計算機圖學單元介紹

一、英文主題：
Chapter 08： Discrete Techniques

二、中文主題：
單元 08：離散技術

三、組別：
第 04 組

四、組員：
B0729061_余承叡；B0729062_李羽喬；B0729063_黃紹禎；B0729065_歐妍君；
B0744149_白儀婕；B0829044_葉丙俊；B0829051_吳沛聲

五、作業分工：
(詳見作業報告)

六、功能簡述：
在原本的 3D 多邊形上設定不同的映射函數，改變物體的紋理、材質、凹凸、透明度、對焦度、鋸齒處理，在顏色的處理上也可以運用矩陣對不同色塊進行混合，行成像是渲染技術以及模糊技術這樣的結果。

七、主要程式碼：

相關檔案：Ch_08_tm4_src1.cpp

```
#define BLACK 0

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include <GL/glut.h>


void draw_pixel(int ix, int iy, int value)
{
  glBegin(GL_POINTS); //繪製點
 glVertex2i( ix, iy); //畫點的位置
  glEnd(); //結束
} //此程式透過繪製每個點並且透過所有點來塑造出一條線的形狀

bres(int x1,int y1,int x2,int y2) //Bresenham's algorithm 用來描繪由兩點所決定的直線的演算法
{
 int dx, dy, i, e;
 int incx, incy, inc1, inc2;
 int x,y;

 dx = x2 - x1;
 dy = y2 - y1;

 if(dx < 0) dx = -dx;
 if(dy < 0) dy = -dy;
```

```
 incx = 1;
 if(x2 < x1) incx = -1;
 incy = 1;
 if(y2 < y1) incy = -1;
 x=x1;
 y=y1;

 if(dx > dy)
  {
    draw_pixel(x,y, BLACK);  //生成出點所在的位置在傳入 draw_pixel 去畫出點
    e = 2*dy - dx;
    inc1 = 2*( dy -dx);
    inc2 = 2*dy;
    for(i = 0; i < dx; i++)
    {
      if(e >= 0)
      {
        y += incy;
        e += inc1;
      }
      else e += inc2;
      x += incx;
      draw_pixel(x,y, BLACK);  //生成出點所在的位置在傳入 draw_pixel 去畫出點
    }
  }
  else
  {
    draw_pixel(x,y, BLACK);   //生成出點所在的位置在傳入 draw_pixel 去畫出點
    e = 2*dx - dy;
    inc1 = 2*( dx - dy);
    inc2 = 2*dx;
    for(i = 0; i < dy; i++)
    {
      if(e >= 0)
      {
        x += incx;
        e += inc1;
      }
      else e += inc2;
      y += incy;
      draw_pixel(x,y, BLACK);   //生成出點所在的位置在傳入 draw_pixel 去畫出點
   }
  }
 }

void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT); //清除目前已啟用色彩寫入的緩衝區
 bres(200, 200, 100, 50);
 glFlush(); //清空所有緩衝區
}

void myinit()
{
 glClearColor(1.0, 1.0, 1.0, 1.0);  //改變顏色背景 (紅色、綠色、藍色、Alpha)
```

```
 glColor3f(1.0, 0.0, 0.0);  //(紅、綠、藍) 將色彩設為紅色
 glPointSize(1.0); //設定點大小為 1 的像素
 glMatrixMode(GL_PROJECTION); //宣告接下來要進行投影
 glLoadIdentity(); //重置當前指定的矩陣?單位矩陣
 gluOrtho2D(0.0, 499.0, 0.0, 499.0); //通過正交投影把模型按照 1:1 的比例繪制到剪裁面上
}

void main(int argc, char** argv)
{

/* Standard GLUT initialization */

   glutInit(&argc,argv);  //初始 GLUT Lib
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); //創建窗口指定顯示模式的類型(單緩衝、顏色
模式)
   glutInitWindowSize(500,500); //窗口的長寬
   glutInitWindowPosition(0,0);  //窗口在螢幕的哪個位置
   glutCreateWindow("Bresenham's Algorithm");  //窗口名稱
   glutDisplayFunc(display); //窗口所要反覆執行哪個程式

   myinit(); //設定參數

   glutMainLoop(); //進入迴圈
}
}
```

