

Computergrafik 1

Tutorium 1

Hinweise zu den Programmieraufgaben

- Übungsblätter
 - Erste Hälfte für betreute Arbeit im Tutorium (nicht bewertet)
 - Zweite Hälfte als Hausaufgabe (bewertet; nur bestanden / nicht bestanden)
 - Hauptsächliches Bewertungskriterium: Aufgabe grundsätzlich gelöst
 - Bestandene Blätter geben einen Bonus in der Klausur (s. Vorlesungsfolien für Genaueres)

- Hinweise zu Chatbots
 - Unsere Erfahrung zeigt: Wer sich zu sehr auf Chatbots etc. verlässt, besteht die Klausur i.d.R. nicht (!!!). Ihr könnt in der Klausur nicht copy-pasten.
 - ChatBots sind gut, um sich Probleme & Sachverhalte erklären zu lassen. Diese muss man am aber im Anschluss nochmal prüfen bzw. durcharbeiten.
 - Die Tutorien sind dafür da, dass ihr an Problemen & Aufgaben arbeitet und Fragen stellt, also keine Scheu :)

- Übungsblätter
 - 4er Gruppen
 - Eintrag über StudIP → Erhältet damit auch Gruppen-Nr.
- Sind via GitLab (<https://gitlab.informatik.uni-bremen.de>) abzugeben!
 - Erstellt ein neues Repository für alle Abgaben.
 - Benennungskonvention: `cg1_25-26-<gruppen-nr.>`, z.B. `cg1_25-26-3`
 - Ladet alle TutorInnen (Judith Boeckers, Hermann Meißenhelter, Navid M. Jadid) als Member (Level: Maintainer, Expiration Date Ende des Semesters!!) ein, damit wir eure Abgaben kontrollieren können.
 - Pusht eure Lösung bis zur Deadline ins Git-Repo – dies gilt als Abgabe!

- Git-Repo Struktur
 - Kopiert die Dateien aus dem Beispiel-Repo (Link auf Webseite) in euer Repository. Struktur sieht dann aus wie folgt:

```

.
|
|—— Blatt 1
|
|   |—— <Dateien für Abgaben>
|   |
|   |—— ...
|
|   ...
|
|—— gruppenmitglieder.md
|—— korrekturen.md
|—— .gitignore
    
```

- gruppenmitglieder.md
 - Einmal alle Gruppenmitglieder eintragen, mit den gelisteten Informationen.
 - Falls es Änderungen gibt (müssen mit uns abgesprochen werden), diese bitte in den entsprechenden Feldern eintragen.

```
gruppenmitglieder.md M X
Beispiel-Repo > gruppenmitglieder.md > # Gruppenmitglieder
1  # Gruppenmitglieder
2
3  | Vorname | Nachname | Matrikelnr. |
4  | -----|-----|-----|
5  |         |         |         |
6  |         |         |         |
7  |         |         |         |
8  |         |         |         |
9
10 Änderungen hier eintragen:
11
12 | Vorname | Nachname | Matrikelnr. | Beigetreten / Verlassen | bei Blatt # |
13 | -----|-----|-----|-----|-----|
14 |         |         |         |         |         |
15 |         |         |         |         |         |
16
```


Tutorium 1

- korrekturen.md
 - Eintrag von Bestehen / Nichtbestehen von Blättern durch die Tutoren.
 - Wir ergänzen ggf. auch Kommentare oder Korrekturen.

```

korrekturen.md x
Beispiel-Repo > korrekturen.md > # Korrekturen
1  # Korrekturen
2
3  | Blatt      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Summe |
4  |-----|---|---|---|---|---|---|---|---|---|---|
5  | Bestanden ✓? |   |   |   |   |   |   |   |   |   |   |
6
7  ## Blatt 1
8
9  Hier können Kommentare stehen...
10
11 ## Blatt 2
12
13 Hier können ebenso...
14
    
```


- 1. Übungsblatt
 - Bereits erschienen (siehe Webseite <https://cgvr.cs.uni-bremen.de>)
 - Deadline ist nächster Dienstag (21.10.2025, um 23:59 Uhr).
 - Abgabe als pdf

