

Documentation

GaeGebra is a coordinate geometry application inspired by GeoGebra. It was written in C99 using SDL2 and uses the AddressSanitizer. Here is a short documentation, but if you want to dive deeper into the source code, you can find a pdf in the docs folder that contains a short description about the functions and structures.

Coordinate geometry

Coordinate system

The first step is to create a coordinate system with `coordinate_system_create`. Then you can add shapes to it that will be stored and freed in the end. You need to call `coordinate_system_draw` to draw it, and call `coordinate_system_destroy` in the end to clean up. There are some other 'methods' that you can use:

- `coordinate_system_is_hovered`
- `coordinate_system_get_hovered_shape`
- `coordinate_system_translate`
- `coordinate_system_zoom`
- `coordinate_system_update_dimensions`

Sometimes, converting points from screen space into coordinate system space can be useful and vica versa. This can be achieved with the following functions:

- `screen_to_coordinates`
- `coordinates_to_screen`

Shapes

There are a number of shapes that can be created:

- Point
- Line
- Circle
- Parallel
- Perpendicular
- Angle Bisector
- Tangent

To create a one, call `[SHAPE]_create` with the name of the shape. The shape will be automatically freed in the end by the parent coordinate system.

API

Project structure

The project is broken down into modules, and the SDL code is *completely* abstracted away:

- App
- Window
- Input
- UI
- Renderer
- Texture
- Font
- Color

The most important ones are presented here:

App

```
Vector<Window*>  
double DeltaTime
```

The application is the base for everything. The app manages the windows that are added to it, and also keep track of the elapsed time between frames. It provides the highest level of abstraction.

Window

```
SDL_Window*  
SDL_Renderer*  
InputData  
UIData
```

You can add multiple windows to the application. Each window has its own `SDL_Window`, `SDL_Renderer`, `InputData` and `UIData`. Before rendering to a window, or retrieving its input state, the window has to be set as the target for the application.

InputData

```
CurrentMouseButtonState  
OldMouseButtonState  
CurrentMousePosition  
OldMousePosition  
CurrentKeyboardState  
OldKeyboardState
```

The `InputData` holds all the key and mouse states for the current and the previous frame. It allows you to listen to 'press', 'hold' and 'release' events. Every window has an `InputData`.

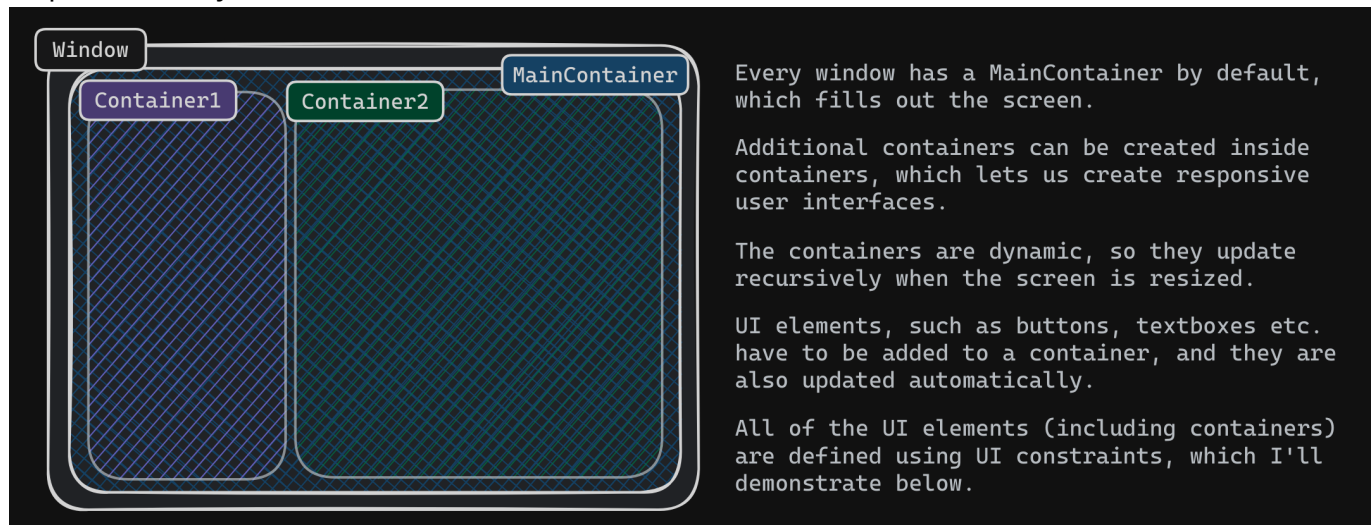
UIData

```
MainContainer  
char TextInput[]  
bool BackspacePressed  
bool MouseCaptured  
ExpandedSplitButton
```

The `UIData` stores information about the current frame for the UI system. `BackspacePressed` and `TextInput` is for the Textboxes, and `MouseCaptured` is needed for not handling clicks twice, and for setting hover states correctly. The expanded Splitbutton is also stored.

GUI

Obviously, a coordinate geometry app would be nothing without a GUI! So I wrote a library that allows you to create basic UI elements easily. The UI is responsive, everything is updated automatically. To make use of responsiveness, you need to use **Containers**:



UI Elements

There are a number of UI elements supported by the library:

- Button
- ImageButton
- Textbox
- Label
- Panel
- Checkbox
- Slider
- SplitButton
- DropDownList



The API is designed in such a way, that it is as easy as possible to create a simple, but fully functional GUI.

To define an element, I chose to use a constraints system:

The constraint system allows you to define UI elements in a way that their positions and sizes are automatically updated when the parent container is resized or moved.

So I designed a flexible, yet easy-to-use API for creating nice, responsive GUIs, which works as follows:

Each UI element has 4 constraints (2 for position, 2 for size)

There are 5 different constraint types:

Pixel Constraint:	it is given in pixels, relative to the parent can be both positional and size constraint can be negative (measured from the opposite side)
Center Constraint:	it places the element in the center of the parent can only positional constraint the value does nothing
Relative Constraint:	it is a ratio, relative to the parent can be both positional and size constraint can only be positive (usually 0-1 decimal)
Offset Constraint:	the offset from the previously added element in pixels can only be positional should only be positive (negative works but not recommended)
Aspect Constraint:	it is a ratio of the other size constraint can only be set for size (and only for one of the two) should only be positive (negative works but not recommended)

How to use

To download the app, execute the following commands:

```
git clone https://github.com/mandliors/GaeGebra
mkdir build;cd build
cmake ..
```

If you want to build and run, you can use the start.sh script in the root directory by calling `./start.sh`