八、程式說明：

(1) draw_pixel
- glBegin(GL_POINTS)：繪製點。
- glVertex2i(ix, iy)：指定點的 x 與 y 座標。
- 此程式透過繪製每個點並且透過所有點來塑造出一條線的形狀。

(2) bres
- Bresenham's Algorithm 是用來描繪由兩點所決定的直線的演算法，它會算出一條線段在 n 維點陣圖上最接近的點。

(3) display
- GlClear 函式：清除緩衝區來設定預設值。
  - GlClear 參數：mask 遮罩的 OR 運算子，表示要清除的緩衝區。
- bres(200, 200, 100, 50)：設定兩點在(200,200), (100,50)。
- glFlush()：清空所有緩衝區。

(4) myinit
- glClearColor(1.0, 1.0, 1.0, 1.0)：改變顏色背景 (紅色、綠色、藍色、Alpha)
- glColor3f(1.0, 0.0, 0.0)：將色彩設為紅色
- glPointSize(1.0)：設定點大小為 1 的像素
- glMatrixMode(GL_PROJECTION)：宣告接下來要進行投影
- glLoadIdentity()：重置當前指定的矩陣為單位矩陣
- gluOrtho2D(0.0, 499.0, 0.0, 499.0)：通過正交投影把模型按照 1:1 的比例繪制到剪裁面上

(5) main
- glutInit(&argc,argv)：初始 GLUT Lib

- glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB)：創建窗口指定顯示模式的類型(單緩衝、顏色模式)
- glutInitWindowSize(500,500)：窗口的長寬
- glutInitWindowPosition(0,0)：窗口在螢幕的哪個位置
- glutCreateWindow("Bresenham's Algorithm")：窗口名稱
- glutDisplayFunc(display)：窗口所要反覆執行哪個程式
- myinit()：設定參數
- glutMainLoop()：進入迴圈

九、延伸應用程式碼：

相關檔案：Ch_08_tm4_src2.cpp

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

/* default data*/
/* can enter other values via command line arguments */

#define CENTERX -0.5
#define CENTERY 0.5
#define HEIGHT 0.5
#define WIDTH 0.5
#define MAX_ITER 100

/* N x M array to be generated */

#define N 500
#define M 500

float height = HEIGHT; /* size of window in complex plane */
float width = WIDTH;
float cx = CENTERX; /* center of window in complex plane */
float cy = CENTERY;
int max = MAX_ITER; /* number of interations per point */

int n=N;
int m=M;

/* use unsigned bytes for image */

GLubyte image[N][M];

/* complex data type and complex add, mult, and magnitude functions
  probably not worth overhead */
```

```
typedef float complex[2];  //declare type of complex

void add(complex a, complex b, complex p) //two complex add
{
   p[0]=a[0]+b[0];
   p[1]=a[1]+b[1];
}

void mult(complex a, complex b, complex p) //two complex mult
{
   p[0]=a[0]*b[0]-a[1]*b[1];
   p[1]=a[0]*b[1]+a[1]*b[0];
}

float mag2(complex a)
{
   return(a[0]*a[0]+a[1]*a[1]);
}

void form(float a, float b, complex p) //put data into a complex
{
   p[0]=a;
   p[1]=b;
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT); //clear the buffer of color
   glDrawPixels(n,m,GL_COLOR_INDEX, GL_UNSIGNED_BYTE, image); //draw a
 glFlush();
}


void myReshape(int w, int h)
{
   glViewport(0, 0, w, h);  //定義視窗矩形的大小位置
   glMatrixMode(GL_PROJECTION); //將當前矩陣指定為投影矩陣
   glLoadIdentity();   //目前的矩陣取代為識別矩陣
   if (w <= h)     //寬比高長
   gluOrtho2D(0.0, 0.0, (GLfloat) n, (GLfloat) m* (GLfloat) h /  //通過正交投影把模型按照 1:1 的比例繪制
到剪裁面上 參數:(左，右，上，下)
                  (GLfloat) w);
   else
   gluOrtho2D(0.0, 0.0, (GLfloat) n * (GLfloat) w / (GLfloat) h,
                  (GLfloat) m);
   glMatrixMode(GL_MODELVIEW);  //畫一個物體 A(看起來是 3D 的)
}
```

