Hinweise: Programmieraufgaben

- Wir programmieren in **C++** und nutzen die **OpenGL 3.3 Core API**

- **Voraussetzungen:**
  1. Eine **IDE** eurer Wahl
  2. Einen **C++ Compiler** eurer Wahl
  3. **CMake**
  4. **OpenGL**

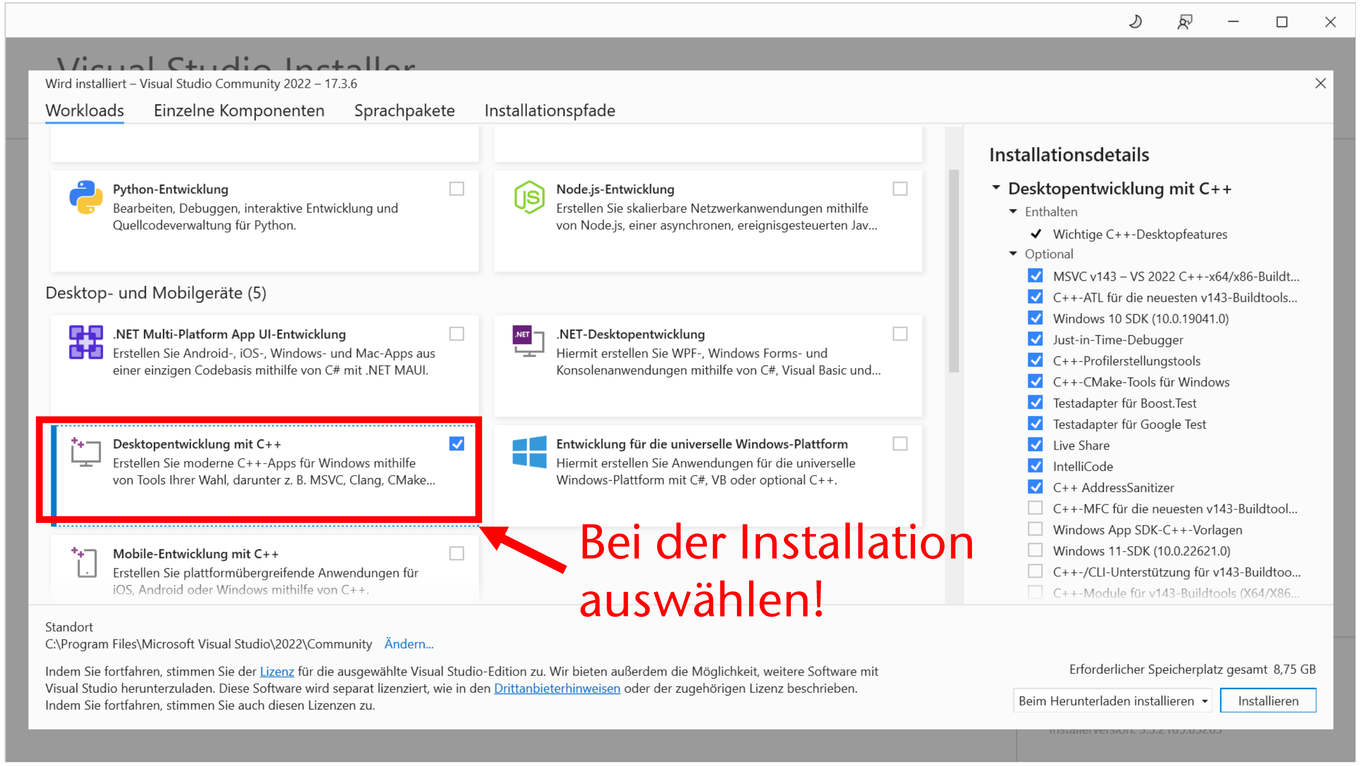
Hinweise: Programmieraufgaben

1. Mögliche **IDEs:**
   - **Visual Studio** *(Windows)*
     - https://visualstudio.microsoft.com/de/downloads/

Hinweise: Programmieraufgaben

1. Mögliche **IDEs:**

- **Visual Studio** *(Windows)*
  - https://visualstudio.microsoft.com/de/downloads/



