Lab 0: Introduction to CMake and ImGUI

Instructor: Ojaswa Sharma
TA: Aarya Patel

August 16, 2024

Introduction to CMake

CMake is an open-source, cross-platform family of tools designed to build, test, and package software. CMake is controlled by writing simple configuration files called 'CMakeLists.txt', which are used to generate standard build files (e.g., Makefiles on Unix or project files on Windows).

Purpose of CMake

The main purpose of CMake is to manage the build process in a platform-independent manner. It provides a simple way to define how to compile and link your program, managing dependencies between source files and libraries. CMake can also be used to automate testing and packaging of software, making it a powerful tool in the software development process.

Compiling C++ Code using CMake

To compile C++ code using CMake, follow these steps:

1. Create a directory for your project and navigate into it.
2. Create a 'CMakeLists.txt' file in the root of your project directory.
3. Write your C++ source code and save it in the project directory.
4. Run CMake to generate the necessary build files (e.g., Makefiles in Linux and project files in Windows).
5. Use the generated build files to compile your project.

Example: Suppose you have a simple C++ file named 'main.cpp':

```
// main.cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

To compile this using CMake, you'll need to write a 'CMakeLists.txt' file.

Writing a CMake File

The 'CMakeLists.txt' file contains the instructions for CMake on how to build your project. Below is an example of a basic 'CMakeLists.txt' file:

```
# Specify the minimum version of CMake required
cmake_minimum_required(VERSION 3.10)

# Set the project name
project(HelloWorld)

# Specify the C++ standard
set(CMAKE_CXX_STANDARD 11)

# Add the executable
add_executable(hello_world src/main.cpp)
```

This ‘CMakeLists.txt‘ file does the following:

- `cmake_minimum_required(VERSION 3.10)`: Specifies the minimum version of CMake required to build the project.
- `project(HelloWorld)`: Sets the name of the project.
- `set(CMAKE_CXX_STANDARD 11)`: Specifies that the C++11 standard should be used for compilation.
- `add_executable(hello_world main.cpp)`: Tells CMake to create an executable named ‘hello_world‘ from the ‘main.cpp‘ source file.

Building the Project

Once the ‘CMakeLists.txt‘ file is set up, you can build the project:

1. Create a build directory:

   ```
   mkdir build
   cd build
   ```

2. Run CMake to generate the build files:

   ```
   cmake ..
   ```

3. Compile the project:

   ```
   make
   ```

4. Run the executable:

   ```
   ./hello_world
   ```

This process will compile your C++ code and produce an executable file named ‘hello_world‘.

1 ImGui

Dear ImGui is a graphical user interface library that allows developers to quickly and easily add GUI elements to their applications. It’s particularly useful in computer graphics applications for the following reasons:

1. **Rapid Prototyping:** ImGui is designed to help developers quickly create and modify GUIs. This is invaluable in the early stages of development and for rapid prototyping.

2. **Minimalist and Efficient:** ImGui has a minimalist design, providing just enough functionality to build robust interfaces without unnecessary overhead.
3. **Immediate Mode GUI:** Unlike traditional retained-mode GUIs, ImGui is an immediate mode GUI (IMGUI), which means that the GUI is drawn and updated every frame. This approach simplifies the state management and makes the GUI logic straightforward and easy to understand.
4. **Integration with Graphics Libraries:** ImGui integrates seamlessly with graphics libraries like OpenGL, Vulkan, and DirectX, making it a versatile choice for developers working with different graphics backends.

1.1 ImGUI Widgets

ImGui provides several widgets for creating beautiful user interfaces. Some commonly used widgets include:

- `ImGui::Text`: Used for creating a text box for displaying text.
- `ImGui::CheckBox`: Used for creating a checkbox.
- `ImGui::SliderFloat`: Creating a slider window.
- `ImGui::Button`: Used for creating a button.
- `ImGui::RadioButton`: Used for creating a radio button.
- `ImGui::ProgressBar`: Used for creating a progress bar.
- `ImGui::InputText`: Used for creating input text box.

1.2 Managing Widgets

ImGui makes it simple to communicate with widgets. When a widget is interacted with, it triggers an event which is then captured by the event dispatcher and returns a boolean upon the trigger.

```
if(ImGui::Button("Click")){
    // if clicked do something
}
```

For interacting with value-based widgets such as text inputs, slider widgets, etc., take a variable to bind a particular value with, which gets updated as the user interacts with the widgets.

```
ImGui::SliderFloat("float", &slider_value, min, max);
// any change is recorded by the value in variable slider_value
```

2 GLFW

GLFW (Graphics Library Framework) is an open-source library for creating and managing windows, handling input, and managing OpenGL contexts for graphics rendering. It provides a simple and cross-platform interface for tasks like creating windows, managing user input, and setting up OpenGL contexts for graphics rendering, much like ImGui.

2.1 Creating Window

GLFW provides a medium to create a window and bind it to OpenGL for rendering. To create a window using GLFW, use the following snippet:

```
GLFWwindow* window = glfwCreateWindow(window_width, window_height, "Window Title", NULL,
```

Next, make the context of the specified window current for the calling thread by calling `glfwMakeContextCurrent(window)`.

2.2 Polling

Once the window is created, you'll want to poll or run continuously to capture I/O events and perform specific rendering changes. This is done until the window is closed.

```
while (!glfwWindowShouldClose(window)){
    // do rendering and handle I/O
}
```

2.3 Key Functions

- `glfwGetFramebufferSize()` Retrieve the size, in pixels, of the framebuffer of the specified window.
- `glViewport()` Set the viewport, set the x, y coordinates, and width height of the viewport.
- `glClearColor()` Specify clear values for the color buffers.
- `glClear()` Set the bitplane area of the window to values previously selected by `glClearColor`.
- `glfwSwapBuffers()` Swap the front and back buffers of the specified window.
- `glfwDestroyWindow()` Destroy the specified window and its context.
- `glfwTerminate()` Destroy all remaining windows.

2.4 Double Buffering

The concept of swapping the front and back buffers is a fundamental technique used in computer graphics to create smooth and flicker-free animations. This process is a key part of double buffering, which is commonly used in graphics programming.

This technique involves using two buffers, the front buffer and the back buffer, to manage rendering and display. The front buffer is what the user sees displayed on the screen, while the back buffer is where the next frame is drawn.

2.4.1 How it works

- `Rendering to the Back Buffer:` All the drawing operations are performed on the back buffer. This means that any changes to the image, such as drawing new graphics or updating existing ones, are done off-screen.
- `Swapping Buffers:` Once rendering is complete for a frame, the back buffer becomes the new front buffer, and the front buffer becomes the new back buffer. This swap is usually performed with a function such as *glfwSwapBuffers(window).*

3 Deliverables

None. Off you go.

How to Run

1. Download the attached zip file and extract it.
2. Go into the Lab0 directory and run `cmake CMakeLists.txt`.
3. Next, run `make`. You should now see an executable.
4. Run `./lab0`.

References

- `https://www.opengl.org/documentation/`
- `https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview`
- `https://www.khronos.org/registry/OpenGL-Refpages`
- `https://www.glfw.org/documentation.html`

Note: Your code should be written by you and be easy to read. You are **NOT** permitted to use any code that is not written by you. (Any code provided by the instructor/TA can be used with proper credits within your program). Theory questions need to be answered by you and not copied from other sources. Please refer to IIIT-Delhi's Policy on Academic Integrity here.