Graphic Interface Part 2

Introduction

In this assignment, you will add functionality to the graphical user interface, allowing the user to control many aspects of the display. Such as:

- switch between Julia set, Mandelbrot set, and ComplexFractal images,
- zoom in or out,
- move left, right, up, or down,
- reset the plane coordinates,
- increase or decrease the maximum escape count,
- select \(a\) and \(b\) values for the Julia set,
- increase or decrease the size of the color table,
- view the output image or the color table.

Assignment

Build on the code from the previous assignments. This assignment should only require changes to the code in the `gui-src` tree.

The program will have two interaction modes:

1- Color table display mode. The display will show the color table from left to right. The first color will be on the left, and the last color will be on the right. The space between the left and right be filled with the other colors.
The color table is always a single gradient. The controlling factors are:

- Color table size
- First color
- Second color

Whenever any of these values change, the color table will be recalculated.

2- Fractal interaction mode. The display will be showing the currently configured and calculated output image. The controlling factors are:

- Fractal mode: Mandelbrot, Julia, ComplexFractal
- Window size: the height and width
- Maximum escape value: the grid’s maximum number
- Julia parameters: a, b if the fractal mode is Julia
- Complex plane boundaries: x_min, x_max, y_min, y_max
- Color table

Whenever any of these values change, the output image will be recalculated.

The following user actions will be supported:

- ‘T’ changes the interaction mode to color table
- ‘t’ changes the interaction mode to fractal
- ‘>’ or ‘.’ increases the color table size
- ‘<’ or ‘,’ decreases the color table size
- ‘b’ changes the fractal mode to Mandelbrot
- ‘n’ changes the fractal mode to Julia
- ‘F’ changes the fractal mode to ComplexFractal
- mouse click sets the Julia parameters a and b, but only if fractal mode is Mandelbrot
- ‘z’ changes the complex plane boundaries so that a smaller area of the plane is viewed
Programming Requirements

GlutApp.h, cpp

Update the GlutApp class.

Enumerations:

- enum InteractionMode { IM_FRACTAL, IM_COLORTABLE }; Used for controlling the |display()| method.
- enum FractalMode { M_MANDELBROT, M_JULIA, M_COMPLEX }; Used to control the |createFractal()| method.

New data members:

- double mMinX, mMaxX, mMinY, mMaxY; Used to call |fractalPlaneSize()| in |createFractal()|.
- double mA, mB; Used to call |juliaParameters()| in |createFractal()|.
- InteractionMode mInteractionMode; Used to choose display mode in |display()|.
- FractalMode mFractalMode; Used to choose the fractal type in |createFractal()|.
- int mMaxNumber; Used to call |configureGrid()| in |createFractal()|.
- Color mColor1; Used to configure the gradient in |setColorTable()|.
- Color mColor2; Used to configure the gradient in |setColorTable()|.
- int mNumColor; Used to configure the color table size in |setColorTable()|.

Update the following methods:

- GlutApp(int height, int width); initialize: mMinX and mMinY to [-2.0], mMaxX and mMaxY to 2.0; mInteractionMode to |IM_FRACTAL|; mFractalMode to |M_MANDELBROT|; mMaxNumber to 200; mColor1 and mColor2 to colors of your choice; mNumColor to 32. In the body call |setColorTable()| and |createFractal()| to setup the color table and initial image display.
- void display(); If the interaction mode is |IM_FRACTAL|, display the output image (which is the existing code in this method), else if the interaction mode is |IM_COLORTABLE|, then call |displayColorTable()|.

Add the following methods:

