COMP3271/CSIS0271
Computer Graphics
Assignment 1

Due date: Feb. 12, 2015 (11:30 pm)

1 Mandelbrot and Julia Sets

Definitions

The 2D fractals to be rendered are structures in the complex plane defined by the Mandelbrot and Julia sets, which are associated with the mapping $z \to z^2 + c$, where $z = x + iy$ and constant $c = a + ib$. The Mandelbrot set is defined in the complex plane of parameter c by

$$M = \left\{ c | z_n \not\to \infty, z_n = z_{n-1}^2 + c, z_0 = 0 \right\}.$$

For every complex c there corresponds a Julia set J_c defined in the complex plane of initial value $z_0 = z$ as the boundary of

$$A_c = \left\{ z | z_n \not\to \infty, z_n = z_{n-1}^2 + c, z_0 = z \right\}.$$

Rendering procedures

Suppose that the window for display is $[0, p_{\max}] \times [0, q_{\max}]$ in pixels, and there are K colors at your disposal, indexed from 0 to $K-1$. Let $R > 0$ be a parameter chosen appropriately.

For rendering the Mandelbrot set, let $[a_{\min}, a_{\max}] \times [b_{\min}, b_{\max}]$ be the area to be displayed in the complex plane of $c = a + ib$. (We recommended $[a_{\min}, a_{\max}] \times [b_{\min}; b_{\max}] = [-2.25, 0.75] \times [-1.5, 1.5]$.)

Algorithm 1 Mandelbrot Set Rendering

```
1: procedure MANDELBROT_SET
2:     for all pixels (p,q) in the window do
3:         $\Delta a = (a_{\max} - a_{\min})/p_{\max}, \Delta b = (b_{\max} - b_{\min})/q_{\max}$
4:         $a = a_{\min} + p\Delta a, b = b_{\min} + q\Delta b$
5:         $x_0 \leftarrow 0, y_0 \leftarrow 0, n \leftarrow 1$
6:         $x_n \leftarrow x_{n-1}^2 - y_{n-1}^2 + a, y_n \leftarrow 2x_{n-1}y_{n-1} + b$
7:         if n = K then
8:             color pixel (p, q) with color 0
9:         else
10:            if ($x_n^2 + y_n^2 < R$) then
11:                $n \leftarrow n + 1$
12:                go to 6
13:            else
14:                color pixel (p, q) with color n
```

For rendering the Julia set, let $[x_{\min}, x_{\max}] \times [y_{\min}; y_{\max}]$ be the area to be displayed in the complex plane of $z_0 = z$. (We recommended $[x_{\min}, x_{\max}] \times [y_{\min}; y_{\max}] = [-2.0, 2.0] \times [-2.0, 2.0]$.)

Algorithm 2 Julia Set Rendering (with $c = a + ib$)

```
procedure JULIA_SET
    for all pixels (p,q) in the window do
        $\Delta x = (x_{\max} - x_{\min})/p_{\max}, \Delta y = (y_{\max} - y_{\min})/q_{\max}$
        $x_0 \leftarrow x_{\min} + p\Delta x, y_0 \leftarrow y_{\min} + q\Delta y, n \leftarrow 1$
        $x_n \leftarrow x_{n-1}^2 - y_{n-1}^2 + a, y_n \leftarrow 2x_{n-1}y_{n-1} + b$
        if n = K then
            color pixel (p, q) with color 0
        else
            if $(x_n^2 + y_n^2 < R)$ then
                $n \leftarrow n + 1$
                go to 5
            else
                color pixel (p, q) with color n
```

Requirements

In this section, you will use a window to display both the Mandelbrot set and Julia set, using colortable provided by the template (or any other color style you like). Interactive techniques, such as magnification, minimization, mouse guided translation are provided by the template. To finish this assignment, you are required to fill in two functions to calculate Mandabrot/Julia set.

Observe the differences among Julia sets when c is chosen at different positions (inside, outside, or on the boundary of the Mandelbrot set). You may assign the color indices carefully and adjust other parameters, such as K and R, to produce good results. You are also encouraged to experiment with rendering fractals using other iterative formulas. (However, please be sure to include your implementation of the original formulas provided here in your submitted code.)

Functions to implement

Your task is to implement the following functions whose interfaces are given in **cg_assignment_1.cpp**

void **setMandabrotSet(const int image_width, const int image_height, const complex_t& bottom_left, const complex_t& top_right, colorlist_t& color_list, image_t& image)**

INPUT: texture image size, window area on the complex plane, color table
OUTPUT: image data colored according to the calculated Mandabrot set.

void **setJuliaSet(const int image_width, const int image_height, const complex_t& bottom_left, const complex_t& top_right, const complex_t& c colorlist_t& color_list, image_t& image)**

INPUT: texture image size, window area on complex plane, c, color table
OUTPUT: Julia set image within selected window area and complex number c

(OPTIONAL) void **setupColorTable(colorlist_t& color_list)**

INPUT: NONE

OUTPUT: color list used to visualize the Mandabrot/Julia set, also used in the next section
TIPS: check `http://www.ncl.ucar.edu/Document/Graphics/color_table_gallery.shtml`

2 Recursive Fractals and OpenGL Transformation

Requirements

In this section, you will make use of the OpenGL ModelView matrix stack to render fractals (different from Section 1) recursively.

Specifically, you will

- Form triangles from interactive mouse input,
- Draw these triangles using OpenGL functions,
- Generate fractals using transformation matrices created from the triangle inputs.

A window should be used to display the fractals. Basic interface functions are provided by the template code. You need to fill in 2 functions to complete the assignment. Observe the different fractals created by different sets of input triangles and color combinations. Try to create images from different arrangements of input triangles and hand them in with the source.

Algorithm 3 Draw Recursive Fractal

```
procedure RECURSIVE_FRACTAL(k)
    if $k > 0$ then
        for each transformation $M_i$ do
            push current OpenGL modelview matrix onto stack
            multiply current modelview matrix by $M_i$
            Recursive_Fractal(k-1)
            pop matrix from stack
    else
        draw triangle $T(P_1, P_2, P_3)$ with current modelview matrix
```

Functions to implement

Your task is to implement the following functions whose interfaces are given in **cg_assignment_1.cpp**.

trans_matrix_t **calcTransformationMatrix(const triangle_t& tri1, const triangle_t& tri2)**

INPUT: two triangles tri1 and tri2
OUTPUT: transformation matrix that transforms tri1 to tri2

void **drawIfsTriangles(objectlist_t& triangle_list, const int k)**

INPUT: triangle list, recursion depth k. each element of the triangle list stores a triangle data, its transformation matrix and color.
OUTPUT: recursively draw triangles using OpenGL

3 Instructions on the Template Code

Project building

We provide project files for Visual Studio 2010/2012, Xcode 6. They were tested on Mac OS X 10.9 and Windows 8. Feel free to contact us if you have problems building on Linux.

User interaction

There are some hot keys for using this template program:

- Press ENTER to switch between three modes of this assignment: MANDABROT, JULIA and IFS_TRIANGLES.
- In MANDABROT and JULIA mode, use left mouse dragging/arrow keys to move the scene, use $+$ / $-$ for magnification/minimization.
- In IFS_TRIANGLES mode, left-click the area within the window to add a point, everytime three new points are clicked, a new triangle made up of these three points will be added into the object list for rendering.

4 What to Submit

Please submit cg_assignmen_1.cpp to Moodle by 11:30 pm on Feb. 15th. Any uploads or modifications of the file after the deadline will be considered a late submission.