Hinweise: Programmieraufgaben

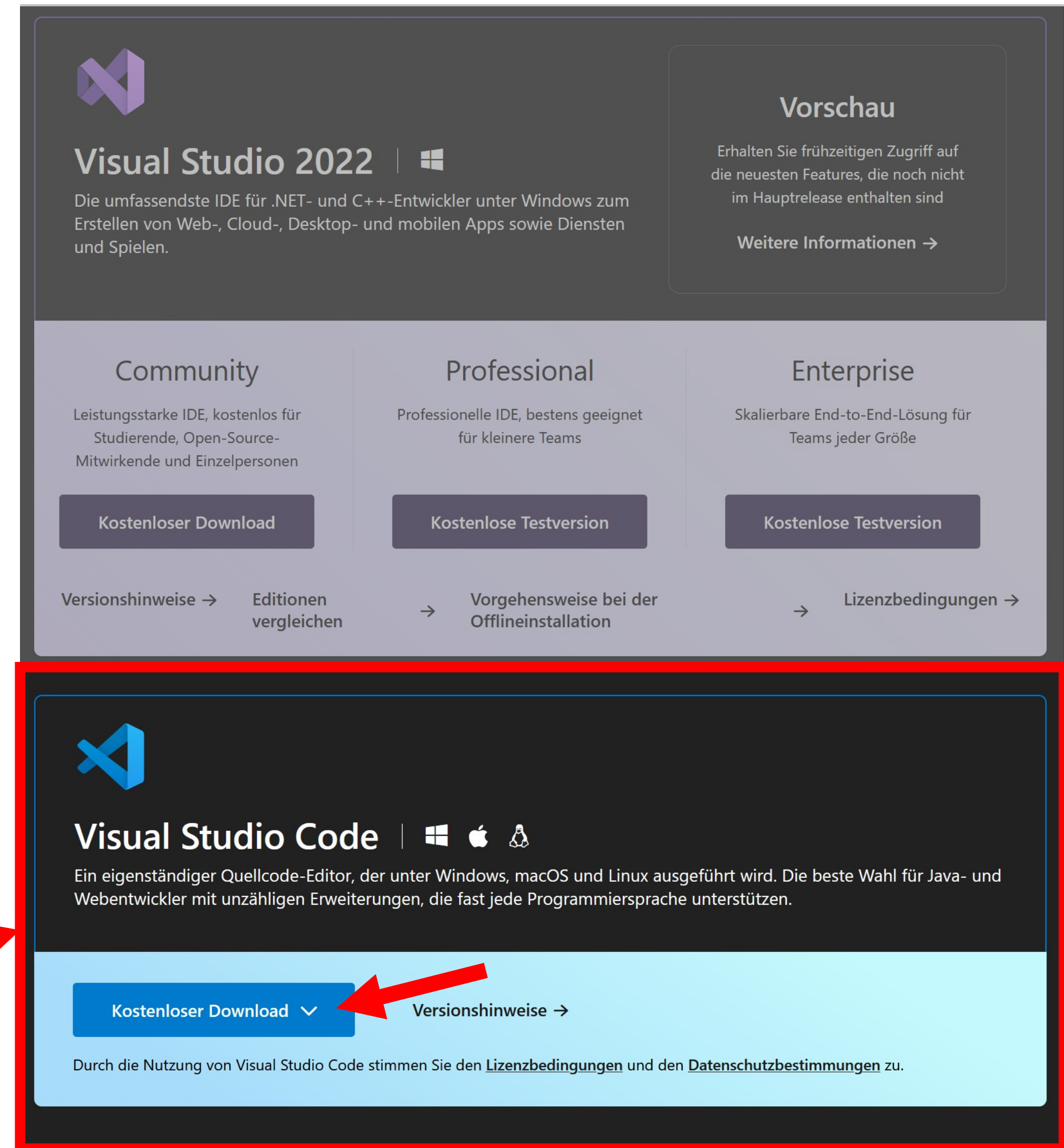
- Wir programmieren in C++ und nutzen die OpenGL 3.3 Core API
- Voraussetzungen:
 1. IDE
 2. C++ Compiler
 3. CMake
 4. OpenGL (unter Windows / Mac vorinstalliert)

