**計算機圖學單元介紹**

1. 英文主題：

Chapter 09: Programming Shader

1. 中文主題：

單元09：可編譯著色器

1. 組別：

第8組

1. 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

1. 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

1. 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;  } |

1. 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);  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);  } | 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  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();  glClearColor(0.0, 0.0, 0.0, 0.0);  } | 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 |
| 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 |

1. 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 |

1. 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/>