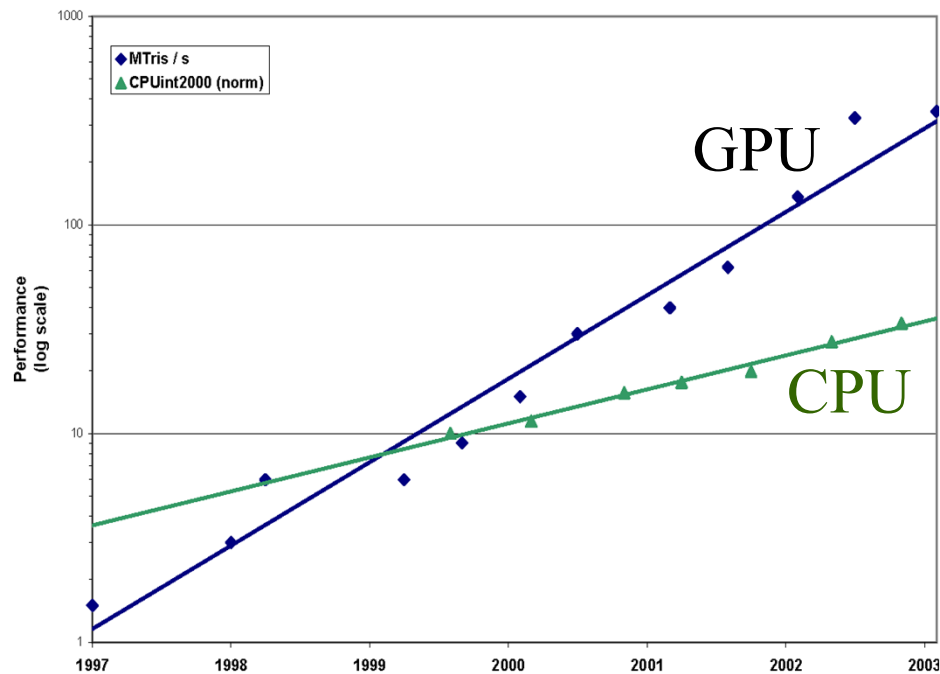
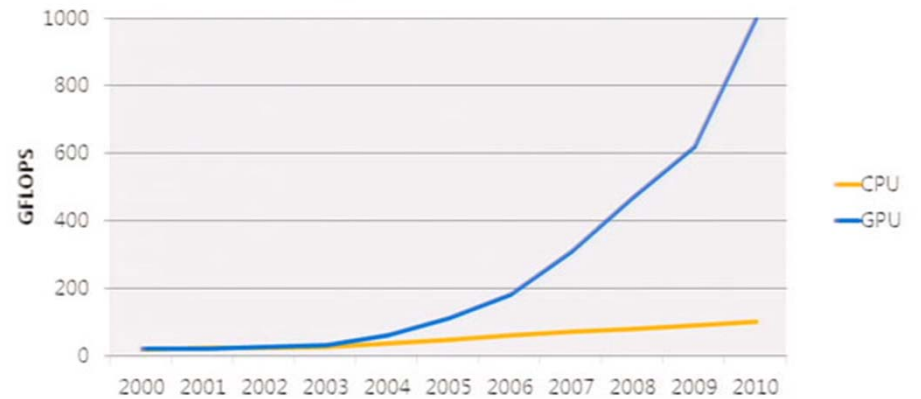
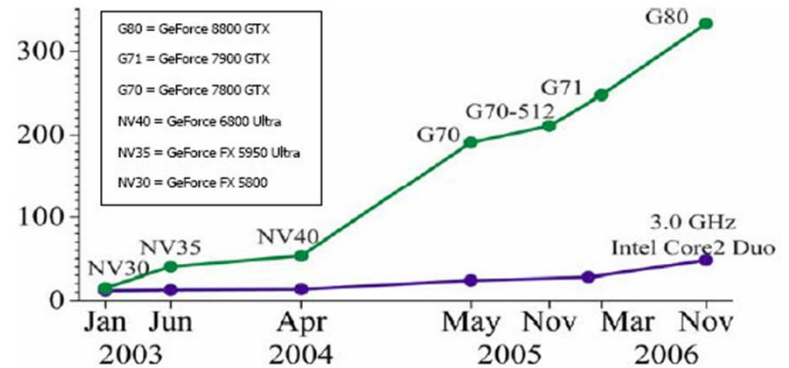


