

BÀI THỰC HÀNH SỐ 2

Lập trình với kiến trúc OpenGL hiện đại

A. Cài đặt các phép biến đổi

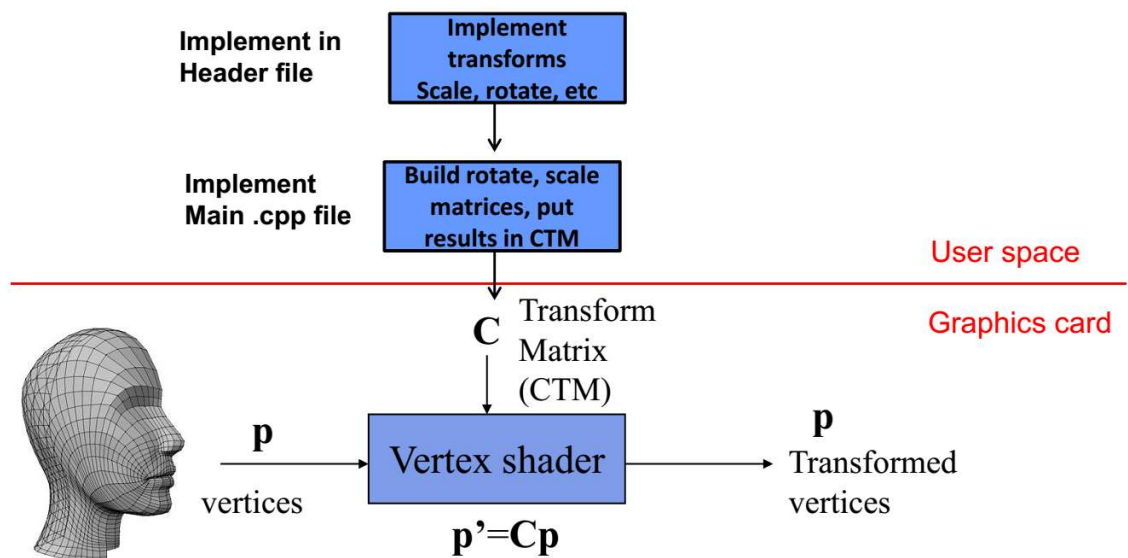
1. **Ví dụ:** Vẽ hình lập phương đơn vị quay theo trục Ox, Oy, Oz được điều khiển bằng chuột.

2. Hướng dẫn

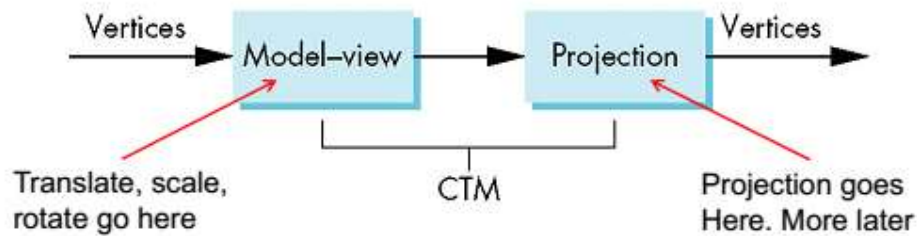
- Trong kiến trúc OpenGL trước phiên bản 3.0, các phép biến đổi được thực hiện bởi các lệnh: `glTranslate`, `glScale`, `glRotate`, ma trận `modelview`, ...
- Từ phiên bản 3.1 trở đi, với kiến trúc lập trình OpenGL hiện đại, nếu lập trình viên cần các phép biến đổi, các ma trận biến đổi thì phải cài đặt chúng.

a) Ma trận biến đổi hiện hành (CTM – Current Transformation Matrix)

- Ma trận biến đổi được cài đặt là một ma trận trong hệ tọa độ thuần nhất với kích thước 4×4 .
- CTM được định nghĩa và cập nhật trong chương trình của người dùng. Thường được cài đặt như sau:
 - o Trong thư viện Header file (`mat.h`): Cài đặt các phép biến đổi như tịnh tiến, quay, biến đổi tỉ lệ, ...
 - o Trong chương trình ứng dụng file `.cpp`: Gọi các phép biến đổi, cài đặt ma trận CTM phù hợp.
 - o Trong Vertex Shader: Cài đặt thi hành, tạo ảnh của đỉnh qua phép biến đổi được truyền CTM vào để tính.



