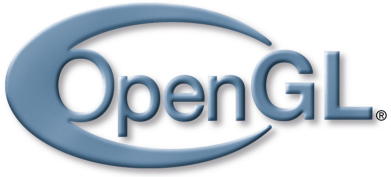
Computação Gráfica

TCC-00291

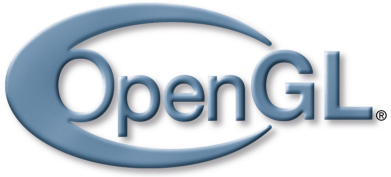
Assunto: GLSL



GLSL
Introdução

O que é? Por que preciso estudar?

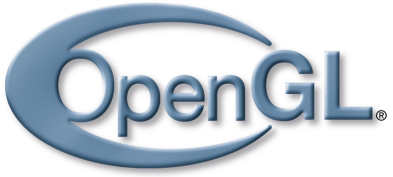
- *OpenGL shading language*.



GLSL
Introdução

O que é? Por que preciso estudar?

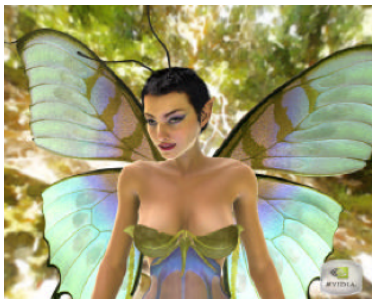
- *OpenGL shading language*.
- Podemos aliviar a CPU de processamento pesado.
- Tempo real.



GLSL
Introdução

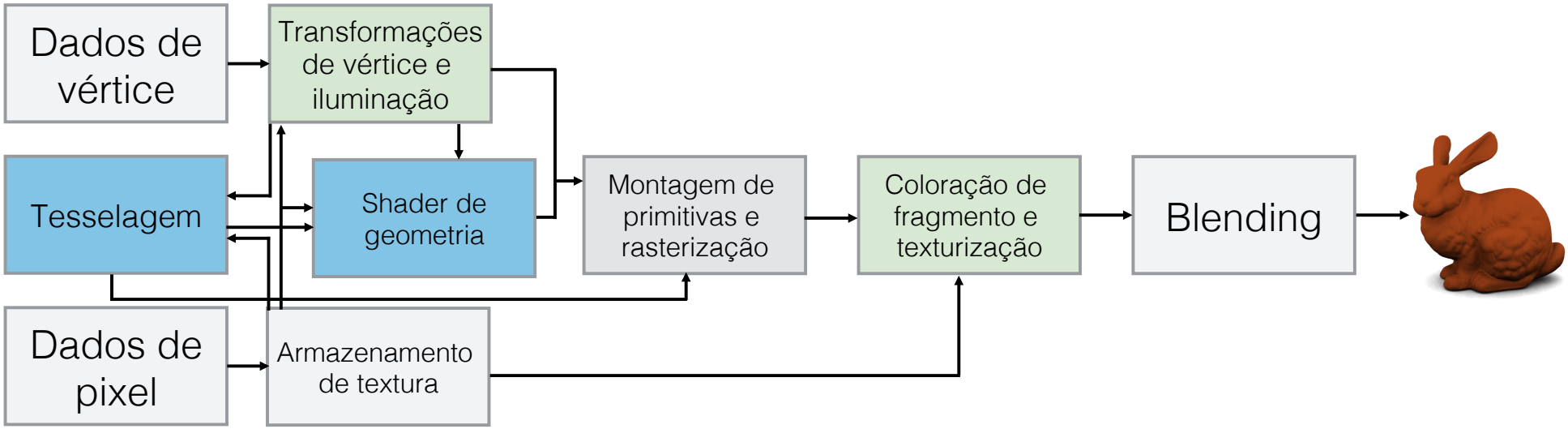
O que é? Por que preciso estudar?

- *OpenGL shading language*.
- Podemos aliviar a CPU de processamento pesado.
- Tempo real.
- Flexibilidade para criar novos “efeitos" visuais.



