```cpp
// src/database.cc

#include "database.hh"

#include <vector>
#include <string>
#include <iostream>
#include <fstream>

using namespace std;

/**
 * @brief Modify a string so that it can be written to CSV properly. This means replacing double quotes with a pair of consecutive double quotes
 *
 * @param s Input string
 * @return A ready-to-write result string
 */
std::string HFractalDatabase::forCSVInner (std::string s) {
    string fixed_quotes = "";
    for (char c : s) {
        fixed_quotes += c;
        if (c == '\"') fixed_quotes += '\"';
    }
    fixed_quotes = "\"" + fixed_quotes + "\"";
    return fixed_quotes;
}

/**
 * @brief Break a record from a CSV file into a sequence of raw string fields
 *
 * @param line Line from CSV file to process
 * @return std::vector of raw string fields (require additional processing)
 */
std::vector<std::string> HFractalDatabase::componentify (std::string line) {
    vector<string> ret_value;
    string current_component;
    int start_index = line.find ("\"");
    int end_index = -1;
    while (start_index < line.length()) {
        end_index = line.find ("\",\"", start_index);
```

```
        if (end_index == string::npos) end_index = line.find ("\"", start_index+1);
        current_component = line.substr (start_index+1, end_index-(start_index+1));
        current_component = fixDoubleQuote (current_component);
        ret_value.push_back (current_component);
        start_index = end_index+2;
    }
    return ret_value;
}

/**
 * @brief Remove double-double quotes from a string. Effectively the reverse of forCSVInner
 *
 * @param s String field to process
 * @return Cleaned-up field string
 */
std::string HFractalDatabase::fixDoubleQuote (std::string s) {
    string result = "";
    char current_char = '\0';
    char next_char = '\0';
    for (int i = 0; i < s.length(); i++) {
        current_char = s[i];
        next_char = (i+1 < s.length()) ? s[i+1] : '\0';
        result.push_back (current_char);
        if (current_char == '\"' && next_char == '\"') i++;
    }
    return result;
}

/**
 * @brief Construct a database instance using a base path to CSV files
 *
 * @param path Base path to the target CSV database. Individual tables must be stored as separate CSV files, so the base path is modified provide paths to the invidiual CSV files
 */
HFractalDatabase::HFractalDatabase (std::string path) {
    // Generate and assign the base path
    int cutoff = path.find(".csv");
    db_path = path.substr (0, cutoff);

```

```
    // Try to read the database, failing that write a new one, failing that, error out
    if (!read()) if (!commit ()) {
        throw new std::runtime_error ("unable to create database");
    }
}

HFractalDatabase::HFractalDatabase () {}

/**
 * @brief Get a list of config profile descriptions paired with their IDs. Allows the GUI to easily grab a profile summary without having to fetch all the data one-by-one
 *
 * @return std::vector of pairs of ID and string values
 */
std::vector<std::pair<long, std::string>>
HFractalDatabase::getConfigDescriptions () {
    std::vector<std::pair<long, std::string>> ret_val;
    for (auto conf : configs) {
        std::pair<long, std::string> desc_pair;
        desc_pair.first = conf.first;
        desc_pair.second = (
            conf.second->name
            + " ("
            + conf.second->equation
            + ")");
        ret_val.push_back (desc_pair);
    }
    return ret_val;
}

/**
 * @brief Function to get a configuration profile by its ID
 *
 * @param id The ID of the profile
 * @return A pointer to the configuration profile
 */
HFractalConfigProfile* HFractalDatabase::getConfig (long id) {
    HFractalConfigProfile *ret = NULL;
    if (configs.count(id) != 0) ret = configs[id];
    return ret;
}
```

```
118
119 /**
120  * @brief Function to get a user profile by its ID
121  *
122  * @param id The ID of the profile
123  * @return A pointer to the user profile
124  */
125 HFractalUserProfile* HFractalDatabase::getUser (long id) {
126     HFractalUserProfile *ret = NULL;
127     if (users.count(id) != 0) ret = users[id];
128     return ret;
129 }
130
131 /**
132  * @brief Insert a configuration profile into the database. Automatically
    generates and assigns a unique ID
133  *
134  * @param c Pointer to the config profile being inserted
135  * @return Generated ID of the profile
136  */
137 long HFractalDatabase::insertConfig (HFractalConfigProfile* c) {
138     long max_id = -1;
139     for (pair<long, HFractalConfigProfile*> p : configs)
140         if (p.first > max_id) max_id = p.first;
141
142     c->profile_id = max_id+1;
143     configs.emplace (c->profile_id, c);
144     return c->profile_id;
145 }
146
147 /**
148  * @brief Insert a user profile into the database. Automatically generates
    and assigns a unique ID
149  *
150  * @param u Pointer to the user profile being inserted
151  * @return Generated ID of the profile
152  */
153 long HFractalDatabase::insertUser (HFractalUserProfile* u) {
154     long max_id = -1;
155     for (pair<long, HFractalUserProfile*> p : users)
156         if (p.first > max_id) max_id = p.first;
157
```

```
    u->user_id = max_id+1;
    users.emplace (u->user_id, u);
    return u->user_id;
}

/**
 * @brief Delete a config profile record from the database
 *
 * @param id ID of the profile to be deleted
 * @return True if the delete succeeded, False if the record did not exist
 */
bool HFractalDatabase::removeConfig (long id) {
    return configs.erase (id);
}

/**
 * @brief Delete a user profile record from the database
 *
 * @param id ID of the profile to be deleted
 * @return True if the delete succeeded, False if the record did not exist
 */
bool HFractalDatabase::removeUser (long id) {
    return users.erase (id);
}

/**
 * @brief Write out the contents of the cached database to CSV files.
 *
 * @return True if the write succeeded, False if it failed
 */
bool HFractalDatabase::commit () {
    // Create path strings and file streams
    string db_path_configs = db_path + "_configs.csv";
    string db_path_users = db_path + "_users.csv";
    ofstream db_file_configs (db_path_configs.c_str());
    ofstream db_file_users (db_path_users.c_str());

    // If the file streams are open, write data
    if (db_file_configs.is_open() && db_file_users.is_open()) {
        // Iterate over config files
        for (auto copair : configs) {
            HFractalConfigProfile* config = copair.second;
```

```
            string line = "";
            line += forCSV (config->profile_id) + ",";
            line += forCSV (config->x_offset) + ",";
            line += forCSV (config->y_offset) + ",";
            line += forCSV (config->zoom) + ",";
            line += forCSV (config->iterations) + ",";
            line += forCSV (config->equation) + ",";
            line += forCSV (config->name) + ",";
            line += forCSV (config->palette) + ",";
            line += forCSV (config->user_id);
            db_file_configs << line.c_str() << endl;
        }

        // Iterate over user files
        for (auto upair : users) {
            HFractalUserProfile* user = upair.second;
            string line = "";
            line += forCSV (user->user_id) + ",";
            line += forCSV (user->user_name);
            db_file_users << line.c_str() << endl;
        }

        // Close the files and return success
        db_file_configs.close();
        db_file_users.close();
        return true;
    } else return false; // Return failure
}

/**
 * @brief Read the contents of CSV files into the cached database
 *
 * @return True if the read succeeded, False if it failed
 */
bool HFractalDatabase::read () {
    // Create paths and file streams
    string db_path_configs = db_path + "_configs.csv";
    string db_path_users = db_path + "_users.csv";
    ifstream db_file_configs (db_path_configs.c_str());
    ifstream db_file_users (db_path_users.c_str());

    // If the file streams are open, read data
```

```
242     if (db_file_configs.is_open() && db_file_users.is_open()) {
243         string line;
244         int line_number = 0;
245 
246         configs.clear();
247         // Read config profiles
248         HFractalConfigProfile* config;
249         while (getline (db_file_configs, line)) {
250             try {
251                 // Attempt to convert the record to a config profile
252                 vector<string> components = componentify (line);
253                 config = new HFractalConfigProfile ();
254                 config->profile_id = stol(components[0]);
255                 config->x_offset = stold(components[1]);
256                 config->y_offset = stold(components[2]);
257                 config->zoom = stold(components[3]);
258                 config->iterations = stoi(components[4]);
259                 config->equation = components[5];
260                 config->name = components[6];
261                 config->palette = stoi(components[7]);
262                 config->user_id = stol(components[8]);
263                 configs.emplace (config->profile_id, config);
264             } catch (std::invalid_argument e) {
265                 // Print a console error if a line could not be read
266                 cout << "Failed to read config profile on line " <<
line_number << endl;
267             }
268             line_number++;
269         }
270 
271         line_number = 0;
272 
273         // Read user profiles
274         users.clear();
275         HFractalUserProfile* user;
276         while (getline (db_file_users, line)) {
277             try {
278                 // Attempt to convert the record to a user profile
279                 vector<string> components = componentify (line);
280                 user = new HFractalUserProfile ();
281                 user->user_id = stol(components[0]);
282                 user->user_name = components[1];
```

```
283                 users.emplace (user->user_id, user);
284             } catch (std::invalid_argument e) {
285                 // Print a console error if a line could not be read
286                 cout << "Failed to read user profile on line " <<
line_number << endl;
287             }
288             line_number++;
289         }
290         // Return success
291         return true;
292     } else return false; // Return failure
293 }
294 
295 /**
296  * @brief Generate a CSV-happy string from a given input
297  *
298  * @param s String input
299  * @return CSV-writeable string
300  */
301 std::string HFractalDatabase::forCSV (std::string s) {
302     return forCSVInner (s);
303 }
304 
305 /**
306  * @brief Generate a CSV-happy string from a given input
307  *
308  * @param l Long integer input
309  * @return CSV-writeable string
310  */
311 std::string HFractalDatabase::forCSV (long l) {
312     return forCSVInner (to_string(l));
313 }
314 
315 /**
316  * @brief Generate a CSV-happy string from a given input
317  *
318  * @param ld Long double input
319  * @return CSV-writeable string
320  */
321 std::string HFractalDatabase::forCSV (long double ld) {
322     return forCSVInner (to_string(ld));
323 }
```

```
324
325 /**
326  * @brief Generate a CSV-happy string from a given input
327  *
328  * @param i Integer input
329  * @return CSV-writeable string
330  */
331 std::string HFractalDatabase::forCSV (int i) {
332     return forCSVInner (to_string(i));
333 }
```

```
// src/database.hh

#ifndef DATABASE_H
#define DATABASE_H

#include <string>
#include <vector>
#include <unordered_map>
#include <cstring>

// Struct describing the Config Profile record type
struct HFractalConfigProfile {
    long profile_id; // Primary key

    long double x_offset;
    long double y_offset;
    long double zoom;
    int iterations;
    std::string equation;
    std::string name;
    int palette;
    long user_id; // Foreign key of HFractalUserProfile

    HFractalConfigProfile () { memset (this, 0, sizeof(HFractalConfigProfile)); }
};

// Struct describing the User Profile record type
struct HFractalUserProfile {
    long user_id; // Primary key

    std::string user_name;

    HFractalUserProfile () { memset (this, 0, sizeof(HFractalUserProfile)); }
};

// Class for managing the database of users and configurations and providing access to data for the GUI
class HFractalDatabase {
private:
    std::string db_path; // Base path to the database
    std::unordered_map<long, HFractalConfigProfile*> configs; // Map of
```

```
config profiles against their IDs
41      std::unordered_map<long, HFractalUserProfile*> users; // Map of user
profiles against their IDs
42      static std::string forCSVInner (std::string); // Static function to
convert a string into a form CSVs will read/write properly
43      std::vector<std::string> componentify (std::string); // Break a line of
CSV into fields
44      std::string fixDoubleQuote (std::string); // Remove double quotes in
strings read from CSV file
45
46      static std::string forCSV (std::string); // Generate a string which can
be written to a CSV file as a field of a record
47      static std::string forCSV (long); // Generate a string which can be
written to a CSV file as a field of a record
48      static std::string forCSV (int); // Generate a string which can be
written to a CSV file as a field of a record
49      static std::string forCSV (long double); // Generate a string which can
be written to a CSV file as a field of a record
50  public:
51      HFractalDatabase (std::string); // Initialise the database from a given
base path
52      HFractalDatabase (); // Dead initialiser for implicit instantiation
53
54      std::vector<std::pair<long, std::string>> getConfigDescriptions (); //
Generate a list of ID and description pairs for the GUI to display
55      HFractalConfigProfile* getConfig (long); // Get a pointer to the config
profile with a given ID
56      HFractalUserProfile* getUser (long); // Get a pointer to the user profile
with a given ID
57
58      long insertConfig (HFractalConfigProfile*); // Insert a new config record
and return its ID
59      long insertUser (HFractalUserProfile*); // Insert a new user record and
return its ID
60
61      bool removeConfig (long); // Remove a config record by ID
62      bool removeUser (long); // Remove a user record by ID
63
64      bool commit (); // Write the contents of the cached database out to CSV
files
65      bool read (); // Read the contents of a CSV file into the database cache
66  };
67
68  #endif
```

```cpp
// src/equationparser.cc

#include "equationparser.hh"

#include <vector>

using namespace std;

/**
 * @brief Clean whitespace out of the input string
 *
 * @param s Input string
 * @return Cleaned string
 */
string HFractalEquationParser::epClean (string s) {
    string ret_val = "";
    for (char c : s) if (c != ' ') ret_val += c;
    return ret_val;
}

/**
 * @brief Check that the input string is valid for the HFractalEquation parser to analyse. Checks for the following and returns an enum value accordingly:
 *
 * 0 - No error found
 * 1 - Bracket error: '()', '(' not equal number to ')', ')...('
 * 2 - Operation error: '**', '*-' or any other repetition of an operation
 * 3 - Implicit multiplication error: 'z2' rather than correct synax '2z'
 * 4 - Floating point error: '.46', '34.'
