計算機圖學單元介紹

一、英文主題：

Chapter 09: Programming Shader

二、中文主題：

單元 09：可編譯著色器

三、組別：

第 8 組

四、組員：

B0829013_林冠斌；B0829017_張睿；B0829029_劉建良；B0829035_莊淙賢；
B0829054_李彥廷；B0829055_毛聖淇；

作業分工：

(詳見作業報告) ...保留此句，本項目不必填寫

五、功能簡述：

例：運用 GLSL 操作著色器，以做出更貼近現實的圖像，像是光影

六、主要程式碼：

相關檔案：Ch_09_tm8_src1.cpp

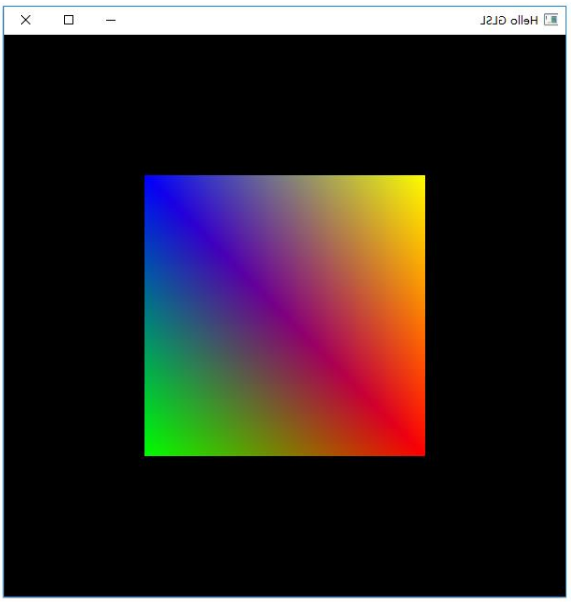
(以 1x1 表格填寫，文字為 “Segoe UI” 11 點字，固定行高 12 點，內容可變更文字顏色)

```
#include <GL/glew.h>
#include "Textfile.h"
#include <GL/freeglut.h>
#include <iostream>
#pragma comment(lib,"glew32.lib")

using namespace std;
GLuint vShader, fShader;
GLuint vaoHandle;

float positionData[] = {
                                    -0.5f,-0.5f,0.0f,1.0f,
                                    0.5f,-0.5f,0.0f,1.0f,
                                    0.5f,0.5f,0.0f,1.0f,
                                    -0.5f,0.5f,0.0f,1.0f
};
float colorData[] = {
                                  1.0f, 0.0f, 0.0f,1.0f,
                                  0.0f, 1.0f, 0.0f,1.0f,
                                  0.0f, 0.0f, 1.0f,1.0f,
                                  1.0f,1.0f,0.0f,1.0f
};

void initShader(const char *VShaderFile, const char *FShaderFile)
{

      const GLubyte *vendor = glGetString(GL_VENDOR);
      const GLubyte *renderer = glGetString(GL_RENDERER);
      const GLubyte *version = glGetString(GL_VERSION);
      const GLubyte *glslVersion = glGetString(GL_SHADING_LANGUAGE_VERSION);
      cout << "GPU     : " << vendor << endl;
```

```
cout << "gpu type        : " << renderer << endl;
cout << "OpenGL version     : " << version << endl;
cout << "GLSLversion        : " << glslVersion << endl;

vShader = glCreateShader(GL_VERTEX_SHADER);
if (0 == vShader)
{
	cerr << "ERROR : Create vertex shader failed" << endl;
	exit(1);
}
const GLchar *vShaderCode = textFileRead(VShaderFile);
const GLchar *vCodeArray[1] = { vShaderCode };

glShaderSource(vShader, 1, vCodeArray, NULL);

glCompileShader(vShader);

GLint compileResult;
glGetShaderiv(vShader, GL_COMPILE_STATUS, &compileResult);
if (GL_FALSE == compileResult)
{
	GLint logLen;
	glGetShaderiv(vShader, GL_INFO_LOG_LENGTH, &logLen);
	if (logLen > 0)
	{
		char *log = (char *)malloc(logLen);
		GLsizei written;
		glGetShaderInfoLog(vShader, logLen, &written, log);
		cerr << "vertex shader compile log : " << endl;
		cerr << log << endl;
		free(log);
	}
}

fShader = glCreateShader(GL_FRAGMENT_SHADER);
if (0 == fShader)
{
	cerr << "ERROR : Create fragment shader failed" << endl;
	exit(1);
}

const GLchar *fShaderCode = textFileRead(FShaderFile);
const GLchar *fCodeArray[1] = { fShaderCode };
glShaderSource(fShader, 1, fCodeArray, NULL);

glCompileShader(fShader);

glGetShaderiv(fShader, GL_COMPILE_STATUS, &compileResult);
if (GL_FALSE == compileResult)
{
	GLint logLen;
