CS 3005: Programming in C++

Complex Fractal Plane

Introduction

In a previous assignment, we created a number grid, that allows the user to store integer numbers at each location in a rectangular, regular grid. We then added the a color table that allows the user to assign a color to each of the integers stored in the grid, allowing a way for users to create images from numbers.

The next series of assignments will add ways for users to insert numbers into the number grid using algorithms. Most of these algorithms work better for floating point numbers, instead of integer numbers.

In this assignment, we'll add a translation between continuous number plane (x,y) coordinates and discrete grid (row, column) coordinates.

Then, we'll create a simple algorithm to calculate numbers using the number plane, and store the numbers in the grid.

Mapping between a regular grid and a number plane

Our end goal is to have a rectangular image with pixel colors assigned based on a mathematical function. The function will take plane coordinates (x,y) as input, and return a number for the number grid. That number will be used to lookup a color in the color table, to set a pixel color in an image.

We make the grid with dimensions of `height` by `width`. The rows of the grid are numbered 0 through `height-1` and the columns are numbered 0 through `width-1`. For each of these grid locations, we want to calculate a point `x,y` that represents the center of the pixel in the number plane.

If two grid locations are horizontal neighbors (for example, column differs by 1, and row is the same), then the plane coordinates of the two points will differ by a fixed amount in the `x` dimension. Similarly, if two grid locations are vertical neighbors (for example, row differs by 1, and column is the same), then the plane coordinates of the two points will differ by a fixed amount in the `y` dimension.

Depending on the mathematical function we use to calculate numbers for our grid, we may want the part of the number plane that is mapped to our grid to be different. We will choose minimum and maximum values for x and y that correspond to the centers of the left and right columns, and the bottom and top rows. (Minimum x is left; maximum x is right; minimum y is bottom; maximum y is top.)

For example, we may want to have our image look at the region of the plane from -1.5 to 0.5 in the x axis and -0.3 to 1.2 in the y axis. For this example, we'll make a grid that is 4 locations wide and 3 locations high. (Very small, but a good example of the calculations.)

Let's calculate the x coordinates of our image. First, the left most column (0) will have `x = -1.5`, because that is the left side of the region of the image. Next, the right most column (3) will have `x = 0.5`, the right side of the region. What are the x coordinates of columns 1 and 2? They need to be regularly spaced. There are `width-1 = 3` gaps between our columns. Draw a picture to convince yourself that `width-1` is the correct number of gaps. We will use the term `delta` to indication the change in a coordinate over a gap.

Since there are 3 gaps to travel from -1.5 to 0.5, how big is each gap? The generic formula for this is `delta_x = (max_x - min_x) / (width - 1)`.

```
delta_x = ( 0.5 - ( -1.5 ) ) / ( 4 - 1 ) = 2.0 / 3 = 0.666666
```

With the delta between columns calculated, we can easily calculate the x coordinate for any column. The generic formula for this calculation is `x = min_x + column * delta_x`.

```
x_column_0 = min_x + 0 * delta_x = -1.5
x_column_1 = min_x + 1 * delta_x = -0.833333
x_column_2 = min_x + 2 * delta_x = -0.166666
x_column_3 = min_x + 3 * delta_x =  0.5
```

Using these two formulas, we can calculate the x coordinate of any column.

A similar discussion for calculating the y coordinate of any row yields these results.

```
delta_y = ( 1.2 - ( -0.3 ) ) / ( 3 - 1 ) = 1.5 / 2 = 0.75
y_row_0 = max_y - 0 * delta_y =  1.2
y_row_1 = max_y - 1 * delta_y =  0.45
y_row_2 = max_y - 2 * delta_y = -0.3
```

Now, for every grid location, we can calculate the point in the plane (x0,y0), from the (column,row) of the pixel. For each grid location, these values are used in our mathematical function to calculate a number to store in the grid.

Assignment

In this assignment you will create a class to map between number plane and grid coordinates. It will inherit from `NumberGrid` for the height, width, max value, and number storage.

You will also extend the `ppm_menu` program to add a few new commands.

The new commands required are:

- `fractal-plane-size`: Set the dimensions of the grid in the complex plane.
- `fractal-calculate`: Calculate the escape values for the fractal.

Programming Requirements

The following files must be updated or created and stored in the `src` directory of your repository.

Updates to `NumberGrid.{h,cpp}`

Updated Methods:

- `virtual void setGridSize(const int& height, const int& width);` No changes in the implementation. Added `virtual` in declaration.

Additional Methods:

- `virtual int calculateNumber(const int& row, const int& column) const = 0;` No implementation. This is a pure virtual method. For this method, declare it in the header file, but do not put anything in the implementation file.
- `virtual void calculateAllNumbers();` For every (row,column) pair, calls `calculateNumber` to get a number and `setNumber` to store it.

Create `ComplexFractal.{h,cpp}`

This class will handle all of the translations between plane coordinates (x,y) and number grid coordinates (column, row). This class inherits publicly from `NumberGrid`.

Data Members:

- 4 `double` data members: Minimum and maximum values for x and y to define the plane size. From our notes, these are min_x, max_x, min_y, and max_y.
- 2 `double` data members: Delta values for x and y to define the grid points in the plane. From our notes, these are delta x and delta y.