 * 5 - Unsupported character error: '$', 'd', or any other character not accounted for
 *
 * @param s Input string
 * @return Either the reference of the first error detected or SUCCESS if no error is found
 */
EP_CHECK_STATUS HFractalEquationParser::epCheck (string s) {
    int bracket_depth = 0;
    char c_last = '\0';
    int index = 0;
    for (char c : s) {
        switch (c) {
```

```
40          case '(':
41              bracket_depth++;
42              if (c_last == '.') return FPOINT_ERROR;
43              break;
44          case ')':
45              bracket_depth--;
46              if (c_last == '(') return BRACKET_ERROR;
47              if (c_last == '.') return FPOINT_ERROR;
48              break;
49          case 'z':
50          case 'c':
51          case 'a':
52          case 'b':
53          case 'x':
54          case 'y':
55          case 'i':
56              if (c_last == '.') return FPOINT_ERROR;
57              break;
58          case '*':
59          case '/':
60          case '+':
61          case '^':
62              if (c_last == '*' || c_last == '/' || c_last == '-' || c_last ==
'+' || c_last == '^') return OPERATION_ERROR;
63              if (c_last == '.') return FPOINT_ERROR;
64              if (index == 0 || index == s.length()-1) return OPERATION_ERROR;
65              break;
66          case '-':
67              if (c_last == '-') return OPERATION_ERROR;
68              if (c_last == '.') return FPOINT_ERROR;
69              if (index == s.length()-1) return OPERATION_ERROR;
70              break;
71          case '0':
72          case '1':
73          case '2':
74          case '3':
75          case '4':
76          case '5':
77          case '6':
78          case '7':
79          case '8':
80          case '9':
```

```
81              if (c_last == 'z' || c_last == 'c' || c_last == 'i' || c_last ==
'a' || c_last == 'b' || c_last == 'x' || c_last == 'y') return IMULT_ERROR;
82              break;
83          case '.':
84              if (!(c_last == '0' || c_last == '1' || c_last == '2' || c_last
== '3' || c_last == '4' || c_last == '5' || c_last == '6' || c_last == '7'
|| c_last == '8' || c_last == '9')) return FPOINT_ERROR;
85              break;
86          default:
87              return UNSUPCHAR_ERROR;
88              break;
89          }
90
91          c_last = c;
92          index++;
93          if (bracket_depth < 0) return BRACKET_ERROR;
94      }
95
96      if (bracket_depth != 0) return BRACKET_ERROR;
97      return SUCCESS;
98  }
99
100 /**
101  * @brief Break string into tokens for processing. Assumes epCheck has been
called on `s` previously and that this has returned 0
102  *
103  * @param s Input string
104  * @return std::vector of tokens
105  */
106 vector<IntermediateToken> HFractalEquationParser::epTokenise (string s) {
107     vector<IntermediateToken> token_vec;
108     string current_token = "";
109     int current_token_type = -1;
110     bool is_last_run = false;
111     bool is_singular_token = false; // Informs the program whether the token
is a single-char token
112
113     for (int i = 0; i < s.length(); i++) {
114         char current_char = s[i];
115         int char_token_type = -1;
116
117         // Decide the type of the current character
118         switch (current_char) {
```

```
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
        case '.':
            char_token_type = 0;
            break;
        case 'z':
            char_token_type = 1;
            break;
        case 'c':
            char_token_type = 2;
            break;
        case 'a':
            char_token_type = 6;
            break;
        case 'b':
            char_token_type = 7;
            break;
        case 'x':
            char_token_type = 8;
            break;
        case 'y':
            char_token_type = 9;
            break;
        case 'i':
            char_token_type = 3;
            break;
        case '*':
        case '/':
        case '-':
        case '+':
        case '^':
            char_token_type = 4;
            break;
        case '(':
```

```
161             char_token_type = 5;
162             break;
163         default:
164             break;
165         }
166 
167         // Save the current token the token vector
168         if (char_token_type != current_token_type || is_last_run ||
is_singular_token) {
169             is_singular_token = false;
170             if (current_token.length () > 0) {
171                 IntermediateToken token;
172                 switch (current_token_type) {
173                 case 0:
174                     token.type = INT_NUMBER;
175                     token.num_val = stod (current_token);
176                     break;
177                 case 1:
178                 case 2:
179                 case 3:
180                 case 6:
181                 case 7:
182                 case 8:
183                 case 9:
184                     token.type = INT_LETTER;
185                     token.let_val = current_token[0];
186                     break;
187                 case 4:
188                     token.type = INT_OPERATION;
189                     token.op_val = current_token[0];
190                     break;
191                 case 5:
192                     token.type = INT_BRACKET;
193                     token.bracket_val = epTokenise (current_token);
194                 default:
195                     break;
196                 }
197                 token_vec.push_back (token);
198             }
199             current_token = "";
200             current_token_type = char_token_type;
201 
```

```
            if (is_last_run) break;
        }

        // Jump automatically to the end of the brackets, recursively processing their contents
        if (char_token_type == 5) {
            int bracket_depth = 1;
            int end = -1;
            for (int j = i+1; j < s.length(); j++) {
                if (s[j] == '(') bracket_depth++;
                if (s[j] == ')') bracket_depth--;
                if (bracket_depth == 0) { end = j; break; }
            }

            current_token = s.substr (i+1, end-(i+1));
            i = end;
        } else { // Otherwise append the current character to the current token
            current_token += s[i];
        }

        // Mark a, b, c, x, y, z, and i as singular
        if ((char_token_type >= 1 && char_token_type <= 3) || (char_token_type >= 6 && char_token_type <= 9)) {
            is_singular_token = true;
        }

        // If we've reached the end of the string, jump back and mark it as a last pass in order to save the current token
        if (i == s.length()-1) {
            is_last_run = true;
            i--;
        }
    }

    // Check through and repair any `-...` expressions to be `0-...`
    for (int i = 0; i < token_vec.size(); i++) {
        if (token_vec[i].type == INT_OPERATION && token_vec[i].op_val == '-') {
            if (i == 0 || (i > 0 && token_vec[i-1].type == INT_OPERATION)) {
                IntermediateToken bracket;
                bracket.type = INT_BRACKET;
                int bracket_length = 1;
```

```
                bracket.bracket_val.push_back ({
                    .type = INT_NUMBER,
                    .num_val = 0
                });
                bracket.bracket_val.push_back ({
                    .type = INT_OPERATION,
                    .op_val = '-'
                });
                while (i+bracket_length < token_vec.size() && token_vec[i+bracket_length].type != INT_OPERATION) {
                    bracket.bracket_val.push_back (token_vec[i+bracket_length]);
                    bracket_length++;
                }

                for (int tmp = 0; tmp < bracket_length; tmp++) token_vec.erase (next(token_vec.begin(), i));
                token_vec.insert (next(token_vec.begin(), i), bracket);
                i -= bracket_length-1;
            }
        }
    }

    return token_vec;
}

/**
 * @brief Search for and replace implicit multiplication (adjacent non-operation tokens such as '5z' or '(...)(...)') with explicit multiplication
 *
 * @param token_vec Token vector to fix
 * @return Token vector with no implicit multiplication
 */
vector<IntermediateToken> HFractalEquationParser::epFixImplicitMul (vector<IntermediateToken> token_vec) {
    vector<IntermediateToken> result = token_vec;
    for (int i = 0; i < result.size()-1; i++) {
        IntermediateToken t1 = result[i];
        IntermediateToken t2 = result[i+1];

        // Fix implicit multiplication within brackets
        if (t1.type == INT_BRACKET) {
            result[i].bracket_val = epFixImplicitMul (t1.bracket_val);
```

```
278             t1 = result[i];
279         }
280 
281         // Detect two adjacent tokens, where neither is an operation
282         if (!(t1.type == INT_OPERATION || t2.type == INT_OPERATION)) {
283             result.erase (next(result.begin(), i));
284             result.erase (next(result.begin(), i));
285             IntermediateToken explicit_mul;
286             explicit_mul.type = INT_BRACKET;
287             explicit_mul.bracket_val.push_back (t1);
288             explicit_mul.bracket_val.push_back ({
289                 .type = INT_OPERATION,
290                 .op_val = '*'
291             });
292             explicit_mul.bracket_val.push_back (t2);
293 
294             result.insert (next(result.begin(), i), explicit_mul);
295             i--;
296         }
297     }
298 
299     IntermediateToken last = result[result.size()-1];
300     if (last.type == INT_BRACKET) {
301         last.bracket_val = epFixImplicitMul (last.bracket_val);
302         result[result.size()-1] = last;
303     }
304 
305     return result;
306 }
307 
308 /**
309  * @brief Ensure BIDMAS (Brackets Indices Division Multiplication Addition Subtraction) order mathematical evaluation by search-and-replacing each with brackets
310  *
311  * @param token_vec Token vector to simplify
312  * @return Token vector which requires only sequential evaluation
313  */
314 vector<IntermediateToken> HFractalEquationParser::epSimplifyBidmas (vector<IntermediateToken> token_vec, bool first_half) {
315     vector<IntermediateToken> result = token_vec;
316     // Recurse down brackets
317     for (int i = 0; i < result.size(); i++) {
```

```
318         if (result[i].type == INT_BRACKET) {
319             result[i].bracket_val = epSimplifyBidmas (result[i].bracket_val,
first_half);
320         }
321     }
322 
323     // Order of operations:
324     //           IDMAS
325     string ops = "^/*+-";
326 
327     // First half allows the parser to make indices and division explicit,
then process implicit multiplication, and then to process other operations
328     // This allows us to maintain BIDMAS even with explicit multiplication
(e.g. `5z^2` should be `5*(z^2)` and not `(5*z)^2`)
329     if (first_half) {
330         ops = "^/";
331     } else {
332         ops = "*+-";
333     }
334 
335     if (result.size() < 5) return result;
336 
337     // Search and replace each sequentially
338     for (char c : ops) {
339         for (int t_ind = 0; t_ind < result.size()-2; t_ind++) {
340             if (t_ind >= result.size()-2) {
341                 break;
342             }
343             if (result[t_ind+1].type == INT_OPERATION &&
result[t_ind+1].op_val == c) {
344                 IntermediateToken bracket;
345                 bracket.type = INT_BRACKET;
346                 bracket.bracket_val.push_back (result[t_ind]);
347                 bracket.bracket_val.push_back (result[t_ind+1]);
348                 bracket.bracket_val.push_back (result[t_ind+2]);
349 
350                 for (int tmp = 0; tmp < 3; tmp++) result.erase
(next(result.begin(), t_ind));
351                 result.insert (next(result.begin(), t_ind), bracket);
352                 t_ind -= 2;
353             }
354         }
355     }
```

```
356
357     return result;
358 }
359
360 /**
361  * @brief Convert intermediate token vector into usable Reverse Polish
Notation token queue
362  * Ensure that everything else is done before calling this function:
363  *  epClean
364  *  epTokenise
365  *  epSimplifyBidmas first_half=true
366  *  epFixImplicitMul
367  *  epSimplifyBidmas first_half=false
368  * These functions ensure the token vector is ready to be linearly parsed
into Reverse Polish
369  *
370  * @param intermediate Token vector to convert
371  * @return Vector of proper tokens, ready to use in the expression evaluator
372  */
373 vector<Token> HFractalEquationParser::epReversePolishConvert
(vector<IntermediateToken> intermediate) {
374     vector<Token> output;
375
376     IntermediateToken operation = {.op_val = '\0'};
377
378     for (int index = 0; index < intermediate.size(); index++) {
379         IntermediateToken current_intermediate_token = intermediate[index];
380         if (current_intermediate_token.type == INT_OPERATION) {
381             operation.op_val = current_intermediate_token.op_val;
382         } else {
383             // Append to token(s) rp notation
384             if (current_intermediate_token.type == INT_BRACKET) {
385                 vector<Token> inner_result = epReversePolishConvert
(current_intermediate_token.bracket_val);
386                 output.insert (output.end(), inner_result.begin(),
inner_result.end());
387             } else {
388                 output.push_back ({
389                     .type = (TOKEN_TYPE)current_intermediate_token.type,
390                     .num_val = current_intermediate_token.type == INT_NUMBER
? current_intermediate_token.num_val : 0,
391                     .other_val = current_intermediate_token.type ==
INT_LETTER ? current_intermediate_token.let_val : '\0'
392                 });
```

```
393             }
394 
395             // If relevant, append operation to rp notation
396             if (operation.op_val != '\0') {
397                 output.push_back ({
398                     .type = OPERATION,
399                     .other_val = operation.op_val
400                 });
401                 operation.op_val = '\0';
402             }
403         }
404     }
405 
406     return output;
407 }
408 
409 /**
410  * @brief Convert a string mathematical expression into an HFractalEquation class instance using Reverse Polish Notation
411  *
412  * @param sequ String containing a mathematical expression to parse
413  * @return Pointer to an HFractalEquation instance representing the input string
414  */
415 HFractalEquation* HFractalEquationParser::extractEquation (string sequ) {
416     if (sequ.length() < 1) return NULL;
417     string cleaned = epClean (sequ);
418     EP_CHECK_STATUS check_result = epCheck (cleaned);
419     if (check_result != SUCCESS) {
420         return NULL;
421     }
422 
423     vector<IntermediateToken> expression = epTokenise (cleaned);
424 
425     expression = epSimplifyBidmas (expression, true);
426     expression = epFixImplicitMul (expression);
427     expression = epSimplifyBidmas (expression, false);
428 
429     vector<Token> reverse_polish_expression = epReversePolishConvert (expression);
430 
431     return new HFractalEquation (reverse_polish_expression);
432 }
```

```cpp
// src/equationparser.hh

#ifndef EQUATIONPARSER_H
#define EQUATIONPARSER_H

#include <string>
#include <vector>

#include "fractal.hh"

// Enum describing the token type for the intermediate parser
enum INTERMEDIATE_TOKEN_TYPE {
    INT_NUMBER,
    INT_LETTER,
    INT_OPERATION,
    INT_BRACKET
};

// Struct describing the token for the intermediate parser