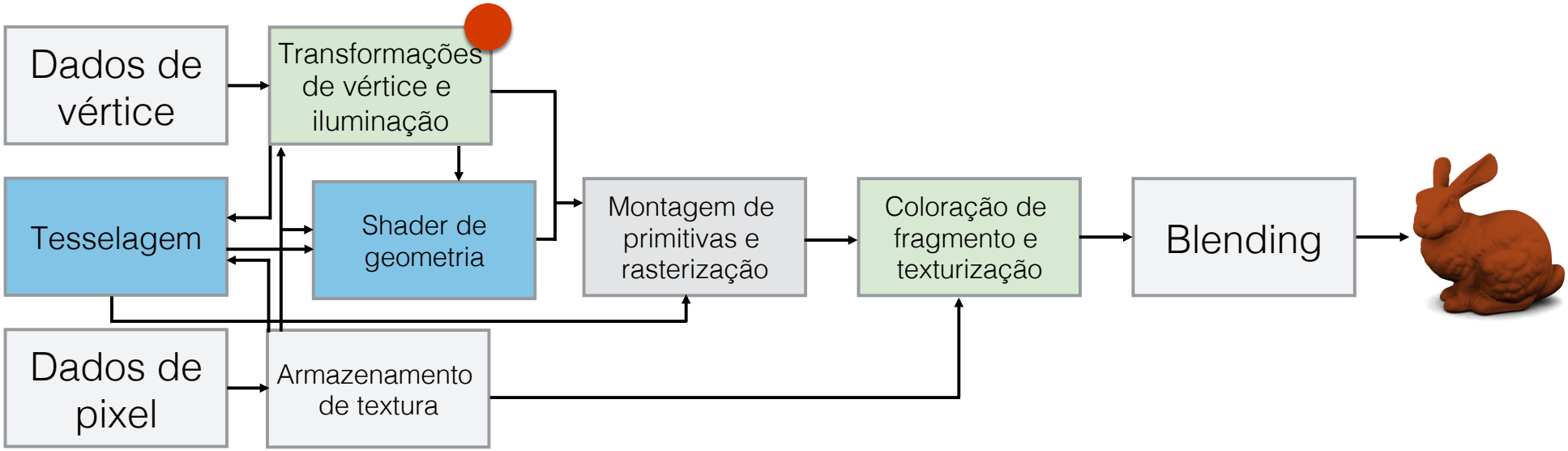
GLSL

Pipeline programável



GLSL

Pipeline programável



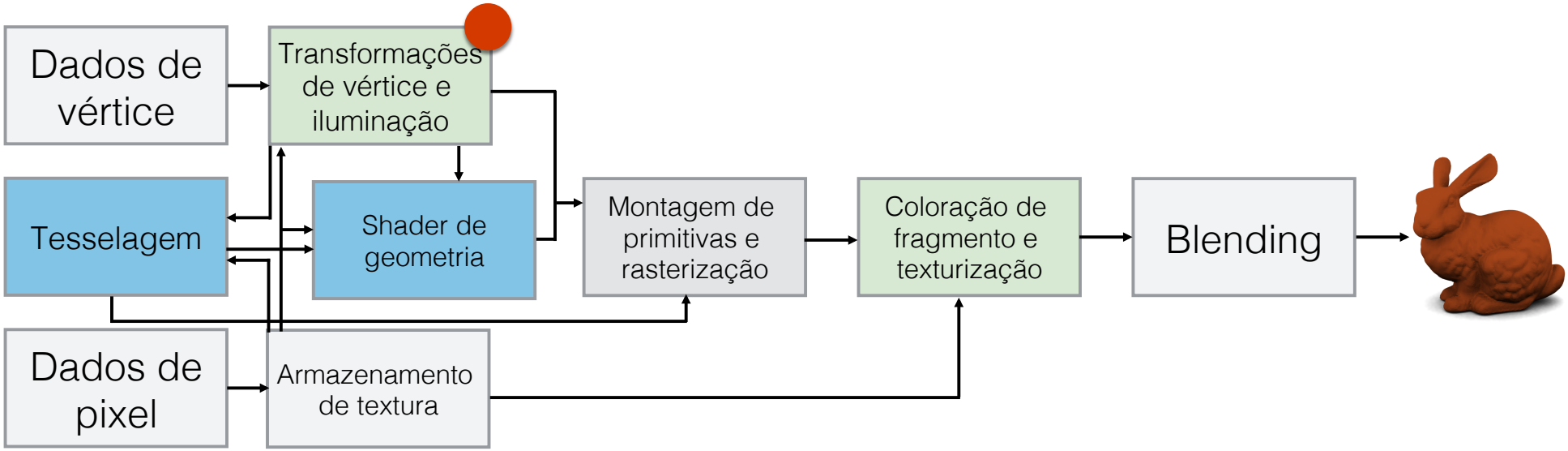
Estágios programáveis:

Vertex shader:

Código fonte GLSL carregado para a GPU que especifica as operações a serem executadas em cada vértice pelo ***processador de vértice***.

GLSL

Pipeline programável



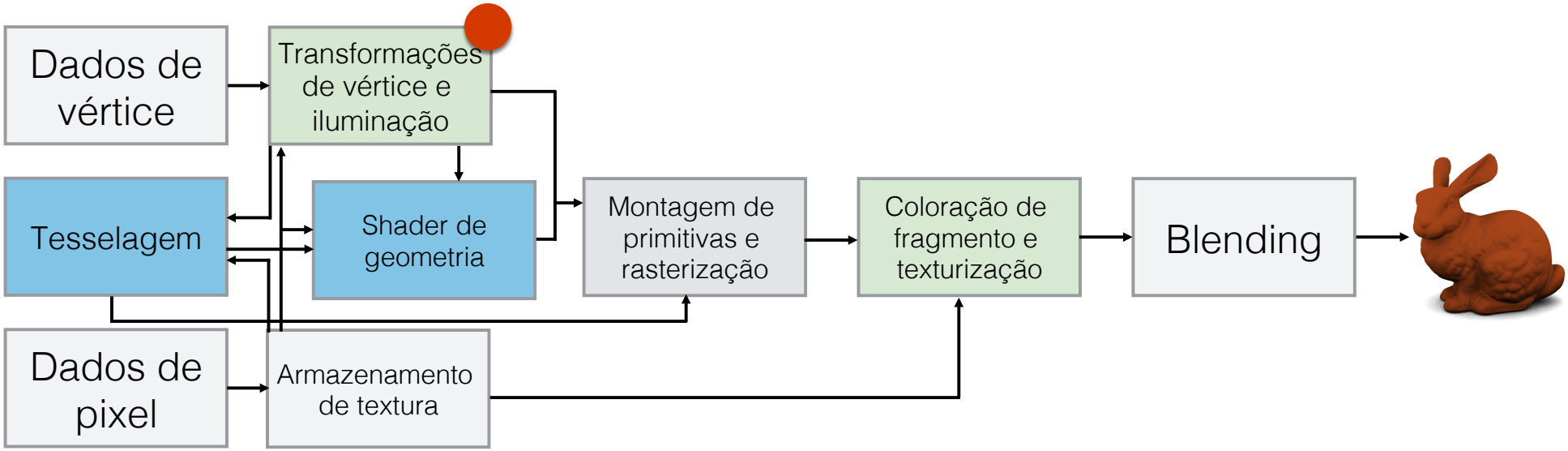
Estágios programáveis:

Vertex shader:

Os vertex shaders em execução no processador de vértice deve calcular a posição do vértice de entrada.

GLSL

Pipeline programável



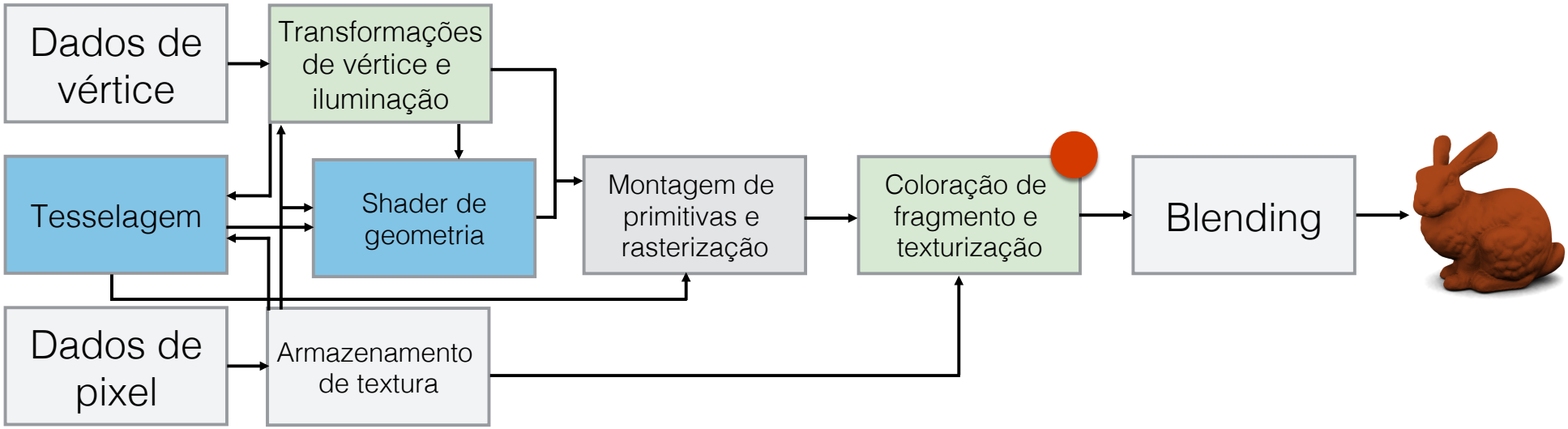
Estágios programáveis:

Processador de vértice:

Uma unidade programável da GPU que opera em cada vértice de entrada e nos dados associados (normais, coordenadas de textura, ...).