	glGetShaderiv(fShader, GL_INFO_LOG_LENGTH, &logLen);
	if (logLen > 0)
	{
		char *log = (char *)malloc(logLen);
		GLsizei written;
		glGetShaderInfoLog(fShader, logLen, &written, log);
		cerr << "fragment shader compile log : " << endl;
		cerr << log << endl;
		free(log);
	}
}
```

```
	GLuint programHandle = glCreateProgram();
	if (!programHandle)
	{
		cerr << "ERROR : create program failed" << endl;
		exit(1);
	}

	glAttachShader(programHandle, vShader);
	glAttachShader(programHandle, fShader);
	glLinkProgram(programHandle);
	GLint linkStatus;
	glGetProgramiv(programHandle, GL_LINK_STATUS, &linkStatus);
	if (GL_FALSE == linkStatus)
	{
		cerr << "ERROR : link shader program failed" << endl;
		GLint logLen;
		glGetProgramiv(programHandle, GL_INFO_LOG_LENGTH,
			&logLen);
		if (logLen > 0)
		{
			char *log = (char *)malloc(logLen);
			GLsizei written;
			glGetProgramInfoLog(programHandle, logLen,
				&written, log);
			cerr << "Program log : " << endl;
			cerr << log << endl;
		}
	}
	else
	{
		glUseProgram(programHandle);
	}
}

void initVBO()
{

	glGenVertexArrays(1, &vaoHandle);
	glBindVertexArray(vaoHandle);


	GLuint vboHandles[2];
	glGenBuffers(2, vboHandles);
	GLuint positionBufferHandle = vboHandles[0];
	GLuint colorBufferHandle = vboHandles[1];


	glBindBuffer(GL_ARRAY_BUFFER, positionBufferHandle);

	glBufferData(GL_ARRAY_BUFFER, 16 * sizeof(float),
		positionData, GL_STATIC_DRAW);


	glBindBuffer(GL_ARRAY_BUFFER, colorBufferHandle);

	glBufferData(GL_ARRAY_BUFFER, 16 * sizeof(float),
		colorData, GL_STATIC_DRAW);

	glEnableVertexAttribArray(0);
	glEnableVertexAttribArray(1);

	glBindBuffer(GL_ARRAY_BUFFER, positionBufferHandle);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLubyte *)NULL);
	glBindBuffer(GL_ARRAY_BUFFER, colorBufferHandle);
```

```
		glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, (GLubyte *)NULL);
	}

	void init()
	{
		GLenum err = glewInit();
		if (GLEW_OK != err)
		{
			cout << "Error initializing GLEW: " << glewGetErrorString(err) << endl;
		}
		initShader("VertexShader.vert","FragmentShader.frag");
		initVBO();
		glClearColor(0.0, 0.0, 0.0, 0.0);
	}

	void display()
	{
		glClear(GL_COLOR_BUFFER_BIT);
		glBindVertexArray(vaoHandle);
		glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
		glBindVertexArray(0);
		glutSwapBuffers();
	}
	 void keyboard(unsigned char key, int x, int y)
	{
		switch (key)
		{
		case 27:
			glDeleteShader(vShader);
			glUseProgram(0);
			glutPostRedisplay();
			break;
		}
	}

	int main(int argc, char** argv)
	{
		glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
		glutInitWindowSize(600, 600);
		glutInitWindowPosition(100, 100);
		glutCreateWindow("Hello GLSL");
		init();
		glutDisplayFunc(display);
		glutKeyboardFunc(keyboard);
		glutMainLoop();
		return 0;
}
```