struct IntermediateToken {
    INTERMEDIATE_TOKEN_TYPE type;
    double num_val;
    char let_val;
    char op_val;
    std::vector<IntermediateToken> bracket_val;
};

// Enum describing the error types from the equation processor checking function
enum EP_CHECK_STATUS {
    SUCCESS,
    BRACKET_ERROR,
    OPERATION_ERROR,
    IMULT_ERROR,
    FPOINT_ERROR,
    UNSUPCHAR_ERROR
};

// Class containing static methods used to parse a string into a postfix token vector
class HFractalEquationParser {
private:
```

```
41      static std::string epClean (std::string); // Preprocess the string to
    remove whitespace
42      static EP_CHECK_STATUS epCheck (std::string); // Check for formatting
    errors in the equation (such as mismatched brackets)
43      static std::vector<IntermediateToken> epTokenise (std::string); // Split
    the string into intermediate tokens
44      static std::vector<IntermediateToken> epFixImplicitMul
    (std::vector<IntermediateToken>); // Remove implicit multiplication
45      static std::vector<IntermediateToken> epSimplifyBidmas
    (std::vector<IntermediateToken>, bool); // Convert BIDMAS rules into explicit
    writing
46      static std::vector<Token> epReversePolishConvert
    (std::vector<IntermediateToken>); // Convert intermediate tokens into a final
    output postfix notation
47
48  public:
49      static HFractalEquation* extractEquation (std::string); // Extract an
    equation containing postfix tokens from a string input
50  };
51
52
53  #endif
```

```cpp
// src/fractal.cc

#include "fractal.hh"

#include <stack>
#include <complex>

#include "utils.hh"

using namespace std;

/**
 * @brief Check if a complex number has tended to infinity. Allows methods which use this check to be implementation independent
 * Tending to infinity is typically defined as |z| > 2, which here is expanded to maximise optimsation
 *
 * @param comp Complex number to check
 * @return True if the number has tended to infinity, False otherwise
 */
bool HFractalEquation::isInfinity (complex<long double> comp) {
    return (comp.real()*comp.real()) + (comp.imag()*comp.imag()) > (long double)4;
}

/**
 * @brief Set the equation preset value
 *
 * @param i Integer representing the preset ID, linked with EQ_PRESETS, or -1 to disable preset mode in this instance
 */
void HFractalEquation::setPreset (int i) {
    is_preset = (i != -1);
    preset = i;
}

/**
 * @brief Parse the Reverse Polish notation Token vector and evaluate the mathematical expression it represents
 *
 * @param z Current value of the z variable to feed in
 * @param c Current value of the c variable to feed in
 * @return Complex number with the value of the evaluated equation
```

```
 */
complex<long double> HFractalEquation::compute (complex<long double> z, complex<long double> c) {
    stack<complex<long double>> value_stack;

    for (Token t : reverse_polish_vector) {
        if (t.type == NUMBER) {
            // Push number arguments onto the stack
            value_stack.push (t.num_val);
        } else if (t.type == LETTER) {
            // Select based on letter and swap in the letter's value, before pushing it onto the stack
            switch (t.other_val) {
            case 'z':
                value_stack.push (z);
                break;
            case 'c':
                value_stack.push (c);
                break;
            case 'a':
                value_stack.push (c.real());
                break;
            case 'b':
                value_stack.push (c.imag());
                break;
            case 'x':
                value_stack.push (z.real());
                break;
            case 'y':
                value_stack.push (z.imag());
                break;
            case 'i':
                value_stack.push (complex<long double> (0,1));
                break;
            default:
                break;
            }
        } else if (t.type == OPERATION) {
            // Perform an actual computation based on an operation token
            complex<long double> v2 = value_stack.top(); value_stack.pop();
            complex<long double> v1 = value_stack.top(); value_stack.pop();
            switch (t.other_val) {
```

```
            case '^':
                value_stack.push (pow (v1, v2));
                break;
            case '/':
                value_stack.push (v1/v2);
                break;
            case '*':
                value_stack.push (v1*v2);
                break;
            case '+':
                value_stack.push (v1+v2);
                break;
            case '-':
                value_stack.push (v1-v2);
                break;
            default:
                break;
            }
        }
    }

    // Return the final value
    return value_stack.top();
}

/**
 * @brief Evaluate a complex coordinate (i.e. a pixel) to find the point at which it tends to infinity, by iteratively applying the equation as a mathematical function
 *
 * @param c Coordinate in the complex plane to initialise with
 * @param limit Limit for the number of iterations to compute before giving up, if the number does not tend to infinity
 * @return Integer representing the number of iterations performed before the number tended to infinity, or the limit if this was reached first
 */
int HFractalEquation::evaluate (complex<long double> c, int limit) {
    complex<long double> last = c;
    if (is_preset && preset == EQ_BURNINGSHIP_MODIFIED) {
        last = complex<long double> (0, 0);
    }

    int depth = 0;
```

```
118     while (depth < limit) {
119         // Switch between custom parsing mode and preset mode for more
    efficient computing of presets
120         if (!is_preset) {
121             last = compute (last, c); // Slow custom compute
122         } else {
123             // Much faster hard coded computation
124             switch (preset) {
125             case EQ_MANDELBROT:
126                 last = (last*last)+c;
127                 break;
128             case EQ_JULIA_1:
129                 last = (last*last)+complex<long double>(0.285, 0.01);
130                 break;
131             case EQ_JULIA_2:
132                 last = (last*last)-complex<long double>(0.70176, 0.3842);
133                 break;
134             case EQ_RECIPROCAL:
135                 last = complex<long double>(1,0)/((last*last)+c);
136                 break;
137             case EQ_ZPOWER:
138                 last = pow(last,last)+c-complex<long double>(0.5, 0);
139                 break;
140             case EQ_BARS:
141                 last = pow(last, c*c);
142                 break;
143             case EQ_BURNINGSHIP_MODIFIED:
144                 last = pow ((complex<long double>(abs(last.real()),0) -
    complex<long double>(0, abs(last.imag()))),2)+c;
145                 break;
146             default:
147                 break;
148             }
149         }
150         depth++;
151         // Check if the value has tended to infinity, and escape the loop if
    so
152         bool b = isInfinity (last);
153         if (b) break;
154     }
155     return depth;
156 }
157
```

```
158 /**
159  * @brief Initialise with the token sequence in postfix form which this
    class should use
160  *
161  * @param rp_vec Reverse Polish formatted vector of tokens
162  */
163 HFractalEquation::HFractalEquation (vector<Token> rp_vec) {
164     reverse_polish_vector = rp_vec;
165 }
166
167 /**
168  * @brief Base initialiser. Should only be used to construct presets, as the
    equation token vector cannot be assigned after initialisation
169  *
170  */
171 HFractalEquation::HFractalEquation () {}
```

```cpp
// src/fractal.hh

#ifndef FRACTAL_H
#define FRACTAL_H

#include <complex>
#include <vector>

// Enum describing the token type
enum TOKEN_TYPE {
    NUMBER,
    LETTER,
    OPERATION
};

// Struct describing the token
struct Token {
    TOKEN_TYPE type;
    double num_val;
    char other_val;
};

// Class holding the equation and providing functions to evaluate it
class HFractalEquation {
private:
    static bool isInfinity (std::complex<long double> comp); // Check if a complex number has exceeded the 'infinity' threshold
    std::vector<Token> reverse_polish_vector; // Sequence of equation tokens in postfix form

    bool is_preset = false; // Records whether this equation is using an equation preset
    int preset = -1; // Records the equation preset being used, if none, set to -1

public:
    void setPreset (int); // Set this equation to be a preset, identified numerically

    std::complex<long double> compute (std::complex<long double>, std::complex<long double>); // Perform a single calculation using the equation and the specified z and c values
    int evaluate (std::complex<long double>, int); // Perform the fractal calculation
```

```
37
38      HFractalEquation (std::vector<Token>); // Initialise with a sequence of
    equation tokens
39      HFractalEquation (); // Base initialiser
40  };
41
42  #endif
```

```cpp
// src/gui.cc

#include "gui.hh"
#include "guimain.hh"

#include <math.h>
#include <algorithm>
#include <thread>

#include "utils.hh"
#include "database.hh"

using namespace std;

/**
 * @brief Automatically configure the styling for the GUI.
 * Uses a stylesheet provided by raysan5, creator of the graphics library used in the project, raylib
 *
 */
void HFractalGui::configureStyling() {
    // This function implements the 'cyber' interface style provided by raygui's documentation.
    const char* stylesheet = R"(p 00 00 0x2f7486ff DEFAULT_BORDER_COLOR_NORMAL
p 00 01 0x024658ff    DEFAULT_BASE_COLOR_NORMAL
p 00 02 0x51bfd3ff    DEFAULT_TEXT_COLOR_NORMAL
p 00 03 0x82cde0ff    DEFAULT_BORDER_COLOR_FOCUSED
p 00 04 0x3299b4ff    DEFAULT_BASE_COLOR_FOCUSED
p 00 05 0xb6e1eaff    DEFAULT_TEXT_COLOR_FOCUSED
p 00 06 0xeb7630ff    DEFAULT_BORDER_COLOR_PRESSED
p 00 07 0xffbc51ff    DEFAULT_BASE_COLOR_PRESSED
p 00 08 0xd86f36ff    DEFAULT_TEXT_COLOR_PRESSED
p 00 09 0x134b5aff    DEFAULT_BORDER_COLOR_DISABLED
p 00 10 0x02313dff    DEFAULT_BASE_COLOR_DISABLED
p 00 11 0x17505fff    DEFAULT_TEXT_COLOR_DISABLED
p 00 16 0x00000012    DEFAULT_TEXT_SIZE
p 00 17 0x00000001    DEFAULT_TEXT_SPACING
p 00 18 0x81c0d0ff    DEFAULT_LINE_COLOR
p 00 19 0x00222bff    DEFAULT_BACKGROUND_COLOR)";
    // Iterate over string and extract styling properties
    int offset = 0;
```

```
    int stylePointIndex = 0;
    string stylePointControl = "";
    string stylePointProperty = "";
    string stylePointValue = "";
    while (offset <= strlen(stylesheet)) {
        if (stylesheet[offset] == ' ') {
            stylePointIndex++;
        } else if (stylesheet[offset] == '\n' || stylesheet[offset] == '\0') {
            GuiSetStyle (stoi(stylePointControl), stoi(stylePointProperty), stoll(stylePointValue, nullptr, 16));
            stylePointControl = "";
            stylePointProperty = "";
            stylePointValue = "";
            stylePointIndex = 0;
        } else {
            switch (stylePointIndex) {
            case 1:
                stylePointControl += stylesheet[offset];
                break;
            case 2:
                stylePointProperty += stylesheet[offset];
                break;
            case 3:
                stylePointValue += stylesheet[offset];
                break;
            default:
                break;
            }
        }
        offset++;
    }
    // Tell raylib we've finished updating the styling
    GuiUpdateStyleComplete();
}

/**
 * @brief Configure the GUI itself and all class properties
 *
 */
void HFractalGui::configureGUI(char* path) {
    // Basic class initialisation
```

```
80      dialog_text = "";
81      console_text = "Ready.";
82      save_name_buffer = "Untitled";
83      for (int i = 0; i < BUTTON_NUM_TOTAL; i++) button_states[i] = false;
84      buffer_image = {};
85      buffer_texture = {};
86      is_rendering = false;
87      is_outdated_render = true;
88      render_percentage = 0;
89      showing_coordinates = false;
90      modal_view_state = MVS_NORMAL;
91      selected_palette = CP_RAINBOW;
92      database_load_dialog_scroll = 0;
93      textbox_focus = TEXT_FOCUS_STATE::TFS_NONE;
94
95      // Fetch configuration
96      unsigned int thread_count = std::thread::hardware_concurrency ();
97      long double start_zoom = 1;
98      long double start_x_offset = 0;
99      long double start_y_offset = 0;
100     image_dimension = WINDOW_INIT_HEIGHT;
101     control_panel_width = WINDOW_INIT_WIDTH - WINDOW_INIT_HEIGHT;
102     equation_buffer = equationPreset (EQ_MANDELBROT, false);
103
104     // GUI configuration
105     SetTraceLogLevel (LOG_WARNING | LOG_ERROR | LOG_DEBUG);
106     InitWindow(WINDOW_INIT_WIDTH, WINDOW_INIT_HEIGHT, "HyperFractal
Mathematical Visualiser");
107     SetWindowState (FLAG_WINDOW_RESIZABLE);
108     SetExitKey(-1);
109     int min_height = std::max (256, CONTROL_MIN_HEIGHT);
110     SetWindowMinSize(min_height+CONTROL_MIN_WIDTH, min_height);
111     SetTargetFPS(24);
112     configureStyling();
113
114     // Initialise database;
115     database = HFractalDatabase
(string(path)+string("FractalSavedStates.csv"));
116
117     // Initialise rendering environment
118     lowres_hm = new HFractalMain();
119     hm = new HFractalMain();
```

```
120
121     // Configure full resolution renderer
122     hm->setResolution (image_dimension);
123     hm->setEquation (equation_buffer);
124     hm->setEvalLimit (200);
125     hm->setWorkerThreads (thread_count);
126     hm->setZoom (start_zoom);
127     hm->setOffsetX (start_x_offset);
128     hm->setOffsetY (start_y_offset);
129
130     // Configure preivew renderer
131     lowres_hm->setResolution (128);
132     lowres_hm->setEquation (equation_buffer);
133     lowres_hm->setEvalLimit (200);
134     lowres_hm->setWorkerThreads (thread_count/2);
135     lowres_hm->setZoom (start_zoom);
136     lowres_hm->setOffsetX (start_x_offset);
137     lowres_hm->setOffsetY (start_y_offset);
138 }
139
140 /**
141  * @brief Called when a rendering parameter has been modified.
142  * Causes the rendered image to become 'out of date' and updates the
    preview render
143  *
144  */
145 void HFractalGui::parametersWereModified() {
146     is_outdated_render = true;
147     console_text = "Outdated render!";
148     updatePreviewRender();
149 }
150
151 /**
152  * @brief Generate and show an updated image from the preview renderer
153  *
154  * @return True if the update was successful, false if the equation was
    invalid
155  */
156 bool HFractalGui::updatePreviewRender() {
157     // Check if the equation is valid
158     if (!lowres_hm->isValidEquation()) return false;
159     lowres_hm->generateImage(true); // If it is, run a render
```

```
160     reloadImageFrom(lowres_hm); // And load it
161     return true;
162 }
163 
164 /**
165  * @brief Trigger a full resolution render to start
166  *
167  * @return True if successful, false if the equation was invalid
168  */
169 bool HFractalGui::startFullRender() {
170     if (!hm->isValidEquation()) { // Check if this is a valid equation
171         console_text = "Invalid equation!";
172         return false;
173     }
174     // If it is, start the render
175     is_rendering = true;
176     console_text = "Rendering...";
177     hm->generateImage(false);
178     is_outdated_render = true;
179     render_percentage = 0;
180     return true;
181 }
182 
183 /**
184  * @brief Check the status of the full resolution render, and produce an
    up-to-date display image showing the render progress.
