

# Willkommen!



C + +  
usergroup  
DRESDEN

## Wer ist SWP?

**swp software systems GmbH & Co. KG aus Dresden**

Lösungsanbieter für den Ein- und Mehrfamilienhausbau

51 Mitarbeiter und Partner Deutschlandweit

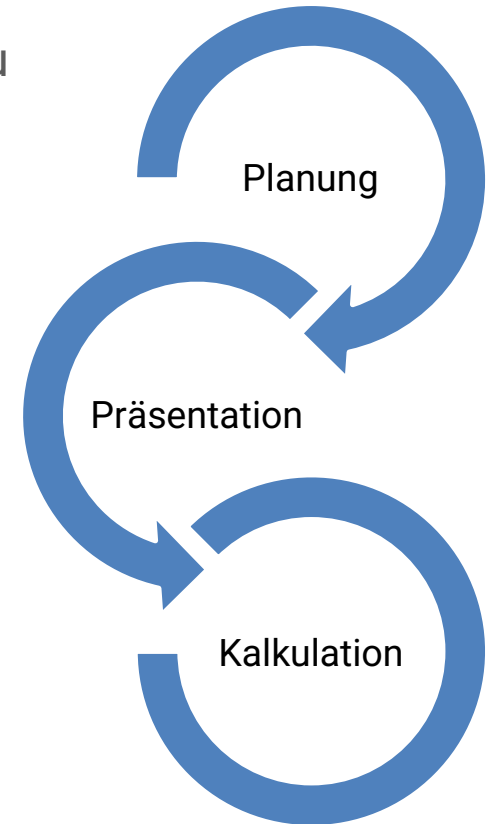
Beratung, Schulung, Dienstleistung und Software

## Mission und Märkte

**Planung, Präsentation & Kalkulation von Ein- und Mehrfamilienhäusern in 1h**

**Kunden (u.a.):**

- Bauunternehmen
- Architekten
- Fertighaushersteller / serielle Fertiger
- Baustoffhändler, -hersteller uvm.
- Bestandsbau



