Introduction to Graphics Programming and its Applications

繪圖程式設計與應用

Appendix: The GLM Library

Instructor: Hung-Kuo Chu

Department of Computer Science

National Tsing Hua University

CS4505

OpenGL Mathematics

GLSL + Optional features = OpenGL Mathematics (GLM)

A C++ mathematics library for graphics programming

THE GLM LIBRARY

GLM

- OpenGL Mathematics (GLM) is a header only C++ mathematics library
- For graphics software programming, based on the OpenGL Shading Language (GLSL) specifications
- Free and open-source, licensed under the MIT License
- http://glm.g-truc.net/

GLM

- Why GLM?
  - ***GLSL-like syntax and vector/matrix object***
  - Header-only, easy to setup
  - Provides replacement function for fixed pipeline
  - Designed for OpenGL, and work perfectly with OpenGL
- Latest release version: 0.9.8.4 (2017/1/22)

GLM

- Code sample

```cpp
#include <glm/vec3.hpp> // glm::vec3
#include <glm/vec4.hpp> // glm::vec4
#include <glm/mat4x4.hpp> // glm::mat4
#include <glm/gtc/matrix_transform.hpp> // glm::translate, glm::rotate, glm::scale

glm::mat4 camera(float Translate, glm::vec2 const & Rotate)
{
    glm::mat4 Projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.f);
    glm::mat4 View = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, -Translate));
    View = glm::rotate(View, Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
    View = glm::rotate(View, Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));
    glm::mat4 Model = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));
    return Projection * View * Model;
}
```

Include Headers

Declare a 4x4 matrix

Use matrix transform functions

GLM Setup

- Include glm/glm.hpp for basic types

```
#include <glm/glm.hpp> // include basic vector and matrix types
```

- Enable swizzle operators (optional)

```
#define GLM_SWIZZLE // enable swizzle operators
#include <glm/glm.hpp>

void foo()
{
    glm::vec4 ColorRGBA(1.0f, 0.5f, 0.0f, 1.0f);
    glm::vec3 ColorBGR = ColorRGBA.bgr();

    glm::vec4 PositionA(1.0f, 0.5f, 0.0f, 1.0f);
    glm::vec3 PositionB = PositionA.xyz() * 2.0f;
}
```

Swizzle operators

GLM Setup

- Compiler compatibility with GLM_SWIZZLE defined

GLM	VS 2010	VS 2013/2015	Xcode 7.1.2
0.9.6.3	✓	✓	✓
0.9.7.3	✗	✓	✓

- Without GLM_SWIZZLE defined

GLM	VS 2010	VS 2013/2015	Xcode 7.1.2
0.9.6.3	✓	✓	✓
0.9.7.3	✓	✓	✓

GLM Setup

- Use namespace glm (optional)

```
#define GLM_SWIZZLE
#include <glm/glm.hpp>

using namespace glm;

mat4 camera(float Translate, vec2 const & Rotate)
{
    mat4 Projection = perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.f);
    mat4 View = translate(mat4(1.0f), vec3(0.0f, 0.0f, -Translate));
    View = rotate(View, Rotate.y, vec3(-1.0f, 0.0f, 0.0f));
    View = rotate(View, Rotate.x, vec3(0.0f, 1.0f, 0.0f));
    mat4 Model = scale(mat4(1.0f), vec3(0.5f));
    return Projection * View * Model;
}
```

Cleaner code!

GLM Setup

- Include GLM GTX/GTC modules for extended functions (optional)
- http://glm.g-truc.net/0.9.8/api/modules.html

```
#define GLM_SWIZZLE
#include <glm/glm.hpp>
#include <glm/gtx/vector_angle.hpp> // include vector angle extension
#include <glm/gtc/matrix_transform.hpp> // include matrix transform extension

using namespace glm;

float vector_angle(vector3 A, vector3 B)
{
    // use glm::angle to compute the angle between A and B
    return angle(normalize(A), normalize(B));
}
```

Vector

- Declare a vector in glm
- Integer type vectors: ivec{2,3,4}
- Float type vectors: vec{2,3,4}

```
ivec2 mivec2; // mivec2 = (0, 0)
ivec3 mivec3(mivec2, 0); // mivec3 = (0, 0, 0)
ivec4 mivec4(mivec3, 0); // mivec4 = (0, 0, 0, 0)

vec2 mvec2; // mvec2 = (0.0f, 0.0f)
vec3 mvec3(1.0f, 2.0f, 3.0f); // mvec3 = (1.0f, 2.0f, 3.0f)
vec4 mvec4(mvec3.zxy(), 4.0f); // mvec4 = (3.0f, 1.0f, 2.0f, 4.0f)
mvec4 = vec4(1.0f, mvec2, 2.0f); // mvec4 = (1.0f, 0.0f, 0.0f, 2.0f)
```