Hinweise: Programmieraufgaben

1. Mögliche **IDEs:**
   - **Visual Studio** *(Windows)*
     - https://visualstudio.microsoft.com/de/downloads/
   - **Xcode** *(Mac)*
     - https://developer.apple.com/xcode/

Hinweise: Programmieraufgaben

1. Mögliche **IDEs:**
   - **Visual Studio** *(Windows)*
     - https://visualstudio.microsoft.com/de/downloads/
   - **Xcode** *(Mac)*
     - https://developer.apple.com/xcode/

Beim ersten Start öffnet sich:

Diese beiden Checkboxen sind ausreichend

Beim ersten Start SDKs installieren

Hinweise: Programmieraufgaben

1. Mögliche **IDEs:**
   - **Visual Studio** *(Windows)*
     - https://visualstudio.microsoft.com/de/downloads/
   - **Xcode** *(Mac)*
     - https://developer.apple.com/xcode/
   - Sonstige *(u.a. Linux):* **QTCreator, CodeLite, Code::Blocks, Visual Studio Code** etc.

Hinweise: Programmieraufgaben

2. Mögliche **C++ Compiler:**

- **MSVC** *(Windows)*
  - Bei Windows in **Visual Studio** enthalten (bzw. bei Installation auswählbar).

- **Clang** *(Mac / Linux):*
  - Beim Mac bereits in **Xcode** enthalten (AppleClang).

- **GCC** *(u.a. Linux):*
  - Installation via Packetmanager (Ubuntu): `sudo apt-get install gcc`

Hinweise: Programmieraufgaben

3. CMake

- Generiert Projektdateien für beliebige IDEs / Compiler aus den Informationen einer `CMakeLists.txt`
  - Auf diese Weise ist es möglich, dass wir ein **einziges** Projekt für **verschiedene** IDEs und Compiler mit den gleichen Einstellungen zur Verfügung stellen können.
  - Wir geben die `CMakeLists.txt` in jedem C++ Projekt vor.

Hinweise: Programmieraufgaben

3. CMake (Installation)

- Unter **Windows** empfohlen:
  - Download von https://cmake.org/
- Unter **Mac** empfohlen:
  - Via Homebrew:
  - Falls Homebrew noch nicht installiert ist, folgenden Befehl im Terminal ausführen:
    - `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
  - Dann:
    - `brew install --cask caskroom/cask/cmake cmake`
- Unter **Linux** empfohlen:
  - Via Packetmanager: `sudo apt-get install cmake`

Hinweise: Programmieraufgaben

4. OpenGL

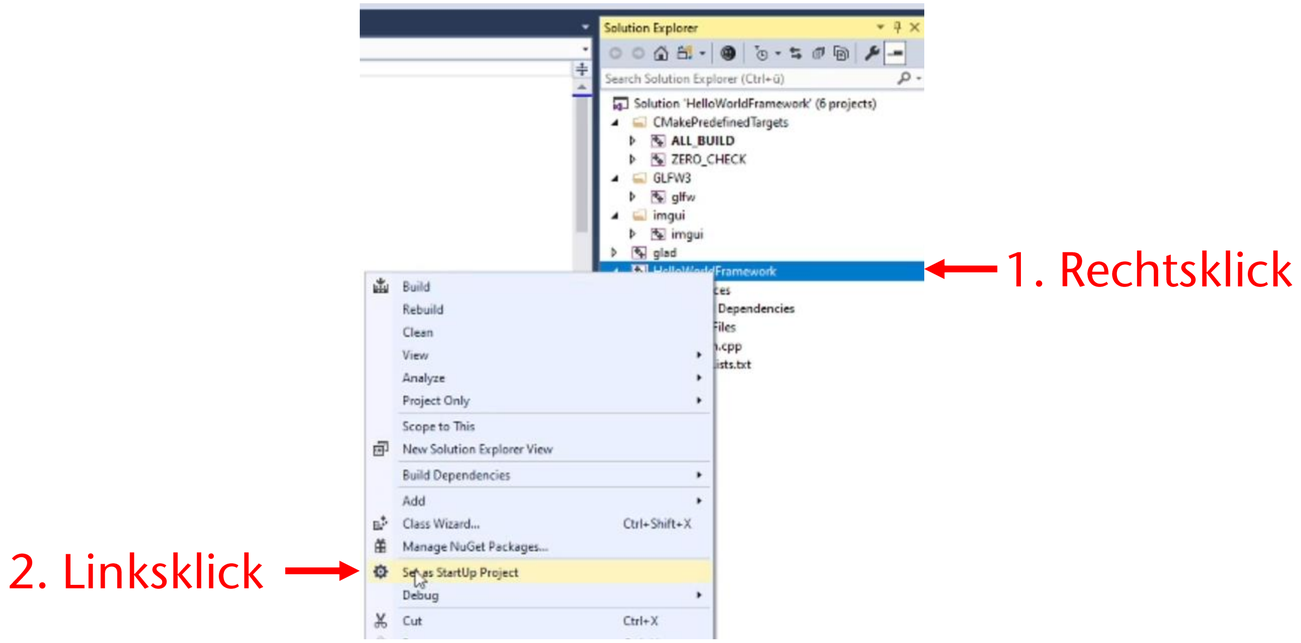
- I.d.R. vorinstalliert
  - ➢ Im aktuellen Grafikkartentreiber enthalten.
- Falls es unter Linux Probleme gibt, Mesa installieren:
  - Installation via Packetmanager (Ubuntu): `sudo apt install mesa-utils`