# Wer sind wir?

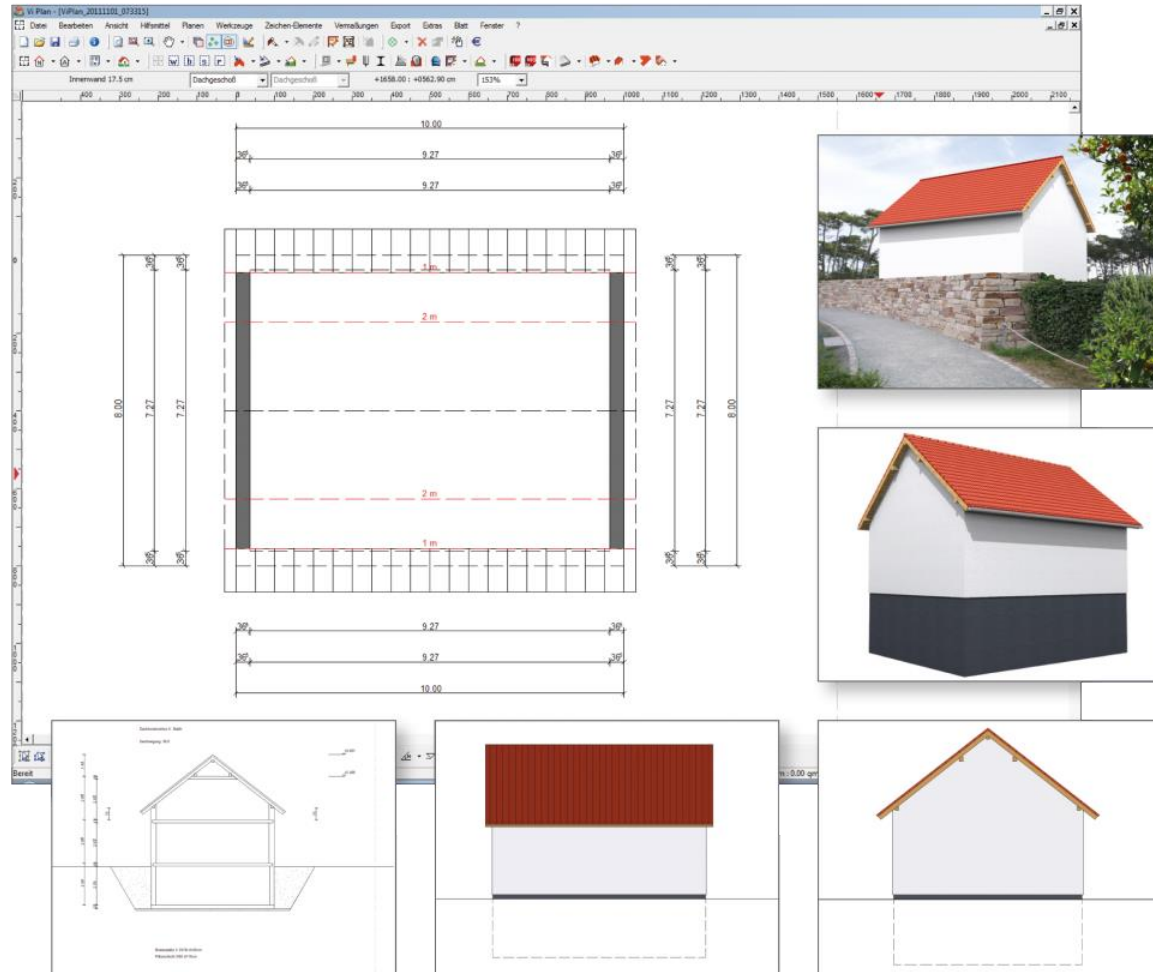
## 1. **Stefan Urlberger**

- Geschäftsführender Gesellschafter
- Zuständigkeitsbereiche: Technik und Softwareentwicklung
- seit ca. 25 Jahren dabei

## 2. **Patrick Morgenstern**

- Softwareentwickler (hauptsächlich C/C++)
- Aufgabenbereich: Wartung und Weiterentwicklung von Vi Plan
- seit ca. 30 Jahren dabei

# Planung



# Präsentation



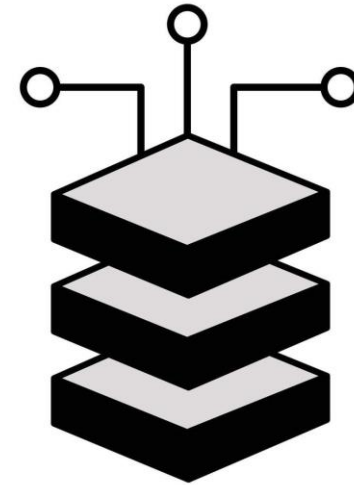
# Technologiestack

## 1. Toolchain / IDE

- MFC / Visual C++ – 2022er Version
- Visual Studio Solution / CMake
- IDE: Visual Studio 2022
- VCS: Git via Gitlab
- Windows only

## 2. Buildserver

- Teamcity



# Warum NuGet / CoApp?

Start: 2015 (vorher gab es keine Packages)

- Erfahrungen im .NET-Bereich mit NuGet war vorhanden, daher erste Wahl
- Beweggründe für Packages generell:
  - Großer Monolith
  - Buildzeiten hoch, Komplexität hoch, Nachvollziehbarkeit gering
  - Grundidee: Nutzung von Packages in C++ und in C# (gleiches Package)
  - Leidensdruck stieg
  - Microservicearchitekturen kamen in Mode



# Warum Conan?



**Start: 2017**

## **Umstiegsgründe:**

- CoApp wurde nicht intensiv gepflegt, bzw. inkompatible Versionen
- Teilweise nur noch per Chocolatey-Package-Manager updatebar
- Viel Sucherei nötig für Patches
- Package-Anweisungen kompliziert und schwer pflegbar
- Dokumentation mäßig, einiges eher experimentell
- Wir hatten immer mehr den Eindruck einer „Bastellösung“

## **Argumente für Conan:**

- Doku wesentlich besser, viele Sonderfälle auch schon dokumentiert
- Viele Scripting-Möglichkeiten mit Python  
(neuer Entwickler mit Python-Erfahrung)



# Warum VCPackage?



Start: 2023

## Umstiegsgründe:

- Komplexität auch bei Conan zu hoch geworden (auch die Migration auf eine neue Conan-Version hätte massiven Aufwand verursacht)
- Umsetzung in Conan etwas unüblich, da wir sämtliche sinnvollen Konfigurationen vorkompilieren
- Bezug von externen Packages über z.B. <https://center.conan.io> instabil
  - Conan-Versionsänderungen in Package von heute auf morgen möglich
  - Alte Packages werden teilweise nicht mehr angeboten oder überschrieben
  - Selbst unser Mirroring von verwendeten externen Packages brachte keinen Erfolg, da in unregelmäßigen Abständen auch diese aktualisiert werden müssen
  - Externe Package-Quellen gingen teilweise „einfach so“ offline und wurden außer Betrieb genommen