GLSL

Pipeline programável



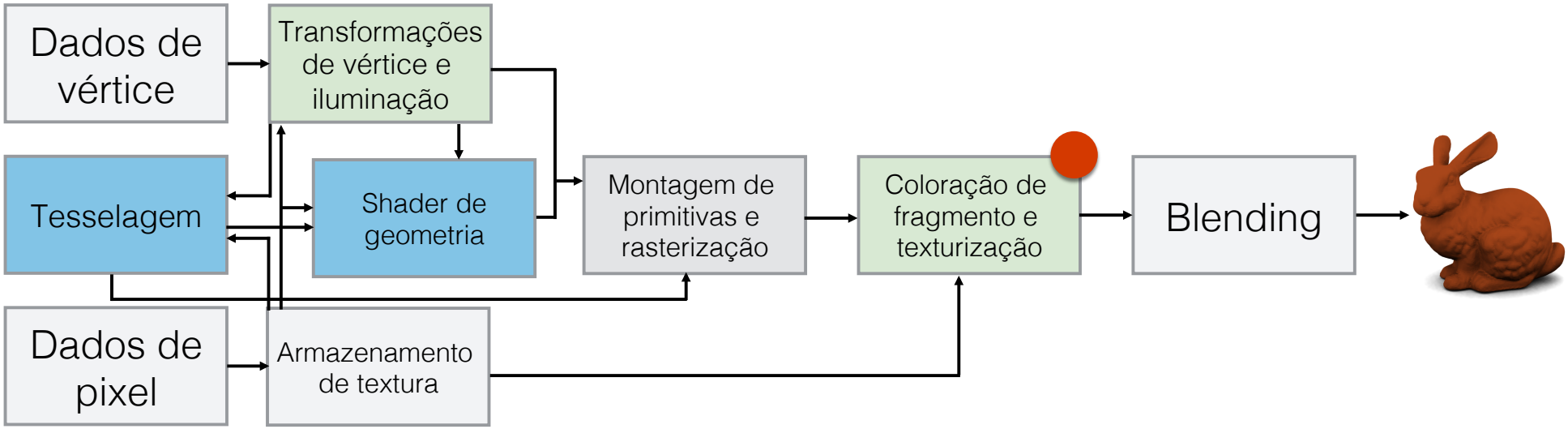
Estágios programáveis:

Fragment shader:

Código fonte GLSL carregado para a GPU que especifica as operações a serem executadas em cada fragmento pelo ***processador fragmento***.

GLSL

Pipeline programável



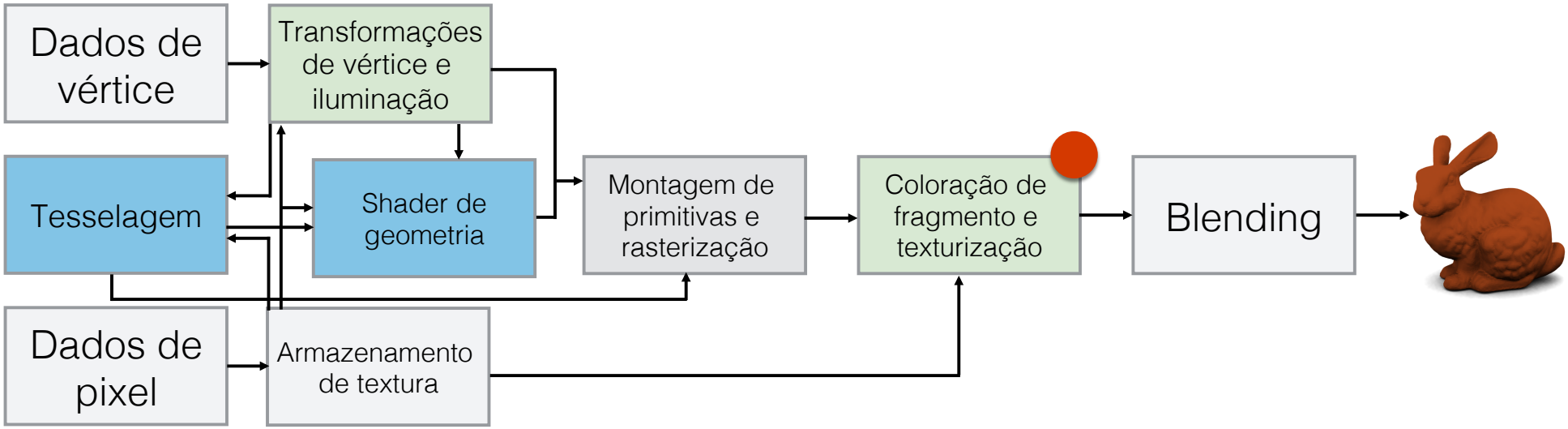
Estágios programáveis:

Fragment shader:

Os valores calculados pelo shader de fragmento são usados para atualizar o frame-buffer.

GLSL

Pipeline programável



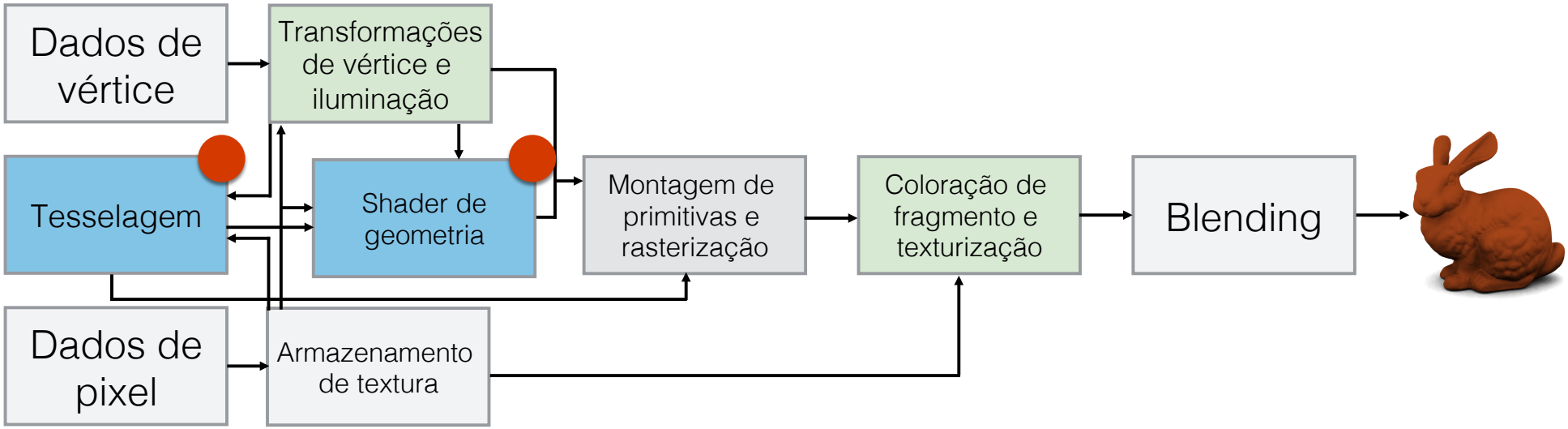
Estágios programáveis:

Fragment shader:

Unidade programável da GPU que opera em cada fragmento produzido durante a ***rasterização*** e seus dados associados (interpolados).

GLSL

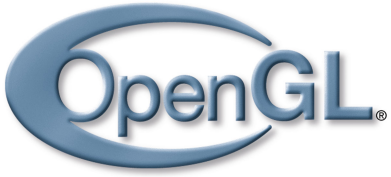
Pipeline programável



Estágios programáveis:

Tesselation e Geometry shaders:

Não estudaremos neste curso.
Na fase atual do aprendizado, podem ser “ignorados” sem prejuízo.