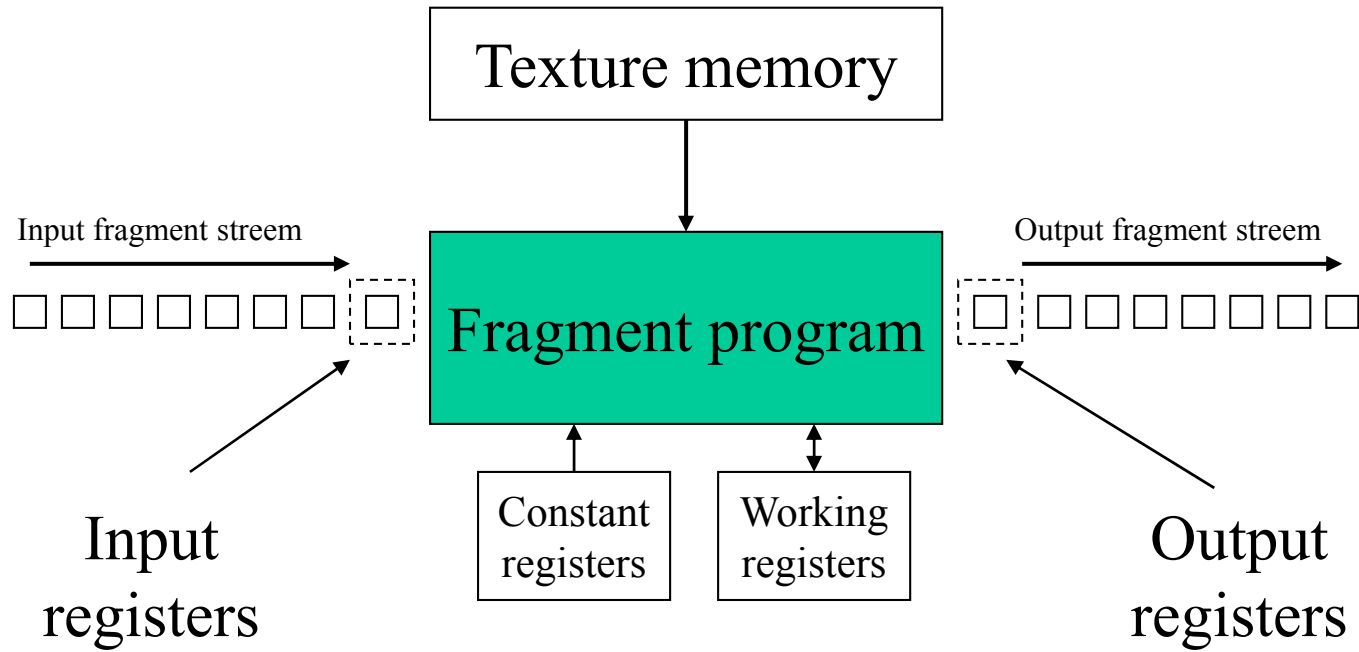
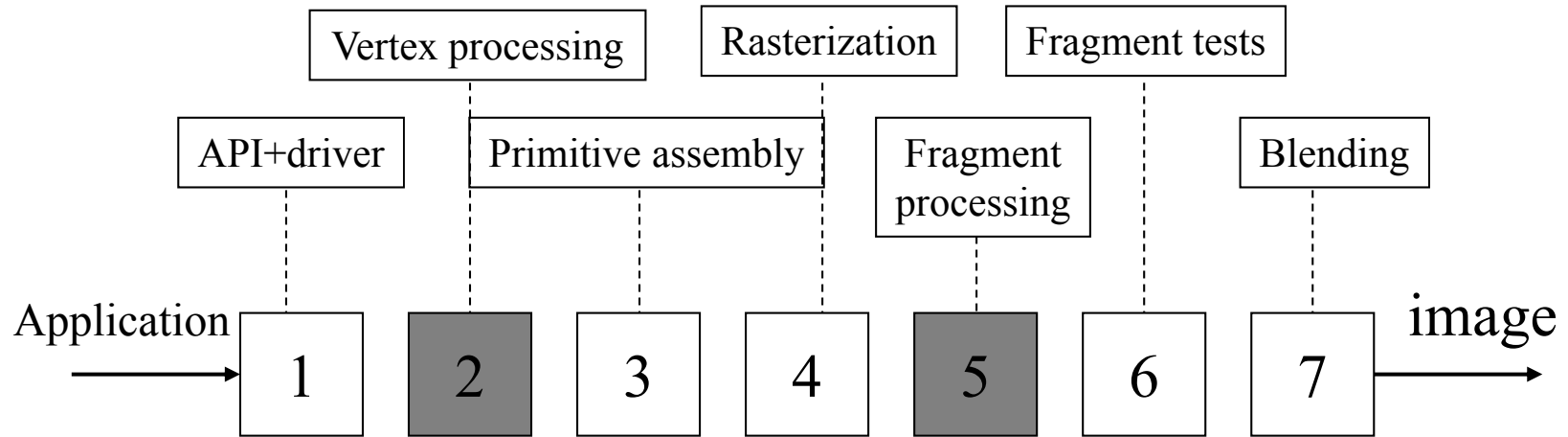
# OpenGL Shading Language