185  * Also updates the completion percentage
186  *
187  * @return True if the image has finished rendering, false otherwise
188  */
189 bool HFractalGui::updateFullRender() {
190     if (!is_rendering) return true;
191     // Update completion percentage
192     render_percentage = round(hm->getImageCompletionPercentage());
193     if (hm->getIsRendering()) { // If still rendering, update the rendered
    image and overlay it onto the preview
194         is_outdated_render = true;
195         Image overlay = getImage(hm);
196         ImageDraw(&buffer_image, overlay, (Rectangle){0,0,(float)hm-
    >getResolution(),(float)hm->getResolution()}, (Rectangle){0,0,(float)hm-
    >getResolution(),(float)hm->getResolution()}, WHITE);
197         UnloadImage(overlay);
198         tryUnloadTexture();
```

```
199         buffer_texture = LoadTextureFromImage(buffer_image);
200         return false;
201     } else { // Otherwise, set states to indicate completion and update the image
202         is_outdated_render = false;
203         is_rendering = false;
204         reloadImageFrom (hm);
205         console_text = "Rendering done.";
206         return true;
207     }
208 }
209 
210 /**
211  * @brief Reload the image and texture used for drawing to the screen from the specified render environment
212  *
213  * @param h Rendering environment to grab the image from
214  */
215 void HFractalGui::reloadImageFrom(HFractalMain* h) {
216     tryUnloadImage(); // Unload image and texture
217     tryUnloadTexture();
218     buffer_image = getImage(h);
219     if (buffer_image.height != image_dimension) // Resize it to fill the frame
220         ImageResize(&buffer_image, image_dimension, image_dimension);
221     buffer_texture = LoadTextureFromImage(buffer_image);
222 }
223 
224 /**
225  * @brief Check if the window has been resized, and handle it if so
226  *
227  */
228 void HFractalGui::checkWindowResize() {
229     if (is_rendering) { // If we're mid-render, snap back to previous window dimensions
230         SetWindowSize(image_dimension+control_panel_width, image_dimension);
231         return;
232     }
233     if (IsWindowResized()) { // Update render resolution and image dimension based on new size
234         image_dimension = std::min(GetScreenWidth()-CONTROL_MIN_WIDTH, GetScreenHeight());
235         control_panel_width = GetScreenWidth()-image_dimension;
```

```
236         hm->setResolution(image_dimension);
237         parametersWereModified(); // Notify that parameters have changed
238     }
239 }
240 
241 /**
242  * @brief Draw the entire interface
243  *
244  */
245 void HFractalGui::drawInterface() {
246     BeginDrawing(); // Tell raylib we're about to start drawing
247 
248     // Clear the background
249     Color bg_col = GetColor (GuiGetStyle(00, BACKGROUND_COLOR));
250     ClearBackground(bg_col);
251 
252     // Draw the rendered HFractalImage
253     Vector2 v {0,0};
254     DrawTextureEx (buffer_texture, v, 0,
(float)image_dimension/(float)buffer_texture.height, WHITE);
255     // Draw Console
256     int button_offset = 0;
257     GuiTextBox((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width, BUTTON_HEIGHT},
(char*)console_text.c_str(), 1, false);
258     // Draw "Render Image" button
259     button_offset++;
260     button_states[BUTTON_ID::BUTTON_ID_RENDER] = GuiButton((Rectangle)
{(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width, BUTTON_HEIGHT}, "Render Image") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
261     // Draw render progress bar
262     button_offset++;
263     GuiProgressBar ((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width, BUTTON_HEIGHT}, "", "",
render_percentage, 0, 100);
264     // Draw zoom buttons
265     button_offset++;
266     button_states[BUTTON_ID::BUTTON_ID_ZOOM_IN] = GuiButton((Rectangle)
{(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width/3, BUTTON_HEIGHT}, "Zoom In") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
267     button_states[BUTTON_ID::BUTTON_ID_ZOOM_RESET] = GuiButton((Rectangle)
{(float)image_dimension+(float)control_panel_width/3, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width/3, BUTTON_HEIGHT}, "Reset
Zoom") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
```

```
268     button_states[BUTTON_ID::BUTTON_ID_ZOOM_OUT] = GuiButton((Rectangle)
{(float)image_dimension+(float)control_panel_width/(3.0f/2.0f),
BUTTON_HEIGHT*(float)button_offset, (float)control_panel_width/3,
BUTTON_HEIGHT}, "Zoom Out") && (modal_view_state ==
MODAL_VIEW_STATE::MVS_NORMAL);
269     // Draw save image button
270     button_offset++;
271     button_states[BUTTON_ID::BUTTON_ID_SAVE_IMAGE] = GuiButton((Rectangle)
{(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width, BUTTON_HEIGHT}, "Save Image") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
272     // Draw render state load/save buttons
273     button_offset++;
274     button_states[BUTTON_ID::BUTTON_ID_SAVE_RSTATE] = GuiButton((Rectangle)
{(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width/2, BUTTON_HEIGHT}, "Save Render State") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
275     button_states[BUTTON_ID::BUTTON_ID_LOAD_RSTATE] = GuiButton((Rectangle)
{(float)image_dimension+(float)control_panel_width/2, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width/2, BUTTON_HEIGHT}, "Load
Render State") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
276     // Draw movement navigation buttons
277     button_offset++;
278     button_states[BUTTON_ID::BUTTON_ID_UP] = GuiButton((Rectangle)
{(float)image_dimension+((float)control_panel_width-40)/2, BUTTON_HEIGHT*
(float)button_offset, (float)40, BUTTON_HEIGHT}, "up") && (modal_view_state
== MODAL_VIEW_STATE::MVS_NORMAL);
279     button_offset++;
280     button_states[BUTTON_ID::BUTTON_ID_LEFT] = GuiButton((Rectangle)
{(float)image_dimension+(((float)control_panel_width)/2)-40, BUTTON_HEIGHT*
(float)button_offset, (float)40, BUTTON_HEIGHT}, "left") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
281     button_states[BUTTON_ID::BUTTON_ID_RIGHT] = GuiButton((Rectangle)
{(float)image_dimension+(((float)control_panel_width)/2), BUTTON_HEIGHT*
(float)button_offset, (float)40, BUTTON_HEIGHT}, "right") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
282     button_offset++;
283     button_states[BUTTON_ID::BUTTON_ID_DOWN] = GuiButton((Rectangle)
{(float)image_dimension+((float)control_panel_width-40)/2, BUTTON_HEIGHT*
(float)button_offset, (float)40, BUTTON_HEIGHT}, "down") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
284     button_offset++;
285     button_states[BUTTON_ID::BUTTON_ID_EQ_PRESETS] = GuiButton((Rectangle)
{(float)image_dimension+(float)control_panel_width/2, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width/2, BUTTON_HEIGHT},
"Equation Presets") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
286
287     // Draw equation input box
288     button_states[BUTTON_ID::BUTTON_ID_EQ_INPUTBOX] = GuiTextBox
((Rectangle){(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width/2, BUTTON_HEIGHT}, equation_buffer.data(), 1,
false) && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
```

```
289     button_offset++;
290 
291     // Coordinate toggle button
292     string coord_button_text = "Hide coordinates";
293     if (!showing_coordinates) coord_button_text = "Show coordinates";
294     button_states[BUTTON_ID::BUTTON_ID_TOGGLE_COORDS] =
GuiButton((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width, BUTTON_HEIGHT},
coord_button_text.c_str()) && (modal_view_state ==
MODAL_VIEW_STATE::MVS_NORMAL);
295     button_offset++;
296 
297     // Eval limit controls
298     button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_LESS] =
GuiButton((Rectangle){(float)image_dimension+(float)control_panel_width/2,
BUTTON_HEIGHT*(float)button_offset, (float)control_panel_width/4,
BUTTON_HEIGHT}, "<") && (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
299     button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_MORE] =
GuiButton((Rectangle){(float)image_dimension+
((float)control_panel_width/4)*3, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width/4, BUTTON_HEIGHT}, ">") && (modal_view_state ==
MODAL_VIEW_STATE::MVS_NORMAL);
300     char evalLimString[16];
301     sprintf (evalLimString, "%d (%d)", hm->getEvalLimit(), lowres_hm-
>getEvalLimit());
302     GuiTextBox ((Rectangle){(float)image_dimension, BUTTON_HEIGHT*
(float)button_offset, (float)control_panel_width/2, BUTTON_HEIGHT},
evalLimString, 1, false);
303     button_offset++;
304 
305     // Colour palette preset selector button
306     button_states[BUTTON_ID::BUTTON_ID_CP_PRESETS] = GuiButton((Rectangle)
{(float)image_dimension, BUTTON_HEIGHT*(float)button_offset,
(float)control_panel_width, BUTTON_HEIGHT}, "Colour Palettes") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
307 
308     // Draw help button
309     button_states[BUTTON_ID::BUTTON_ID_HELP] = GuiButton((Rectangle)
{(float)image_dimension, (float)GetScreenHeight()-(2*BUTTON_HEIGHT),
(float)control_panel_width, BUTTON_HEIGHT*2}, "Help & Instructions") &&
(modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL);
310 
311     // Draw the equation preset dialog
312     if (modal_view_state == MODAL_VIEW_STATE::MVS_EQUATION_PRESET_SELECTOR)
{
313         float preset_dialog_x = (float)image_dimension+
(float)control_panel_width/2;
314         float preset_dialog_y = BUTTON_HEIGHT*10.0f;
315         for (int e = 0; e < NUM_EQUATION_PRESETS; e++) {
```

```
316             // Draw a button for each option
317             if (
318                 GuiButton((Rectangle){preset_dialog_x, preset_dialog_y+(BUTTON_HEIGHT*e), (float)control_panel_width/2, BUTTON_HEIGHT}, equationPreset((EQ_PRESETS)e, true).c_str())
319             && !is_rendering
320             ) {
321                 escapeEquationPresetDialog(e);
322             }
323         }
324         if (GetMouseX() < preset_dialog_x || GetMouseX() > preset_dialog_x + (float)control_panel_width/2 ||  GetMouseY() < preset_dialog_y - BUTTON_HEIGHT || GetMouseY() > preset_dialog_y + (BUTTON_HEIGHT*NUM_EQUATION_PRESETS)) {
325             escapeEquationPresetDialog(-1);
326         }
327     }
328
329     // Draw the colour palette preset dialog
330     if (modal_view_state == MODAL_VIEW_STATE::MVS_COLOUR_PRESET_SELECTOR) {
331         float preset_dialog_x = (float)image_dimension;
332         float preset_dialog_y = BUTTON_HEIGHT*13.0f;
333         for (int c = 0; c < NUM_COLOUR_PRESETS; c++) {
334             // Draw a button for each option
335             if (
336                 GuiButton((Rectangle){preset_dialog_x, preset_dialog_y+(BUTTON_HEIGHT*c), (float)control_panel_width, BUTTON_HEIGHT}, colourPalettePreset((CP_PRESETS)c).c_str())
337             && !is_rendering
338             ) {
339                 escapeColourPalettePresetDialog(c);
340             }
341         }
342         if (GetMouseX() < preset_dialog_x || GetMouseX() > preset_dialog_x + (float)control_panel_width ||  GetMouseY() < preset_dialog_y - BUTTON_HEIGHT || GetMouseY() > preset_dialog_y + (BUTTON_HEIGHT*NUM_COLOUR_PRESETS)) {
343             escapeColourPalettePresetDialog(-1);
344         }
345     }
346
347     // Draw the info dialog
348     if (modal_view_state == MODAL_VIEW_STATE::MVS_TEXT_DIALOG) {
349         float box_width = (2.0/3.0)*GetScreenWidth();
350         DrawRectangle (0, 0, GetScreenWidth(), GetScreenHeight(), (Color)
```

```
{200, 200, 200, 128});
351         Rectangle text_rec = (Rectangle){
352             ((float)GetScreenWidth()-box_width-10)/2,
353             ((float)GetScreenHeight()-DIALOG_TEXT_SIZE-10)/2,
354             box_width+10,
355             DIALOG_TEXT_SIZE
356         };
357         GuiDrawText (dialog_text.c_str(), text_rec, GUI_TEXT_ALIGN_CENTER,
BLACK);
358         button_states[BUTTON_ID::BUTTON_ID_TEXT_DIALOG_CLOSE] =
GuiButton((Rectangle){(float)(GetScreenWidth()-box_width-10)/2, (float)
(GetScreenHeight()*(3.0/4.0)+10), (float)(box_width+10), (float)
(DIALOG_TEXT_SIZE+10)}, "OK");
359     }
360
361     // Draw database dialog
362     if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG ||
modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG) {
363         DrawRectangle (0, 0, GetScreenWidth(), GetScreenHeight(), (Color)
{200, 200, 200, 128});
364         float box_width = (2.0/3.0)*GetScreenWidth();
365         button_states[BUTTON_ID::BUTTON_ID_DATABASE_CANCEL] = GuiButton(
366             (Rectangle){
367                 (float)(GetScreenWidth()-box_width-10)/2,
368                 (float)(GetScreenHeight()*(4.0/5.0)+10),
369                 (float)((box_width+10)/2.0),
370                 (float)(DIALOG_TEXT_SIZE+10)
371                 },
372         "Cancel");
373
374         // Branch depending on whether the saving dialog or the loading
dialog is open
375         if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG)
{
376             button_states[BUTTON_ID::BUTTON_ID_SAVE] = GuiButton(
377                 (Rectangle){
378                     (float)(GetScreenWidth()-10)/2,
379                     (float)(GetScreenHeight()*(4.0/5.0)+10),
380                     (float)((box_width+10)/2.0),
381                     (float)(DIALOG_TEXT_SIZE+10)
382                     },
383             "Save");
384
385             button_states[BUTTON_ID::BUTTON_ID_SAVE_NAME_INPUTBOX] =
GuiTextBox (
```

```
                (Rectangle){
                    (float)((GetScreenWidth()-box_width)/2.0),
                    (float)(GetScreenHeight()*(1.0/5.0)),
                    (float)(box_width),
                    (float)(BUTTON_HEIGHT*2)
                    },
            save_name_buffer.data(), 2, false);

        } else if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG) {
            button_states[BUTTON_ID::BUTTON_ID_LOAD] = GuiButton(
                (Rectangle){
                    (float)(GetScreenWidth()-10)/2,
                    (float)(GetScreenHeight()*(4.0/5.0)+10),
                    (float)((box_width+10)/2.0),
                    (float)(DIALOG_TEXT_SIZE+10)
                    },
            "Load (overwrites current config)");

            button_states[BUTTON_ID::BUTTON_ID_SCROLL_UP] = GuiButton(
                (Rectangle){
                    (float)(GetScreenWidth()/2),
                    (float)(GetScreenHeight()*(1.0/5.0))+9*BUTTON_HEIGHT,
                    (float)120,
                    (float)(BUTTON_HEIGHT)
                    },
            "Scroll up");

            button_states[BUTTON_ID::BUTTON_ID_SCROLL_DOWN] = GuiButton(
                (Rectangle){
                    (float)(GetScreenWidth()/2),
                    (float)(GetScreenHeight()*(1.0/5.0))+10*BUTTON_HEIGHT,
                    (float)120,
                    (float)(BUTTON_HEIGHT)
                    },
            "Scroll down");

            auto descriptions = database.getConfigDescriptions();
            int row_offset = 0;

            for (auto item : descriptions) {
                int draw_row = row_offset-database_load_dialog_scroll;
```

```
                if (draw_row >= 0 && draw_row <= 8) {
                    if (
                        GuiButton(
                    (Rectangle){
                        (float)(GetScreenWidth()-box_width)/2,
                        (float)(GetScreenHeight()*(1.0/5.0) +
BUTTON_HEIGHT*draw_row),
                        (float)((box_width)-120),
                        (float)(BUTTON_HEIGHT)
                        },
                    (((item.first == selected_profile_id) ? "(x)" : "( )")
+ item.second).c_str())
                    ) {
                        selected_profile_id = item.first;
                    }
                    if (
                        GuiButton(
                    (Rectangle){
                        (float)((GetScreenWidth()+box_width)/2)-120,
                        (float)(GetScreenHeight()*(1.0/5.0) +
BUTTON_HEIGHT*draw_row),
                        (float)(120),
                        (float)(BUTTON_HEIGHT)
                        },
                    "Delete? (!)")