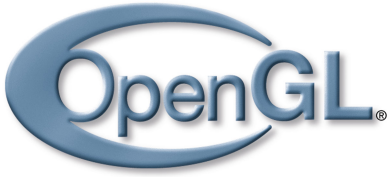


GLSL

Características

Parecido com a linguagem C

```
#version 330

layout (location = 0) in vec4 position;
layout (location = 1) in vec4 color;

smooth out vec4 theColor;

void main()
{
    gl_Position = position;
    theColor = color;
}
```

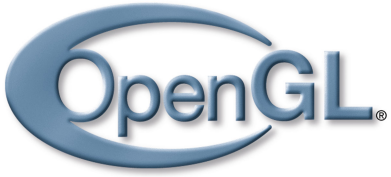


GLSL

Características

Parecido com a linguagem C

```
#version 330

layout (location = 0) in vec4 position;
layout (location = 1) in vec4 color;

smooth out vec4 theColor;

void main() ⟶ Ponto de entrada
{
    gl_Position = position;
    theColor = color;
}
```



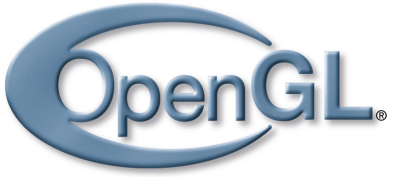
GLSL

Características

Parecido com a linguagem C

```
#version 330

layout (location = 0) in vec4 position;
layout (location = 1) in vec4 color;

smooth out vec4 theColor;

void main()
{
    gl_Position = position;
    theColor = color;
}
```

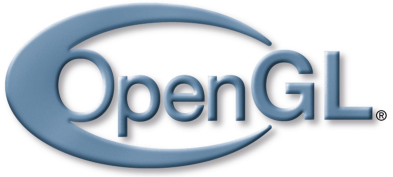
operadores, structs, funções, arrays, etc …



GLSL

Tipos de dados

- Tipos escalares:
  `float`, `int`, `bool`;
- Tipos vetoriais:
  `vec2`, `vec3`, `vec4`,
  `ivec2`, `ivec3`, `ivec4`,
  `bvec2`, `bvec3`, `bvec4`.
- Tipos Matriciais:
  `mat2`, `mat3`, `mat4`,
- Texturas:
  `sampler1D`, `sampler2D`,
  `sampler3D`, `samplerCube`

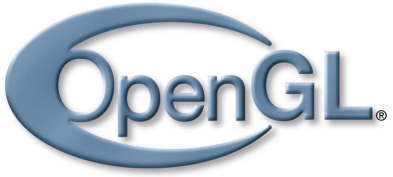


GLSL

Tipos de dados

Construtores

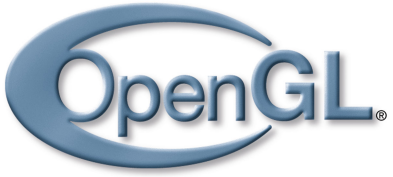
```
vec2 a = vec2(1.0, 2.0);
vec3 b = vec3(-1.0, 0.0, 0.0);
vec4 c = vec4(0.0, 0.0, 0.0, 1.0);

vec4 a = vec4(0.0); // = vec4(0.0, 0.0, 0.0, 0.0)

vec4 a = vec4(-1.0, 2.5, 4.0, 1.0);
vec3 b = vec3(a); // = vec3(-1.0, 2.5, 4.0)
vec2 c = vec2(b); // = vec2(-1.0, 2.5)

vec2 a = vec2(0.1, 0.2);
vec3 b = vec3(0.0, a); // = vec3(0.0, 0.1, 0.2)
vec4 c = vec4(b, 1.0); // = vec4(0.0, 0.1, 0.2, 1.0)
```



GLSL
Sintaxe

Construtores

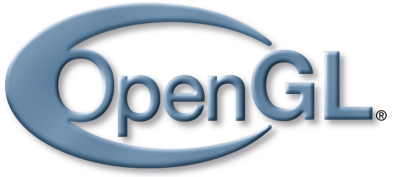
```
mat3 m = mat3(
   1.1, 2.1, 3.1, // primeira coluna
   1.2, 2.2, 3.2, // segunda coluna
   1.3, 2.3, 3.3  // terceira coluna
);

mat3 id = mat3(1.0); // 1.0 na diagonal
                     // todos os outros componentes são 0.0

vec3 column0 = vec3(0.0, 1.0, 0.0);
vec3 column1 = vec3(1.0, 0.0, 0.0);
vec3 column2 = vec3(0.0, 0.0, 1.0);
mat3 n = mat3(column0, column1, column2);
```



GLSL
Sintaxe

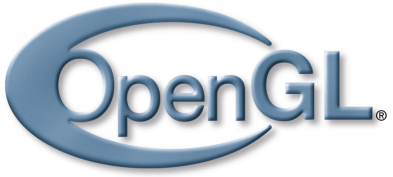
Construtores

```
mat2 m2x2 = mat2(
   1.1, 2.1,
   1.2, 2.2
);
mat3 m3x3 = mat3(m2x2); // = mat3(
   // 1.1, 2.1, 0.0,
   // 1.2, 2.2, 0.0,
   // 0.0, 0.0, 1.0)
mat2 mm2x2 = mat2(m3x3); // = m2x2
```



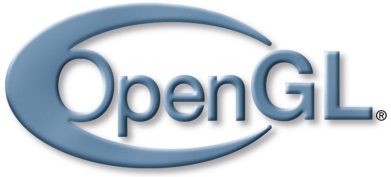
Operadores de acesso

```
vec4 v = vec4(1.1, 2.2, 3.3, 4.4);
float a = v[3]; // = 4.4
float b = v.w;  // = 4.4
float c = v.a;  // = 4.4
float d = v.q;  // = 4.4

vec4 v = vec4(1.1, 2.2, 3.3, 4.4);
vec3 a = v.xyz; // = vec3(1.1, 2.2, 3.3)
vec3 b = v.bgr; // = vec3(3.3, 2.2, 1.1)
vec2 c = v.tt;  // = vec2(2.2, 2.2)
```



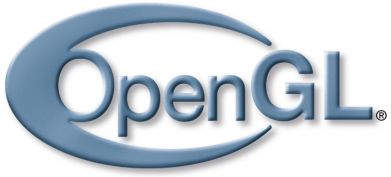
Operadores de acesso

```
mat3 m = mat3(
   1.1, 2.1, 3.1, // primeira coluna
   1.2, 2.2, 3.2, // segunda coluna
   1.3, 2.3, 3.3  // terceira coluna
);
vec3 column2 = m[2]; // = vec3(1.3, 2.3, 3.3)
vec3 m20 = m[2][0];  // = 1.3
vec3 m21 = m[2].y;   // = 2.3
```



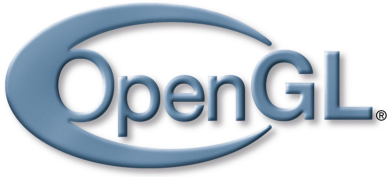
Operadores aritméticos

```
vec3 a = vec3(1.0, 2.0, 3.0);
vec3 b = vec3(0.1, 0.2, 0.3);
vec3 c = a + b; // = vec3(1.1, 2.2, 3.3)
vec3 d = a * b; // = vec3(0.1, 0.4, 0.9)

mat2 a = mat2( 1.,  2.,   3.,  4.);
mat2 b = mat2(10., 20.,  30., 40.);
mat2 c = a * b; // = mat2(
                // 1. * 10. + 3. * 20., 2. * 10. + 4. * 20.,
                // 1. * 30. + 3. * 40., 2. * 30. + 4. * 40.)
```



Operadores aritméticos

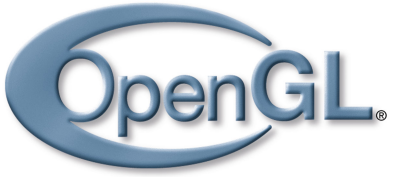
```
vec3 a = vec3(1.0, 2.0, 3.0);
mat3 m = mat3(1.0);
float s = 10.0;
vec3 b = s * a;  // vec3(10.0, 20.0, 30.0)
vec3 c = a * s;  // vec3(10.0, 20.0, 30.0)
mat3 m2 = s * m; // = mat3(10.0)
mat3 m3 = m * s; // = mat3(10.0)

vec2 v = vec2(10., 20.);
mat2 m = mat2(1., 2.,  3., 4.);
vec2 w = m * v;  // = vec2(1. * 10. + 3. * 20., 2. * 10. + 4. * 20.)

vec2 v = vec2(10., 20.);
mat2 m = mat2(1., 2.,  3., 4.);
vec2 w = v * m;  // = vec2(1. * 10. + 2. * 20., 3. * 10. + 4. * 20.)
```



GLSL
Sintaxe

Qualificadores

Definição:
Modificam o comportamento variáveis globais e locais.