## - Vývoj CPU a GPU:



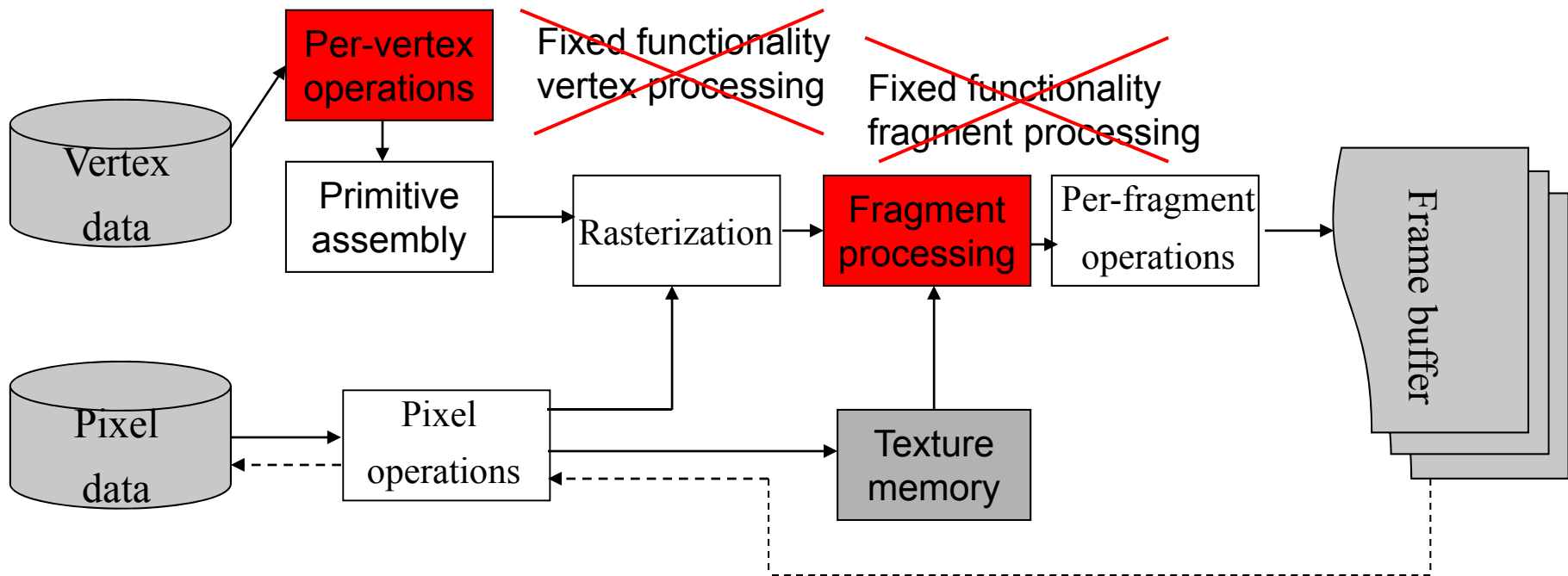
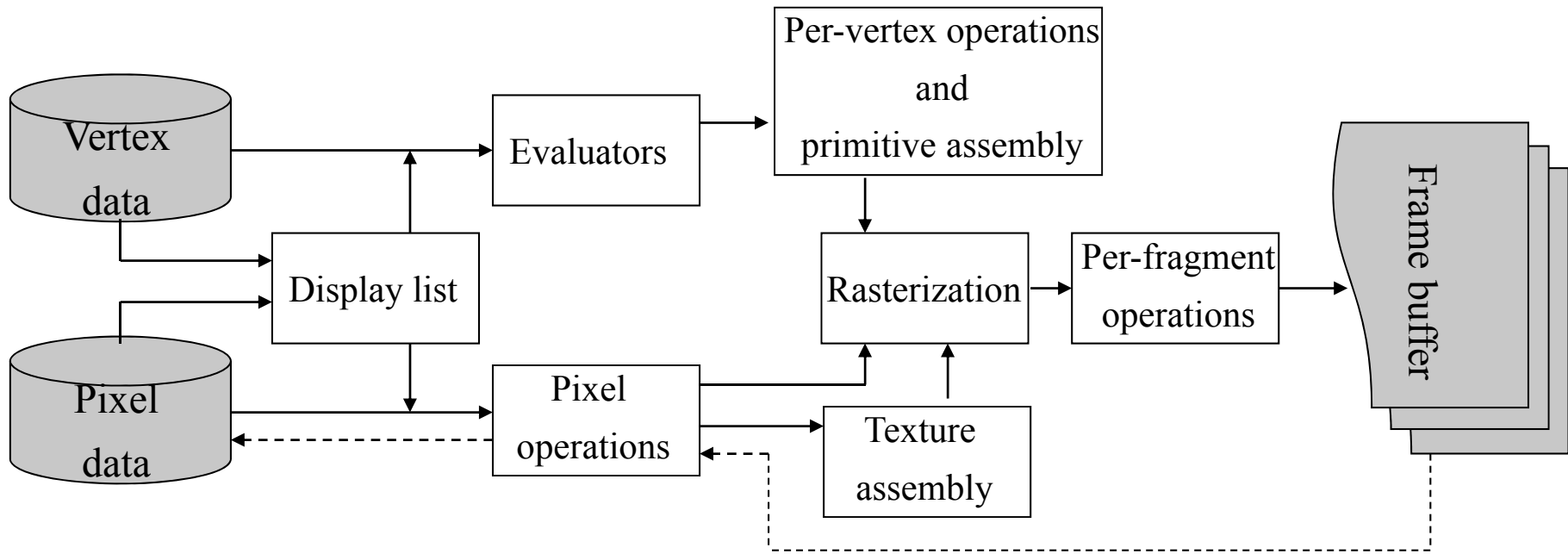
GFLOPS





# Instrukční sada

- Obecné instrukce : MOV, ...
- Aritmetické instrukce : ADD, SUB, ...
- Matematické instrukce : SIN, ...
- Řídící instrukce: KIL, ...
- Spec. grafické instr. : DST, ...
- Texturovací instrukce: TEX, ...