                    ) {
                        database.removeConfig(item.first);
                        database.commit();
                    }
                }
                row_offset++;
            }
        }
    }

    // Draw coordinates text next to cursor
    if (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL &&
showing_coordinates && GetMouseX() <= image_dimension && GetMouseY() <=
image_dimension) {
        float left = GetMouseX()+15;
        float top = GetMouseY()+15;
        Color col {250, 250, 250, 200};

```

```
465         long double location_x = hm->getOffsetX() + ((long double)(((long double)GetMouseX()/(image_dimension/2))-1))/hm->getZoom();
466         long double location_y = hm->getOffsetY() - ((long double)(((long double)GetMouseY()/(image_dimension/2))-1))/hm->getZoom();
467         char t[142];
468         sprintf (t, "%.10Lf\n%.10Lf", location_x, location_y);
469         DrawRectangle (left, top, 115, 40, col);
470         DrawText (t, left+5, top, 15, BLACK);
471     }
472 
473     EndDrawing(); // Tell raylib we're done drawing
474 }
475 
476 /**
477  * @brief Close the equation preset dialog, and switch to a given preset
478  *
479  * @param e The equation preset to switch to, or -1 if the dialog was cancelled
480  */
481 void HFractalGui::escapeEquationPresetDialog(int e) {
482     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL; // Switch back to normal mode
483     if (is_rendering) return;
484     if (e != -1) { // If an option was selected, make it the current equation and notify that parameters have changed
485         equation_buffer = equationPreset ((EQ_PRESETS)e, false);
486         hm->setEquation (equation_buffer);
487         lowres_hm->setEquation (equation_buffer);
488         // Check whether the equation is valid
489         if (!hm->isValidEquation()) console_text = "Invalid equation input";
490         else {
491             parametersWereModified();
492         }
493     }
494 }
495 
496 /**
497  * @brief Show the equation preset dialog
498  *
499  */
500 void HFractalGui::enterEquationPresetDialog() {
501     if (is_rendering) return;
502     // Switch to equation preset selector mode
```

```
503     modal_view_state = MODAL_VIEW_STATE::MVS_EQUATION_PRESET_SELECTOR;
504 }
505 
506 /**
507  * @brief Show the colour palette preset dialog
508  *
509  */
510 void HFractalGui::enterColourPalettePresetDialog() {
511     if (is_rendering) return;
512     // Switch to colour preset selector mode
513     modal_view_state = MODAL_VIEW_STATE::MVS_COLOUR_PRESET_SELECTOR;
514 }
515 
516 /**
517  * @brief Close the colour palette preset dialog and switch to a given
    palette
518  *
519  * @param c Palette to switch to, or -1 if the dialog was cancelled
520  */
521 void HFractalGui::escapeColourPalettePresetDialog(int c) {
522     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL; // Return to normal
    GUI mode
523     if (is_rendering) return;
524     if (c != -1) {
525         // If an option was selected, reload the image with the selected
    palette (no rerender necessarry)
526         selected_palette = (CP_PRESETS)c;
527         if (is_outdated_render) {
528             reloadImageFrom(lowres_hm);
529         } else {
530             reloadImageFrom(hm);
531         }
532     }
533 }
534 
535 /**
536  * @brief Get an image handleable by raylib from a given rendering
    environment
537  *
538  * @param h Rendering environment to extract from
539  * @return A raylib-style image for drawing into the GUI
540  */
541 Image HFractalGui::getImage(HFractalMain* h) {
```

```
542     // Fetch a 32 bit RGBA image in the selected colour palette
543     int size = h->getResolution();
544     uint32_t *data = h->getRGBAImage(selected_palette);
545     Color *pixels = (Color *)malloc (size*size*sizeof(Color));
546     // Convert the image data to a format raylib will accept
547     for (int i = 0; i < size*size; i++) pixels[i] = GetColor(data[i]);
548     delete data;
549     // Construct a raylib image from the data
550     Image img = {
551         .data = pixels,
552         .width = size,
553         .height = size,
554         .mipmaps = 1,
555         .format = PIXELFORMAT_UNCOMPRESSED_R8G8B8A8
556     };
557     return img;
558 }
559 
560 /**
561  * @brief Handle when the user clicks on the image.
562  * Automatically centres the area they clicked and notifies the GUI that
    rendering parameters have been modified, triggering a preview update
563  *
564  * @return True if a click was handled, otherwise false
565  */
566 bool HFractalGui::handleClickNavigation() {
567     if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON) && !is_rendering) {
568         Vector2 mpos = GetMousePosition();
569         // Check if the mouse click was inside the image
570         if (mpos.x <= image_dimension && mpos.y <= image_dimension) {
571             long double change_in_x = (long double)((mpos.x /
    (image_dimension / 2)) - 1) / hm->getZoom();
572             long double change_in_y = (long double)((mpos.y /
    (image_dimension / 2)) - 1) / hm->getZoom();
573             long double new_offset_x = hm->getOffsetX() + change_in_x;
574             long double new_offset_y = hm->getOffsetY() - change_in_y;
575             // Update parameters and notify of the modification
576             lowres_hm->setOffsetX(new_offset_x);
577             lowres_hm->setOffsetY(new_offset_y);
578             hm->setOffsetX(new_offset_x);
579             hm->setOffsetY(new_offset_y);
580             parametersWereModified();
581             return true;
```

```
582         }
583     }
584     return false;
585 }
586 
587 /**
588  * @brief Show a text dialog with a given string as text
589  *
590  * @param text Text to display
591  */
592 void HFractalGui::launchTextDialog(std::string text) {
593     modal_view_state = MODAL_VIEW_STATE::MVS_TEXT_DIALOG;
594     dialog_text = text;
595 }
596 
597 /**
598  * @brief Close the currently open text dialog and go back to normal GUI mode
599  *
600  */
601 void HFractalGui::closeTextDialog() {
602     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL;
603     dialog_text = "";
604 }
605 
606 /**
607  * @brief Handler for Zoom In button
608  *
609  */
610 void HFractalGui::zoomIn() {
611     if (hm->getZoom() <= SCALE_DEPTH_LIMIT) { // Check the zoom has not exceeded the depth limit
612         long double new_zoom = hm->getZoom() * SCALE_STEP_FACTOR;
613         lowres_hm->setZoom (new_zoom);
614         hm->setZoom (new_zoom);
615         parametersWereModified();
616     } else launchTextDialog ("Zoom precision limit reached"); // Present a text dialog to report the issue to the user
617 }
618 
619 /**
620  * @brief Handler for Zoom Out button
621  *
```

```
 */
void HFractalGui::zoomOut() {
    long double new_zoom = hm->getZoom() / SCALE_STEP_FACTOR;
    lowres_hm->setZoom (new_zoom);
    hm->setZoom (new_zoom);
    parametersWereModified();
}

/**
 * @brief Handler for Reset Zoom button
 *
 */
void HFractalGui::resetZoom() {
    lowres_hm->setZoom(1);
    hm->setZoom(1);
    parametersWereModified();
}

/**
 * @brief Handler for Save Image button
 *
 */
void HFractalGui::saveImage() {
    bool result = false;
    // Switch depending on whether there is a full render available to save
    if (is_outdated_render) {
        result = lowres_hm->autoWriteImage(IMAGE_TYPE::PGM);
        console_text = "Saved preview render to desktop.";
    } else {
        result = hm->autoWriteImage (IMAGE_TYPE::PGM);
        console_text = "Saved render to desktop.";
    }
    if (!result) {
        console_text = "Image saving failed.";
    }
}

/**
 * @brief Make the save render state dialog visible
 *
 */
```

```
663 void HFractalGui::showSaveStateDialog() {
664     if (is_rendering) return;
665     modal_view_state = MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG;
666 }
667 
668 /**
669  * @brief Make the load render state dialog visible
670  *
671  */
672 void HFractalGui::showLoadStateDialog() {
673     if (is_rendering) return;
674     modal_view_state = MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG;
675     database_load_dialog_scroll = 0;
676 }
677 
678 /**
679  * @brief Save the current render state to the database and close the
    dialog
680  *
681  */
682 void HFractalGui::saveStateToDatabase() {
683     // Create the new config profile and populate its fields
684     HFractalConfigProfile *cp = new HFractalConfigProfile();
685     cp->equation = hm->getEquation();
686     cp->iterations = hm->getEvalLimit();
687     cp->name = save_name_buffer;
688     cp->palette = selected_palette;
689     cp->x_offset = hm->getOffsetX();
690     cp->y_offset = hm->getOffsetY();
691     cp->zoom = hm->getZoom();
692 
693     // Fetch or create the default user if necessarry
694     HFractalUserProfile *default_user = database.getUser(0);
695     if (default_user == NULL) {
696         default_user = new HFractalUserProfile();
697         default_user->user_name = "default";
698         database.insertUser (default_user);
699     }
700 
701     cp->user_id = default_user->user_id;
702 
703     // Insert the new profile into the database
```

```
704     database.insertConfig(cp);
705     database.commit();
706     console_text = "Profile saved to database!";
707     closeDatabaseDialog(); // Escape from the dialog
708 }
709 
710 /**
711  * @brief Load the selected render state from the database and close the
    dialog
712  *
713  */
714 void HFractalGui::loadStateFromDatabase() {
715     // Try to fetch the config profile
716     HFractalConfigProfile *cp = database.getConfig (selected_profile_id);
717     if (cp == NULL) {
718         console_text = "No profile selected to load.";
719     } else { // On success, load all properties into the rendering
    environments
720         hm->setEquation(cp->equation);
721         lowres_hm->setEquation(cp->equation);
722 
723         hm->setEvalLimit(cp->iterations);
724         lowres_hm->setEvalLimit(cp->iterations);
725 
726         hm->setOffsetX(cp->x_offset);
727         lowres_hm->setOffsetX(cp->x_offset);
728 
729         hm->setOffsetY(cp->y_offset);
730         lowres_hm->setOffsetY(cp->y_offset);
731 
732         hm->setZoom(cp->zoom);
733         lowres_hm->setZoom(cp->zoom);
734 
735         selected_palette = (CP_PRESETS)cp->palette;
736         save_name_buffer = cp->name;
737 
738         updatePreviewRender();
739         console_text = "Profile '" + save_name_buffer + "' loaded from
    database.";
740     }
741     closeDatabaseDialog(); // Close the dialog
742 }
743 
```

```
744 /**
745  * @brief Hide the database dialog and return to normal GUI mode
746  *
747  */
748 void HFractalGui::closeDatabaseDialog() {
749     modal_view_state = MODAL_VIEW_STATE::MVS_NORMAL;
750 }
751
752 /**
753  * @brief Scroll down inside the load render state dialog
754  *
755  */
756 void HFractalGui::databaseLoadScrollDown() {
757     database_load_dialog_scroll++;
758 }
759
760 /**
761  * @brief Scroll up inside the load render state dialog
762  *
763  */
764 void HFractalGui::databaseLoadScrollUp() {
765     database_load_dialog_scroll--;
766     if (database_load_dialog_scroll < 0) database_load_dialog_scroll = 0;
767 }
768
769 /**
770  * @brief Handler for Move Up button
771  *
772  */
773 void HFractalGui::moveUp() {
774     long double new_offset = hm->getOffsetY() + (MOVE_STEP_FACTOR/hm-
>getZoom());
775     hm->setOffsetY (new_offset);
776     lowres_hm->setOffsetY (new_offset);
777     parametersWereModified();
778 }
779
780 /**
781  * @brief Handler for Move Left button
782  *
783  */
784 void HFractalGui::moveLeft() {
```

```
785     long double new_offset = hm->getOffsetX() - (MOVE_STEP_FACTOR/hm-
>getZoom());
786     hm->setOffsetX (new_offset);
787     lowres_hm->setOffsetX (new_offset);
788     parametersWereModified();
789 }
790 
791 /**
792  * @brief Handler for Move Right button
793  *
794  */
795 void HFractalGui::moveRight() {
796     long double new_offset = hm->getOffsetX() + (MOVE_STEP_FACTOR/hm-
>getZoom());
797     hm->setOffsetX (new_offset);
798     lowres_hm->setOffsetX (new_offset);
799     parametersWereModified();
800 }
801 
802 /**
803  * @brief Handler for Move Down button
804  *
805  */
806 void HFractalGui::moveDown() {
807     long double new_offset = hm->getOffsetY() - (MOVE_STEP_FACTOR/hm-
>getZoom());
808     hm->setOffsetY (new_offset);
809     lowres_hm->setOffsetY (new_offset);
810     parametersWereModified();
811 }
812 
813 /**
814  * @brief Handler for Show/Hide Coordinates button
815  *
816  */
817 void HFractalGui::toggleCoords() {
818     showing_coordinates = !showing_coordinates;
819 }
820 
821 /**
822  * @brief Handlder for '<' button
823  *
824  */
```

```
825 void HFractalGui::evalLimitLess() {
826     int new_el = hm->getEvalLimit();
827     // Allow faster jumping if shift is held
828     if (IsKeyDown(KEY_LEFT_SHIFT) || IsKeyDown (KEY_RIGHT_SHIFT)) {
829         new_el -= 10;
830     } else {
831         new_el--;
832     }
833     hm->setEvalLimit (new_el);
834     lowres_hm->setEvalLimit (new_el);
835     parametersWereModified();
836 }
837 
838 /**
839  * @brief Handler for '>' button
840  *
841  */
842 void HFractalGui::evalLimitMore() {
843     int new_el = hm->getEvalLimit();
844     // Allow faster jumping if shift is held
845     if (IsKeyDown(KEY_LEFT_SHIFT) || IsKeyDown (KEY_RIGHT_SHIFT)) {
846         new_el += 10;
847     } else {
848         new_el++;
849     }
850     hm->setEvalLimit (new_el);
851     lowres_hm->setEvalLimit (new_el);
852     parametersWereModified();
853 }
854 
855 /**
856  * @brief Handler for Help & Instructions button
857  *
858  */
859 void HFractalGui::showHelp() {
860     // Open the help page in the repository, cross-platform
861     #ifdef _WIN32
862         system("explorer
https://github.com/JkyProgrammer/HyperFractal/blob/main/README.md#help--
instructions");
863     #else
864         system("open
https://github.com/JkyProgrammer/HyperFractal/blob/main/README.md#help--
```

```
instructions");
865     #endif
866 }
867
868 /**
869  * @brief Handle the user pressing a GUI button
870  *
871  * @return True if a button press was handled, otherwise false
872  */
873 bool HFractalGui::handleButtonPresses() {
874     if (is_rendering) return false;
875     // Branch to different handling modes depending on the dialog state,
allowing certain sets of buttons to be disabled when dialogs are open
876     if (modal_view_state == MODAL_VIEW_STATE::MVS_TEXT_DIALOG) {
877         if (button_states[BUTTON_ID::BUTTON_ID_TEXT_DIALOG_CLOSE]) {
closeTextDialog(); return true; }
878     } else if (modal_view_state == MODAL_VIEW_STATE::MVS_NORMAL) {
879         if (button_states[BUTTON_ID::BUTTON_ID_RENDER]) {
startFullRender(); return true; }
880         if (button_states[BUTTON_ID::BUTTON_ID_ZOOM_IN]) { zoomIn(); return
true; }
881         if (button_states[BUTTON_ID::BUTTON_ID_ZOOM_OUT]) { zoomOut();
return true; }
882         if (button_states[BUTTON_ID::BUTTON_ID_SAVE_IMAGE]) { saveImage();