Hinweise: Programmieraufgaben

Wenn ihr alles installiert habt, könnt ihr das **Beispielprojekt** von der Webseite herunterladen und mit **CMake** konfigurieren – siehe dazu das Video auf der Webseite „Video Walkthrough Windows”

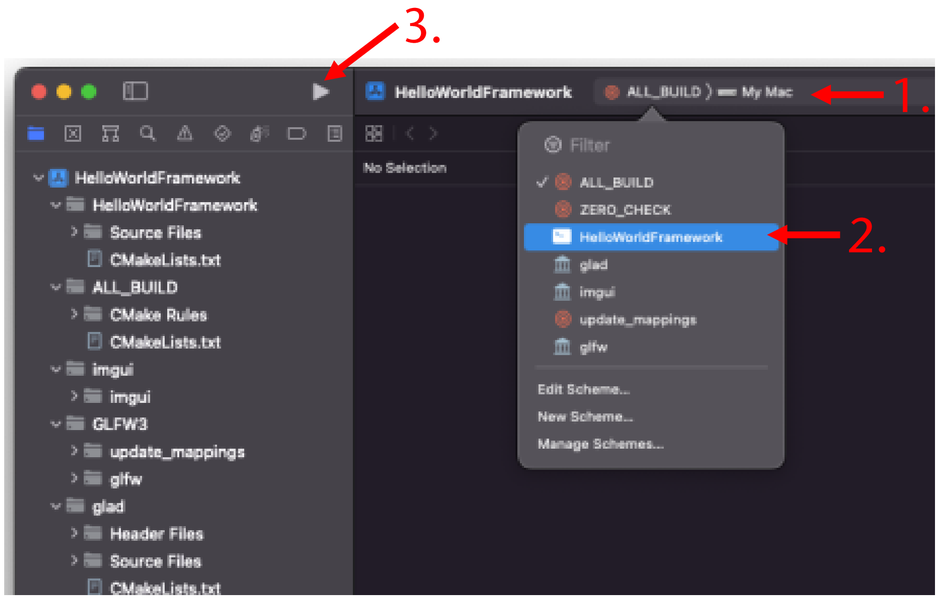
Hinweise: Programmieraufgaben

- **Hinweis zum Kompilieren und Ausführen**
  - In **Visual Studio:**
    - Wie im Video (siehe vorherige Folie) gezeigt, muss das Framework als aktives Startprojekt gesetzt werden „Set as Startup project“



Hinweise: Programmieraufgaben

- **Hinweis zum Kompilieren und Ausführen**
  - In **Xcode:**



Der Klick auf den Playbutton kompiliert den Quellcode und führt das Programm dann aus

Hinweise: Programmieraufgaben

Über die **Software-Voraussetzungen** wisst ihr jetzt bescheid...

... nachfolgend ein paar Informationen dazu, wie die vorgegebenen C++ Projekte aufgebaut sein werden.