- CTM sẽ bao gồm các phép biến đổi về model, view và phép chiếu – projection.



b) Các cài đặt trong thư viện mat.h

- Thư viện mat.h là thư viện tự cài đặt (được cung cấp kèm project, do Prof. Angel phát triển)
 - Nội dung cài đặt:
 - o Kiểu ma trận: mat4 (ma trận 4x4), mat3 (ma trận 3x3), ...
 - o Các toán tử cho ma trận
 - o Các hàm biến đổi: Translate, Scale, Rotate, ...
- mat4 Translate(const GLfloat x, const GLfloat y, const GLfloat z)
 mat4 Scale(const GLfloat x, const GLfloat y, const GLfloat z)

Ví dụ: Trong chương trình chính có khai báo `#include "mat.h"`, ta có:

```
mat4 m; //m là một ma trận 4x4 và được gán là ma trận đơn vị
mat4 ctm=Translate(3, 6, 4);
```

Khi đó ta có:

$$\text{CTM} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{Translation Matrix}$$

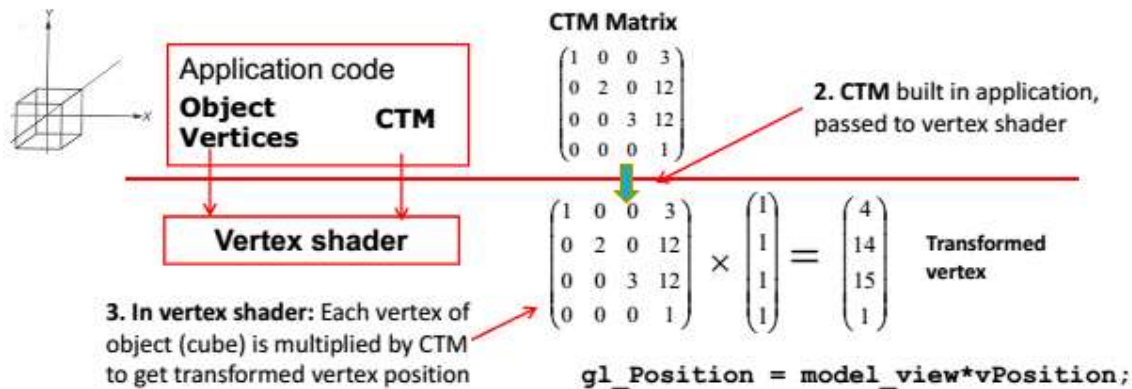
c) Cách áp dụng các ma trận biến đổi

Xây dựng ma trận CTM phù hợp:

```
mat4 m = Identity();
mat4 s = Scale(1,2,3);
mat4 t = Translate(3,6,4);
m = m*s*t;
colorcube( );
```

1. In application:

Load object vertices into points[] array -> VBO
Call glDrawArrays



d) Chuyển biến CTM trong ứng dụng đến Vertexshader

```
void display( ){
    .....
    mat4 m = Identity();
    mat4 s = Scale(1,2,3);
    mat4 t = Translate(3,6,4);
    m = m*s*t;

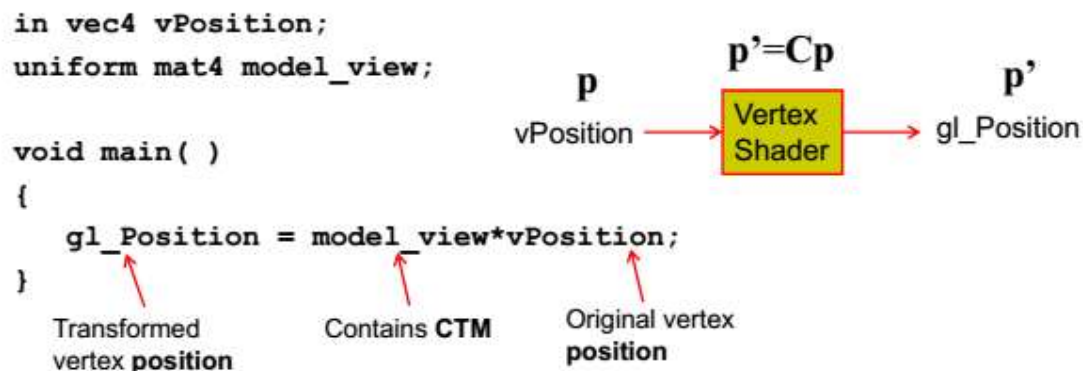
    // find location of matrix variable "model_view" in shader
    // then pass matrix to shader

    matrix_loc = glGetUniformLocation(program, "model_view");
    glUniformMatrix4fv(matrix_loc, 1, GL_TRUE, m);
    .....
}
```

