CS 3005: Programming in C++

Open GL Interface

Assignment

In this assignment, you will add an Open GL interface to the software project you’ve been building. When completed, the `ppm_menu` program will still work the same as before, and you will have a new program `glut_main` that will give a graphical interface for users to browse the Julia and Mandelbrot sets and save images of them.

This assignment has the following steps:

- Add a method to the `ComplexFractal` class.
- Create the `MyApp` class.
- Copy `app.cpp`, `gl_draw.cpp`, `glut_app.h`, `glut_callback.cpp`, and `glut_main.cpp` from the class example.
- Make small modifications to the previous files.
- Update the `Makefile`.
- Test.

Programming Requirements

**ComplexFractal** class

```cpp
void setPlaneSizeNoLimits(const double& min_x, const double& max_x, const double& min_y, const double& max_y);```

Add this method to the class. It has the same behavior as `setPlaneSize`, except that it does not force the plane coordinates to be between -2.0 and 2.0.

**MyApp** class

Create this class in `MyApp.h` and `MyApp.cpp`. This class will control the graphical interface program much like the `image_menu.cpp` functions control the operations for `ppm_menu`. The data members used in this class are to keep information that the user interface needs as it switches between display modes and fractal types.

The class declaration should look like this:

```cpp
class MyApp {
public:
    enum Mode { MODE_DISPLAY_OUTPUT_IMAGE, MODE_JULIA_PARAMETERS };

    MyApp(int height, int width);
    int getHeight() const;
    int getWidth() const;
    void setSize(int height, int width);
    int getMaxNumber() const;
    void setMaxNumber(int max_number);
    void setFractalPlaneSize(double min_x, double max_x, double min_y, double max_y);
    void fixFractalPlaneSize();
    void calculateFractal();
    void applyColorTable();
    PPM& getOutputImage();
    NumberGrid& getGrid();
    void setGrid(NumberGrid *grid);
    ColorTable& getTable();
    PPM& getDisplayPPM();
    void setCenter(int x, int y);
    void setJuliaParameters(int x, int y);
    void zoomIn();
    void zoomOut();
    void julia();
    void mandelbrot();
    void setMode(Mode mode);
    Mode getMode() const;
    void writePPMFile();
    void setColorTable(int which);