Hinweise: Programmieraufgaben

- Alle vorgegebenen **C++ Projekte** verwenden diese Bibliotheken:
  - **GLFW:** Erlaubt es, einfache Fenster unabhängig vom Betriebsystem zu erstellen, in denen man mit OpenGL zeichnen kann.
  - **GLAD:** Erlaubt den Zugriff auf aktuelle OpenGL-Funktionen, in unserem Fall die OpenGL 3.3 Core Funktionen
  - **Dear ImGui:** Erlaubt es, einfache GUIs (in unserem Fall mit OpenGL) zu erstellen und Maus- und Tastatur-Events abzufragen.

Hinweise: Programmieraufgaben

- Verwendete externe Bibliotheken:
  - Müssen nicht installiert werden, sondern befinden sich bereits in den vorgegebenen Projekten (siehe Ordner **lib**).
  - `CMakeLists.txt` ist so konfiguriert, dass diese Bibliotheken automatisch mitkompiliert und verwendet werden.
  - Der Code, welcher die API dieser Bibliotheken nutzt, ist i.d.R. vorgegeben
    - Ausnahme: Maus- und Tastaturinteraktionen, dazu in den Übungsblättern mehr.

Hinweise: Programmieraufgaben

Ihr müsst euch mit diesem Bibliotheken **nicht** auskennen, um die Programmieraufgaben zu meistern!

Ihr sollt euch nur nicht wundern, warum diese Bibliotheken in den C++ Projekten enthalten sind und warum wir diese benötigen ;-)

Hinweise: Programmieraufgaben

- `main`-Funktion:
  - Alle vorgegebenen Projekte enthalten eine `main.cpp`-Datei, die eine `main`-Funktion enthält
    - Diese `main`-Funktion ist der Startpunkt des Programmes.
    - In dieser Funktion werden das Fenster erstellt und Einstellungen für die OpenGL-Version festgelegt.

Hinweise: Programmieraufgaben

FYI

- Die Initialisierung sieht bei uns wie folgt aus:

```
int main(int, char**)
{
    // Setup window:
    glfwSetErrorCallback([](int error, const char* description) {
        fprintf(stderr, "Glfw Error %d: %s\n", error, description);
    });

    if (!glfwInit())
        return 1;

    // Decide GL+GLSL versions:
    const char* glsl_version = "#version 330 core";
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

    // Create window with graphics context:
    GLFWwindow* window = glfwCreateWindow(1090, 560, "Computergrafik 1", NULL, NULL);
    if (window == NULL)
        return 1;

    glfwMakeContextCurrent(window);
    glfwSwapInterval(1);

    // Initialize GLAD functions:
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // Setup Dear ImGui context:
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;
    io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;     // Enable Keyboard Controls

    // Setup Dear ImGui style:
    ImGui::StyleColorsDark();

    // Setup Platform/Renderer backends:
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init(glsl_version);

    [...]
```

Sage GLFW, wie im Falle eines Fehlers diese Fehler gezeigt werden sollen (via Lambda Funktion)

Falls GLFW nicht initialisert werden konnte, breche ab.

Lege OpenGL Versionen fest.

Erstelle das Fenster in der Größe von 1090 x 560 Pixeln

„Binde“ OpenGL an das Fenster

Lädt Pointer zu neueren OpenGL Funktionen, sodass wir diese nutzen können (muss man nicht verstehen).

Initialisiert Dear ImGui (kurz: ImGui).

Konfiguriert ImGui, sodass es GLFW mit OpenGL zum Rendern der GUI nutzt.

Hinweise: Programmieraufgaben

- **Main-Loop**
  - Die main-Funktionen in unseren Projekten enthalten **immer** eine while-Schleife
    - Diese Schleife wird Main-Loop, Render-Loop o. Game-Loop genannt (vgl. Wikipedia)
    - Innerhalb der Schleife steht das, was beim Zeichnen **eines** Frames ausgeführt wird.
      - Diese Schleife wird je nach Computer/Bildschirm verschieden oft pro Sekunde ausgeführt.
    - Die Schleife wird erst dann verlassen, wenn das Programm beendet werden soll.