## Vertex Processor

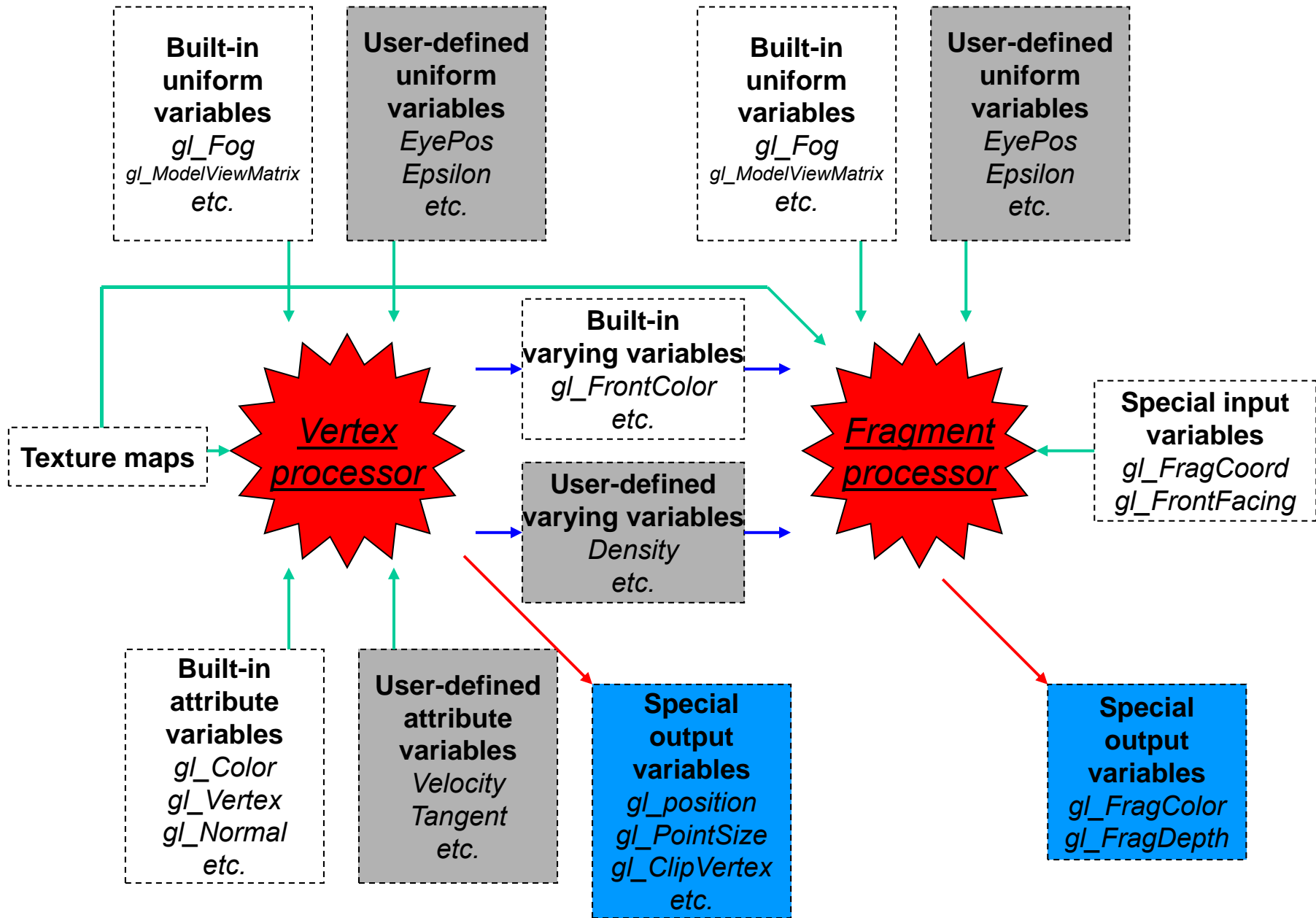
- Transformace vrcholů
- Transformace normál a normalizace
- Generování texturových souřadnic
- Transformace texturových souřadnic
- Přístup k texturám
- Osvětlení
- Nastavení barev materiálu
- *Vertex shaders* mohou být použity ke specifikaci zcela obecných sekvencí operací, které se aplikují na každý vrchol a s ním asociovaná data.
- *Vertex shaders* které provádí některé z výpočtů uvedených v předchozím seznamu, musí provést všechny požadované funkce z tohoto seznamu..

## Fragment Processor

- Operace na interpolovaných hodnotách
  - Přístup k texturám
  - Aplikace textur
  - Fog
  - Součty barev
- 
- *Fragment shaders* jsou použity pro algoritmy prováděné na fragmentových procesorech, které produkují výstupní hodnoty založené na poskytovaných vstupních hodnotách.
  - *Fragment shader* nemůže měnit  $x/y$  pozici fragmentu.
  - *Fragment shaders*, které provádí některé z výpočtů uvedených v předchozím seznamu, musí provést všechny požadované funkce z tohoto seznamu.

## Fragment Processor

- Primárním vstupem fragmentového shaderu jsou **interpolované** *varying* proměnné, které jsou výsledkem rasterizace.
- Fragmentový procesor nenahradí grafické operace, které požadují informace o několika fragmentech v jedné chvíli.
- Z důvodu podpory paralelizmu na úrovni zpracování fragmentu jsou fragmentové shadery psány způsobem který vyjadřuje požadovaný výpočet pro jeden fragment a přístup k sousedním fragmentům není povolen.





## První příklad

```
// Vertex shader
uniform float CoolestTemp;
uniform float TempRange;
attribute float VertexTemp;
varying float Temperature;

void main()
{
    Temperature= (VertexTemp - CoolestTemp)/TempRange;
    gl_Position= gl_ModelViewProjectionMatrix * gl_Vertex;
}

-----

// Fragment shader
uniform vec3 CoolestColor;
uniform vec3 HottestColor;
varying float Temperature;

void main()
{
    vec3 color= mix(CoolestColor, HottestColor, Temperature);
    gl_FragColor= vec4(color, 1.0);
}
```

## Definice jazyka

- OpenGL Shading language je založen na systému programovacího jazyka ANSI C.