- void displayColorTable(); For each row in the display, do the same thing. For each column in the display: calculate the index into the color table using: |i = column * color_table_size / width_of_display|. Use the |i|th color from |mActionData|’s color table. Prepare each color channel (red, green, blue) for OpenGL by dividing by 255.0, then use |gColor3d(red,green,blue)| to set the color. Finally, draw the screen pixel using |glVertex2i(column, row)|. Repeat this process for every pixel in the display.
- void setInteractionMode(InteractionMode mode); assign to the correct data member.
- void setColorTable(); Use |set-color-table-size| and |set-color-gradient| to configure the color table using |mNumColor|, |mColor1|, and |mColor2|.
- void decreaseColorTableSize(); If the number of colors is more than 10, decrease the number of colors by dividing by 1.1. Then use |setColorTable()| and |gridApplyColorTable()| to update the output image.
- void increaseColorTableSize(); If the number of colors is less than 1024, increase the number of colors by multiplying by 1.1. Then use |setColorTable()| and |gridApplyColorTable()| to update the output image.
- void zoomIn(); Decrease the size of the view window to |0.9| the size. Calculate |dx| as |(1.0 - 0.9) * (mMaxX - mMinX) / 2.0|. Add |dx| to |mMinX| and subtract it from |mMaxX|. Do similar for the y dimension.
- void zoomOut(); Increase the size of the view window to |1.1| the size. Calculate |dx| as |(1.0 - 0.9) * (mMinX - mMaxX) / 2.0|. Subtract |dx| from |mMinX| and add it to |mMaxX|. Do similar for the y dimension. Only do this zoom operation if it will not cause any of the plane values to go past -2.0 or 2.0.
- void moveLeft(); Move the view port to the left by the fraction 0.05. Calculate |dx| as |(1.0 - 0.9) * (mMaxX-mMinX) / 2.0|. If |mMinX - dx| is at least -2.0, then subtract |dx| from |mMinX| and |mMaxX|.
- void moveRight(); Move the view port to the right by the fraction 0.05, similar to |moveLeft()|, except add to |mMinX| and |mMaxX|.
- void moveDown(); Like |moveLeft()|, but for the y dimension.
- void moveUp(); Like |moveRight()|, but for the y dimension.
- void setFractalMode(FractalMode mode); Change the value of |mFractalMode|.
- void increaseMaxNumber(); If the |mMaxNumber| is less than 2048, increase it by multiplying by 1.1.
- void decreaseMaxNumber(); If the |mMaxNumber| is greater than 10, decrease it by dividing by 1.1.
**void setAB(int x, int y);** If the `mFractalMode` is `M_MANDELBROT`, and the `mActionData` grid is a `ComplexFractal`, then set `mA` to `mMinX + x * delta_x`, and similar for `b` and `y`. `delta_x` is obtained from the dynamically cast `ComplexFractal` pointer with `getDeltaX()`.

**void resetPlane();** Sets `xMinX` and the other three data members to `-2.0` or `2.0` to make the default square.

**void createFractal();** Uses `mFractalMode` to choose whether to `selectMandelbrot()`, `selectJulia()` or `selectComplexFractal()`. For Julia, also calls `juliaParameters()` as well. Calls `configureGrid()`, `fractalPlaneSize()`, `fractalCalculate()`, and `gridApplyColorTable()` to calculate the output image. Uses data members for parameters to these functions.

**glut_callback.cpp**

Update the following functions.

- **void keyboard_cb(unsigned char c, int x, int y);** Add cases to cover the key presses listed above. (All but the arrow keys).
- **void special_cb(unsigned char c, int x, int y);** Add cases to cover the key presses listed above. (The arrow keys).
- **void mouse_cb(int mouse_button, int state, int x, int y);** For the left button down, call `setAB()` passing `xdisplay` and `ydisplay`.

<table>
<thead>
<tr>
<th>Key/Action</th>
<th>Method(s) called</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>setInteractionMode()</td>
</tr>
<tr>
<td>'t'</td>
<td>setInteractionMode()</td>
</tr>
<tr>
<td>'&gt;' or ','</td>
<td>increaseColorTableSize()</td>
</tr>
<tr>
<td>'&lt;' or ','</td>
<td>decreaseColorTableSize()</td>
</tr>
<tr>
<td>'b'</td>
<td><code>setFractalMode()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>'n'</td>
<td><code>setFractalMode()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>'F'</td>
<td><code>setFractalMode()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>left click</td>
<td><code>setAB()</code></td>
</tr>
<tr>
<td>'Z'</td>
<td><code>zoomIn()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>'Z'</td>
<td><code>zoomOut()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>left arrow</td>
<td><code>moveLeft()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>right arrow</td>
<td><code>moveRight()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>down arrow</td>
<td><code>moveDown()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>up arrow</td>
<td><code>moveUp()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>'R'</td>
<td><code>resetPlane()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>'+' or '='</td>
<td><code>increaseMaxNumber()</code>, <code>createFractal()</code></td>
</tr>
<tr>
<td>','</td>
<td><code>decreaseMaxNumber()</code>, <code>createFractal()</code></td>
</tr>
</tbody>
</table>

**Update src/Makefile**

No changes here: The following commands should work correctly.

- **make hello** - builds the hello program
- **make questions_3** - builds the questions 3 program
- **make ascii_image** - builds the ascii image program
- **make image_file** - builds the image file program
- **make ppm_menu** - builds the ppm_menu program
- **make all** - builds all programs
- **make** - builds all programs (same as **make all**)
- **make clean** - removes all .o files, and all executable programs

**Update gui-src/Makefile**

Should be able to use the file as is.

- **make glut_main** - builds the application.
- **make clean** - removes all .o files, and all executable programs

**Additional Documentation**

- **C++ Reference**
Show Off Your Work

To receive credit for this assignment, you must

- complete the unit tests available in CodeGrinder (if available)
- use git to add, commit and push your solution to your repository for this class.

Additionally, the program must build, run and give correct output.

Extra Challenges (Not Required)

- Allow the user to design their own color tables.
- Other interesting user interactions.