```
void myinit()
{
   float redmap[256], greenmap[256],bluemap[256];
   int i;

   glClearColor (1.0, 1.0, 1.0, 1.0);  //改變顏色背景 (紅色、綠色、藍色、Alpha)
   gluOrtho2D(0.0, 0.0, (GLfloat) n, (GLfloat) m);  //通過正交投影把模型按照 1:1 的比例繪制到剪裁面上
參數:(左，右，上，下)

/* define pseudocolor maps, ramps for red and blue,
  random for green */


   for(i=0;i<256;i++)  //使用公式去定義 redmap 與 bluemap 再用亂數去定義 greenmap
   {
      redmap[i]=i/255.;
      greenmap[i]=rand()%255;
      bluemap[i]=1.0-i/255.;
   }

   glPixelMapfv(GL_PIXEL_MAP_I_TO_R, 256, redmap);  //像素交換圖 map 顏色索引到紅色分量
   glPixelMapfv(GL_PIXEL_MAP_I_TO_G, 256, greenmap); //到綠色分量
   glPixelMapfv(GL_PIXEL_MAP_I_TO_B, 256, bluemap); //到藍色分量
}


main(int argc, char *argv[])
{
   int i, j, k;  //declare variable
   float x, y, v;
   complex c0, c, d;

/* uncomment to define your own parameters */

/*    scanf("%f", &cx); /* center x */
/*    scanf("%f", &cy); /* center y */
 /*   scanf("%f", &width); /* rectangle width */
 /*   height=width; /* rectangle height */
/*    scanf("%d",&max); /* maximum iterations */

   for (i=0; i<n; i++) for(j=0; j<m; j++)
   {

/* starting point */

   x= i *(width/(n-1)) + cx -width/2;    //produce the point of complex c0
```

```
    y= j *(height/(m-1)) + cy -height/2;

    form(0,0,c);        //put data into complex c,c0
    form(x,y,c0);



    for(k=0; k<max; k++)      //complex calculate and iteration
      {
      mult(c,c,d);
      add(d,c0,c);
      v=mag2(c);
      if(v>4.0) break; /* assume not in set if mag > 4 */
      }

/* assign gray level to point based on its magnitude */
      if(v>1.0) v=1.0; /* clamp if > 1 */
      image[i][j]=255*v;      //透過迭代來算出 image 中各 pixel 的值
    }



    glutInit(&argc, argv);      //initialize glut lib
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB ); //define mode
    glutInitWindowSize(N, M);     //define window
    glutCreateWindow("mandlebrot");
    myinit();
    glutReshapeFunc(myReshape);     //當窗口尺寸改變時，圖形比例不發生變化
    glutDisplayFunc(display);     //窗口所要反覆執行哪個程式