return true; }
883         if (button_states[BUTTON_ID::BUTTON_ID_SAVE_RSTATE]) {
showSaveStateDialog(); return true; }
884         if (button_states[BUTTON_ID::BUTTON_ID_LOAD_RSTATE]) {
showLoadStateDialog(); return true; }
885         if (button_states[BUTTON_ID::BUTTON_ID_UP]) { moveUp(); return
true; }
886         if (button_states[BUTTON_ID::BUTTON_ID_LEFT]) { moveLeft(); return
true; }
887         if (button_states[BUTTON_ID::BUTTON_ID_RIGHT]) { moveRight();
return true; }
888         if (button_states[BUTTON_ID::BUTTON_ID_DOWN]) { moveDown(); return
true; }
889         if (button_states[BUTTON_ID::BUTTON_ID_EQ_PRESETS]) {
enterEquationPresetDialog(); return true; }
890         if (button_states[BUTTON_ID::BUTTON_ID_ZOOM_RESET]) { resetZoom();
return true; }
891         if (button_states[BUTTON_ID::BUTTON_ID_TOGGLE_COORDS]) {
toggleCoords(); return true; }
892         if (button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_LESS]) {
evalLimitLess(); return true; }
893         if (button_states[BUTTON_ID::BUTTON_ID_EVAL_LIM_MORE]) {
evalLimitMore(); return true; }
```

```
894         if (button_states[BUTTON_ID::BUTTON_ID_HELP]) { showHelp(); return true; }
895         if (button_states[BUTTON_ID::BUTTON_ID_EQ_INPUTBOX]) { textbox_focus = TEXT_FOCUS_STATE::TFS_EQUATION; return true; }
896         if (button_states[BUTTON_ID::BUTTON_ID_CP_PRESETS]) { enterColourPalettePresetDialog(); return true; }
897     } else if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_SAVE_DIALOG) {
898         if (button_states[BUTTON_ID::BUTTON_ID_SAVE_NAME_INPUTBOX]) { textbox_focus = TEXT_FOCUS_STATE::TFS_SAVE_NAME; return true; }
899         if (button_states[BUTTON_ID::BUTTON_ID_SAVE]) { saveStateToDatabase(); return true; }
900         if (button_states[BUTTON_ID::BUTTON_ID_DATABASE_CANCEL]) { closeDatabaseDialog(); return true; }
901     } else if (modal_view_state == MODAL_VIEW_STATE::MVS_DATABASE_LOAD_DIALOG) {
902         if (button_states[BUTTON_ID::BUTTON_ID_LOAD]) { loadStateFromDatabase(); return true; }
903         if (button_states[BUTTON_ID::BUTTON_ID_SCROLL_DOWN]) { databaseLoadScrollDown(); return true; }
904         if (button_states[BUTTON_ID::BUTTON_ID_SCROLL_UP]) { databaseLoadScrollUp(); return true; }
905         if (button_states[BUTTON_ID::BUTTON_ID_DATABASE_CANCEL]) { closeDatabaseDialog(); return true; }
906     }
907
908     return false;
909 }
910
911 /**
912  * @brief Clear the contents of the button states array to prevent unhandled button presses hanging over to the next update
913  *
914  */
915 void HFractalGui::clearButtonStates() {
916     for (int i = 0; i < BUTTON_NUM_TOTAL; i++) {
917         button_states[i] = false;
918     }
919 }
920
921 /**
922  * @brief Unload the image buffer to prevent memory leaks
923  *
924  */
925 void HFractalGui::tryUnloadImage() {
926     UnloadImage (buffer_image);
```

```
927 }
928
929 /**
930  * @brief Unload the texture buffer to prevent memory leaks
931  *
932  */
933 void HFractalGui::tryUnloadTexture() {
934     UnloadTexture (buffer_texture);
935 }
936
937 /**
938  * @brief Handle when the user presses a key
939  *
940  * @return True if a key press was handled, false otherwise
941  */
942 bool HFractalGui::handleKeyPresses() {
943     // Escape currently editing text box when escape is pressed
944     if (IsKeyDown(KEY_ESCAPE)) { textbox_focus =
TEXT_FOCUS_STATE::TFS_NONE; return true; }
945
946     // Handle keys depending on which text box is focussed (if none, use
them for navigation)
947     if (textbox_focus == TEXT_FOCUS_STATE::TFS_NONE) {
948         for (auto key : key_map) {
949             if (IsKeyDown (key.first)) {
950                 button_states[key.second] = true;
951                 return true;
952             }
953         }
954     } else if (textbox_focus == TEXT_FOCUS_STATE::TFS_EQUATION) {
955         if (IsKeyDown(KEY_ENTER)) {
button_states[BUTTON_ID::BUTTON_ID_RENDER] = true; return true; }
956         int key = GetCharPressed();
957         if ((((int)'a' <= key && key <= (int)'c') || ((int)'x' <= key &&
key <= (int)'z') || key == 122 || (key >= 48 && key <= 57) || key == 94 ||
(key >= 40 && key <= 43) || key == 45 || key == 46 || key == 47 || key ==
'i') && !is_rendering) {
958             equation_buffer += (char)key;
959             hm->setEquation (equation_buffer);
960             lowres_hm->setEquation (equation_buffer);
961             if (!hm->isValidEquation()) console_text = "Invalid equation
input";
962             else parametersWereModified();
963         } else if (GetKeyPressed () == KEY_BACKSPACE && !is_rendering &&
```

```
equation_buffer.length() > 0) {
964             equation_buffer.pop_back();
965             hm->setEquation(equation_buffer);
966             lowres_hm->setEquation(equation_buffer);
967             if (!hm->isValidEquation()) console_text = "Invalid equation
input";
968             else parametersWereModified();
969         }
970     } else if (textbox_focus == TEXT_FOCUS_STATE::TFS_SAVE_NAME) {
971         int key = GetCharPressed();
972         if (((int)'a' <= key && key <= (int)'z') || ((int)'A' <= key && key
<= (int)'Z')) {
973             save_name_buffer += (char)key;
974         } else if (GetKeyPressed() == KEY_BACKSPACE) {
975             if (save_name_buffer.length() > 0) save_name_buffer.pop_back();
976         }
977     }
978     return false;
979 }
980
981 /**
982  * @brief Start the GUI and run the mainloop.
983  * Blocks on current thread
984  *
985  * @return Integer showing exit status
986  */
987 int HFractalGui::guiMain(char* path) {
988     // Run the setup code
989     configureGUI(path);
990     parametersWereModified();
991     while(!WindowShouldClose()) { // Loop until the application closes
992         checkWindowResize();
993         if (!is_rendering && modal_view_state == MVS_NORMAL) {
994             bool click_handled = handleClickNavigation();
995             // Defocus the textbox if a click is handled somewhere
996             if (click_handled) { textbox_focus =
TEXT_FOCUS_STATE::TFS_NONE; }
997         }
998         handleKeyPresses();
999         bool button_pressed = handleButtonPresses();
1000        // Defocus the textbox if a button press is handled
1001        if (button_pressed &&
!button_states[BUTTON_ID::BUTTON_ID_EQ_INPUTBOX] &&
```

```
!button_states[BUTTON_ID::BUTTON_ID_SAVE_NAME_INPUTBOX]) { textbox_focus = TEXT_FOCUS_STATE::TFS_NONE; }
1002            clearButtonStates();
1003            // If a render is in progress, update the status of it
1004            if (is_rendering) updateFullRender();
1005            // Finally, draw everything
1006            drawInterface();
1007        }
1008
1009        // Release resources and close
1010        tryUnloadImage();
1011        tryUnloadTexture();
1012        CloseWindow();
1013        return 0;
1014    }
1015
1016    /**
1017     * @brief Construct a new GUI object
1018     *
1019     */
1020    HFractalGui::HFractalGui() {}
1021
1022    /**
1023     * @brief Method to start the GUI, isolates the GUI module from the main module to prevent linker conflicts with raylib
1024     *
1025     * @return Integer showing exit status
1026     */
1027    int guiMain (char* path) {
1028        HFractalGui gui = HFractalGui ();
1029        int res = gui.guiMain(path);
1030        return res;
1031    }
```

```cpp
// src/gui.hh

#ifndef GUI_H
#define GUI_H

#include <map>

#define RAYGUI_IMPLEMENTATION
#define RAYGUI_SUPPORT_ICONS
#include "../lib/raygui.h"
#include "../lib/ricons.h"

#include "hyperfractal.hh"
#include "utils.hh"
#include "database.hh"

#define SCALE_STEP_FACTOR 1.5       // Factor by which scaling changes
#define SCALE_DEPTH_LIMIT 1.0e15    // Limit to prevent user from going too deep due to limited precision
#define MOVE_STEP_FACTOR 0.1        // Factor by which position changes
#define WINDOW_INIT_WIDTH 900       // Initial window - width
#define WINDOW_INIT_HEIGHT 550      //                - height
#define BUTTON_HEIGHT 30            // Height of a single button in the interface
#define ELEMENT_NUM_VERTICAL 15     // Number of vertical elements
#define BUTTON_NUM_TOTAL 25         // Total number of buttons in the interface
#define CONTROL_MIN_WIDTH 400       // Minimum width of the control panel
#define CONTROL_MIN_HEIGHT BUTTON_HEIGHT*ELEMENT_NUM_VERTICAL // Minimum height of the panel
#define DIALOG_TEXT_SIZE 25         // Size of text in dialog windows

// Enum listing button IDs to abstract and make code clearer
enum BUTTON_ID {
    BUTTON_ID_RENDER = 0,
    BUTTON_ID_ZOOM_IN,
    BUTTON_ID_ZOOM_OUT,
    BUTTON_ID_SAVE_IMAGE,
    BUTTON_ID_SAVE_RSTATE,
    BUTTON_ID_LOAD_RSTATE,
    BUTTON_ID_UP,
    BUTTON_ID_LEFT,
    BUTTON_ID_RIGHT,
```

```
    BUTTON_ID_DOWN,
    BUTTON_ID_EQ_PRESETS,
    BUTTON_ID_ZOOM_RESET,
    BUTTON_ID_TOGGLE_COORDS,
    BUTTON_ID_EVAL_LIM_LESS,
    BUTTON_ID_EVAL_LIM_MORE,
    BUTTON_ID_HELP,
    BUTTON_ID_TEXT_DIALOG_CLOSE,
    BUTTON_ID_EQ_INPUTBOX,
    BUTTON_ID_CP_PRESETS,
    BUTTON_ID_SAVE_NAME_INPUTBOX,
    BUTTON_ID_SAVE,
    BUTTON_ID_LOAD,
    BUTTON_ID_SCROLL_DOWN,
    BUTTON_ID_SCROLL_UP,
    BUTTON_ID_DATABASE_CANCEL
};

// Enum listing GUI states for cases when a dialog or modal is open (i.e. to disable certain interface elements)
enum MODAL_VIEW_STATE {
    MVS_NORMAL,
    MVS_TEXT_DIALOG,
    MVS_DATABASE_SAVE_DIALOG,
    MVS_DATABASE_LOAD_DIALOG,
    MVS_EQUATION_PRESET_SELECTOR,
    MVS_COLOUR_PRESET_SELECTOR
};

// Enum listing text focus states to enable/disable input to specific fields
enum TEXT_FOCUS_STATE {
    TFS_NONE,
    TFS_EQUATION,
    TFS_SAVE_NAME
};

// Class managing the GUI environment
class HFractalGui {
private:
    // Lists which keys on the keyboard map to which interface buttons
    std::map<KeyboardKey, BUTTON_ID> key_map = {
        {KEY_ENTER, BUTTON_ID::BUTTON_ID_RENDER},
```

```
81          {KEY_EQUAL, BUTTON_ID::BUTTON_ID_ZOOM_IN},
82          {KEY_MINUS, BUTTON_ID::BUTTON_ID_ZOOM_OUT},
83          {KEY_UP, BUTTON_ID::BUTTON_ID_UP},
84          {KEY_DOWN, BUTTON_ID::BUTTON_ID_DOWN},
85          {KEY_LEFT, BUTTON_ID::BUTTON_ID_LEFT},
86          {KEY_RIGHT, BUTTON_ID::BUTTON_ID_RIGHT},
87          {KEY_LEFT_BRACKET, BUTTON_ID::BUTTON_ID_EVAL_LIM_LESS},
88          {KEY_RIGHT_BRACKET, BUTTON_ID::BUTTON_ID_EVAL_LIM_MORE}
89      };
90
91      HFractalMain* hm; // Pointer to main rendering environment
92      HFractalMain* lowres_hm; // Pointer to an identical rendering
    environment, but with a lower resolution for preview renders
93
94      std::string dialog_text; // Text to show in the text dialog widget
95      std::string console_text; // Text to show in the application console
96      std::string equation_buffer; // Contains the equation being used by both
    renderer classes
97      std::string save_name_buffer; // Contains the text shown/edited in the
    name field in the save render state dialog
98      bool button_states[BUTTON_NUM_TOTAL]; // Contains the current states of
    every button in the GUI (true for pressed, false for not pressed)
99      Image buffer_image; // Image being used by raygui for displaying the
    render result
100     Texture2D buffer_texture; // Texture being used by raygui for displaying
    the render result
101     bool is_rendering; // Stores whether the GUI is currently waiting on a
    full-resolution render (and thus should freeze controls)
102     bool is_outdated_render; // Stores whether the GUI is showing a preview
    render (i.e. needs a full-resolution render to be run by the user)
103     TEXT_FOCUS_STATE textbox_focus; // Stores the currently focussed text
    box
104     int render_percentage; // Stores the percentage completion of the
    current render
105     bool showing_coordinates; // Stores whether coordinates are currently
    being shown on the mouse cursor
106     MODAL_VIEW_STATE modal_view_state; // Stores the current modal state of
    the GUI, allowing certain controls to be enabled and disabled in different
    modes
107     int image_dimension; // Stores the size of the image, used for sizing
    the window, scaling and rendering images, and positioning elements
108     int control_panel_width; // Stores the width of the control panel
109     CP_PRESETS selected_palette; // Determines the colour palette in which
    the GUI is currently displaying the rendered image
110     HFractalDatabase database; // Database which manages saved profile
    states
111     long selected_profile_id; // Records the ID of the profile currently
```

```
    selected in the load render state dialog
112     int database_load_dialog_scroll; // Records the current amount of scroll
    in the load render state dialog
113 
114     void configureStyling(); // Configures the GUI styling from a stylesheet
    provided by raylib's creator as part of the library
115     void configureGUI(char*); // Configures the GUI and initialises all
    class variables ready for the first GUI mainloop update
116 
117     void parametersWereModified(); // Marks the GUI as using an outdated
    render and triggers a preview render update
118     bool updatePreviewRender(); // Rerenders the preview image
119     bool startFullRender(); // Triggers a full resolution render
120     bool updateFullRender(); // Updates the image and texture buffers from
    the partially-finished rendering environment image, and finalises if the
    render has completed
121     void reloadImageFrom(HFractalMain*); // Automatically fetch and reload
    the image and texture buffers from a given rendering environment
122 
123     void checkWindowResize(); // Check to see if the window has been
    resized, and handle it
124 
125     bool handleClickNavigation(); // Check to see if the user has clicked
    somewhere on the image, and jump to focus that location if so
126     bool handleButtonPresses(); // Handle any interface button presses the
    user has made since the last update
127     bool handleKeyPresses(); // Handle any keyboard key presses the user has
    made since the last update
128     void drawInterface(); // Draw the entire interface, called each update
129 
130     Image getImage(HFractalMain*); // Extract image data from a rendering
    environment
131 
132     void enterEquationPresetDialog(); // Show the equation preset selector
    and disable other GUI controls
133     void escapeEquationPresetDialog(int); // Close the equation preset
    selector and return to normal GUI mode
134     void enterColourPalettePresetDialog(); // Show the colour palette preset
    selector and disable other GUI controls
135     void escapeColourPalettePresetDialog(int); // Close the colour palette
    preset selector and return to normal GUI mode
136     void launchTextDialog(std::string); // Show a text dialog over the
    window with a given string as text
137     void closeTextDialog(); // Close the text dialog currently being shown
138 
139     void zoomIn(); // Handler for Zoom In button
140     void zoomOut(); // Handler for Zoom Out button
```

```
141     void resetZoom(); // Handler for Reset Zoom button