protected:
```
The methods have these behaviors:

- MyApp(int height, int width); Initializes the data members as follows: mHeight to height and mWidth to width, mMaxNumber to 300, mMinX to -2.0, mMaxX to 2.0, mMinY to -2.0, mMaxY to 2.0, mA to 0.8, mB to 0.156, mZoomFactor to 0.9, mGrid to 0, mTable to size 16, mMode to MODE_DISPLAY_OUTPUT_IMAGE, mFileNumber to 1, and mWhichColor to 0.

Also inserts a gradient into the color table, the same as the ImageMenu() function did; sets mJuliaParameterGrid to a heap allocated MandelbrotSet with height and width from the parameters, and plane boundaries set to the [-2.0, 2.0] range; sets the mJuliaParameterGrid maximum number to 100, tells it to calculate all numbers, and calls setPPM on it to set mJuliaParameterImage, using mTable; Additionally uses setGrid to set a new MandelbrotSet with height, width and plane parameters set from the data members; finally calls setMaxNumber and setSize with the data member values as parameters.

All of this work is necessary to set up the initial default display.

- int getHeight() const; Returns the data member.

- void setSize(int height, int width); If height and width are both at least 2, updates the data members, calls setGridSize on mGrid, calls fixFractalPlaneSize, calculateFractal, and applyColorTable. Additionally calls setGridSize, calculateAllNumbers, and setPPM on mJuliaParameterGrid.

- int getMaxNumber() const; Returns the data member.

- void setMaxNumber(int max_number); If max_number is positive, sets the data member, calls setMaxNumber on mGrid, and calls setColorTable with mWhichColor as the parameter.

- void setFractalPlaneSize(double min_x, double max_x, double min_y, double max_y); If mGrid can be dynamic_cast to ComplexFractal*, call setPlaneSizeNoLimits on the cast pointer with this function's parameters as arguments, then call fixFractalPlaneSize, calculateFractal, and applyColorTable.

- void fixFractalPlaneSize(); If mGrid can be dynamic_cast to ComplexFractal*, Get the min/max/delta x/y values from the cast pointer. If mWidth is bigger than mHeight, calculate mid_y as the average of min_y and max_y. Update min_y to be mid_y - dx*mHeight/2.0, max_y is updated the same, but with a + instead of -. If mWidth is not greater than mHeight, instead calculate mid_x as the average of min_x and max_x, then update [min_x] and [max_x] from similar formulas. Note that the y values are updated using [dx] (the delta x value), and x values are updated using the [dy] value. Also note that mHeight is used for the y value calculation and mWidth is used for the x value calculation. Finally, call setPlaneSizeNoLimits on the cast pointer, using the min/max x/y values, two of which should have been updated, and the other two should not have been changed.

- void calculateFractal(); Calls calculateAllNumbers on mGrid.

- void applyColorTable(); Calls setPPM on mGrid, passing mOutputImage and mTable as parameters.

- PPM& getOutputImage(); Returns the data member.

- NumberGrid& getGrid(); Returns mGrid.

- void setGrid(NumberGrid *grid); Much like ActionData::setGrid. If mGrid is not 0, delete it. Then sets mGrid to the parameter.

- ColorTable& getTable(); Returns the data member.

- PPM& getDisplayPPM(); If mMode is MODE_DISPLAY_OUTPUT_IMAGE, returns mOutputImage, if it is MODE_JULIA_PARAMETERS, returns mJuliaParameterImage.

- void setCenter(int x, int y); If mGrid can be dynamic_cast to ComplexFractal*, The x and y parameters are the column and row of the image clicked by the user. To calculate the new plane size values: calculate the column and row shifts to center with [x-mGrid->getWidth()/2] and similar for the y coordinate. To find the shift distance in the plane, multiply these results by delta x and delta y, respectively. These can be fetched from the cast pointer with getDeltaX() and getDeltaY(). The new minimum x value is mMinX + dx. Similar values can be found for maximum x, and min/max y. All use addition. Call setPlaneSizeNoLimits on the cast pointer, using the min/max x/y values calculated. Finally, calls [fixFractalPlaneSize], calculateFractal, and applyColorTable.

- void setJuliaParameters(int x, int y); x and y are the column and row clicked by the user in the image. If mJuliaParameterGrid can be dynamic_cast to ComplexFractal*, call calculatePlaneCoordinatesFromPixelCoordinates on the cast pointer, passing y and x as row and column
parameters, and \( m_A \) and \( m_B \) as the location parameters. If \( mGrid \) can be \texttt{dynamic\_cast} to \texttt{JuliaSet*}, call \texttt{setParameters} on it with \( m_A \) and \( m_B \) as parameters. Also call \texttt{calculateFractal}, and \texttt{applyColorTable}. Finally if \( mMode \) is \texttt{MODE\_JULIA\_PARAMETERS}, then call \texttt{setMode} to set it to \texttt{MODE\_DISPLAY\_OUTPUT\_IMAGE}.