Hinweise: Programmieraufgaben

FYI

- **Main-Loop (1/2)**

```
// Main loop which is executed every frame until the window is closed:
while (!glfwWindowShouldClose(window))
{
    // Processes all glfw events:
    glfwPollEvents();

    // Start the Dear ImGui frame
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Get the size of the window:
    int display_w, display_h;
    glfwGetFramebufferSize(window, &display_w, &display_h);

    // Setup the GL viewport
    glViewport(0, 0, display_w, display_h);
    glClearColor(0.6f, 0.725f, 0.8f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
```

Wiederhole solange, bis das Fenster geschlossen wird.

Sorge dafür, dass in unserem Falle ImGui alle Mausklicks und Tastaturevents mitbekommt, die seit dem letzten Frame stattfanden.

Sage ImGui, dass der nächste Frame gerendert wird.

Lese aus, wie groß das Fenster ist, in dem mittels OpenGL gerendert werden kann (so bekommen wir mit, wenn das Fenster vergrößert / verkleinert wird).

Sage OpenGL, wie groß der Bereich ist, in dem wir zeichnen wollen.

Sage OpenGL, dass wir den gesamten Bild mit der RGB-Farbe (0.6, 0.725, 0.8) überschreiben wollen.

Hinweise: Programmieraufgaben

FYI

- **Main-Loop (2/2)**

```
    [...]

    // Render the GUI and draw it to the screen:
    ImGui::Render();
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

    // Swap Buffers:
    glfwSwapBuffers(window);
}
```

Hier fügen wir später unseren Code in den Aufgabenblättern ein, um 3D-Objekte mithilfe von OpenGL innerhalb eines Frames zu zeichnen!

Sorgt schließlich dafür, dass die GUI von Dear ImGui mithilfe von OpenGL in das Fenster gezeichnet wird.

Tauscht Front- und Back-Buffer. Dabei wird i.d.R. auf die vertikale Synchronisation gewartet (sodass die `while`-Schleife nicht mehr als 60 Mal pro Sekunde ausgeführt wird, oder auch 90, 120 Mal, je nach Monitor).

Hinweise: Programmieraufgaben

FYI

- **Nach der Main-Loop:**

```
    // Swap Buffers:
    glfwSwapBuffers(window);
}

// Cleanup
ImGui_ImplOpenGL3_Shutdown();
ImGui_ImplGlfw_Shutdown();
ImGui::DestroyContext();

glfwDestroyWindow(window);
glfwTerminate();

return 0;
```

← Sorgt dafür, dass alle mögliche an Speicher freigegeben wird, damit das Programm sauber beendet wird.

← Sorgt dafür, dass das Fenster zerstört wird.

← Beende den `main`-Funktionsaufruf (und damit das Programm) schließlich und gebe 0 zurück (dies steht bei C++ für eine erfolgreiche Programmausführung).

Hinweise: Programmieraufgaben

- **Wir erwarten:**
  - **Leserlichen, verständlichen** Code.
  - Erklärt uns ggf. in den **C++ Kommentare** kurz, was / warum ihr getan habt.
    - Der von uns vorgegebene C++ Code darf als Vorbild dienen.
    - Ihr müsst euren Code allerdings **nicht** in der PDF-Abgabe beschreiben, wie es ggf. in PI1 oder PI2 verlangt wird!
  - Bei unleserlichem oder **sehr ineffizienten** Code kann es zu Punktabzügen kommen!

Hinweise: Programmieraufgaben

- **C++ Hinweisblätter**
  - In diesem Kurs werden grundlegende Kenntnisse in C++ erwartet.
    - Es gibt ein **C++ Propädeutikum** jedes Jahr Ende September, dessen Teilnahme in den Studiengängen empfohlen wird.
  - Da einige von euch dennoch **wenig C++ Erfahrung** haben, geben wir zu den ersten C++ Übungsblätter ein paar C++ Hinweisblätter mit heraus.
    - Diese erklären ein paar für das Übungsblatt relevante Unterschiede zwischen **C++** und **Java / Processing**, sind allerdings nicht zwangsläufig vollständig.
    - ➢ Wir erwarten von euch, dass ihr euch bei Bedarf **selbstständig** mit C++ beschäftigt.
      - ➢ Im Sinne einer Universität