Build CTM in application

CTM matrix *m* in application is same as *model_view* in shader

e) Cài đặt Vertex Shader



- Lưu ý: Biến **Uniform**

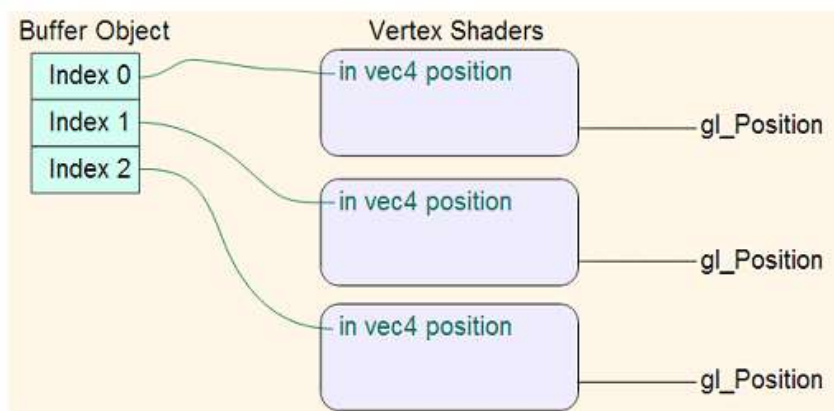
- + Được áp dụng không thay đổi trên toàn bộ các đỉnh của nguyên mẫu
 - + Không thay đổi giá trị trong shader
- Khi thực hiện lệnh `glDrawArrays()`, vertex shader đã gọi với `vPosition` khác nhau trên shader.

Ví dụ: `colorcube()` khi vẽ bằng lệnh `glDrawArrays` có 36 đỉnh, mỗi vertex shader nhận một đỉnh lưu trữ trong `vPosition`.

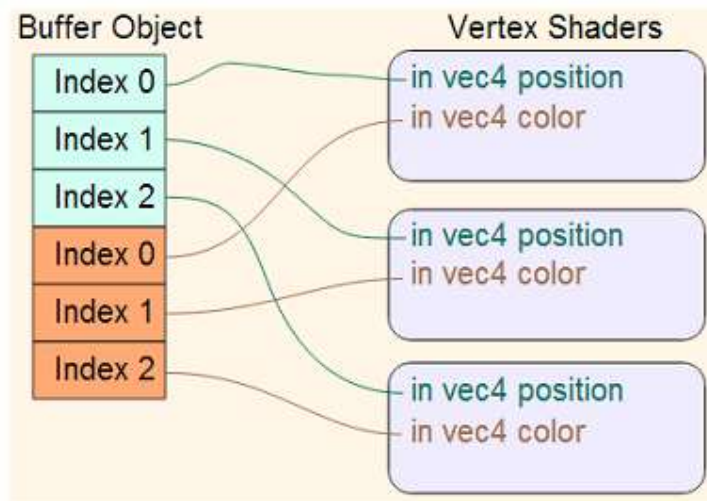
- Shader tính toán vị trí đỉnh được biến đổi và lưu vào `gl_Position`.