## Warum VCPackage?

- Einbindung in Visual Studio-Projekte aufwändig / keine native Integration
  - Debugging / Weiterentwicklung von Packages kompliziert (auch durch unseren eigenen Devtools-Layer)
- Schwergewichtiger Toolstack
  - Python: Module in „richtigen“ Versionen erforderlich
  - Unsere eigenen DevTools auf Basis von Python wurden schwer wartbar
  - CMake
  - Git
  - MSBuild
- Alles muss in der „korrekten“ Version vorliegen
- Integration neuer Compiler-Versionen zieht neue Conan-Version nach sich und muss in eigene Devtools integriert werden
- Erfolg nach größeren Umstellungen wg. Breaking Changes erst spät sichtbar
- Eigener Fehler: Start mit BETA-Version, es fielen Funktionen weg

## Erkenntnisse zu vcpkg

- Technologiestack „leichtgewichtiger“
- Geringerer Administrationsaufwand
  - CMake und „unsere“ DevTools auf CMake-Basis
  - Git
  - MSBuild
  - CMake kann in VisualStudio Code problemlos debugged werden ☺
- lediglich DevTools und CMake müssen vorliegen
- native Integration in andere Buildsysteme bereits vorhanden
  - Make
  - Meson
  - MSBuild
  - OS X
- Self-Maintained (installiert / wartet benötigte Tools selbst)
- Debugging / Änderung von Packages funktioniert problemlos



## Einige Begrifflichkeiten

- Registry
  - Sammlung von Ports und deren Versionen
- Port
  - entspricht im Wesentlichen einem Recipe
  - Bildungsvorschrift, wie Package gebildet wird
  - Implementiert in „portfile.cmake“
- Triplet
  - Zielkonfiguration für benötigte Libraries
  - <https://learn.microsoft.com/en-us/vcpkg/users/triplets>
- Binary Caching
  - Speicherung des kompletten Package-Inhaltes einer Library im Dateisystem oder anderweitig, um Buildzeiten zu reduzieren
- Classic-/Manifest-Mode
  - im Classic-Mode verhält sich vcpkg ähnlich zu apt dh. eine Konfiguration der Abhängigkeiten PRO Projekt ist nicht möglich
  - im Manifest-Mode können z.B. Abhängigkeiten zu Packages in der Datei „vcpkg.json“ PRO Projekt definiert werden



# Demo



C + +  
usergroup  
DRESDEN

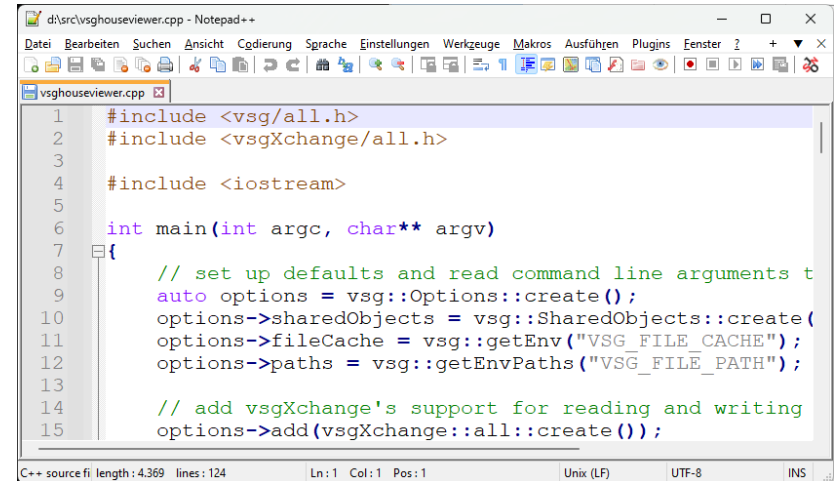
# Zielstellung

- benötigt wird ein Viewer zum Darstellen von beliebigen 3D-Szenen
- soll auf Vulkan aufsetzen
- C++ mit Sprachstandard C++17
- CMake als Buildsystem
- Support für Windows und Linux
  - x86
  - x64