GLSL
Sintaxe

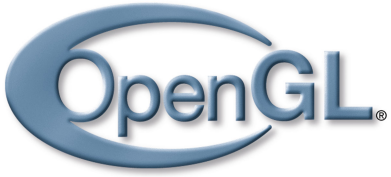
Qualificadores

Definição:
Modificam o comportamento variáveis globais e locais.

- in, out

Variáveis globais declaradas com os qualificadores são vari

```
in vec3 position;
out vec4 color;
```



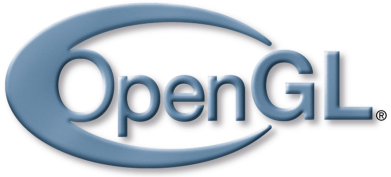
Qualificadores

Definição:
Modificam o comportamento variáveis globais e locais.

- uniform

Variáveis constantes do shader, provenientes da aplicação.

```
uniform float type;
uniform vec4 rotation;
```



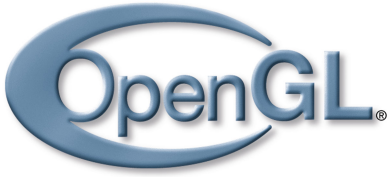
GLSL
Sintaxe

Qualificadores

Definição:
Modificam o comportamento variáveis globais e locais.

Existem muitos outros…

https://www.opengl.org/wiki/Type_Qualifier_(GLSL)

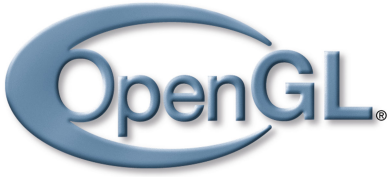


Funções

A linguagem provê uma série de operações.

- Vetores e matrizes

  `sqrt, power, abs, …`

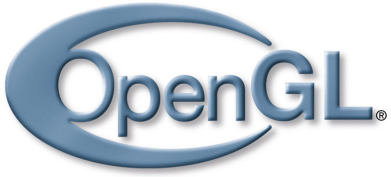


GLSL
Sintaxe

Funções

A linguagem provê uma série de operações.

- Vetores e matrizes
  `sqrt, power, abs, …`
- Trigonometria
  `sin, cos, …`

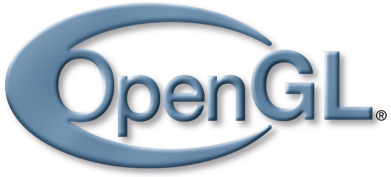


GLSL
Sintaxe

Funções

A linguagem provê uma série de operações.

- Vetores e matrizes
  `sqrt, power, abs, …`
- Trigonometria
  `sin, cos, …`
- Outros
  `length, reflect, …`



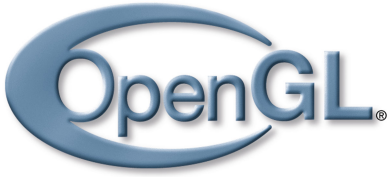
GLSL
Sintaxe

Funções

A linguagem provê uma série de operações.

- Vetores e matrizes
  `sqrt, power, abs, …`
- Trigonometria
  `sin, cos, …`
- Outros
  `length, reflect, …`

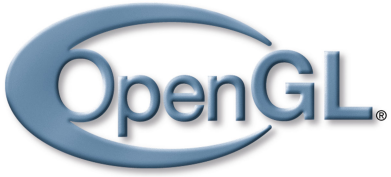
não há suporte para a leitura de arquivos e impressão de valores



GLSL
Sintaxe

Variáveis

São responsáveis por manter a consistência entre os estágios do pipeline gráfico.

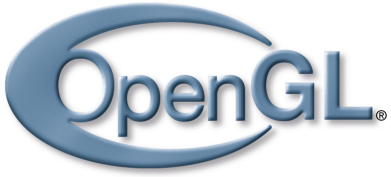


GLSL
Sintaxe

Variáveis

São responsáveis por manter a consistência entre os estágios do pipeline gráfico.

Por convenção, todas as variáveis predefinidas começam com `gl_`.



GLSL
Sintaxe

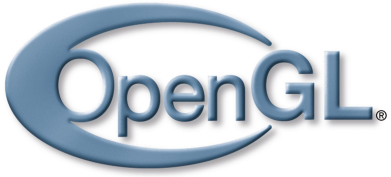
Variáveis

São responsáveis por manter a consistência entre os estágios do pipeline gráfico.

Por convenção, todas as variáveis predefinidas começam com `gl_`.

`gl_Position`

Saída do shader de vértices. Sua atribuição é obrigatória.



GLSL
Sintaxe

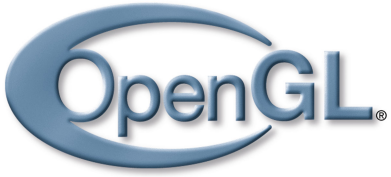
Variáveis

São responsáveis por manter a consistência entre os estágios do pipeline gráfico.

Por convenção, todas as variáveis predefinidas começam com `gl_`.

`gl_FragCoord`

Entrada do shader de fragmento que armazena a coordenada do fragmento.



GLSL
Sintaxe

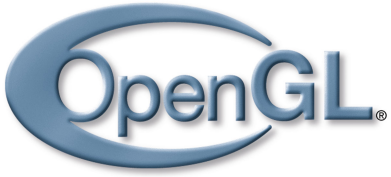
Variáveis

São responsáveis por manter a consistência entre os estágios do pipeline gráfico.

Por convenção, todas as variáveis predefinidas começam com `gl_`.

`gl_FragDepth`

Entrada do shader de fragmento que armazena a profundidade do fragmento.



GLSL
Sintaxe

Variáveis

São responsáveis por manter a consistência entre os estágios do pipeline gráfico.

Por convenção, todas as variáveis predefinidas começam com `gl_`.