Cụ thể:

- + Các thuộc tính vị trí của đỉnh:

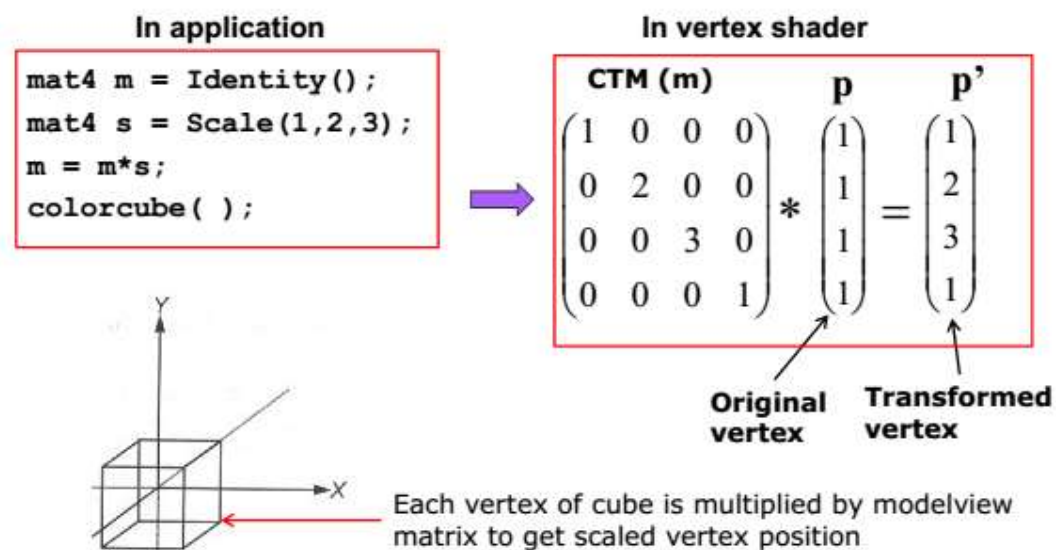


+ Khi VBO lưu trữ nhiều loại thuộc tính đỉnh:

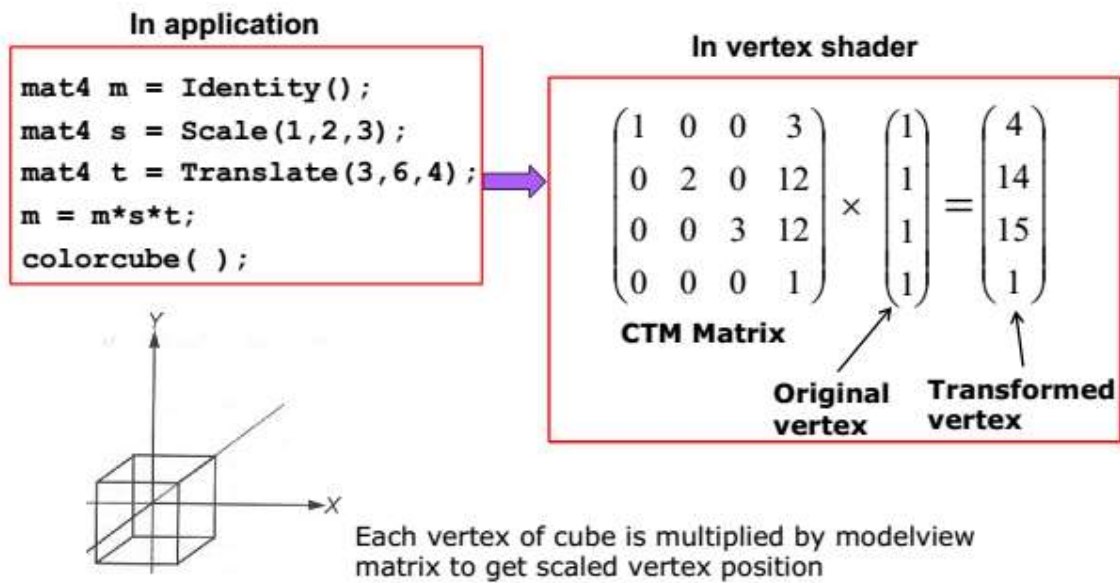


f) Ví dụ

Giả sử có đỉnh (1, 1, 1) là một trong 8 đỉnh của hình lập phương



Hoặc



3. Bài tập

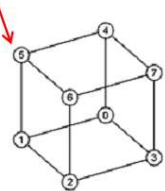
- 3.1. Vẽ hình lập phương đơn vị khi quay theo trục Oy góc quay 30 độ rồi quay theo trục Ox góc 30 độ.
- 3.2. Vẽ hình hàng rào bằng các thanh hình hộp thẳng đứng với mỗi thanh rào có kích thước: 0.15 x 1 x 0.03.
- 3.4. Vẽ hình 2D chuyển động: vẽ cảnh ô tô chuyển động ngang màn hình.
- 3.5. Đọc về cài đặt cánh tay robot theo tài liệu [1]