七、程式說明：

GLuint vShader , fShader	頂點、片段著色器對象
GLuint vaoHandle	VAO 對象
Float positionData[]	頂點位置陣列
Float colorData[]	頂點顏色陣列
Const Glubyte *vendor=glGetString(GL_VENDOR)	顯卡品牌
Const Glubyte *renderer=glGetString(GL_RENDERER)	顯卡型號
Const Glubyte *version=glGetString(GL_VERSION)	OpenGL 版本
Const Glubyte *glsVersion=glGetString(GL_SHADING_LANGUAGE_VERSION)	GLSL 版本
vShader =glCreateShader(GL_VERTEX_SHADER)	創建頂點著色器
Const GLchar *vShaderCode = textFileRead(VShaderFile)	著色器 code 和著色器對象綁定
Const GLchar *vCodeArray[1] = {vShaderCode}	著色器 code 和著色器對象綁定
GLShaderSource(vShader ,1 ,vCodeArray , NULL)	將陣列丟給對應的著色器對象
Glint compileResult	檢查編譯是否成功
GLAttachShader(programHandle,vShader)	將頂點著色器鏈結到程序中
GLAttachShader(programHandle,vShader)	將片元著色器鏈結到程序中

void initVBO()
{
glGenVertexArrays(1, &vaoHandle);
glBindVertexArray(vaoHandle);
GLuint vboHandles[2];
glGenBuffers(2, vboHandles);
GLuint positionBufferHandle = vboHandles[0];
GLuint colorBufferHandle = vboHandles[1];
glBindBuffer(GL_ARRAY_BUFFER,
positionBufferHandle);
	绑定 VAO，創造並填充 buffer 的物件
	綁定 VBO 以供使用，加載數據到 VBO

glBufferData(GL_ARRAY_BUFFER, 16 * sizeof(float),
positionData, GL_STATIC_DRAW);
 	
glBindBuffer(GL_ARRAY_BUFFER, colorBufferHandle);
glBufferData(GL_ARRAY_BUFFER, 16 * sizeof(float),
colorData, GL_STATIC_DRAW);
 	
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
 	
glBindBuffer(GL_ARRAY_BUFFER,
positionBufferHandle);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0,
(GLubyte *)NULL);
glBindBuffer(GL_ARRAY_BUFFER, colorBufferHandle);
glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0,
(GLubyte *)NULL);
}
	頂點座標
	頂點顏色
	
	調用 glVertexAttribPointer 之前需要綁定

void init()
{
GLenum err = glewInit();
if (GLEW_OK != err)
{
cout << "Error initializing GLEW: " <<
glewGetErrorString(err) << endl;
}上
initShader("VertexShader.vert","FragmentShader.frag");
initVBO();
	初始化 GLEW 函式庫，並加載頂點器和片元著色器到程序上，最後在綁定並加載 VAO、VBO

glClearColor(0.0, 0.0, 0.0, 0.0);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glBindVertexArray(vaoHandle);
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
glBindVertexArray(0);
glutSwapBuffers();
}
	使用 VAO、VBO 繪製

void keyboard(unsigned char key, int x, int y)
{
switch (key)
{
case 27:
glDeleteShader(vShader);
glUseProgram(0);
glutPostRedisplay();
break;
}
}
	ESC 可以直接退出著色器
	glutPostRedisplay() 會刷新顯示