Use swizzle operator and constructor like GLSL

Vector

```
vec3 V(3.0f, 4.0f, 5.0f); // a non-unit vector V
float V_len = V.length(); // find the length of V by .length() member
vec3 V_unit = V / V_len; // find the unit vector by division
V_unit = normalize(V); // or by glm::normalize function
```

Vector Dot Product

```
vec3 A(x1, y1, z1), B(x2, y2, z2);
// calculate dot product by definition
float dotProduct = A.x * B.x + A.y * B.y + A.z * B.z;
dotProduct = dot(A, B); // or use glm::dot function
// calculate angle by definition (note that 0 <= angleRad <= PI)
float angleRad = acos(dotProduct / A.length() / B.length());
float angleDegree = angleRad / PI * 180.0f;
// or use glm::angle from GTX_vector_angle
angleRad = angle(normalize(A), normalize(B));
```

Vector Cross Product

```
vec3 A(x1, y1, z1), B(x2, y2, z2);
// calculate dot product by definition
vec3 crossProduct = vec3(
    A.y * B.z – A.z * B.y,
    A.z * B.x – A.x * B.z,
    A.x * B.y – A.y * B.x
);
crossProduct = cross(A, B); // or use glm::cross function
```

Matrix

- Declare a matrix in glm

```
mat4 M1; // create an identity matrix
mat3 M2(2.0f); // create a 3x3 matrix with diagonal components set to 2.0f
mat4 M3(vec4(1, 2, 3, 4), vec4(1, 2, 3, 4),
        vec4(1, 2, 3, 4), vec4(1, 2, 3, 4)); // create a matrix by 4D vectors
float c00 = M3[0][0]; // access the matrix component
```

- Apply an identity matrix to a vector

```
vec4 V(1, 2, 3, 1);
mat4 M4(1.0f); // create a 4x4 identity matrix
vec4 V2 = M4 * V; // apply the identity matrix
if(V == V2)
    printf("V is the same as V2");
```

Translation Matrix

- Construct a translation matrix by (T_x, T_y, T_z)

```
#include <glm/gtx/transform.hpp>


vec4 V(1, 2, 3, 1);
mat4 M4(1.0);
M4 = translate(M4, vec3(Tx, Ty, Tz));
vec4 V2 = M4 * V; // apply the translation matrix
```

Scaling Matrix

- Construct a scaling matrix by (S_x, S_y, S_z)

```
#include <glm/gtx/transform.hpp>


vec4 V(1, 2, 3, 1);
mat4 M4(1.0);
M4 = scale(M4, vec3(Sx, Sy, Sz));
vec4 V2 = M4 * V; // apply the scaling matrix
```

Rotation around Given Axis

- Construct a rotation matrix by given axis $(\mathrm{R}_x, \mathrm{R}_y, \mathrm{R}_z)$ and angle θ

```
#include <glm/gtx/transform.hpp>
#define PI 3.14159265358979323846
#define deg2rad(x) ((x)*((PI)/(180.0)))

vec4 V(1, 2, 3, 1);
mat4 M4(1.0);
M4 = rotate(M4, deg2rad(theta), vec3(Rx, Ry, Rz));
vec4 V2 = M4 * V; // apply the rotation matrix
```

Viewing Matrix

- Construct a viewing matrix

```
#include <glm/gtx/transform.hpp>

vec4 V(1, 2, 3, 1);
mat4 M4 = lookAt(vec3(Px, Py, Pz), vec3(Ex, Ey, Ez), vec3(Ux, Uy, Uz));
vec4 V2 = M4 * V; // apply the viewing matrix
```

Perspective Projection Matrix

- Construct a perspective projection matrix

```
#include <glm/gtx/transform.hpp>
#define PI 3.14159265358979323846
#define deg2rad(x) ((x)*((PI)/(180.0)))

vec4 V(1, 2, 3, 1);
mat4 M4 = frustum(left, right, bottom, top, nearVal, farVal); // use frustum
M4 = perspective(deg2rad(fovy), aspect, nearVal, farVal); // or perspective
vec4 V2 = M4 * V; // apply the projection matrix
```

Orthographic Projection Matrix

- Construct an orthographic projection matrix

```
#include <glm/gtx/transform.hpp>


vec4 V(1, 2, 3, 1);
mat4 M4 = ortho(left, right, bottom, top);
M4 = ortho(left, right, bottom, top, 0, 1); // this is the same as above
M4 = ortho(left, right, bottom, top, nearVal, farVal);
vec4 V2 = M4 * V; // apply the projection matrix
```

Matrix Transformation

- V' = Projection * Viewing * Modeling * V

main.cpp

```
mat4 model;
mat4 view;
mat4 proj;
…
mat4 mvp = proj * view * model;
glUniformMatrix4fv(0, 1, GL_FALSE, &mvp[0][0]);
```

vert.glsl

```
mat4 um4mvp;
in vec4 vertex;
void main()
{
    gl_Position = um4mvp * vertex;
}
```