Hinweise: Programmieraufgaben

1. Empfohlene IDE:

- Visual Studio Code
 - Über Webseite <https://visualstudio.microsoft.com/de/downloads/>
 - Ggf. Paketmanager oder App-Store
(z.B. Ubuntu Software Center)

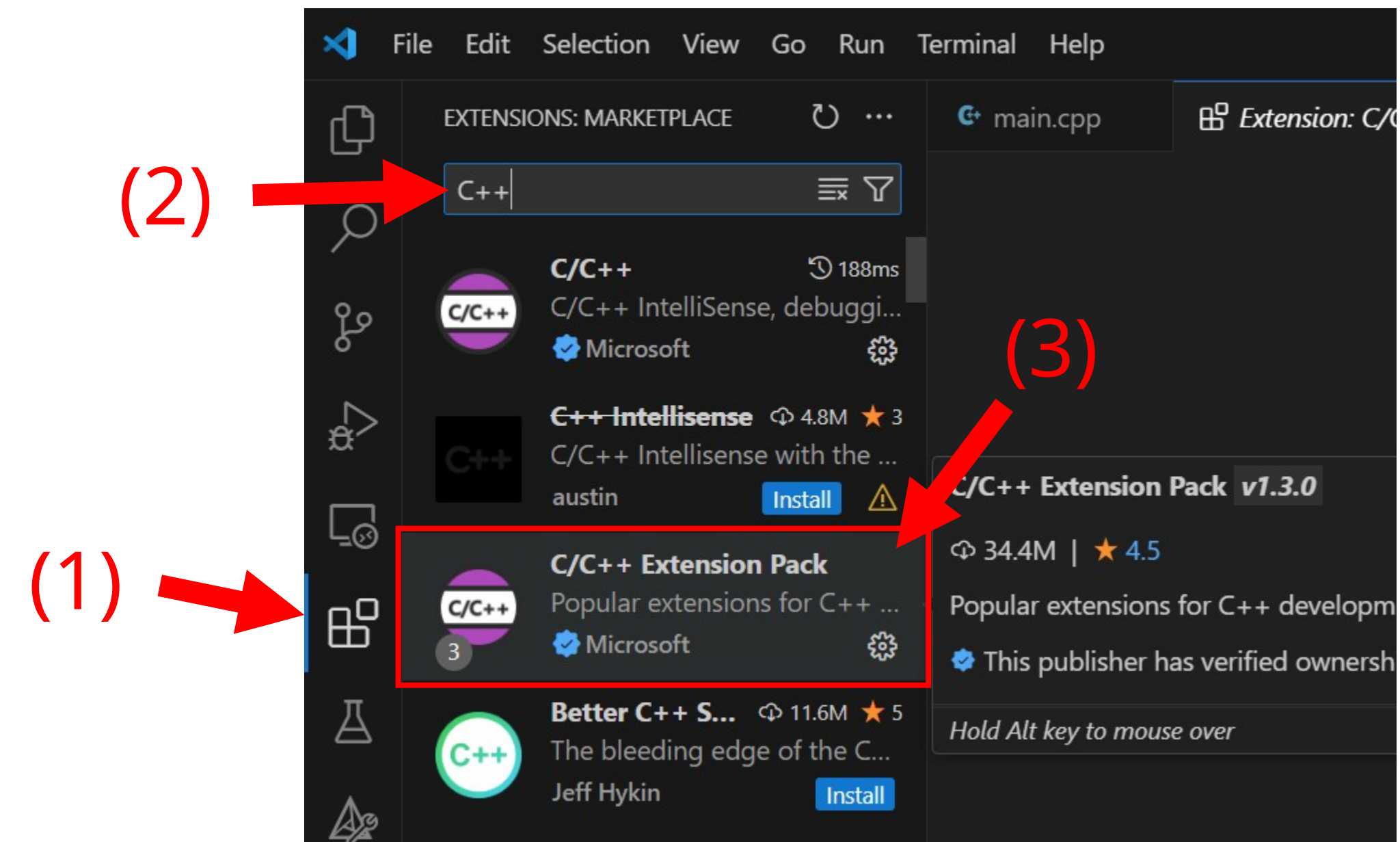
Auf der Webseite
das hier wählen



Hinweise: Programmieraufgaben

1. Empfohlene IDE:

- Visual Studio Code
 - Nach erstem Start die „C/C++ Extension“ installieren
 - Vereinfacht Kompilieren & Debuggen der Programme