Veja mais em:

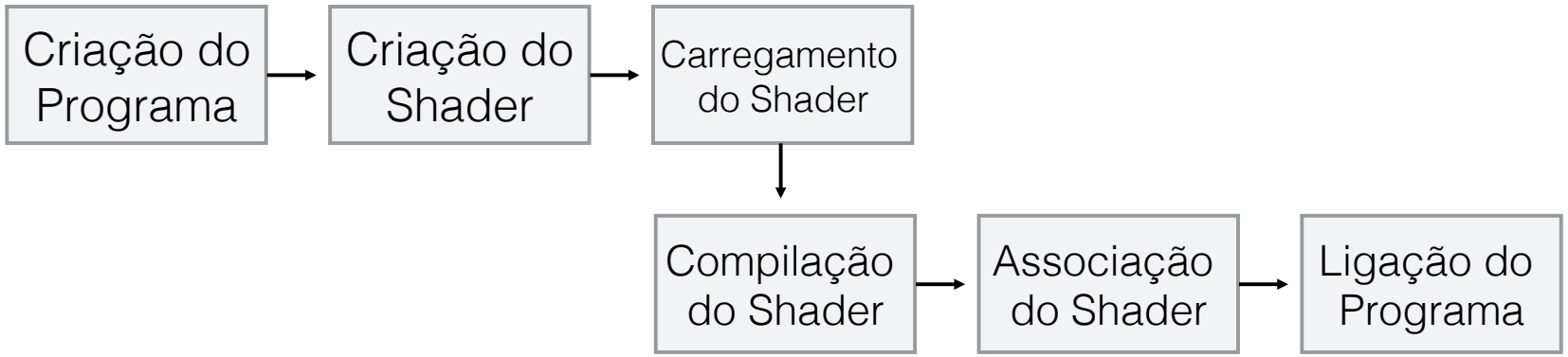
http://www.opengl.org/wiki/Built-in_Variable_(GLSL)