    glutMainLoop();       //進入迴圈
}
```

十、應用說明：

程式目的：曼德博集合是一種在複數平面上組成碎形的點的集合，利用 GlPixelMapfv 函式設定圖元傳輸對應，繪出色塊渲染畫面。

程式說明：

(1) Complex data type and complex add, mult, and magnitude functions probably not worth overhead
利用 add( ), mult( )進行複數的加法與乘法， mag2( ), form( )去組成一個複數。

(2) Display
- GlClear 函式：清除緩衝區來設定預設值。
  - GlClear 參數：mask 遮罩的 OR 運算子，表示要清除的緩衝區。
- glDrawPixels(n,m,GL_COLOR_INDEX, GL_UNSIGNED_BYTE, image)：將圖元區塊寫入至畫面格緩衝區。

- glDrawPixels 參數
  - width, height, format, type
  - GL_COLOR_INDEX：每個圖元都是單一值，也就是色彩索引。
  - GlDrawPixels 函式會將每個圖元轉換成固定點格式，而不會有二進位點右邊未指定的位數數目，而不論記憶體資料類型為何。 浮點值會轉換成真正的固定點值。 GlDrawPixels 函式會將帶正負號和不帶正負號的整數資料轉換成設定為零的所有小數位。 函數會將點陣圖資料轉換為 0.0 或 1.0。
  - GlDrawPixels 函式會將 GL_INDEX_SHIFT 位左方的每個固定點索引移至 GL_INDEX_OFFSET，並將它加入至。 如果 GL_INDEX_SHIFT 為負數，則向右移位。 無論是哪一種情況，在結果中都不會以零位填滿指定的位位置。
  - 在 RGBA 模式中， glDrawPixels 會使用 GL_PIXEL_MAP_I_TO_R、GL_PIXEL_MAP_I_TO_G、GL_PIXEL_MAP_I_TO_B 和 GL_PIXEL_MAP_I_TO_A 資料表，將產生的索引轉換成 RGBA 圖元。 當在色彩索引模式中，且 GL_MAP_COLOR 為 true 時，會將索引取代為查閱資料表中 glDrawPixels 參考的值 GL_PIXEL_MAP_I_TO_I。
  - 無論索引的查閱取代是否已完成，索引的整數部分都是 和 2b -1，其中 b 是色彩索引緩衝區中的位數。
  - 接著，會將目前的點陣位置 z 座標和材質座標附加至每個圖元，然後將 x 和 y 視窗座標指派給 n 個片段（例如 x），以將產生的索引或 RGBA 色彩轉換成片段： = xr + n mod 寬度 y？ = yr + n/寬度 其中 (xr ， yr) 是目前的點陣位置。
  - GlDrawPixels 函式會將這些圖元片段視為與透過將點、線條或多邊形所產生的片段一樣。 它會先套用材質對應、霧化和所有片段作業，然後再將片段寫入至畫面格緩衝區。
- GlFlush 函式：在有限的時間內強制執行 OpenGL 函數。

(3) myReshape
- glViewport：指定從標準化裝置座標到視窗座標之 x 和 y 的仿射轉換。 讓(xnd， ynd) 為標準化裝置座標。
- glMatrixMode(GL_PROJECTION)：設定目前的矩陣模式為投影。
- glLoadIdentity:會將目前的矩陣取代為識別矩陣。
- GluOrtho2D 函式：定義 2d 正向投射矩陣。

(4) myinit
- define pseudocolor maps, ramps for red and blue, random for green
- glPixelMapfv：會設定圖元傳輸對應。本程式分別建立地圖紅色, 綠色, 藍色元件的色彩索引。
  - glPixelMapfv 參數：map 符號對應名稱。

(5) main
- 設定畫面起始點
- 進行複數 iteration 運算
- 根據其幅度為點分配灰階程度
- OpenGL 中的 glutInitDisplayMode( )函數的作用主要是在創建窗口的時候，指定其顯示模式的類型。
- glutInitDisplayMode：在創建窗口的時候，指定其顯示模式的類型。
- glutInitWindowSize：確定窗口大小。
- glutCreateWindow：設定窗口名稱。
- glutReshapeFunc：設置窗口調整大小的函數。
- glutDisplayFunc：設置顯示函數

十一、其他範例程式：

(1) Ch_08_tm4_src3.cpp

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

GLfloat planes[]= {-1.0, 0.0, 1.0, 0.0};
GLfloat planet[]= {0.0, -1.0,  0.0, 1.0};
GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
   {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
   {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};
GLfloat colors[][4] = {{0.0,0.0,0.0,0.5},{1.0,0.0,0.0,0.5},
   {1.0,1.0,0.0,0.5}, {0.0,1.0,0.0,0.5}, {0.0,0.0,1.0,0.5},
   {1.0,0.0,1.0,0.5}, {1.0,1.0,1.0,0.5}, {0.0,1.0,1.0,0.5}};

void polygon(int a, int b, int c, int d)
{
   glBegin(GL_POLYGON);  //繪製一個多邊形
   glColor4fv(colors[a]);  //設定顏色 4 個參數(紅色、綠色、藍色、alpha 值)
   glTexCoord2f(0.0,0.0);  //配置紋理坐標
   glVertex3fv(vertices[a]); //配置圖形坐標，此兩個函式可以將圖形關聯至一個多邊形上，呈現真實視
覺效果
   glColor4fv(colors[b]);  //再設定顏色
   glTexCoord2f(0.0,1.0);
   glVertex3fv(vertices[b]);
   glColor4fv(colors[c]);
   glTexCoord2f(1.0,1.0);
   glVertex3fv(vertices[c]);
   glColor4fv(colors[d]);
   glTexCoord2f(1.0,0.0);
   glVertex3fv(vertices[d]);
   glEnd();
}

void colorcube()   //製造多邊形
