1. *Download the sources from GitHub.*

   This is ideally done with the *git* VCS, however grabbing a zipped snapshot of the repository head via HTTP is an option for users lacking a copy of *git*. Choose the appropriate method for your environment and execute *one* of the following the commands in a shell:

   - `git clone https://github.com/liballeg/allegro5.git`
   - `wget https://github.com/liballeg/allegro5/archive/refs/heads/master.zip`

   If the second method was chosen, unzip and rename the sources:

   - `unzip -q master.zip && mv allegro5-master allgero5 && rm master.zip`

2. *Navigate into the sources and create the build directory.*

   - `cd allegro5 && mkdir build && cd build`

3. *Configure the sources to determine the availability of dependencies on your system.*

   - `cmake ..`

   Always review the configuration summary for any obvious mistakes. Most reasonable systems will have the majority of necessary libraries installed, however users running minimal installations, such as kernels cloned directly from `kernel.org`, or those without a windowing system, may have to do additional work to satisfy the minimum set of dependencies. Some libraries perform similar functions, such as `OSS`, `PulseAudio`, and `ALSA`, so users needn't worry if only one of these dependencies is satisfied.

4. *Build the sources.*

   - `make`

   At this stage, all temporary build files have been generated by the configuration script, so no errors should occur. Compilation and linking is a CPU-intensive job, especially for large libraries like Allegro, so this operation may take a while[1].

5. **(Optional: requires root access.)** *Install the binaries.*

   For users wishing to configure a more permanent Allegro development environment, it is highly recommended to place the generated shared object files, which currently exist inside `build/lib`, inside the standard system path. Symbolic links are generated to support the standard UNIX semantic versioning schema. By installing the sources, they are being placed into the system library directory; typically `/usr/lib`, `/usr/lib64`, or `/usr/local/lib` for easy use by any linker operating on the filesystem. This is especially important in the case of dynamic linking.

   - `make install`

   If this stage is not completed, users wishing to link to Allegro must explicitly specify the absolute location of the `build/` directory.

6. *Building a Test Program with Allegro.*

   At this stage, the development headers and library binaries exist somewhere on your system. If you completed the `make install` stage, the headers reside in `/usr/local/include` and the binaries reside in `/usr/local/lib`. If the installation stage was not completed, the headers exist in the original repository's `include/allegro5` directory, with the binaries in `build/lib`. Now we must perform two stages: informing the runtime linker to the location of the shared objects (for dynamic linking), and provide the include and build directories to the linker at compiletime.

[1] 6m:50s on an Intel® Core™ 2 P8400 @ 2.26GHz and 0m:26s on an Intel® Core™ i7-5820K @ 3.30GHz. Users familiar with `make` may wish to make use of the `-j` parallelisation option to significantly reduce build time; use `grep cpu\ cores -m 1 /proc/cpuinfo` to determine the number of available CPU cores.

(a) To allow for runtime linking, the linker must be aware of the binaries' location. This is done by appending the `/usr/local/lib` or absolute `build/lib` path to the environment variable `LD_LIBRARY_PATH`. Assuming Bash, enter the following commands to permanently update the environment:

i. `echo 'LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/lib"' >> ~/.bashrc`

ii. `source ~/.bashrc`

(b) Alter the compilation command to include the aforementioned hints. This will need to be replicated for each build. Remember to alter `-I` and `-L` accordingly to match your Allegro installation; `test.c` should be replaced with your real source files[2].

- `gcc test.c -I/usr/local/include -L/usr/local/lib -lallegro`

You should now be able to run your Allegro-dependent program on your system by executing `./a.out`. To report any errors in this document, please e-mail `od641@york.ac.uk`[3].

[2]The CFLAGS build path can also be attained with `PKG_CONFIG_PATH=/usr/local/lib/pkgconfig/ pkg-config --cflags --libs allegro-5`, altering `PKG_CONFIG_PATH` accordingly. For makefiles, calculating the compiler flags with `pkg-config` is always the preferred method, as the `.pc` files update with the library.

[3]X11, OpenGL, and ALSA dependency failures may occur, but please do not report them to this e-mail address, as there are too many distributions and package manager arrangements to cover everything in this short document. Someone on-line will have had the same problem as you.