GLSL
Exemplo

GLSL

Carregando, compilando e ligando

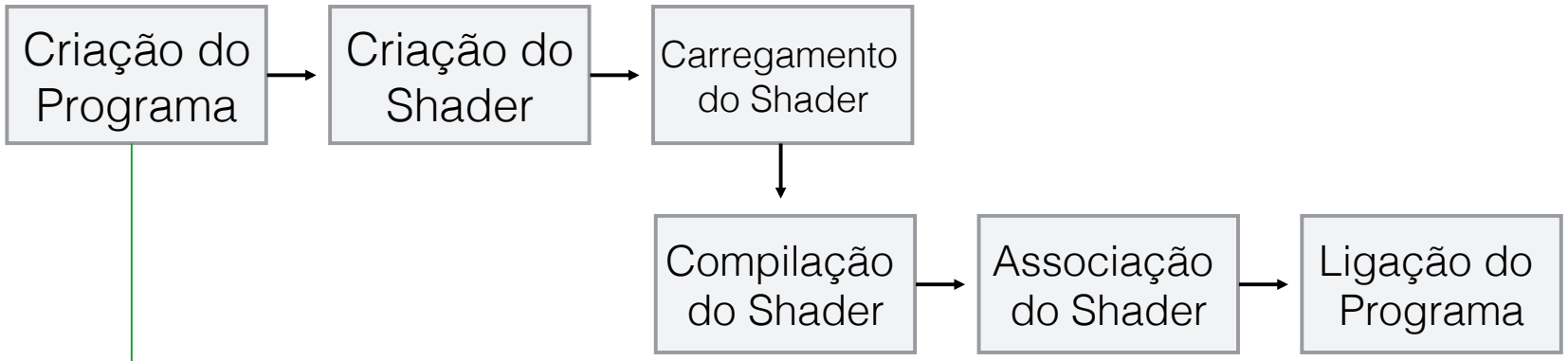
Etapas



GLSL

Carregando, compilando e ligando

Etapas

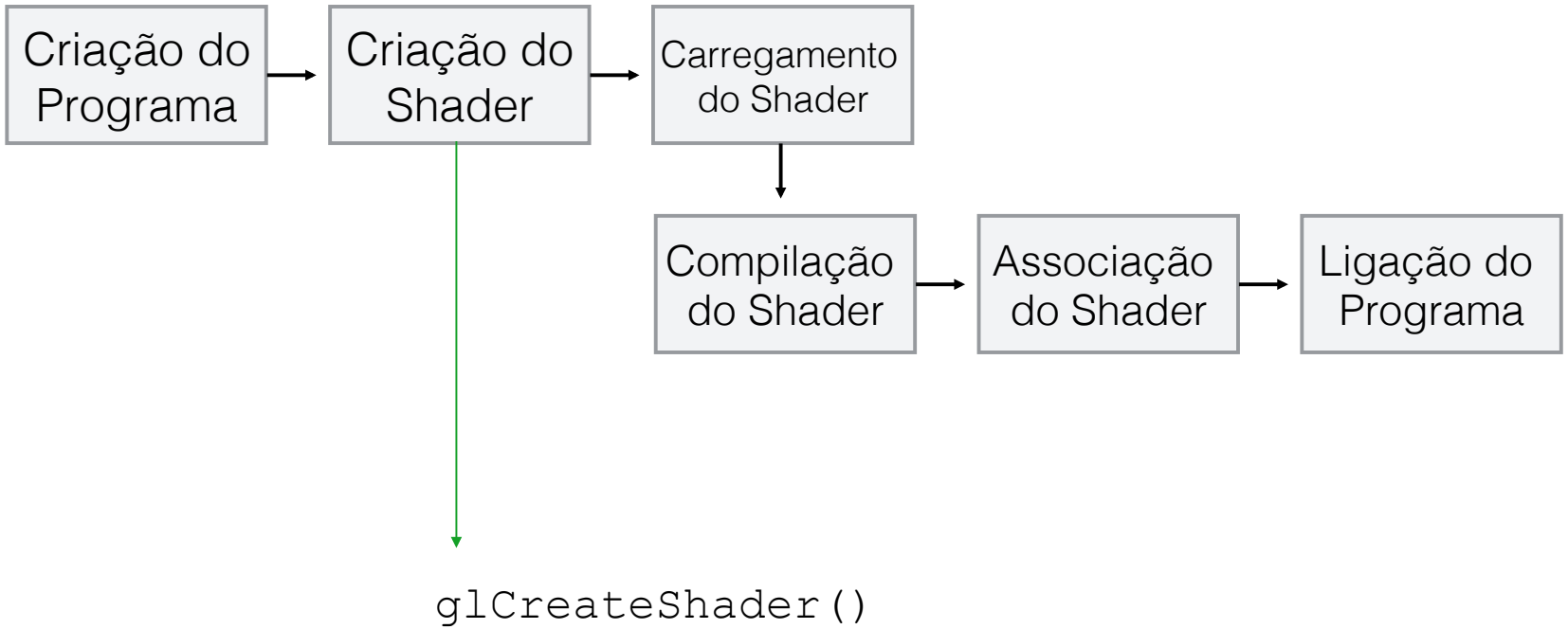



```
glCreateProgram()
```

GLSL

Carregando, compilando e ligando

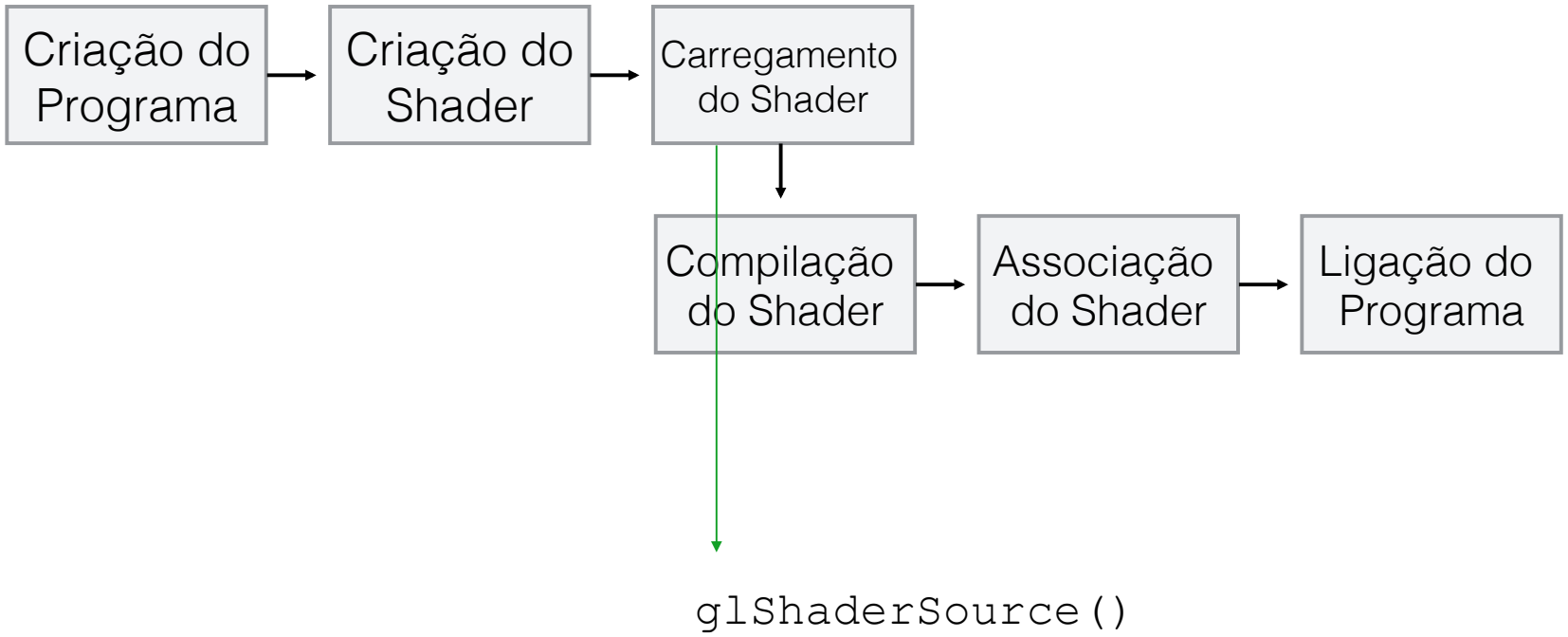
Etapas



GLSL

Carregando, compilando e ligando

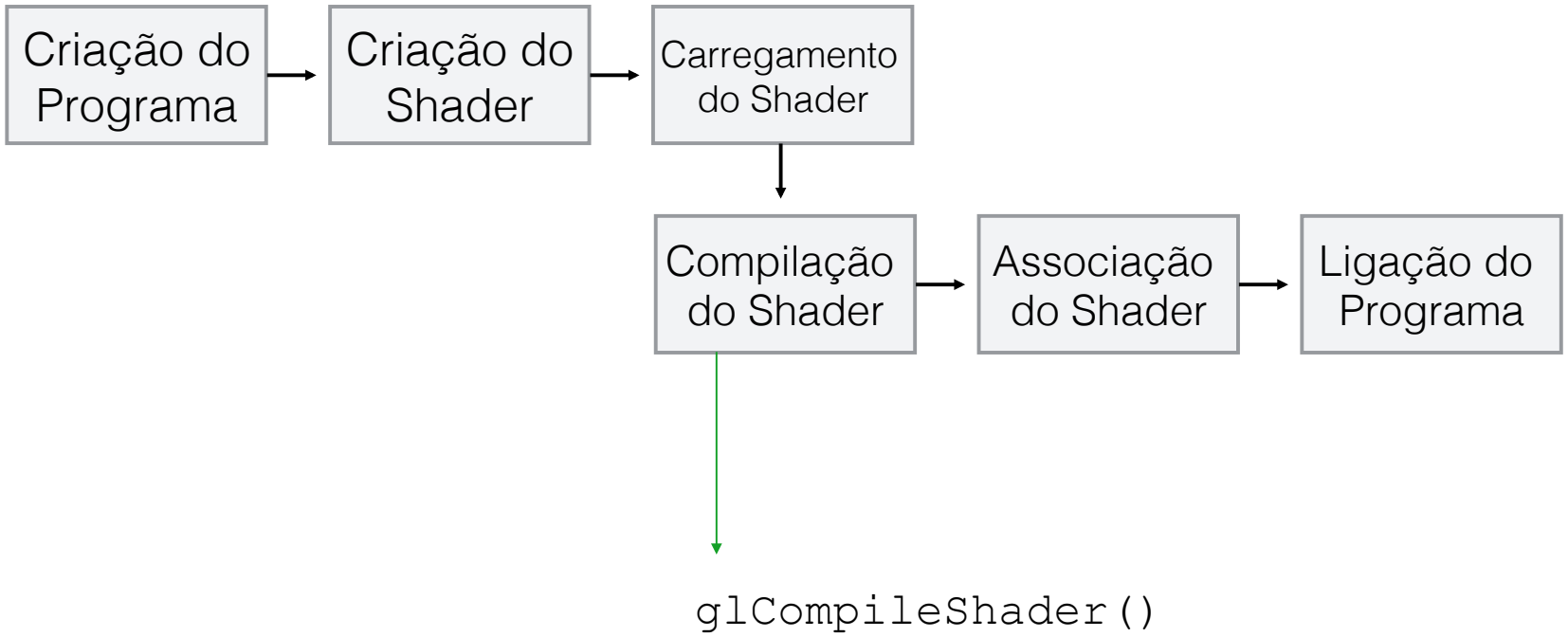
Etapas



GLSL

Carregando, compilando e ligando