## Doplňky k C

- vektorové typy jsou podporovány pro hodnoty float, int a bool

- `vec{2,3,4}`, `ivec{2,3,4}`, `bvec{2,3,4}`,

```
vec4 v=vec4(1.0, 0.5, 0.0, 1.0);  
vec4 u=vec4(5.0, 3.0, 2.0, 4.0);  
vec4 w;  
vec2 w2;  
float a,b;
```

```
w= v+u;  
a= v.x; b= v.z;  
w2= u.zw; (swizzling)
```

```
// x,y,z,w - Treats a vector as a position or direction  
// r,g,b,a - Treats a vector as a color  
// s,t,p,q - Treats a texture coordinate
```

## Definice jazyka

### - Matice

mat2, mat3, mat4 -matice 2x2, 3x3, 4x4

K matici můžete přistupovat jako k vektoru po sloupcích.

```
vec4 u;  
mat4 m;
```

```
m= ...  
u= m[3];
```

### - Samplers

sampler{1,2,3}D

samplerCube

textura mapovaná na kostku.

sampler{1,2}Dshadow

hloubková textura s porovnáním.

## Definice jazyka

Shadery nemohou samy inicializovat samplery. Mohou je pouze obdržet od aplikace přes kvalifikátor, nebo je předat k dalšímu zpracování vestavěné funkci.

Sampler nemůže být modifikován.

```
Uniform sampler2D Grass;
```

```
// texture2D use the texture coordinate vec2 coord to do a texture  
// lookup in the 2D texture currently specified by sampler  
vec4 color= texture2D(Grass, coord);
```

### - Kvalifikátory

attribute	Pro často měněné informace (jednou za vrchol)
uniform	Pro zřídka měněné informace (polygony)
varying	Pro interpolované informace
const	Deklaruje neměnné, časově konstantní proměnné

## Definice jazyka

### - Inicializátory a konstruktory

- Proměnná shaderu může být inicializována v okamžiku deklarace.
- Attribute, uniform a varying nemohou být inicializovány v okamžiku deklarace.
- Existují konstruktory pro všechny vestavěné typy (s výjimkou samplerů) a pro struktury.

```
vec4 v=vec4(1.0, 0.5, 0.0, 1.0);  
---
```

```
vec4 v;  
v= vec4(1.0, 0.5, 0.0, 1.0);  
mat2 m=(1.0, 2.0, 3.0, 4.0);  
---
```

## Definice jazyka

```
vec3 color= vec3(0.2, 0.5, 0.6);
struct light
{
    vec4 position;
    struct lightColor
    {
        vec3 color;
        float intensity;
    }
}light1 = light(v, lightColor(color, 0.9));
```

## - Typová konverze

Explicitní typová konverze se taktéž provádí pomocí konstruktorů.

OpenGL Shading language neposkytuje syntaxi jazyka pro převod typů. Namísto toho jsou pro konverzi použity konstruktory.

```
float f= 2.3;
bool b= bool(f);
float f= float(3); // convert integer 3 to 3.0
float g= float(b); // convert Boolean b to floating point
vec4 v= vec4(2)    // all components of v are set to 2.0
```

## Definice jazyka

### Flow Control

**for, while, do-while, break, continue**

**if, if-else**

**:?**

**goto, switch** – není povoleno

**discard** – zabrání fragmentu provést update frame-bufferu. V okamžiku průchodu přes toto klíčové slovo se zpracováváný fragment označí jakožto skartovaný.

### Swizzling

```
vec4 v4;
```

```
v4.rgba;
```

```
v4.rgb;
```

```
v4.b;
```

```
v4.xy;
```

```
v4.xgba; //!! Illegal - the component names do not come from the same set.
```

## Definice jazyka

### Operace po komponentách

- S několika málo výjimkami, kdy se operátor aplikuje na vektor, se operátor chová, jako by byl aplikován na každou komponentu vektoru zvlášť.

```
vec3 u,v, w;  
float f;
```

```
v= u+f; ~ v.x= u.x+f; v.y= u.y+f; v.z= u.z+f;
```

```
w= u+v; ~ w.x= u.x+v.x; w.y= u.y+v.y; w.z= u.z+v.z;
```

- Logické operátory `!`, `&&`, `||`, `^^` fungují pouze na výrazech, které jsou označeny jakožto scalar Booleans, a jejich výsledkem je scalar Boolean

- Relační operátory (`<`, `>`, `<=`, `>=`) fungují pouze na floating-point a integer skalárech a výsledkem je scalar Boolean.

- Operátory rovnosti (`==`, `!=`) fungují na všech typech s výjimkou polí. Výsledkem je scalar Boolean.



## Vestavěné funkce – Trigonometrické funkce

```
float radians(float degrees)  
vec{1,2,3} radians(vec{1,2,3} degrees)
```

Konvertuje *stupně* na radiány a vrátí výsledek

```
float degrees(float radians)  
vec{1,2,3} degrees(vec{1,2,3} radians)
```

Konvertuje *radiány* na stupně a vrátí výsledek

```
float sin(float radians)  
vec{1,2,3} sin(vec{1,2,3} radians)
```

Standardní trigonometrická funkce sinus

**cos, tan, asin, acos, atan**

## Vestavěné funkce

```
float pow(float x, float y)  
vec{1,2,3} pow(vec{1,2,3} x, vec{1,2,3} y)
```

Vrátí  $x$  umocněno na  $y$

```
float exp2(float x)  
vec{1,2,3} exp2(vec{1,2,3} x)
```

vrátí 2 umocněno na  $x$

**log2, sqrt, inversesqrt**

## Vestavěné funkce

```
float abs(float x, float y)  
vec{1,2,3,4} abs(vec{1,2,3,4})
```

Vrátí  $x$  jestliže  $x \geq 0$ ; jinak vrátí  $-x$

**sign** ...

**floor** Vrátí hodnotu rovnou nejbližšímu celému číslu které je menší než  
nebo rovno  $x$

**ceil** Vrátí hodnotu rovnou nejbližšímu celému číslu které je větší než  
nebo rovno  $x$

**fract** Vrátí  $x - \text{floor}(x)$

## Vestavěné funkce

```
float mod(float x, float y)  
vec{2,3,4} mod(vec{2,3,4} x, vec{2,3,4} y)
```

Vrátí  $x - y * \text{floor}(x/y)$  pro každou komponentu z  $x$  za použití hodnoty (odpovídající komponenty)  $y$ .