ĐỌC THÊM - Vẽ hình lập phương đơn vị

a) Thiết lập các thuộc tính đỉnh, màu cho mỗi đỉnh

Declare array of (x,y,z,w) vertex positions
for a unit cube centered at origin
(Sides aligned with axes)

```
point4 vertices[8] = {
0 point4( -0.5, -0.5,  0.5, 1.0 ),
1 point4( -0.5,  0.5,  0.5, 1.0 ),
2 point4(  0.5,  0.5,  0.5, 1.0 ),
3 point4(  0.5, -0.5,  0.5, 1.0 ),
4 point4( -0.5, -0.5, -0.5, 1.0 ),
5 point4( -0.5,  0.5, -0.5, 1.0 ),
6 point4(  0.5,  0.5, -0.5, 1.0 ),
7 point4(  0.5, -0.5, -0.5, 1.0 )
};
```

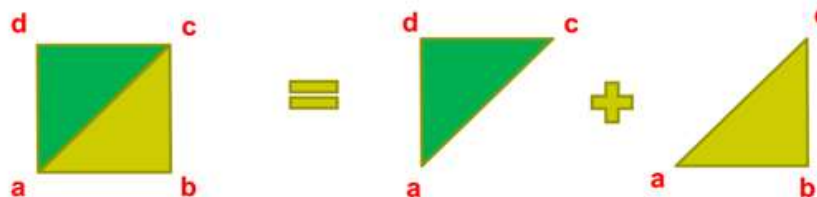


Declare array of vertex colors
(set of RGBA colors vertex can have)

```
color4 vertex_colors[8] = {
color4( 0.0, 0.0, 0.0, 1.0 ), // black
color4( 1.0, 0.0, 0.0, 1.0 ), // red
color4( 1.0, 1.0, 0.0, 1.0 ), // yellow
color4( 0.0, 1.0, 0.0, 1.0 ), // green
color4( 0.0, 0.0, 1.0, 1.0 ), // blue
color4( 1.0, 0.0, 1.0, 1.0 ), // magenta
color4( 1.0, 1.0, 1.0, 1.0 ), // white
color4( 0.0, 1.0, 1.0, 1.0 ) // cyan
};
```

b) Hàm quad thực hiện thiết lập 1 mặt hình lập phương bằng 2 tam giác

- Khi vẽ bằng `glDrawArray` với nguyên thể là `TRIANGLES`, 1 mặt hình lập phương được vẽ bằng 2 tam giác)



```
// quad generates two triangles (a,b,c) and (a,c,d) for each face
// and assigns colors to the vertices
```

```
int Index = 0; // Index goes 0 to 5, one for each vertex of face
```

```
void quad( int a, int b, int c, int d )
```

```
{
0 colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
1 colors[Index] = vertex_colors[b]; points[Index] = vertices[b]; Index++;
2 colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++;
3 colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
4 colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++;
5 colors[Index] = vertex_colors[d]; points[Index] = vertices[d]; Index++;
}
```

```
quad 0 = points[0 - 5]
quad 1 = points[6 - 11]
quad 2 = points[12 - 17] ...etc
```

Points[] array to be
Sent to GPU

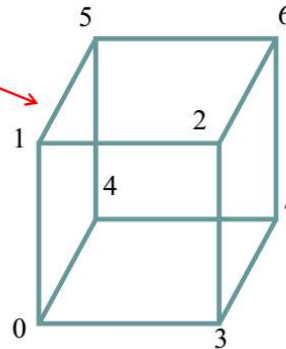
Read from appropriate index
of unique positions declared

c) Thiết lập các mặt cho hình lập phương

```
// generate 6 quads,
// sides of cube

void colorcube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```