Hinweise: Programmieraufgaben

2. Empfohlene C++ Compiler:

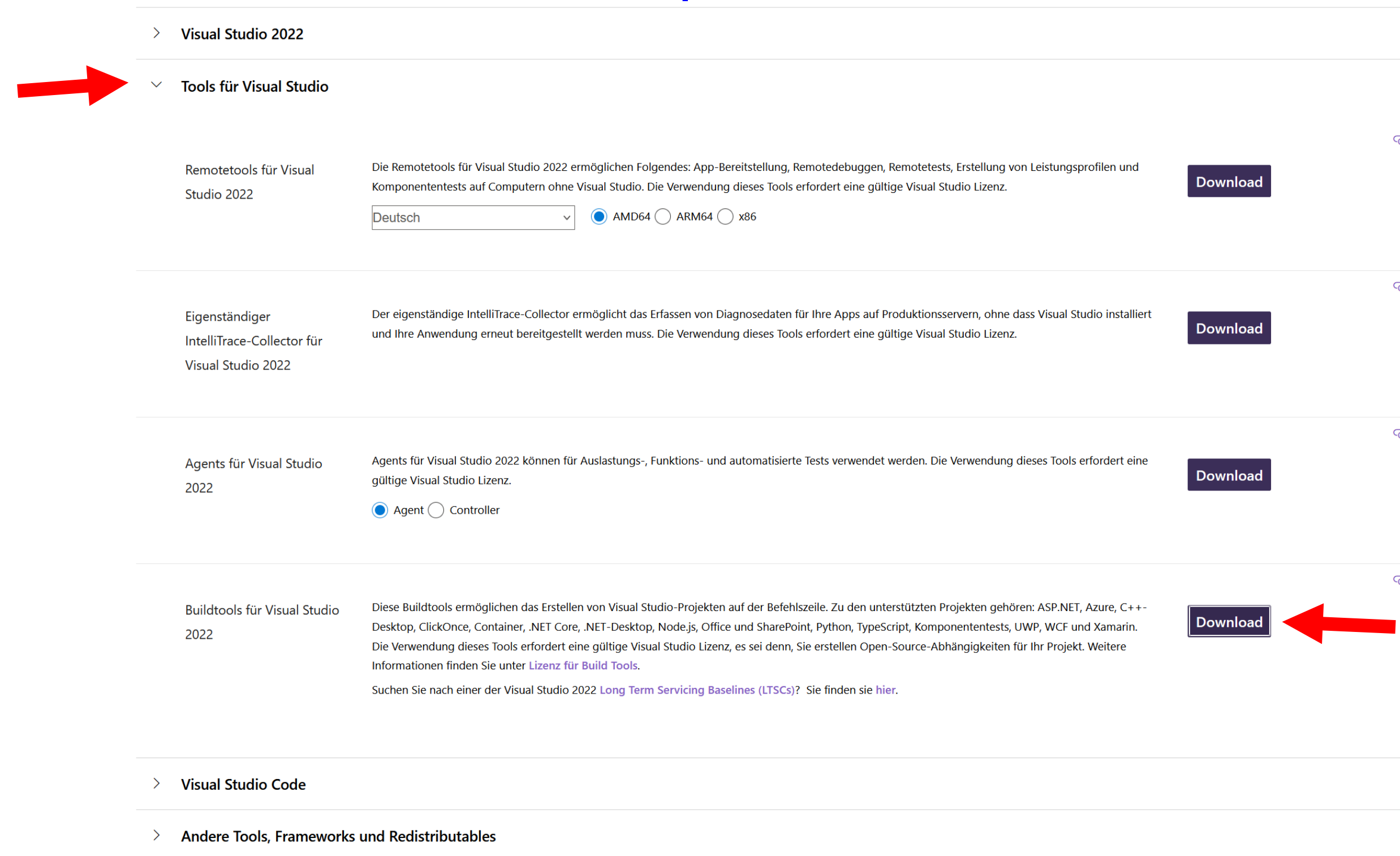
- MSVC (Windows)
- Clang (Mac / Linux):
 - In Xcode enthalten (AppleClang).
- GCC (u.a. Linux):
 - Installation via Paketmanager (Ubuntu): `sudo apt-get install gcc`
 - I.d.R. vorinstalliert