### **min, max...**

```
float clamp(float x, float minVal, float maxVal)  
vec{2,3,4} clamp(vec{2,3,4} x, float minVal, float maxVal)
```

Vrátí  $\text{min}(\text{max}(x, \text{minVal}), \text{maxVal})$

```
float mix(float x, float y, float a)  
vec{2,3,4} mix(vec{2,3,4} x, vec{2,3,4} y, float a)
```

Vrátí  $x * (1.0 - a) + y * a$

## Vestavěné funkce

```
float step(float edge, float x)  
vec{2,3,4} step(vec{2,3,4} edge, vec{2,3,4} x)
```

Vrátí 0 jestliže  $x \leq edge$ ; jinak vrátí 1.0.

```
float smoothstep(float edge0, float edge1, float x)  
vec{2,3,4} smoothstep(vec{2,3,4} edge0, vec{2,3,4} edge1,  
float x)
```

Vrátí 0 jestliže  $x \leq edge0$  a 1.0 jestliže  $x \geq edge1$  a provede hladkou Hermitovskou interpolaci mezi 0 a 1.0 kde  $edge0 < x < edge1$ .

## Vestavěné funkce – Geometrické funkce

float **length**(float  $x$ )  
vec{2,3,4} **length**(vec{2,3,4}  $x$ )

Vrátí délku vektoru  $x$ .

float **distance**(float  $p0$ , float  $p1$ ), ...

Vrátí vzdálenost mezi  $p0$  a  $p1$ .

float **dot**(float  $p0$ , float  $p1$ ), ...

Vrátí skalární součin  $p0$  a  $p1$ .

float **cross**(float  $p0$ , float  $p1$ ), ...

Vrátí vektorový součin  $p0$  a  $p1$ .

float **normalize**(float  $x$ ), ...

Vrátí normalizovaný vektor  $x$ .

## Vestavěné funkce

### **vec4 ftransform()**

Pouze pro vertexové shadery. Tato funkce zajistí, že pozice vstupního vrcholu bude transformována způsobem, který poskytne přesně stejné výsledky jako vestavěné transformace OpenGL's. Tato funkce je určena k výpočtu hodnoty pro `gl_Position`.

```
gl_Position= ftransform();
```

### **float faceforward(float *N*, float *I*, float *Nref*)...**

Jestliže **dot(*Nref*, *I*) < 0.0**, vrátí *N*. jinak vrátí **-*N***.

### **float reflect(float *I*, float *N*)...**

Pro kolizní vektor *I* a orientaci povrchu *N* vrátí směr odrazu:

**$I - 2.0 * \text{dot}(N, I) * N$** . *N* by mělo být normalizováno.

## Vestavěné funkce – Funkce s maticemi

- Operátor “\*” provádí lineární algebraické násobení matic.

```
mat2 matrixcompmult(mat{2,3,4} x, mat{2,3,4} y)
```

Násobí matici  $x$  maticí  $y$  po komponentách.



## Vestavěné funkce – Funkce na vektorech

`bvec{2,3,4} lessThen([i]vec{2,3,4} x, [i]vec{2,3,4} x)`

Vrátí srovnání po komponentách  $x < y$ .

`bvec{2,3,4} lessThenEqual([i]vec{2,3,4} x, [i]vec{2,3,4} x)`

Vrátí srovnání po komponentách  $x \leq y$ .

`bvec{2,3,4} greatherThen([i]vec{2,3,4} x, [i]vec{2,3,4} x)`

Vrátí srovnání po komponentách  $x > y$ .

`bvec{2,3,4} greatherThenEqual([i]vec{2,3,4} x, [i]vec{2,3,4} x)`

Vrátí srovnání po komponentách  $x \geq y$ .

`bvec{2,3,4} Equal([i,b]vec{2,3,4} x, [i,b]vec{2,3,4} x)`

Vrátí srovnání po komponentách  $x == y$ .

## Vestavěné funkce

`bvec{2,3,4} notEqual([i,b]vec{2,3,4} x, [i,b]vec{2,3,4} x)`

Vrátí srovnání po komponentách  $x \neq y$ .

`bool any(bvec{2,3,4} x)`

Vrátí *true* jestliže nějaká komponenta  $x$  je *true*.

`bool all(bvec{2,3,4} x)`

Vrátí *true* pouze pokud každá komponenta  $x$  je *true*.

`bool not(bvec{2,3,4} x)`

Vrátí logický komplement vektoru  $x$  po složkách.

## Vestavěné funkce – Funkce pro přístup k texturám

- Vestavěné funkce se suffixem “Lod” jsou povoleny pouze ve vertex shaderu. Pro funkce “Lod”, *lod* je přímo použito jako level of detail.
- Vestavěné funkce se suffixem “Proj” mohou být použity pro projektivní texturování. Umožňují textuře projektovat se na objekt obdobně jako jsou promítány slajdy. Může být použito k výpočtu stínových map.

```
vec4 texture1D(sampler1D sampler, float coord [,float bias])  
vec4 texture1DProj(sampler1D sampler, vec2 coord [,float bias])  
vec4 texture1DProj(sampler1D sampler, vec4 coord [,float bias])  
vec4 texture1DLod(sampler1D sampler, float coord [,float bias])  
vec4 texture1DProjLod(sampler1D sampler, vec2 coord [,float bias])  
vec4 texture1DProjLod(sampler1D sampler, vec4 coord [,float bias])
```

- Texturové souřadnice *coord* se používají k vyzvednutí informací z 1D textury aktuálně specifikované *samplerem*. Pro projektivní (Proj) verze, texturová souřadnice *coord.s* je dělena poslední komponentou *coord*. Druhá a třetí komponenta *coord* je pro variantu *vec4 coord* ignorována.