- \texttt{void zoomIn()}; Like other methods, \texttt{dynamic\_cast} \( mGrid \) to a \texttt{ComplexFractal*}. If successful, calculate new values for \( mMaxX \), \( mMinX \), etc. The formulas can be found in the notes listed below. The formulas for \( y \) are the same as for \( x \). Call \texttt{setPlaneSizeNoLimits} on the cast pointer, with the newly calculated limits as parameters, then call \texttt{fixFractalPlaneSize}, \texttt{calculateFractal}, and \texttt{applyColorTable}.

- \texttt{void zoomOut()}; This looks the same as \texttt{zoomIn}, but using the other formulas.

- \texttt{void julia()}; Calls \texttt{setGrid} with a new \texttt{JuliaSet} initialized with \( mHeight \), \( mWidth \), \( mMinX \), ... Calls \texttt{setMaxNumber} and \texttt{setSize}, using the data members as parameters.

- \texttt{void mandelbrot()}; Calls \texttt{setGrid} with a new \texttt{MandelbrotSet} initialized with \( mHeight \), \( mWidth \), \( mMinX \), ... Calls \texttt{setMaxNumber} and \texttt{setSize}, using the data members as parameters.

- \texttt{void setMode(Mode mode);} Assigns the parameter to \( mMode \).

- \texttt{Mode getMode() const;}; Returns the data member.

- \texttt{void writePPMFile();}; Constructs a filename using \texttt{std::stringstream} \texttt{gui\_image\_} and \texttt{mFileNumber}. Writes \texttt{mOutputImage} to the file, then increments \texttt{mFileNumber}.

- \texttt{void setColorTable(int which);} Sets the data member to the parameter. If the value is 0, sets the color table to the one configured in the constructor. If the value is 1, 2, or 3, configure a different color table. You choose what these 3 extra color tables will look like. Make them interesting as possible. Finally, calls \texttt{applyColorTable}.

\texttt{app.cpp}

- Uncomment the global variable declaration for \( g\_app\_data \).
- Add parameters \texttt{int height, int width to initialize\_app}.
- Initialize the global parameter in \texttt{initialize\_app} to a new \texttt{MyApp} with the height and width parameters.

\texttt{glut\_app.h}

- Remove \texttt{g\_screen\_x} and \texttt{g\_screen\_y}.
- Uncomment the \texttt{extern} declaration of \texttt{g\_app\_data}.
- Include \texttt{MyApp.h}.
- Add the parameters to \texttt{initialize\_app}.

\texttt{glut\_callback.cpp}

- Remove all of the \texttt{g\_*} variables.
- In \texttt{keyboard}, keep the \texttt{q, 27} case to exit. Remove all other cases. Add cases for \texttt{z to g\_app\_data->zoomIn(), Z to zoom out, j to julia, m to mandelbrot, J to set the mode to MODE\_JULIA\_PARAMETERS, w to writePPMFile, 0 1 2 3 to call setColorTable with the correct number.}
- In \texttt{reshape}, remove the \texttt{g\_screen\_x, g\_screen\_y} uses, instead call \texttt{setSize} on \texttt{g\_app\_data} with \texttt{h} and \texttt{w} as parameters.
- In \texttt{mouse}, replace \texttt{g\_screen\_y} with a call to \texttt{g\_app\_data->getHeight()}. On the left down click, if the mode is to display the output image, call \texttt{setCenter}, if it’s julia parameters, call \texttt{setJuliaParameters}.
- In \texttt{display}, replace the drawing code with that shown below.

The \texttt{display} function code.

\begin{verbatim}
gClear(GL\_COLOR\_BUFFER\_BIT);
PPM &p = g\_app\_data->getDisplayPPM();

double max = p.getMaxColorValue();
double r, g, b;
int row, column;
gBegin( GL\_POINTS );
for(row = 0; row < p.getHeight(); row++) {
   for(column = 0; column < p.getWidth(); column++) {
      r = p.getChannel(row, column, 0) / max;
      g = p.getChannel(row, column, 1) / max;
      b = p.getChannel(row, column, 2) / max;
      glColor3d(r, g, b);
      glVertex2i(column, p.getHeight()-row-1);
   }
}
gEnd( );
gluSwapBuffers();
\end{verbatim}

\texttt{glut\_main.cpp}
• Remove `g_screen_x` and `g_screen_y` declarations.
• Move the `initialize_app` call to the first thing in `main()` with 500 and 700 as the height and width.
• For `glutInitWindowSize`, replace `g_screen_x`, `g_screen_y` with calls to `getWidth` and `getHeight` on `g_app_data`.

**Makefile**

Consider using the [Makefile](#) provided to build both `ppm_menu` and `glut_main`.

**Additional Documentation**

- [C++ Reference](#)
- [Examples from class](#)
- [Zoom In Formulas](#)
- [Zoom Out Formulas](#)
- [Debugging Display Problems](#)

**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.
- Allow the user to change the maximum number (max escape count).
- Allow the user to move the center point with the arrow keys.
- Other interesting user interactions.