Hinweise: Programmieraufgaben

2. Empfohlene Compiler (Windows)

- MSVC Compiler

- Ganz unten auf <https://visualstudio.microsoft.com/de/downloads/>:



The screenshot shows the Visual Studio 2022 download page. A red arrow points to the 'Tools für Visual Studio' section. Below this section, there are four rows of tools, each with a 'Download' button. The last row, 'Buildtools für Visual Studio 2022', has a red arrow pointing to its 'Download' button.

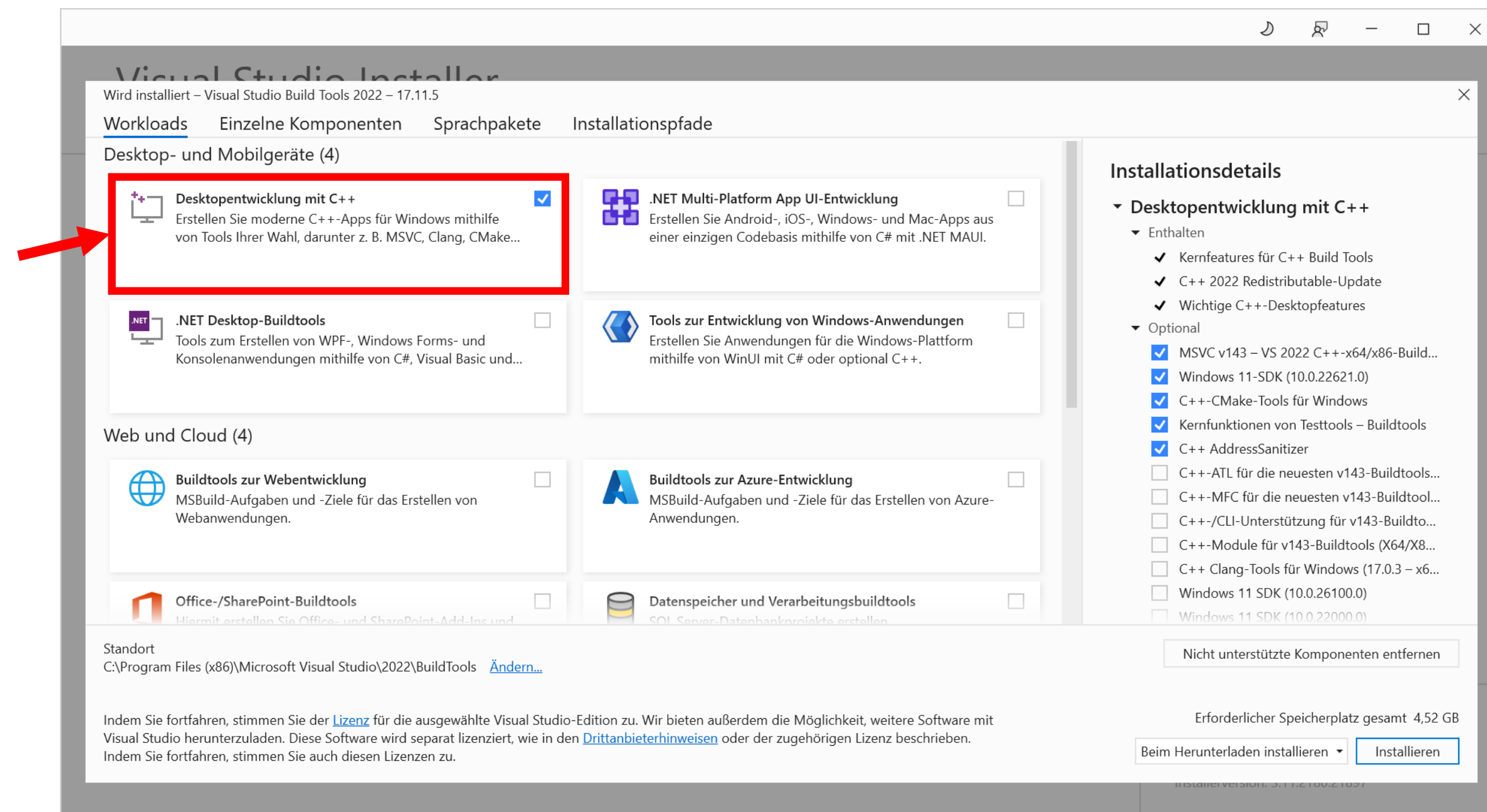
Tool Name	Description	Download Button
Remotetools für Visual Studio 2022	Die Remotetools für Visual Studio 2022 ermöglichen Folgendes: App-Bereitstellung, Remotedebuggen, Remotetests, Erstellung von Leistungsprofilen und Komponententests auf Computern ohne Visual Studio. Die Verwendung dieses Tools erfordert eine gültige Visual Studio Lizenz.	Download
Eigenständiger IntelliTrace-Collector für Visual Studio 2022	Der eigenständige IntelliTrace-Collector ermöglicht das Erfassen von Diagnosedaten für Ihre Apps auf Produktionsservern, ohne dass Visual Studio installiert und Ihre Anwendung erneut bereitgestellt werden muss. Die Verwendung dieses Tools erfordert eine gültige Visual Studio Lizenz.	Download
Agents für Visual Studio 2022	Agents für Visual Studio 2022 können für Auslastungs-, Funktions- und automatisierte Tests verwendet werden. Die Verwendung dieses Tools erfordert eine gültige Visual Studio Lizenz.	Download
Buildtools für Visual Studio 2022	Diese Buildtools ermöglichen das Erstellen von Visual Studio-Projekten auf der Befehlszeile. Zu den unterstützten Projekten gehören: ASP.NET, Azure, C++-Desktop, ClickOnce, Container, .NET Core, .NET-Desktop, Node.js, Office und SharePoint, Python, TypeScript, Komponententests, UWP, WCF und Xamarin. Die Verwendung dieses Tools erfordert eine gültige Visual Studio Lizenz, es sei denn, Sie erstellen Open-Source-Abhängigkeiten für Ihr Projekt. Weitere Informationen finden Sie unter Lizenz für Build Tools . Suchen Sie nach einer der Visual Studio 2022 Long Term Servicing Baselines (LTSCs) ? Sie finden sie hier .	Download