## Vestavěné funkce

- Obdobně **texture2D**
- Další položky: **texture3D**, **textureCube**, **shadow1D**, **shadow2D**.

## Funkce zpracovávající fragmenty

- Pouze na fragment processoru
- Funkce derivací, **dFdx** a **dFdy** jsou použity k určení rychlosti změn výrazu.

```
float dFdx(float rho), ...
```

Vrátí derivaci v *x* pro vstupní argument *rho*.

```
float dFdy(float rho), ...
```

Vrátí derivaci v *y* pro vstupní argument *rho*.

## Šumové funkce ...

## Důležité vestavěné proměnné

### Vertex Attributes

```
attribute vec4 gl_Color;  
attribute vec4 gl_SecondaryColor;  
attribute vec4 gl_Normal;  
attribute vec4 gl_MultiTexCoord0;  
attribute vec4 gl_MultiTexCoord1;  
attribute vec4 gl_MultiTexCoord2;  
//... up to gl_MultiTexCoordN-1, where N = gl_MaxTextureCoords  
attribute float gl_FogCoord;
```

### Uniform Variables

Shadery mohou přistupovat k aktuálnímu stavu OpenGL přes vestavěné proměnné obsahující rezervovaný prefix “gl\_”

```
gl_ModelViewMatrix
```

```
...
```

## Důležité vestavěné proměnné

### Speciální výstupní proměnné

```
vec4 gl_position; // must be written to
```

Proměnná `gl_position` je určena pro zápis pozice vrcholu v clipping coordinates, poté co byly vypočteny ve vertex shaderu.

```
float gl_PointSize; // may be written to
```

```
vec4 gl_ClipVertex; // may be written to
```

### Varying Variables

```
varying vec4 gl_FrontColor;
```

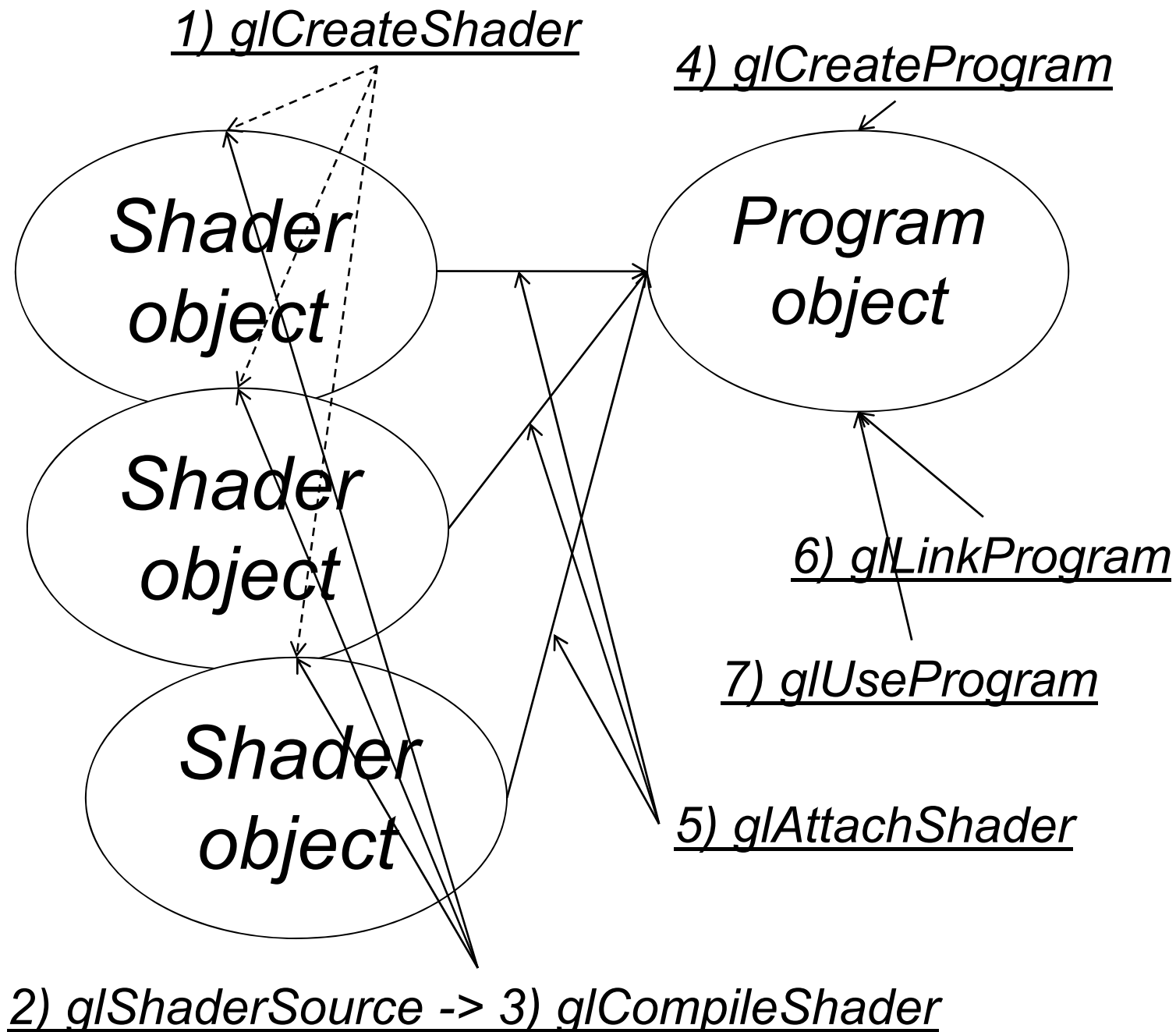
```
varying vec4 gl_BackColor;
```

```
varying vec4 gl_FrontSecondaryColor;
```

```
varying vec4 gl_BackSecondaryColor;
```

```
varying vec4 gl_TexCoord[gl_MaxTextureCoords];
```

```
varying float gl_FogFragCoord;
```