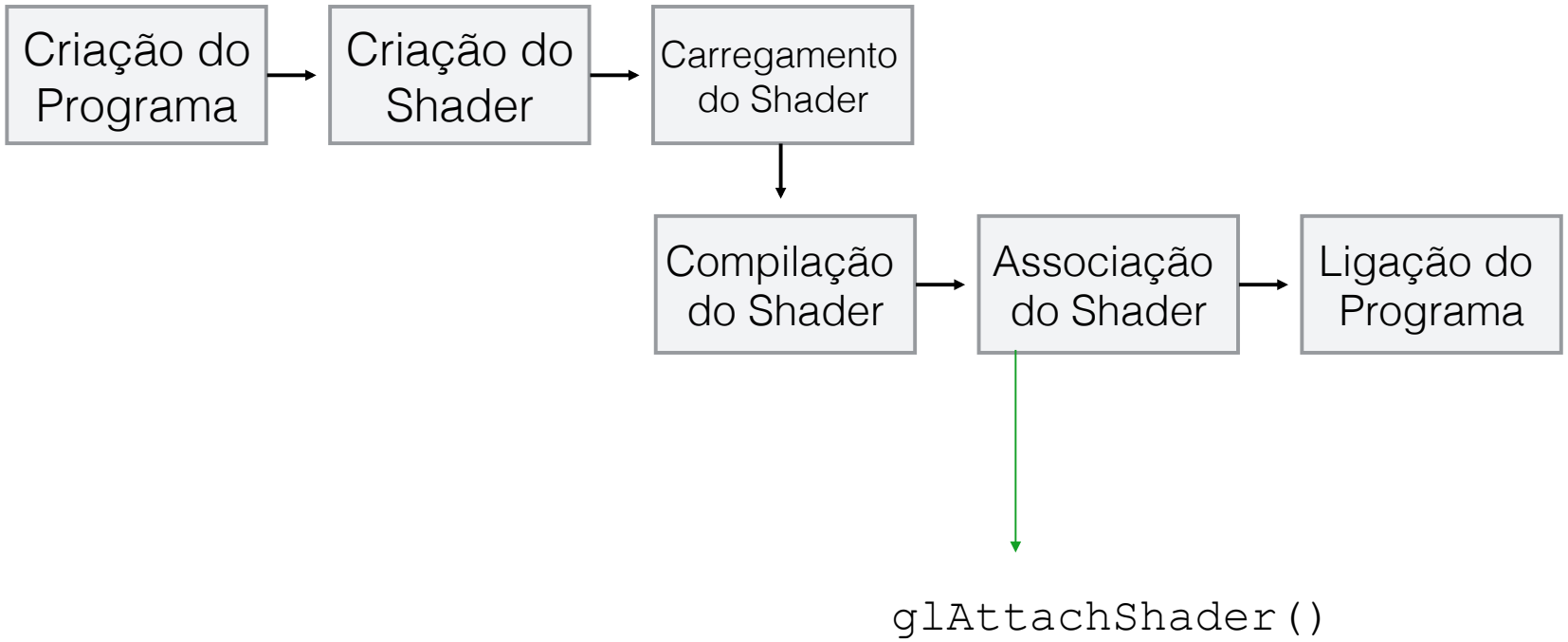
Etapas



GLSL

Carregando, compilando e ligando

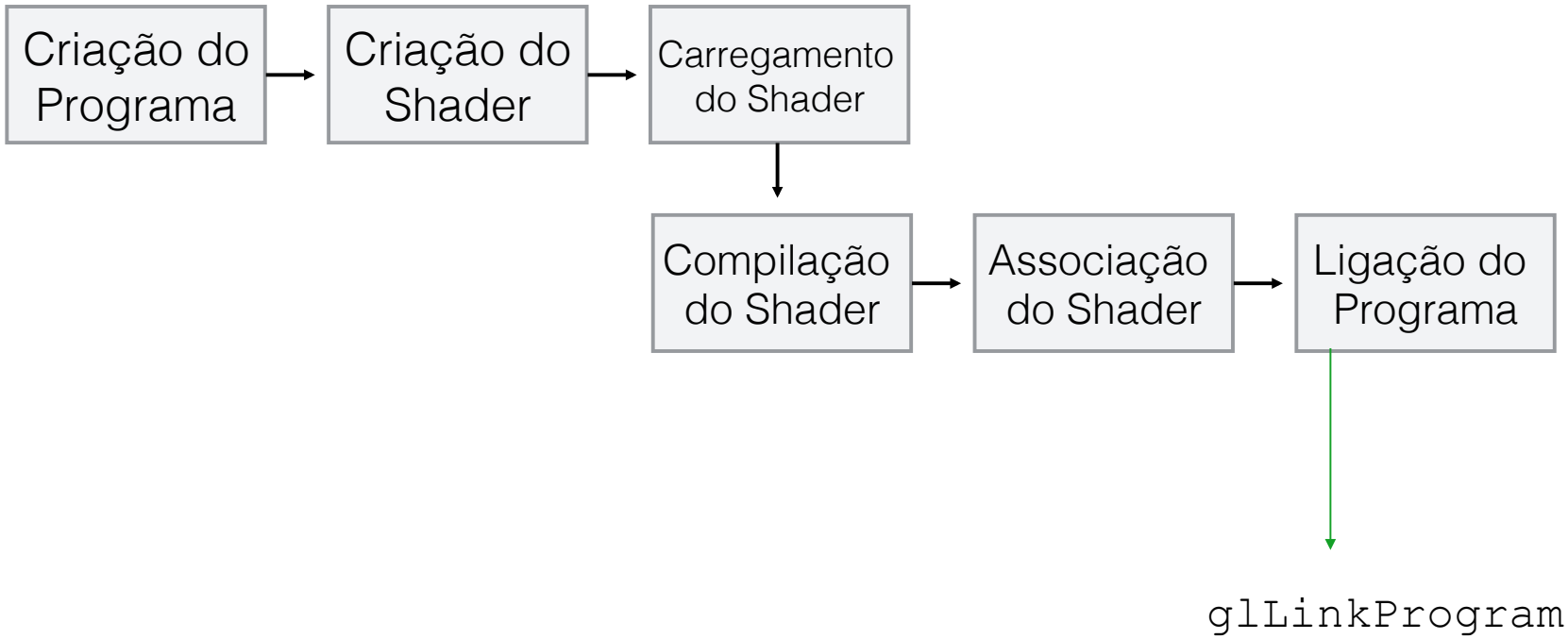
Etapas



GLSL

Carregando, compilando e ligando

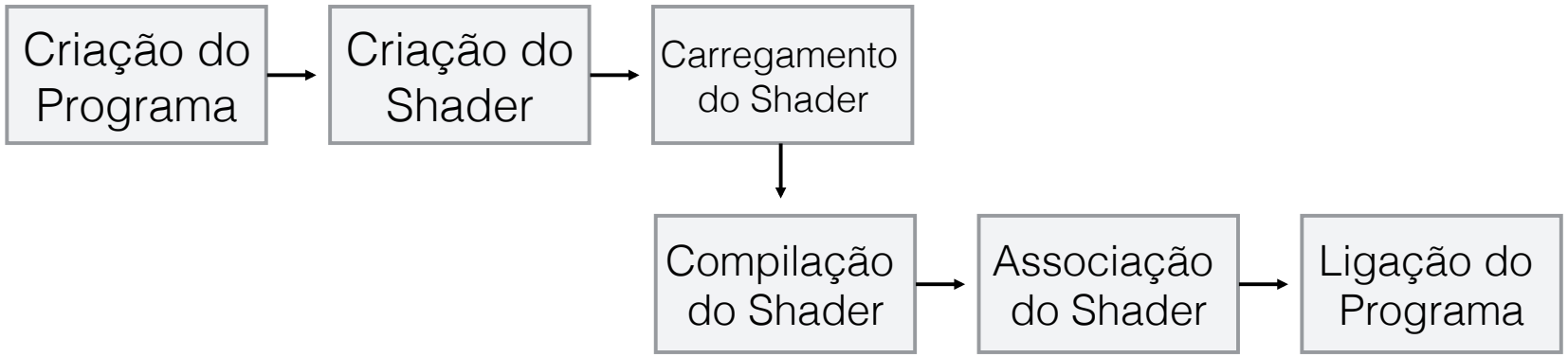
Etapas



GLSL

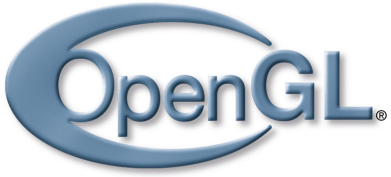
Carregando, compilando e ligando

Etapas



Por fim:

```
glUseProgram()
```



GLSL

Classe ShaderProgram

código fonte!

Computação Gráfica

TCC-00291

Assunto: GLSL