Libigl analysis

Felix An

Libigl is a C++ library for geometry processing tasks, such as computing distances, angles, and area ratios between surfaces, as well as performing mesh editing operations and optimization. It was developed by Alec Jacobson et al. at the University of Toronto. It is built on top of the Eigen linear algebra library, which provides support for efficient computations with matrices and vectors. Libigl has a lot of data structures for manipulating 3D models. The libigl library provides classes for storing and manipulating 3D meshes, plus representing curves and surfaces. libigl also provides algorithms for computing distances, angles, area ratios between surfaces, mesh optimization and editing. Several techniques from the field of computational geometry were used to implement the algorithms.

Libigl provides a number of classes for storing and manipulating geometric models. These include classes for 3D meshes, curves, and surfaces, as well as supporting data structures such as matrices and vectors. Let’s take a look at the corresponding .cpp file igl/opengl/glfw/Viewer.cpp.

# Analysis of Viewer.cpp

The Viewer class defined in this file, part of the libigl library, a collection of tools for geometry processing in C++, provides a GUI for viewing 3D models using the OpenGL and GLFW libraries. It provides functions for handling user input and rendering the scene, as well as various other tasks such as loading and saving models and computing geometric quantities.

The interface is implemented using the OpenGL and GLFW libraries, which allow the viewer to render 3D graphics and handle user input. The Viewer class also uses several functions from other parts of the libigl library, such as igl::project and igl::read\_triangle\_mesh, to perform tasks such as projecting 3D points onto the screen and reading in triangle mesh models from file. The file also includes code for handling various events, such as mouse and keyboard input, using the GLFW library.

The code begins by including several headers for various libraries and other parts of the libigl library. The first header, "Viewer.h", is likely the header file for the Viewer class being defined in this file. The next few headers, such as <chrono> and <thread>, provide standard C++ functionality for working with time and threads. The <Eigen/LU> header includes functionality for performing linear algebra operations using the Eigen library, which is a popular C++ library for numerical computing. The "gl.h" header includes the basic OpenGL functions, while "report\_gl\_error.h" likely contains functions for checking and reporting any errors that occur while using OpenGL. The "GLFW/glfw3.h" header includes functions for creating windows and handling user input using the GLFW library, which is a popular library for creating windows and handling user input in OpenGL applications. The remaining headers provide various functionality for tasks such as reading and writing geometry files, computing geometric quantities, displaying dialogs for opening and saving files, and handling quaternions and trackballs for 3D rotation.

The next block of code defines several global variables used for handling events in the viewer using the GLFW library. The "\_\_viewer" variable is a pointer to the Viewer instance being used by the application, while "highdpi" and the "scroll\_x" and "scroll\_y" variables are used for handling high-resolution displays and mouse scrolling, respectively. Four functions are then defined that are used as callbacks for handling events in the viewer. The "glfw\_mouse\_press" function is called when a mouse button is pressed or released, and it translates the button and action arguments into the corresponding values for the Viewer class and calls the appropriate "mouse\_down" or "mouse\_up" function. The "glfw\_error\_callback" function is called when an error occurs in the GLFW library, and it simply prints the error description to stderr, the standard error output. The "glfw\_char\_mods\_callback" function is called when a character is input with certain modifier keys pressed (which are keys such as function keys and the control key), and it passes the character code and modifier information to the Viewer's "key\_pressed" function. The "glfw\_key\_callback" function is called when a key is pressed or released, and it translates the key and action arguments into the corresponding values for the Viewer class and calls the appropriate "key\_down" or "key\_up" function. The "glfw\_window\_size" function is called when the size of the viewer window is changed, and it updates the dimensions of the OpenGL viewport to match the new window size. The "glfw\_scroll\_callback" function is called when the mouse scroll wheel is used, and it updates the "scroll\_x" and "scroll\_y" variables with the amount of scrolling that occurred. Similarly, the "glfw\_mouse\_move" function is called when the mouse is moved, and it passes the mouse position to the Viewer's "mouse\_move" function.

In the next part of the code, the "Viewer" class is defined. It appears to be a subclass of the "igl::opengl::glfw::Viewer" class from the libigl library. The class has several member variables, including arrays for storing vertex positions and colors, a 4x4 model-view-projection matrix for transforming 3D points to screen coordinates, and various flags and parameters for controlling the appearance and behavior of the viewer. The Viewer class has several member functions for initializing and interacting with the viewer. Looking at one of them, one can see that the "init" function initializes the viewer by creating a window using the GLFW library and setting up the OpenGL context. It also sets up the event handling callbacks defined earlier in the file and initializes the viewer's internal state.

The "launch" function is the main loop of the viewer, which continually renders the scene and processes user input until the window is closed. It starts by setting up the OpenGL viewport and clearing the color and depth buffers. It then calls the "draw" function, which is responsible for rendering the scene. This function sets up the projection matrix and model-view matrix and then calls OpenGL functions to draw the vertices and edges of the 3D model being viewed.

After the scene has been drawn, the "launch" function processes any pending user input events using the GLFW library and then waits for a short time to prevent the application from using too much CPU time.

The Viewer class also has several functions for handling user input events, including "mouse\_down", "mouse\_up", "mouse\_move", "key\_down", and "key\_up". One can see that these functions are called by the event handling callbacks defined earlier in the file and update the internal state of the viewer class in response to user actions. The "mouse\_down" function might start a drag operation when the left mouse button is pressed, while the "mouse\_move" function might update the position of the camera in response to mouse movements during a drag.

The Viewer class can also load and save 3D models from files, compute geometric quantities, and display dialogs for opening and saving files. Various functions are defined to perform these tasks.

# Analysis of libigl sample code

<https://github.com/libigl/libigl-example-project/blob/main/main.cpp>

This C++ program creates an inline mesh of a cube and displays it using the Viewer class from the libigl library.

The program begins by defining the vertices and faces of the cube. The vertices are stored in the matrix V, and the faces are stored in the matrix F.

Next, the program creates an instance of the igl::opengl::glfw::Viewer class and uses it to display the mesh. The set\_mesh() method is used to set the vertices and faces of the mesh, and the set\_face\_based() method is used to specify that the faces of the mesh are defined by a list of vertices.

Finally, the launch() method is called to start the viewer and display the mesh.

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. It is developed with the goal of providing high-level performance and ease of use.

In the sample code, Eigen is used to create the matrices V and F, which represent the vertices and faces of the mesh of a cube. The V matrix is created using an initializer list and the finished() function, which converts it to an Eigen matrix. The F matrix is created in a similar way.

Eigen matrices are used extensively in the libigl library for storing and manipulating geometric data. The library provides functions for operations such as matrix decompositions, eigenvalue and eigenvector computation, and numerical solvers. It also provides support for various matrix storage layouts and efficient operations on sparse matrices.