Hinweise: Programmieraufgaben

2. Empfohlene Compiler (Windows)

- MSVC Compiler
- Bei Installation „Desktopentwicklung mit C++“ auswählen:

Bei der Installation auswählen!



Hinweise: Programmieraufgaben

2. Empfohlene Compiler (Mac)

- Apple Clang via XCode
 - Ganz unten auf <https://developer.apple.com/xcode/>

Beim ersten Start öffnet sich:

Diese beiden Checkboxes
sind ausreichend

Beim ersten Start
SDKs installieren



Hinweise: Programmieraufgaben

3. CMake

- Generiert Projektdaten 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 homebrew/cask/cmake cmake`
- Unter Linux empfohlen:
 - Via Packetmanager: `sudo apt-get install cmake`

Hinweise: Programmieraufgaben

3. CMake (VSCode Extension)

- Extension: Cmake Tools (von Microsoft)
- Im VSCode Marketplace

Hinweise: Programmieraufgaben

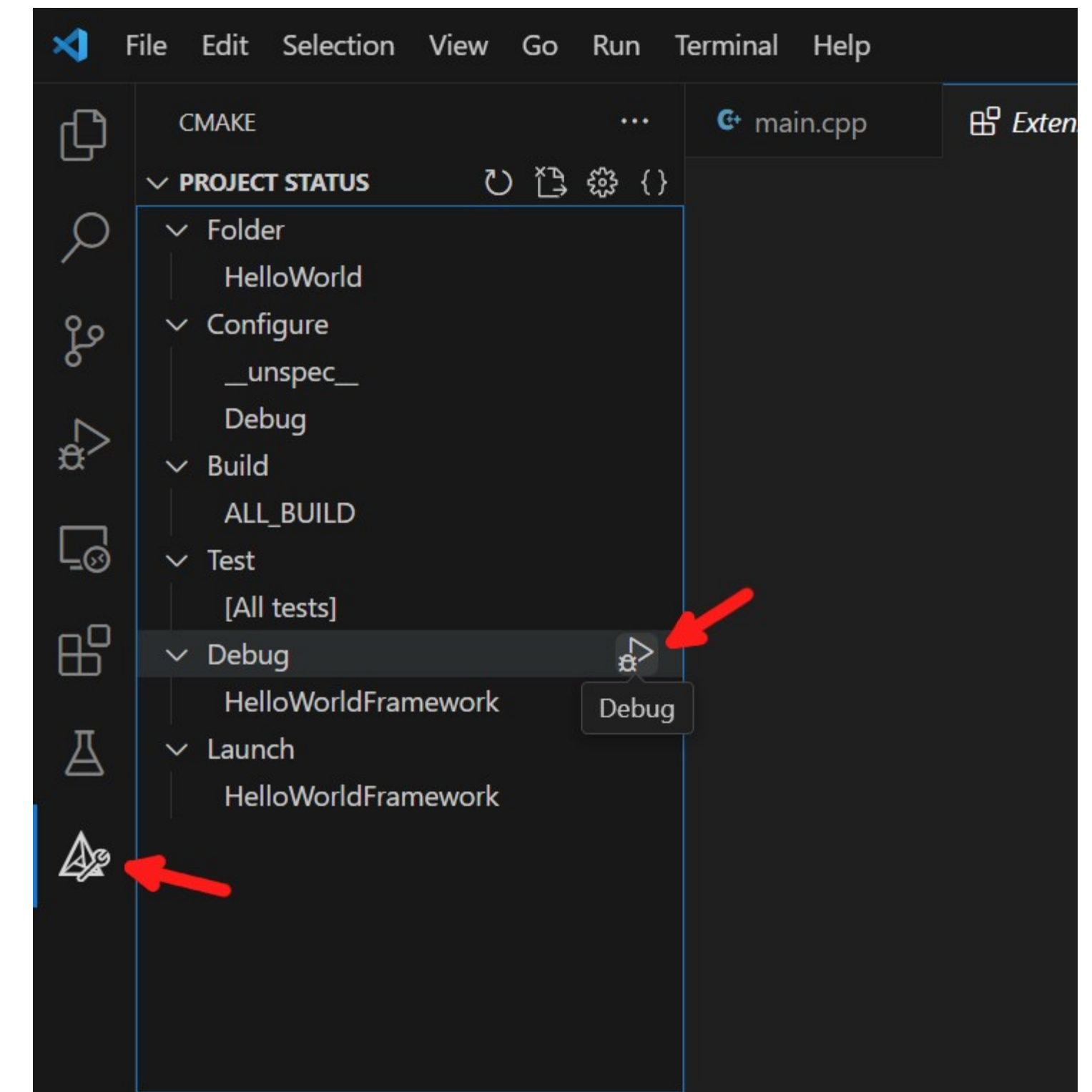
4. OpenGL

- 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

Testen der Installation

- Zum Testen der Installation das Beispielpjekt „Hello World“ herunterladen
 - <https://cgvr.cs.uni-bremen.de/teaching/cg1/>
 - Den entpackten Ordner in Visual Studio Code öffnen
- Falls kein Startbutton gezeigt wird, oben in der Kommandozeile nach „cmake build“ suchen.



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 Betriebssystem 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 diesen 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 initialisiert 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 einer PDF-Abgabe beschreiben, wie es ggf. in PI1 oder PI2 verlangt wird!
 - Bei unleserlichem oder sehr ineffizienten Code kann es zu Abzügen kommen (→ nicht bestehen)!

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