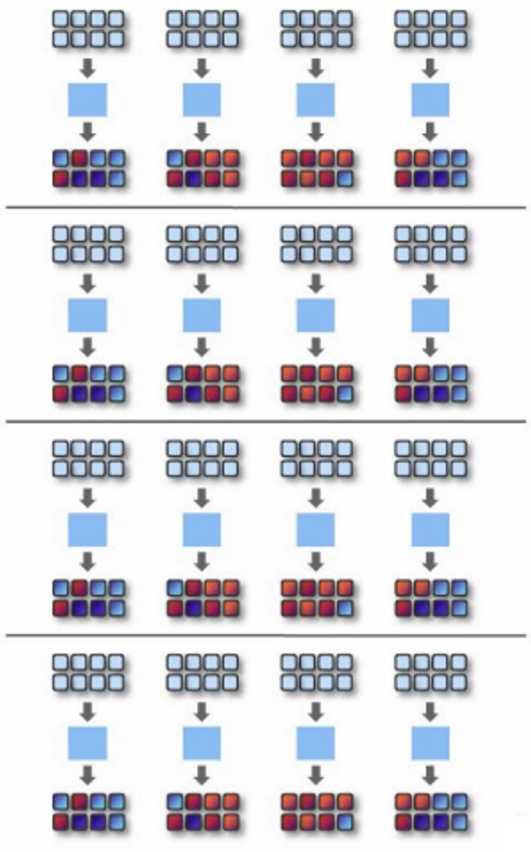


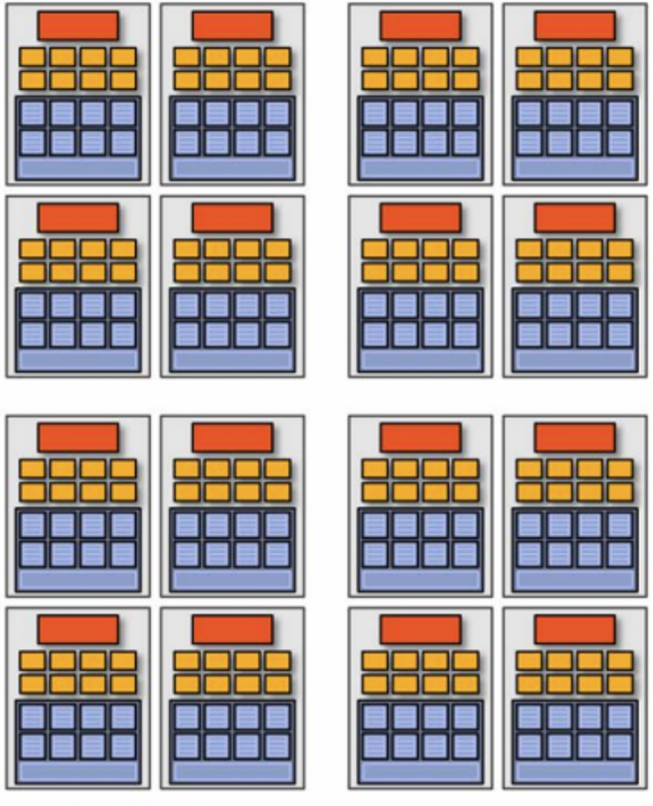


Insper

Adicionando tudo: vários núcleos SIMD

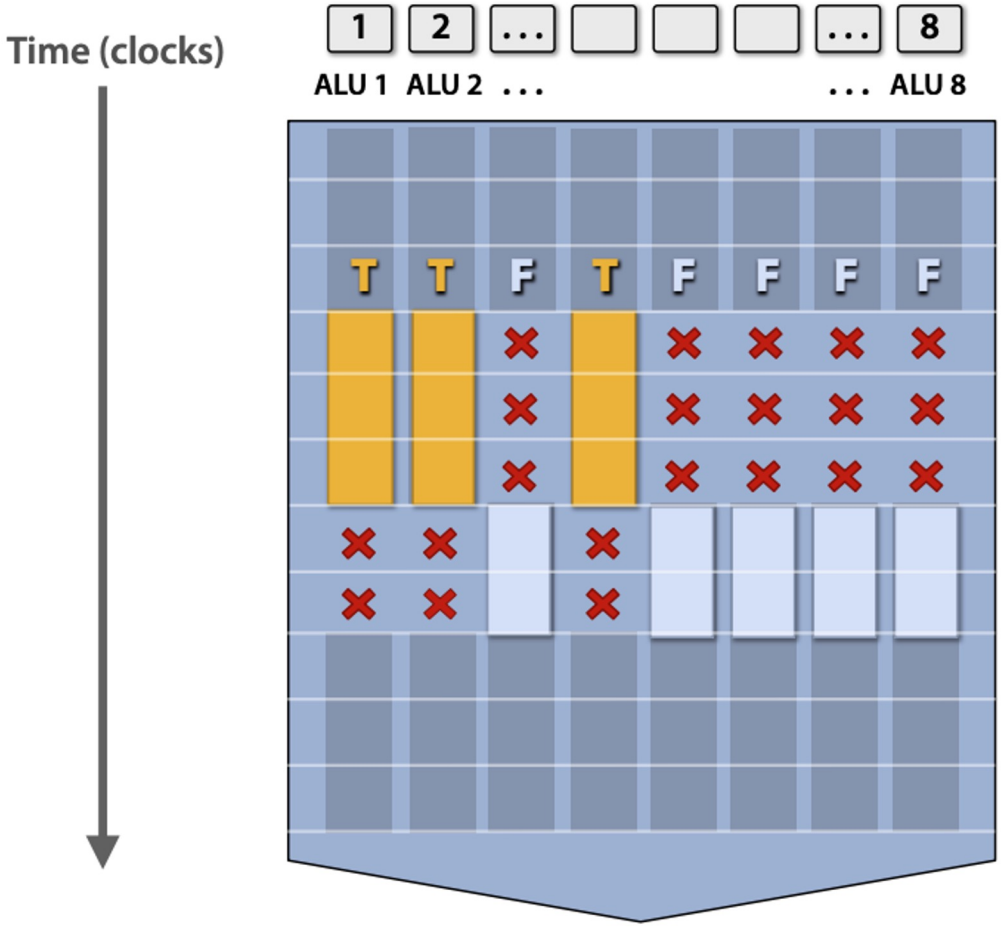
Neste exemplo: 128 dados processados simultaneamente





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>
```


Insper

Insper

Computação Gráfica

Luciano Soares
<lpsoares@insper.edu.br>