{

/* map vertices to faces */

   polygon(0,3,2,1);
   polygon(2,3,7,6);
   polygon(0,4,7,3);
   polygon(1,2,6,5);
   polygon(4,5,6,7);
```

```
   polygon(0,1,5,4);
}

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

void display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //清除目前已啟用色彩與深度寫入的緩衝區
   glLoadIdentity();        //目前的矩陣取代為識別矩陣，拋棄之前的矩陣
   glRotatef(theta[0], 1.0, 0.0, 0.0);     //繞 x 軸旋轉
   glRotatef(theta[1], 0.0, 1.0, 0.0);     //繞 y 軸旋轉
   glRotatef(theta[2], 0.0, 0.0, 1.0);     //繞 z 軸旋轉
   colorcube();
   glutSwapBuffers();        //將後台的緩衝區跟前台交換
}

void spinCube()  //物體旋轉的公式
{
   theta[axis] += 2.0;
   if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
   glutPostRedisplay(); //每次 loop 循環時，當前視窗需要重新繪製
}

void mouse(int btn, int state, int x, int y)
{
   if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;  //當按右鍵時，cube 方向以 axis 0 改
變
   if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;  //當按左鍵時，cude 方向會以 axis
1 改變
   if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;  //當按下滾輪鍵，cude 會以 axis 2 方
向改變
}

void myReshape(int w, int h)
{
   glViewport(0, 0, w, h);    //定義視窗矩形的大小位置
   glMatrixMode(GL_PROJECTION);   //將當前矩陣指定為投影矩陣
   glLoadIdentity();     //目前的矩陣取代為識別矩陣
   if (w <= h)      //寬比高等於或較長
      glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,   //通過正交投影把模型按照 1:1 的比例繪制到剪裁
面上 參數:(左，右，上，下)
         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(unsigned char k, int x, int y)
{
   if(k == '1') glutIdleFunc(spinCube);  //按下"1"時 旋轉 cube
   if(k == '2') glutIdleFunc(NULL);   //按下"2"時 暫停 cube
   if(k == 'q') exit(0);    //按下"q"時 跳出程式
}

int main(int argc, char **argv)
{
  GLubyte image[64][64][3];    //宣告相關數值
  int i, j,  c;
  for(i=0;i<64;i++)
  {
    for(j=0;j<64;j++)                //將 image 陣列裡面的所有值畫成 8*8 方格棋盤(每個方個由 8*8 個像點組成)
    {
      c = ((((i&0x8)==0)^((j&0x8))==0))*255;
      image[i][j][0]= (GLubyte) c;
      image[i][j][1]= (GLubyte) c;
      image[i][j][2]= (GLubyte) c;
    }
  }
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH); //創建窗口指定顯示模式的類型(雙緩衝、顏色模式、深度模式)
   glutInitWindowSize(500, 500);   //窗手的長寬
   glutCreateWindow("colorcube");   //窗口名稱
   glutReshapeFunc(myReshape);    //當窗口尺寸改變時，圖形比例不發生變化
   glutDisplayFunc(display);
   glutIdleFunc(spinCube);    //設置不斷調用顯示函數
   glutMouseFunc(mouse);    //使用滑鼠 click 的函數 mouse
   glEnable(GL_DEPTH_TEST);    //啟用深度
   glEnable(GL_TEXTURE_2D);    //啟用 2 維紋理
   glTexImage2D(GL_TEXTURE_2D,0,3,64,64,0,GL_RGB,GL_UNSIGNED_BYTE, image); //指定的參數生成一個 2D 紋理 RGB 64*64 的大小
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);  //設置紋理相關參數(2 維、以 x 軸延伸、重複紋理)
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);  //(2 維、以 y 軸延伸、重複紋理)
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);  //(2 維、設置最大過濾、最接近的一個像素的顏色作為繪製像素的顏色)
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);  //(2 維、設置最小過濾、最接近的一個像素的顏色作為繪製像素的顏色)
```

```
  glutKeyboardFunc(key);    //使用鍵盤的函數 mouse
  glClearColor(1.0,1.0,1.0,1.0);   //清除目前已啟用色彩與深度寫入的緩衝區
  glutMainLoop();    //進入迴圈
}
```