142     void saveImage(); // Handler for Save Image button
143 
144     void moveUp(); // Handler for Move Up button
145     void moveLeft(); // Handler for Move Left button
146     void moveRight(); // Handler for Move Right button
147     void moveDown(); // Handler for Move Down button
148 
149     void toggleCoords(); // Handler for Show/Hide Coordinates button
150 
151     void evalLimitLess(); // Handler for '<' button
152     void evalLimitMore(); // Handler for '>' button
153 
154     void showHelp(); // Handler for Help & Instructions button
155     void clearButtonStates(); // Clears current button states to ignore
    unhandled button presses
156 
157     void tryUnloadImage(); // Unload the image buffer, prevents memory leaks
158     void tryUnloadTexture(); // Unload the texture buffer, prevents memory
    leaks
159 
160     void showSaveStateDialog(); // Make the save render state dialog visible
161     void showLoadStateDialog(); // Make the load render state dialog visible
162     void saveStateToDatabase(); // Save the current render state to the
    database
163     void loadStateFromDatabase(); // Load the selected config profile from
    the database to be the current render state
164     void closeDatabaseDialog(); // Hide the save/load render state dialog
165     void databaseLoadScrollDown(); // Scroll down in the load render state
    dialog
166     void databaseLoadScrollUp(); // Scroll up in the load render state
    dialog
167 public:
168     int guiMain(char*); // Start and run the entire GUI. Blocks on current
    thread
169 
170     HFractalGui(); // Basic constructor
171 };
172 
173 #endif
```

```
// src/guimain.hh

#ifndef GUIMAIN_H
#define GUIMAIN_H

// GUI Main function to isolate the GUI module from the main module to prevent linker conflicts
int guiMain(char*);

#endif
```

```cpp
// src/hyperfractal.cc

#include "hyperfractal.hh"

#include <iostream>
#include <iomanip>
#include <chrono>
#include <thread>

#include "utils.hh"

using namespace std;
using namespace std::chrono;

/**
 * @brief Main function called when each worker thread starts. Contains code to actually fetch and render pixels
 *
 */
void HFractalMain::threadMain () {
    // Pre-compute constants to increase performance
    long double p = 2/(zoom*resolution);
    long double q = (1/zoom)-offset_x;
    long double r = (1/zoom)+offset_y;

    // Get the next unrendered pixel
    int next = img->getUncompleted();
    while (next != -1) {
        // Find the x and y coordinates based on the pixel index
        int x = next%resolution;
        int y = next/resolution;
        // Apply the mathematical transformation of offsets and zoom to find a and b, which form a coordinate pair representing this pixel in the complex plane
        long double a = (p*x) - q;
        long double b = r - (p*y);
        // Construct the initial coordinate value, and perform the evaluation on the main equation
        complex<long double> c = complex<long double> (a,b);
        int res = (main_equation->evaluate (c, eval_limit));
        // Set the result back into the image class, and get the next available unrendered pixel
        img->set (x, y, res);
```

```
39          next = img->getUncompleted();
40      }
41
42      // When there appear to be no more pixels to compute, mark this thread as completed
43      thread_completion[std::this_thread::get_id()] = true;
44
45      // Check to see if any other threads are still rendering, if not then set the flag to mark the environment as no longer rendering
46      bool is_incomplete = false;
47      for (auto p : thread_completion) is_incomplete |= !p.second;
48      if (!is_incomplete) is_rendering = false;
49  }
50
51  /**
52   * @brief Generate a fractal image based on all the environment parameters
53   *
54   * @param wait Whether to wait and block the current thread until the image has been fully computed, useful if you want to avoid concurrency somewhere else (functionality hiding)
55   * @return Integer representing status code, 0 for success, else for failure
56   */
57  int HFractalMain::generateImage (bool wait=true) {
58      if (getIsRendering()) { std::cout << "Aborting!" << std::endl; return 2; } // Prevent overlapping renders from starting
59      // Output a summary of the rendering parameters
60      std::setprecision (100);
61      std::cout << "Rendering with parameters: " << std::endl;
62      std::cout << "Resolution=" << resolution << std::endl;
63      std::cout << "EvaluationLimit=" << eval_limit << std::endl;
64      std::cout << "Threads=" << worker_threads << std::endl;
65      std::cout << "Zoom="; printf ("%Le", zoom); std::cout << std::endl;
66      std::cout << "OffsetX="; printf ("%.70Lf", offset_x); std::cout << std::endl;
67      std::cout << "OffsetY="; printf ("%.70Lf", offset_y); std::cout << std::endl;
68
69      // Abort rendering if the equation is invalid
70      if (!isValidEquation()) { std::cout << "Aborting!" << std::endl; return 1; }
71
72      // Mark the environment as now rendering, locking resources/parameters
73      is_rendering = true;
74
```

```
75      // Clear and reinitialise the image class with the requested resolution
76      if (img != NULL) img->~HFractalImage();
77      img = new HFractalImage (resolution, resolution);
78
79      // Clear the thread pool, and populate it with fresh worker threads
80      thread_pool.clear();
81      thread_completion.clear();
82      for (int i = 0; i < worker_threads; i++) {
83          std::thread *t = new std::thread(&HFractalMain::threadMain, this);
84          thread_completion[t->get_id()] = false;
85          thread_pool.push_back(t);
86      }
87
88      // Optionally, wait for the render to complete before returning
89      if (wait) {
90          while (true) {
91              // If enabled at compile time, show a progress bar in the
terminal
92              #ifdef TERMINAL_UPDATES
93              float percent = getImageCompletionPercentage();
94              std::cout << "\r";
95              std::cout << "Working: ";
96              for (int k = 2; k <= 100; k+=2) { if (k <= percent) std::cout <<
"▉"; else std::cout << "_"; }
97              std::cout << " | ";
98              std::cout << round(percent) << "%";
99              #endif
100             // Break out when the image has been fully completed (all pixels
computed)
101             if (img->isDone()) break;
102             crossPlatformDelay (10);
103         }
104         // Wait for all the threads to join, then finish up
105         for (auto th : thread_pool) th->join();
106         is_rendering = false;
107     }
108     std::cout << std::endl << "Rendering done." << std::endl;
109     return 0;
110 }
111
112 /**
113  * @brief Construct a new rendering environment, with blank parameters
114  *
```

```
 */
HFractalMain::HFractalMain () {
    resolution = 1;
    offset_x = 0;
    offset_y = 0;
    zoom = 1;
    img = NULL;
}

/**
 * @brief Convert the raw data stored in the image class into a coloured RGBA 32 bit image using a particular colour scheme preset
 *
 * @param colour_preset The colour scheme to use
 * @return uint32_t* Pointer to the image stored in memory as an array of 4-byte chunks
 */
uint32_t* HFractalMain::getRGBAImage (int colour_preset) {
    // Return a blank result if the image is uninitialised, other conditions should ensure this never occurs
    if (img == NULL) {
        return (uint32_t *)malloc(0);
    }

    // Copy parameters to local
    int size = resolution;
    int limit = eval_limit;

    // Construct a pixel buffer with RGBA channels
    uint32_t *pixels = (uint32_t *)malloc(size*size*sizeof(uint32_t));
    for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
            int v = img->get(x,y);
            pixels[(y*size)+x] = (v == limit) ? 0x000000ff : HFractalImage::colourFromValue(v, colour_preset);

            // If the pixel has not been computed, make it transparent
            if (img->completed[(y*size)+x] != 2) pixels[(y*size)+x] = 0;
        }
    }

    // Return the pointer to the pixel buffer
    return pixels;
```

```
154 }
155
156 /**
157  * @brief Get the percentage of pixels in the image which have been computed
158  *
159  * @return Unrounded percentage
160  */
161 float HFractalMain::getImageCompletionPercentage () {
162     if (img == NULL) return 100;
163     return ((float)(img->getInd())/(float)(resolution*resolution))*100;
164 }
165
166 /**
167  * @brief Automatically write an image to a generated file address.
168  * File path will be dynamically constructed so that the file name is
    unique, timestamped, and placed on the user's desktop
169  *
170  * @param type Image format to write image out to
171  * @return True for success, false for failure
172  */
173 bool HFractalMain::autoWriteImage (IMAGE_TYPE type) {
174     string image_name = "Fractal render from ";
175
176     // Get current system time
177     auto time = system_clock::to_time_t (system_clock::now());
178     string c_time = string (ctime (&time));
179
180     // Separate ctime result into components
181     vector<string> time_components;
182     string current_component = "";
183     for (char c : c_time) {
184         if (c == ' ') {
185             time_components.push_back (current_component);
186             current_component = "";
187         } else if (c != '\n') current_component.push_back (c != ':' ? c :
    '.');
188     }
189     time_components.push_back (current_component);
190
191     // Get milliseconds, not part of ctime
192     system_clock::duration dur = system_clock::now().time_since_epoch();
193     seconds s = duration_cast<seconds> (dur);
```

```
    dur -= s;
    milliseconds ms = duration_cast<milliseconds> (dur);

    // Components are in the form: dayofweek month day hour:minute:second year
    image_name += time_components[2] + " ";
    image_name += time_components[1] + " ";
    image_name += time_components[4] + " ";
    image_name += "at ";
    image_name += time_components[3];
    image_name += ".";
    image_name += to_string(ms.count());

    cout << image_name << endl;

    string image_path = "";

    image_path += getDesktopPath();
    image_path += image_name;

    // Call into the image's writer to write out data
    switch (type) {
    case PGM:
        image_path += ".pgm";
        return img->writePGM (image_path);
    default:
        return false;
    }
}
```

```cpp
// src/hyperfractal.hh

#ifndef HYPERFRACTAL_H
#define HYPERFRACTAL_H

#include <string>
#include <thread>
#include <vector>
#include <map>

#include "image.hh"
#include "fractal.hh"
#include "utils.hh"
#include "equationparser.hh"

// When defined, progress updates will be written to terminal.
#define TERMINAL_UPDATES

// Class defining a fractal rendering environment, fully encapsulated
class HFractalMain {
private:
    int resolution; // Horizontal and vertical dimension of the desired image
    long double offset_x; // Horizontal offset in the complex plane
    long double offset_y; // Vertical offset in the complex plane
    long double zoom; // Scaling value for the image (i.e. zooming in)

    std::string eq; // String equation being used
    HFractalEquation *main_equation; // Actual pointer to the equation manager class being used for computation

    int worker_threads; // Number of worker threads to be used for computation
    int eval_limit; // Evaluation limit for the rendering environment

    HFractalImage *img = new HFractalImage(0,0); // Pointer to the image class containing data for the rendered image

    std::vector<std::thread*> thread_pool; // Thread pool containing currently active threads
    std::map<std::thread::id, bool> thread_completion; // Map of which threads have finished computing pixels
    bool is_rendering = false; // Marks whether there is currently a render ongoing (locking resources to prevent concurrent modification e.g. changing resolution mid-render)
```

```
38
39      void threadMain (); // Method called on each thread when it starts,
contains the worker/rendering code
40
41  public:
42      int generateImage (bool); // Perform the render, and optionally block the
current thread until it is done
43
44      HFractalMain (); // Base initialiser
45
46      int getResolution () { return resolution; } // Inline methods to get/set
the resolution
47      void setResolution (int resolution_) { if (!getIsRendering()) resolution
= resolution_; }
48
49      long double getOffsetX () { return offset_x; } // Inline methods to
get/set the x offset
50      void setOffsetX (long double offset_x_) { if (!getIsRendering()) offset_x
= offset_x_; }
51
52      long double getOffsetY () { return offset_y; } // Inline methods to
get/set the y offset
53      void setOffsetY (long double offset_y_) { if (!getIsRendering()) offset_y
= offset_y_; }
54
55      long double getZoom () { return zoom; } // Inline methods to get/set the
zoom
56      void setZoom (long double zoom_) { if (!getIsRendering()) zoom = zoom_; }
57
58      std::string getEquation () { return eq; } // Inline methods to get/set
the equation
59      void setEquation (std::string eq_) {
60          if (!getIsRendering()) {
61              eq = eq_;
62              main_equation = HFractalEquationParser::extractEquation (eq);
63              if (main_equation == NULL) return;
64              // Detect if the equation matches the blueprint of a preset
65              int preset = -1;
66              for (int i = 0; i < NUM_EQUATION_PRESETS; i++) {
67                  if (eq == equationPreset ((EQ_PRESETS)i, false)) {
68                      preset = i;
69                      break;
70              main_equation->setPreset (preset);
71          }
72      }
```

```
73      main_equation->setPreset (preset);
74          }
75      }
76
77      int getWorkerThreads () { return worker_threads; } // Inline methods to
get/set the number of worker threads
78      void setWorkerThreads (int wt_) { if (!getIsRendering()) worker_threads =
wt_; }
79
80      int getEvalLimit () { return eval_limit; } // Inline methods to get/set
the evaluation limit
81      void setEvalLimit (int el_) { if (!getIsRendering()) eval_limit = el_; }
82
83      bool isValidEquation () { return main_equation != NULL; } // Check if the
equation the user entered was parsed correctly last time it was set
84
85      bool getIsRendering() { return is_rendering; } // Get if there is
currently a render happening in this environment
86
87      uint32_t* getRGBAImage (int); // Return a pointer to a 32 bit RGBA
formatted image, produced using a particular colour scheme preset, from the
generated image
88
89      float getImageCompletionPercentage (); // Get the current percentage of
pixels that have been actually computed
90
91      bool autoWriteImage (IMAGE_TYPE); // Automatically write out the render
to desktop using a particular image type
92  };
93  #endif
```

```cpp
// src/image.cc

#include "image.hh"

#include <ostream>
#include <math.h>

/**
 * @brief Set the value of a pixel, and automatically mark it as complete