八、延伸應用程式碼：

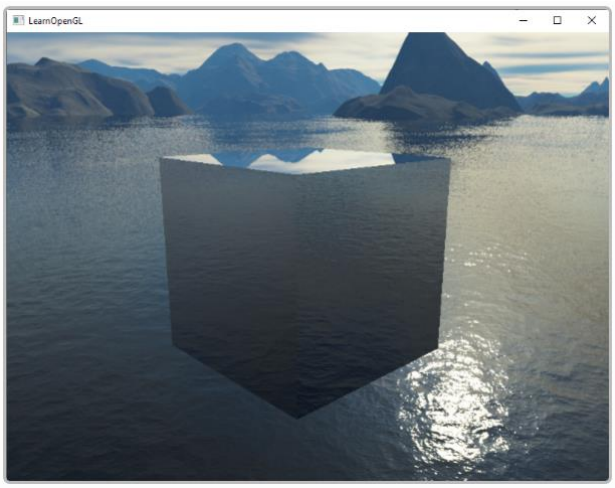
相關檔案：Ch_00_tm0_src2.cpp

(以 1x1 表格填寫，文字為 “Segoe UI” 11 點字，固定行高 12 點，內容可變更文字顏色)

```
#include <stdlib.h>
#include <GL/glut.h>


static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
```

```
	GLuint tex;

void display(void)
{

		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		glBindTexture(GL_TEXTURE_CUBE_MAP, tex);

	glLoadIdentity();
	glRotatef(theta[0], 1.0, 0.0, 0.0); //將目前矩陣乘旋轉矩陣
	glRotatef(theta[1], 0.0, 1.0, 0.0);
	glRotatef(theta[2], 0.0, 0.0, 1.0);

	glutSolidTeapot(1.0);    // call teapot function


	glutSwapBuffers();
}

void spinCube() //    call spin function
{

	theta[axis] += 2.0;
	if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
	glutPostRedisplay();
}

void mouse(int btn, int state, int x, int y)
{

	if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
	if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
	if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}

void myReshape(int w, int h)
{
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,
			2.0 * (GLfloat) h / (GLfloat) w, -10.0, 10.0);
	else
		glOrtho(-2.0 * (GLfloat) w / (GLfloat) h,
			2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);
	glMatrixMode(GL_MODELVIEW);
}

void key(char k, int x, int y)
{
	if(k == '1') glutIdleFunc(spinCube);
	if(k == '2') glutIdleFunc(NULL);
		if(k == 'q') exit(0);
}

void init()
{

	GLubyte red[3] = {255, 0, 0};      // set color
	GLubyte green[3] = {0, 255, 0};
	GLubyte blue[3] = {0, 0, 255};
```

```
        GLubyte cyan[3] = {0, 255, 255};
        GLubyte magenta[3] = {255, 0, 255};
        GLubyte yellow[3] = {255, 255, 0};

        glEnable(GL_DEPTH_TEST);    // make the opengl function run
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);
        glEnable(GL_TEXTURE_GEN_R);
        glEnable(GL_TEXTURE_CUBE_MAP);

      glGenTextures(1, &tex);
      glBindTexture(GL_TEXTURE_CUBE_MAP, tex);

      glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,GL_REFLECTION_MAP); //控制材質座標產生
      glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,GL_REFLECTION_MAP);
      glTexGeni(GL_R, GL_TEXTURE_GEN_MODE,GL_REFLECTION_MAP);
      glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X ,0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE,
red);
      glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X ,0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE,
green);
      glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y ,0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE,
blue);
      glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y ,0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE,
cyan);
      glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z ,0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE,
magenta);
      glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z ,0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE,
yellow);
      glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_WRAP_S,GL_REPEAT); //設置材質參
數
      glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_WRAP_T,GL_REPEAT);
      glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_WRAP_R,GL_REPEAT);
      glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
      glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_MIN_FILTER,GL_NEAREST);

      glClearColor(1.0,1.0,1.0,1.0);
}

void
main(int argc, char **argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
      glutInitWindowSize(500, 500);
      glutCreateWindow("colorcube");

      init();

      glutReshapeFunc(myReshape);
      glutDisplayFunc(display);
      glutIdleFunc(spinCube);
      glutIdleFunc(NULL);
      glutMouseFunc(mouse);

      glutKeyboardFunc(key);
      glutMainLoop();
}
```



glTexImage()	設置正方體的六個面
glTexParameter()	設置材質參數
glTexGeni()	控制材質座標產生
glGenTextures(1, &tex); glBindTexture(GL_TEXTURE_CUBE_MAP, tex);	創立一個立方體貼圖
glRotatef()	將目前矩陣乘旋轉矩陣

九、應用說明：

實作用 GLSL 操作 shader 並展現出更為細緻必貼近現實的效果

十、參考資料：

(1) https://blog.csdn.net/dcrmg/article/details/53648306

(2) https://learnopengl-cn.readthedocs.io/zh/latest/04%20Advanced%20OpenGL/06%20Cubemaps/