程式目的：建立一個會旋轉並且具有棋盤紋理的立方體

程式說明：

(1)polygon : 立方體繪製，同時設定顏色參數

- glBegin(GL_POLYGON) :繪製一個多邊形
- glColor4fv(colors[a]) : 設定顏色 4 個參數(紅色、綠色、藍色、alpha 值)
- glTexCoord2f(0.0,0.0) : 配置紋理坐標
- glVertex3fv(vertices[a]) : 配置圖形坐標，此兩個函式可以將圖形關聯至一個多邊形上，

呈現真實視覺效果

- glColor4fv(colors[b]) : 設定顏色

(2)key : 控制立方體旋轉或停止

- glutIdleFunc(spinCube) ： 旋轉 cube
- glutIdleFunc(NULL) ：暫停 cube

(3)colorcube：製造多個多邊形代入函式 polygon 提供多邊形切換功能

(4)spinCube：設定立方體旋轉，改變 theta 角度

- glutPostRedisplay()：每次 loop 循環時，當前視窗需要重新繪製

(5)display:

- glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //清除目前已啟用色彩與深度寫入的緩衝區
- glLoadIdentity()：目前的矩陣取代為識別矩陣，拋棄之前的矩陣
- glRotatef() : 將目前的矩陣乘以旋轉矩陣。
  參數： 旋轉的角度（以度為單位）。
  x：向量的 x 座標。
  Y：向量的 y 座標。
  Z：向量的 y 座標。

(6)main

- 首先設定起始窗口畫面，設置立方體 display 函數
- glEnable(GL_DEPTH_TEST) : 啟用更新深度緩衝區的功能
- glEnable(GL_TEXTURE_2D); //啟用 2 維紋理映像
- glTexImage2D(GL_TEXTURE_2D,0,3,64,64,0,GL_RGB,GL_UNSIGNED_BYTE, image); //指定的參數生成一個 2D 紋理 RGB 64*64 的大小
- glTexParameterf(int target, int pname, float param)：(目標紋理、設置紋理映射過程中的映射像素問題、pname 值)
- glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT)：設置紋理相關參數(2 維、以 x 軸延伸、重複紋理)
- glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT)：(2 維、以 y 軸延伸、重複紋理)
- glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST); //(2 維、設置最大過

濾、最接近的一個像素的顏色作為繪製像素的顏色)

- glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST); //(2 維、設置最小過濾、最接近的一個像素的顏色作為繪製像素的顏色)

十二、注意事項：

建立一個渲染色塊需要先注意每個圖層疊加的順序，以及迭代的方法，迭代主要目的是為了求出每個點的 pixel 值，因此在迴圈的設定上就更重要，若設定錯誤會影響整個顏色的走向。

十三、參考資料：

(1) gltexparameter：http://blog.sina.com.cn/s/blog_e77c5f870102x0kf.html

(2) CS 352: Computer Graphics Chapter8 : DiscreteTechniques: https://studylib.net/doc/11251837/discrete-techniques-cs-352--computer-graphics-chapter-8-

(3)[openGL]Texture 函式參數：https://cg2010studio.com/2011/09/05/opengl-texture-%E5%87%BD%E5%BC%8F%E5%8F%83%E6%95%B8/