# Program a kompilace

GLSL musí obsahovat podporu pro uložení zdrojového kódu, překompilovaného kódu a výsledného spustitelného kódu.

Vytvoření objektu shaderu.

```
GLuint glCreateShader(GLenum shaderType)
```

*shaderType*: GL\_VERTEX\_SHADER,  
GL\_FRAGMENT\_SHADER

Vytvoří prázdný objekt shaderu a vrátí nenulovou referenční hodnotu. V okamžiku vytvoření je parametr GL\_SHADER\_TYPE vážící se k tomuto objektu nastaven na hodnotu *shaderType*



## Program a kompilace

Uložení zdrojového kódu.

```
void glShaderSource(GLuint shader,  
                   GLsizei count,  
                   const GLchar **string,  
                   const GLint *length)
```

*shader* : Odkaz na objekt shaderu

*string* : Pole řetězců obsahující program

*count* : Počet řetězců v poli *string*

*length* : Délky řetězců. Jeli *length* NULL, předpokládá se ukončení řetězců nulou.

## Program a kompilace

Kompilace objektů shaderu.

```
void glCompileShader(GLuint shader)
```

*shader* : Odkaz na objekt shaderu

Provede kompilaci zdrojového kódu. Kontrola může být provedena pomocí **glGetShader**(*shader*, GL\_COMPILE\_STATUS, *parameters*).

Informace o kompilaci mohou být získány pomocí **glGetShaderInfoLog**

## Program a kompilace

### Linkování.

```
GLuint glCreateProgram(void)
```

Vytvoří prázdný objekt programu

```
void glAttachShader(GLuint program,  
                    GLuint shader)
```

Připojí objekt shaderu k programu objektu. Není žádné omezení na množství připojených objektů shaderů. Je možné připojit objekt před nahráním zdrojového kódu, nebo před kompilací. Je také možné připojit jeden objekt shaderu k více objektům programu. Tímto je pouze specifikována množina objektů pro slinkování.

## Program a kompilace

### Linkování.

```
void glLinkProgram(GLuint program)
```

Provede slinkování.

Kontrola může být provedena pomocí **glGetProgram(GL\_LINK\_STATUS)** a **glGetProgramInfoLog**

- Výsledkem úspěšného slinkování je nastavení všech aktivních uživatelem definovaných uniformních proměnných patřících k *program* na 0.
- Všem aktivním *uniformním* proměnných je přiřazena pozice, na kterou je možné se ptát pomocí **glGetUniformLocation**
- Všechny aktivní uživatelsky definované *atributové* proměnné jsou navázány

## Program a kompilace

### Použití.

```
void glUseProgram(GLuint program)
```

Instaluje objekt programu specifikovaného pomocí *program*, jakožto součást vykreslovací pipeline.

### Smazání.

```
void glDeleteShader(GLuint shader).  
void glDeleteProgram(GLuint program)  
void glDetachShader(GLuint shader)
```

Uvolní paměť a zneplatní jméno asociované s objektem. Jestliže smazání není možné (je připojen k objektu, nebo součástí vykreslovacího stavu), je pouze označen pro smazání.

## Program a kompilace

### Dotazy.

```
void glGetShaderiv(GLuint shader,  
                  GLenum, pname,  
                  GLint, *params)
```

*V params vrátí hodnotu parametru pro specifikovaný objekt shaderu.*

*pname:* GL\_SHADER\_TYPE,  
GL\_DELETE\_STATUS,  
GL\_COMPILE\_STATUS,  
GL\_INFO\_LOG\_LENGTH,  
GL\_SHADER\_SOURCE\_LENGTH

## Program a kompilace

### Dotazy.

```
void glGetProgramiv(GLuint program,  
                    GLenum, pname,  
                    GLint, *params)
```

*V params vrátí hodnotu parametru pro specifikovaný objekt programu.*

*pname:* GL\_DELETE\_STATUS,  
GL\_LINK\_STATUS,  
GL\_VALIDATE\_STATUS,  
GL\_INFO\_LOG\_LENGTH,  
GL\_ATTACHED\_SHADERS,  
GL\_ACTIVE\_ATTRIBUTES,  
GL\_ACTIVE\_ATTRIBUTE\_MAX\_LENGTH,  
GL\_ACTIVE\_UNIFORMS  
GL\_ACTIVE\_UNIFORM\_MAX\_LENGTH

## Program a kompilace

### Dotazy.

```
void glGetShaderInfoLog(GLuint shader,  
                        GLsizei maxLength,  
                        GLsizei *length,  
                        GLchar *infoLog)
```

Vrátí informační log pro specifikovaný objekt shaderu.

*infoLog* - vrácená hodnota

*maxLength* – maximální délka vrácené zprávy

*length* – skutečná délka vrácené zprávy

```
void glGetProgramInfoLog(GLuint program,  
                          GLsizei maxLength,  
                          GLsizei *length,  
                          GLchar *infoLog)
```