 *
 * @param x Horizontal coordinate
 * @param y Vertical coordinate
 * @param p Value of the pixel to assign
 */
void HFractalImage::set(int x, int y, uint16_t p) {
    int offset = ((y*width)+x);
    data_image[offset] = p;
    completed[offset] = 2;
}

/**
 * @brief Get the value of the pixel at the specified coordinates, as measured from top-left
 *
 * @param x Horizontal coordinate
 * @param y Vertical coordinate
 * @return The value of the pixel at the coordinates
 */
uint16_t HFractalImage::get(int x, int y) {
    return data_image[(y*width)+x];
}

/**
 * @brief Initialise a new image with a specified width and height
 *
 * @param w Horizontal size
 * @param h Vertical size
 */
HFractalImage::HFractalImage(int w, int h) {
    width = w;
    height = h;
    c_ind = 0;
```

```
42      data_image = new uint16_t[width*height];
43      completed = new uint8_t[width*height];
44      // Clear both buffers
45      for (int i = 0; i < width*height; i++) { data_image[i] = 0xffff;
completed[i] = 0; }
46  }
47
48  /**
49   * @brief Write the contents of the image buffer out to a PGM file (a
minimal image format using grayscale linear colour space)
50   *
51   * @param path Path to the output file
52   * @return True for success, false for failure
53   */
54  bool HFractalImage::writePGM (std::string path) {
55      // Abort if the image is incomplete
56      if (!isDone()) return false;
57      FILE *img_file;
58      img_file = fopen(path.c_str(),"wb");
59
60      // Write the header
61      fprintf(img_file,"P5\n");
62      fprintf(img_file,"%d %d\n",width,height);
63      fprintf(img_file,"65535\n");
64
65      // Write each pixel
66      for(int y = 0; y < height; y++){
67          for(int x = 0; x < width; x++){
68              uint16_t p = data_image[(y*width)+x];
69
70              fputc (p & 0x00ff, img_file);
71              fputc ((p & 0xff00) >> 8, img_file);
72          }
73      }
74
75      // Close and return success
76      fclose(img_file);
77      return true;
78  }
79
80  /**
81   * @brief Create an RGBA 32 bit colour from hue, saturation, value
components
```

```
 *
 * @param h Hue value
 * @param s Saturation value
 * @param v Value (brightness) value
 * @return A 32 bit colour in RGBA form
 */
uint32_t HFractalImage::HSVToRGB (float h, float s, float v) { // TODO: Test this
    float c = v * s;
    float h_ = fmod(h,1)*6;
    float x = c * (1 - fabsf(fmodf(h_, 2)-1));
    float m = v - c;

    float r;
    float g;
    float b;

    if (h_ >= 0 && h_ < 1) {
        r = c; g = x; b = 0;
    } else if (h_ < 2) {
        r = x; g = c; b = 0;
    } else if (h_ < 3) {
        r = 0; g = c; b = x;
    } else if (h_ < 4) {
        r = 0; g = x; b = c;
    } else if (h_ < 5) {
        r = x; g = 0; b = c;
    } else if (h_ < 6) {
        r = c; g = 0; b = x;
    }

    r = r + m;
    g = g + m;
    b = b + m;

    uint32_t final_value = 0x000000ff;
    final_value |= (int)(r*255) << (8*3);
    final_value |= (int)(g*255) << (8*2);
    final_value |= (int)(b*255) << (8*1);

    return final_value;
}
```

```
123
124 /**
125  * @brief Convert a computed value (i.e. from the image_data buffer) into a
    renderable RGBA 32 bit colour
126  *
127  * @param value The value to convert
128  * @param colour_preset Colour palette preset to map colour onto
129  * @return The converted colour as a 32 bit integer
130  */
131 uint32_t HFractalImage::colourFromValue (uint16_t value, int colour_preset)
    {
132     uint32_t col = 0x000000ff;
133     if (colour_preset == 0) {
134         col |= 0x3311ff00;
135         col |= ((value % 256) << (8*3)) + 0x33000000;
136     } else if (colour_preset == 1) {
137         uint8_t looped = 255-(uint8_t)(value % 256);
138         col |= looped << (8*3);
139         col |= (2*looped) << (8*2);
140         col |= 0x00007000;
141     } else if (colour_preset == 2) {
142         float hue = (float)value/(float)0xffff;
143         hue = fmod(512*hue, 1);
144         col = HFractalImage::HSVToRGB (hue, 0.45, 0.8);
145     } else if (colour_preset == 3) {
146         uint8_t looped = 255-(uint8_t)(value % 256);
147         col |= looped << (8*1); // B
148         col |= looped << (8*2); // G
149         col |= looped << (8*3); // R
150     } else if (colour_preset == 4) {
151         uint8_t looped = (uint8_t)(value % 256);
152         col |= looped << (8*1); // B
153         col |= looped << (8*2); // G
154         col |= looped << (8*3); // R
155     }
156
157
158     return col;
159 }
160
161 /**
162  * @brief Destroy the image class, freeing the buffers
```

```
163  */
164 HFractalImage::~HFractalImage () {
165     free (data_image);
166     free (completed);
167 }
168 
169 /**
170  * @brief Fetch the index (i.e. (y*width)+x) of the next pixel which needs
    to be computed
171  *
172  * @return The index of the next pixel to compute, -1 if there is no
    available pixel
173  */
174 int HFractalImage::getUncompleted () {
175     // Lock resources to prevent collisions
176     mut.lock();
177     int i = -1;
178     // Find the next available pixel index and increment c_ind
179     if (c_ind < height*width) {
180         i = c_ind;
181         c_ind++;
182     }
183     // Unlock before returning
184     mut.unlock();
185     return i;
186 }
187 
188 /**
189  * @brief Check every pixel to see if the image is fully computed. Use with
    caution, especially with large images
190  *
191  * @return True if the image is complete, false otherwise
192  */
193 bool HFractalImage::isDone () {
194     // Iterate over every pixel to check its status
195     for (int i = 0; i < height*width; i++) {
196         if (completed[i] != 2) {
197             // Fail if the pixel is not complete
198             return false;
199         }
200     }
201     // Succeed if every pixel is fully computed
202     return true;
```

```
}

/**
 * @brief Get the current completion index of the image
 *
 * @return The current completion index
 */
int HFractalImage::getInd () { return c_ind; }
```

```
1  // src/image.hh
2
3  #ifndef IMAGE_H
4  #define IMAGE_H
5
6  #include <mutex>
7
8  // Class containing information about an image currently being generated
9  class HFractalImage {
10 private:
11     int width; // Width of the image
12     int height; // Heigh of the image
13     uint16_t * data_image; // Computed data values of the image
14     int c_ind = 0; // Index of the next pixel to be sent out to a rendering
   thread
15     std::mutex mut; // Mutex object used to lock class resources during
   multi-threading events
16
17 public:
18     HFractalImage (int, int); // Constructor, creates a new image buffer of
   the specified size
19     ~HFractalImage (); // Destructor, destroys and deallocates resources used
   in the current image
20
21     void set (int, int, uint16_t); // Set the value of a pixel
22     uint16_t get (int, int); // Get the value of a pixel
23     uint8_t * completed; // Stores the completion status of each pixel, 0 =
   not computed, 1 = in progress, 2 = computed
24     int getUncompleted (); // Get the index of an uncomputed pixel, to be
   sent to a rendering thread, and update completion data
25     bool isDone (); // Check if the image has been completed or not
26     int getInd (); // Get the current completion index
27     bool writePGM (std::string); // Write out the contents of the data buffer
   to a simple image file, PGM format, with the given path
28
29     static uint32_t HSVToRGB (float h, float s, float v); // Create a 32 bit
   RGB colour from hue, saturation, value components
30     static uint32_t colourFromValue (uint16_t, int); // Convert a computed
   value into a 32 bit RGBA colour value, using the specified palette
31 };
32
33 #endif
```

```cpp
// src/main.cc

#include <iostream>

#include "hyperfractal.hh"
#include "guimain.hh"
#include "utils.hh"

using namespace std;

/**
 * Naming Convention:
 * Classes & Structs - CapitalisedCamelCase
 * Variables - snake_case
 * Functions - uncapitalisedCamelCase
 * Constants - SCREAMING_SNAKE_CASE
 *
 **/

int main (int argc, char *argv[]) {
    if (argc == 8) {
        // If we have the required arguments, run a console-only render
        HFractalMain hm;
        int argument_error = 0;
        try {
            hm.setResolution (stoi (argv[1]));
            if (hm.getResolution() <= 0) throw runtime_error("Specified resolution too low.");
            argument_error++;
            hm.setOffsetX (stod (argv[2]));
            argument_error++;
            hm.setOffsetY (stod (argv[3]));
            argument_error++;
            hm.setZoom (stod (argv[4]));
            argument_error++;
            hm.setEquation (string (argv[5]));
            if (!hm.isValidEquation()) throw runtime_error("Specified equation is invalid.");
            argument_error++;
            hm.setWorkerThreads (stoi (argv[6]));
            if (hm.getWorkerThreads() <= 0) throw runtime_error("Must use at least one worker thread.");
```

```
            argument_error++;
            hm.setEvalLimit (stoi (argv[7]));
            if (hm.getEvalLimit() <= 0) throw runtime_error("Must use at least one evaluation iteration.");
            argument_error++;
            hm.generateImage(true);
            return !hm.autoWriteImage (IMAGE_TYPE::PGM);
        } catch (runtime_error e) {
            cout << "Parameter error on argument number " << argument_error << ":" << endl;
            cout << "   " << e.what() << endl;
            return 1;
        }
    } else if (argc != 1) {
        // If we have only some arguments, show the user what arguments they need to provide
        cout << "Provide all the correct arguments please:" << endl;
        cout << "int resolution, long double offset_x, long double offset_y, long double zoom, string equation, int worker_threads, int eval_limit" << endl;
        return 1;
    } else {
        // Otherwise, start the GUI
        cout << trimExecutableFromPath(argv[0]) << endl;
        return guiMain(trimExecutableFromPath(argv[0]));
    }
}
```

```cpp
// src/utils.cc

#include "utils.hh"

#ifdef _WIN32
#include <windows.h>
#include <shlobj.h>
#else
    #include <unistd.h>
#endif

using namespace std;

/**
 * @brief Get the details of an equation preset
 *
 * @param p The preset to return
 * @param t True - return the name of the preset, False - return the string equation of it instead
 * @return Either the name or the equation string of the equation preset
 */
string equationPreset (EQ_PRESETS p, bool t) {
    switch (p) {
    case EQ_MANDELBROT:
        return t ? "Mandelbrot" : "(z^2)+c";
    case EQ_JULIA_1:
        return t ? "Juila 1" : "(z^2)+(0.285+0.01i)";
    case EQ_JULIA_2:
        return t ? "Julia 2" : "(z^2)+(-0.70176-0.3842i)";
    case EQ_RECIPROCAL:
        return t ? "Reciprocal" : "1/((z^2)+c)";
    case EQ_ZPOWER:
        return t ? "Z Power" : "(z^z)+(c-0.5)";
    case EQ_BARS:
        return t ? "Bars" : "z^(c^2)";
    case EQ_BURNINGSHIP_MODIFIED:
        return t ? "Burning Ship Modified" : "((x^2)^0.5-((y^2)^0.5)i)^2+c";
    default:
        return "NONE";
    }
    return "";
}
```

```
42
43 /**
44  * @brief Return the name of a colour palette preset
45  *
46  * @param p Preset to return
47  * @return The string name of the colour palette
48  */
49 string colourPalettePreset (CP_PRESETS p) {
50     switch (p) {
51     case CP_VAPORWAVE:
52         return "Vaporwave";
53     case CP_YELLOWGREEN:
54         return "Yellow-Green";
55     case CP_RAINBOW:
56         return "Rainbow";
57     case CP_GREYSCALE_BRIGHT:
58         return "Greyscale Bright";
59     case CP_GREYSCALE_DARK:
60         return "Greyscale Dark";
61     default:
62         return "NONE";
63     }
64     return "";
65 }
66
67 /**
68  * @brief Wrap text given a certain line length
69  *
70  * @param s String to wrap
71  * @param line_length Numbere of characters which limit the length of the
    line
72  * @return string
73  */
74 string textWrap (string s, int line_length) {
75     string output = "";
76     int line_offset = 0;
77     for (char c : s) {
78         if (c == '\n') line_offset = -1;
79         if (line_offset == line_length-1) { if (c != ' ') output += "-";
    output += "\n"; line_offset = 0; }
80         if (!(line_offset == 0 && c == ' ')) {
81             output += c;
```

```
            line_offset++;
        }
    }
    return output;
}

/**
 * @brief Get the desktop path of the user
 *
 * @return The user's desktop path
 */
string getDesktopPath () {
    #ifdef _WIN32
    static char path[MAX_PATH+1];
    if (SHGetSpecialFolderPathA(HWND_DESKTOP, path, CSIDL_DESKTOP, FALSE))
        return string(path) + string("\\");
    else
        return "";
    #else
    return string(getenv ("HOME")) + string("/Desktop/");
    #endif
}

/**
 * @brief Delay for a number of millisecods, across any platform
 *
 * @param milliseconds Time to delay
 */
void crossPlatformDelay(int milliseconds) {
    #ifdef _WIN32
        Sleep(milliseconds);
    #else
        usleep(milliseconds * 1000);
    #endif
}

/**
 * @brief Trim the executable name from the end of the path, returning just the working directory
 *
 * @param path Input path from command line arguments
 * @return Trimmed path
```

```
 */
char* trimExecutableFromPath(char* path) {
    int str_len = 0;
    while (true) {
        if (path[str_len] == '\0') break;
        str_len++;
    }

    uint32_t slash_index = -1;
    for (int i = str_len-1; i >= 0; i--) {
        if (path[i] == '\\' || path[i] == '/') {
            slash_index = i;
            break;
        }
    }

    char* result = (char*)malloc(slash_index+2);

    for (int i = 0; i < slash_index+1; i++) {
        result[i] = path[i];
    }
    result[slash_index+1] = '\0';

    return result;
}
```

```cpp
// src/utils.hh

#ifndef UTILS_H
#define UTILS_H

#include <string>

#define NUM_EQUATION_PRESETS 7
#define NUM_COLOUR_PRESETS 5

// Wrap text along a line length
std::string textWrap (std::string, int);

// Enum describing possible equation presets
enum EQ_PRESETS {
    EQ_MANDELBROT = 0, // "(z^2)+c"
    EQ_JULIA_1, // "(z^2)+(0.285+0.01i)"
    EQ_JULIA_2, // "(z^2)+(-0.70176-0.3842i)"
    EQ_RECIPROCAL, // "1/((z^2)+c)"
    EQ_ZPOWER, // "(z^z)+(c-0.5)"
    EQ_BARS, // "z^(c^2)"
    EQ_BURNINGSHIP_MODIFIED // "((x^2)^0.5-((y^2)^0.5)i)^2+c"
};

// Enum describing possible colour palette presets
enum CP_PRESETS {
    CP_VAPORWAVE = 0,
    CP_YELLOWGREEN,
    CP_RAINBOW,
    CP_GREYSCALE_BRIGHT,
    CP_GREYSCALE_DARK
};

// Enum describing available image types which can be saved to disk
enum IMAGE_TYPE {
    PGM
};

// Delay for a given number of milliseconds
void crossPlatformDelay (int);

```

```
42 // Get the user's desktop path
43 std::string getDesktopPath ();
44
45 // Get information about an equation preset
46 std::string equationPreset (EQ_PRESETS, bool);
47 // Get information about a colour palette preset
48 std::string colourPalettePreset (CP_PRESETS);
49
50 // Get the current working directory from the path argument
51 char* trimExecutableFromPath (char*);
52
53 #endif
```