```
point4 vertices[8] = {
    0 point4( -0.5, -0.5,  0.5, 1.0 ),
    1 point4( -0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5, -0.5,  0.5, 1.0 ),
    4 point4( -0.5, -0.5, -0.5, 1.0 ),
    5 point4( -0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5, -0.5, -0.5, 1.0 )
};
```



d) Thiết lập VAO, VBO và chuyển dữ liệu vào GPU

```
void init()
{
    colorcube(); // Generates cube data in application using quads

    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays ( 1, &vao );
    glBindVertexArray ( vao );

    // Create a buffer object and move data to GPU
    GLuint buffer;
    glGenBuffers( 1, &buffer );
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points) +
        sizeof(colors), NULL, GL_STATIC_DRAW );
```



Send `points[]` and `colors[]` data to GPU separately using `glBufferSubData`

```
glBufferSubData( GL_ARRAY_BUFFER, 0, sizeof(points), points );  
glBufferSubData( GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors );
```



```
// Load vertex and fragment shaders and use the resulting shader program  
GLuint program = InitShader( "vshader36.glsl", "fshader36.glsl" );  
glUseProgram( program );
```

```
// set up vertex arrays
```

```
GLuint vPosition = glGetAttribLocation( program, "vPosition" );  
glEnableVertexAttribArray( vPosition );  
glVertexAttribPointer( vPosition, 4, GL_FLOAT, GL_FALSE, 0,   
    BUFFER_OFFSET(0) );
```

Specify vertex data

```
GLuint vColor = glGetAttribLocation( program, "vColor" );  
glEnableVertexAttribArray( vColor );  
glVertexAttribPointer( vColor, 4, GL_FLOAT, GL_FALSE, 0,   
    BUFFER_OFFSET(sizeof(points)) );
```



Specify color data

e) Hàm display()

```
void display( void )  
{  
    glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT );  
    glDrawArrays( GL_TRIANGLES, 0, NumVertices );  
    glutSwapBuffers();  
}
```

Draw series of triangles forming cube

f) Hàm idle() – được gọi khi GPU không làm gì

```
void idle( void )  
{  
    theta[axis] += 0.01;  
  
    if ( theta[axis] > 360.0 ) {  
        theta[axis] -= 360.0;  
    }  
  
    glutPostRedisplay();  
}
```

The idle() function is called whenever nothing to do

Use it to increment rotation angle in steps of theta = 0.01 around currently selected axis

g) Hàm mouse()

```
...  
enum { Xaxis = 0, Yaxis = 1, Zaxis = 2, NumAxes = 3 };
```

```
void mouse( int button, int state, int x, int y )  
{  
    if ( state == GLUT_DOWN ) {  
        switch( button ) {  
            case GLUT_LEFT_BUTTON:    axis = Xaxis; break;  
            case GLUT_MIDDLE_BUTTON:   axis = Yaxis; break;  
            case GLUT_RIGHT_BUTTON:    axis = Zaxis; break;  
        }  
    }  
}
```

Select axis (x,y,z) to rotate around
Using mouse click

Một số hàm OpenGL

- `void glBufferSubData(GLenum target, GLintptr offset, GLsizeiptr size, const GLvoid * data);`

Thiết lập việc lưu trữ dữ liệu con của glBufferData. Cụ thể:

target

Buffer Binding Target	Purpose
GL_ARRAY_BUFFER	Vertex attributes
...	...

offset

Chỉ rõ độ rời, vị trí đầu tiên lưu trữ dữ liệu của đối tượng buffer, đơn vị tính là bytes.

size

Chỉ rõ kích cỡ tính bằng byte của vùng lưu trữ dữ liệu sẽ được thay thế.

data

Chỉ rõ một con trỏ đến dữ liệu mới mà sẽ được copy vào vị trí lưu trữ dữ liệu.

Ví dụ:

```
glBufferSubData( GL_ARRAY_BUFFER, 0, sizeof(points), points );
```

```
glBufferSubData( GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors );
```

- `GLint glGetUniformLocation(GLuint program, const GLchar *name);`

Trả ra một giá trị kiểu nguyên là vị trí của biến uniform trong đối tượng *program*. Trong đó:

program

Chỉ rõ đối tượng *program* được yêu cầu.

name

Chỉ đến một xâu được kết thúc bằng ký tự null chứa tên của biến uniform mà vị trí của biến được yêu cầu.