Methods:

- `ComplexFractal( );` Default constructor. Sets up for a 301x201 grid. For the plane coordinates uses the 3x2 rectangle centered on the origin. Sets the default value for delta_x and delta_y to 0.01. What values of min_x and max_x would give you a rectangle of width 3 and centered on the origin? Be sure to use constructor chaining.
- `ComplexFractal( const int& height, const int& width, const double& min_x, const double& max_x, const double& min_y, const double& max_y );` Constructor. Sets up the `NumberGrid` and `ComplexFractal` data members from parameters. Be sure to use constructor chaining.
- `virtual ~ComplexFractal( );` Must exist, but has empty code block.
- `double getMinX( ) const;` Return the minimum X value for the plane coordinates.
- `double getMaxX( ) const;` Return the maximum X value for the plane coordinates.
- `double getMinY( ) const;` Return the minimum Y value for the plane coordinates.
- `double getMaxY( ) const;` Return the maximum Y value for the plane coordinates.
- `virtual void setGridSize( const int& height, const int& width );` This method overrides the `NumberGrid` version. Only makes changes if both height and width are at least 2. If so, it calls

`NumberGrid::setGridSize()`. If a change is made, updates the values of delta x and delta y data members. Uses `calculateDeltaX()`, `calculateDeltaY()`, and `setDeltas()`.
- `void setPlaneSize( const double& min_x, const double& max_x, const double& min_y, const double& max_y );` Sets the 4 plane coordinates. Only makes a change if all of the coordinate values are between -2.0 and 2.0, inclusive. Only make changes if the minimum and maximum value for a dimension are different. If the minimum value for a dimension is greater than the maximum value for the dimension, automatically swap them. If a change is made, updates the values of delta x and delta y data members. Uses `calculateDeltaX()`, `calculateDeltaY()`, and `setDeltas()`.
- `double getDeltaX( ) const;` Returns the horizontal delta value from the data member.
- `double getDeltaY( ) const;` Returns the vertical delta value from the data member.
- `void setDeltas( const double& delta_x, const double& delta_y );` Assigns the deltas to data members. Only assigns if both values are positive.
- `double calculateDeltaY( ) const;` Calculate the vertical plane distance between neighboring pixel rows. This is the delta value discussed above. Note this method calculates the value and returns it. It does not set the data member.
- `double calculateDeltaX( ) const;` Calculate the horizontal plane distance between neighboring pixel columns. This is the delta value discussed above. Note this method calculates the value and returns it. It does not set the data member.
- `double calculatePlaneXFromPixelColumn( const int& column ) const;` Calculate the plane x value for a given column. If the column index is out of range (if `column` is less than zero or `column` is greater than or equal to the grid width), return 0. Do not call `calculateDeltaX()` here. Use `getDeltaX()` or directly access the data member. The value should have already been calculated previously.
- `double calculatePlaneYFromPixelRow( const int& row ) const;` Calculate the plane y value for a given row. If the row index is out of range (if `row` is less than zero or `row` is greater than or equal to the grid height), return 0. Do not call `calculateDeltaY()` here. Use `getDeltaY()` or directly access the data member. The value should have already been calculated previously.
- `void calculatePlaneCoordinatesFromPixelCoordinates( const int& row, const int& column, double& x, double& y ) const;` Sets x and y to the plane coordinates for the row and column. If either row or column is out of range, set both x and y to 0. Notice x and y are return by reference.
- `virtual int calculateNumber( const int& row, const int& column ) const;` If the row and column will make a valid index, calculate values for x and y from row and column. Then use the formula: `std::abs(getMaxNumber() * std::sin(10*x) * std::cos(10*y))` to calculate an integer value. Return this value. If row and column are not valid, return -1.

Update `image_menu.h` and `image_drawing.cpp`

The follow functions must be declared and implemented.

- `void setFractalPlaneSize(ActionData& action_data);` Asks the user for the `double`s “Min X? “, “Max X? “, “Min Y? ” and “Max Y? “, then sets the plane size. Only does this work if the `grid` is actually a `ComplexFractal` object. Otherwise, gives a message “Not a ComplexFractal object. Can’t set plane size.”.
- `void calculateFractal(ActionData& action_data);` Calculates all numbers for the grid stored in `action_data`.

Update Functions in `controllers.cpp`

- `void configureMenu( MenuData& menu_data )` add the new actions with the names and descriptions listed below.
- `int imageMenu( std::istream& is, std::ostream& os );` Instead of passing a `new NumberGrid` to the `ActionData`’s `setGrid()`, send a `new ComplexFractal`.

Table of New Commands

Command Name	Function Name	Description
fractal-plane-size	setFractalPlaneSize	Set the dimensions of the grid in the complex plane.
fractal-calculate	calculateFractal	Calculate the escape values for the fractal.

Update `Makefile`

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 image_file program
- `make all` - builds all programs
- `make` - builds all programs (same as `make all`)
- `make clean` - removes all .o files, and all executable programs

Additional Documentation

- C++ Reference
- Examples from class

Show Off Your Work

To receive credit for this assignment, you must

- use git to add, commit and push your solution to your repository for this class.
- successfully pass all unit tests and acceptance tests

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

Extra Challenges (Not Required)

- Create classes that inherit from `ComplexFractal` that have different functions, `calculateNumber()`, for calculating numbers. Add the ability to use them from the program.
- Make methods of the `ComplexFractal` class that allow for zooming in or out in the plane.
- Try other ways to modify the plane and parameters that would make it easier to create interesting images.