# Anforderungsanalyse

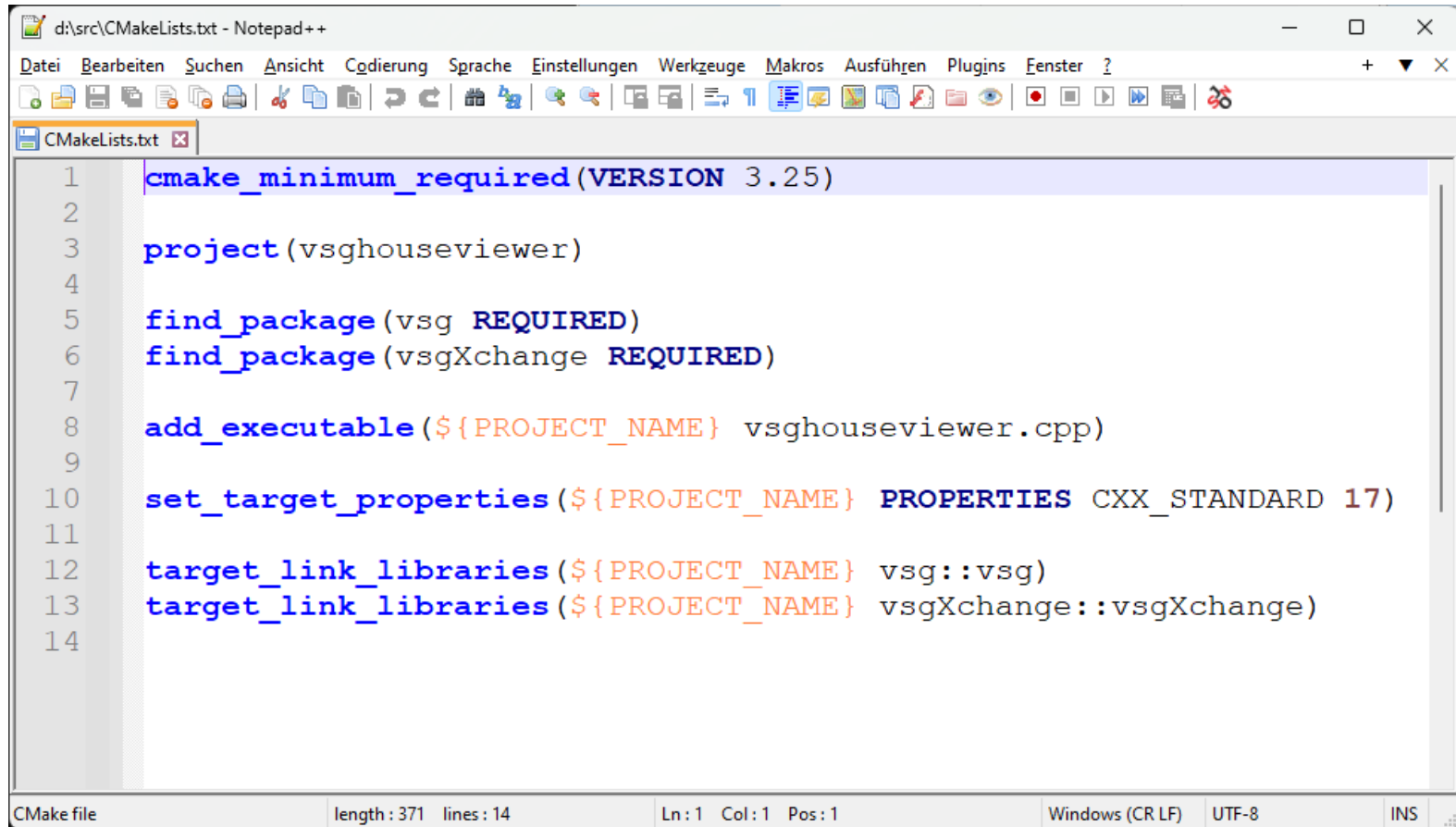
- Quellcode des Viewers?
  - bereits vorhanden
- Welche Libraries werden benötigt?
  - C++ Standard Template Library (STL)
  - VulkanSceneGraph (vsg)
  - VulkanSceneGraph Exchange (vsgXchange)
  - Vulkan SDK
- Existieren alle benötigten Libraries als vcpkg?
  - alle Libraries, außer das Vulkan SDK
  - Vulkan SDK muss manuell auf dem System installiert werden
- Was fehlt?
  - Anbindung an CMake-Buildsystem
  - Anbindung an vcpkg über Manifest



```
d:\src\vsg\houseviewer.cpp - Notepad++
Datei Bearbeiten Suchen Ansicht Codierung Sprache Einstellungen Werkzeuge Makros Ausführen Plugins Fenster 2 + v x
vsgHouseviewer.cpp
1 #include <vsg/all.h>
2 #include <vsgXchange/all.h>
3
4 #include <iostream>
5
6 int main(int argc, char** argv)
7 {
8     // set up defaults and read command line arguments t
9     auto options = vsg::Options::create();
10    options->sharedObjects = vsg::SharedObjects::create(
11    options->fileCache = vsg::getEnv("VSG_FILE_CACHE");
12    options->paths = vsg::getEnvPaths("VSG_FILE_PATH");
13
14    // add vsgXchange's support for reading and writing
15    options->add(vsgXchange::all::create());
```

# Anbindung an CMake

## CMakeLists.txt:



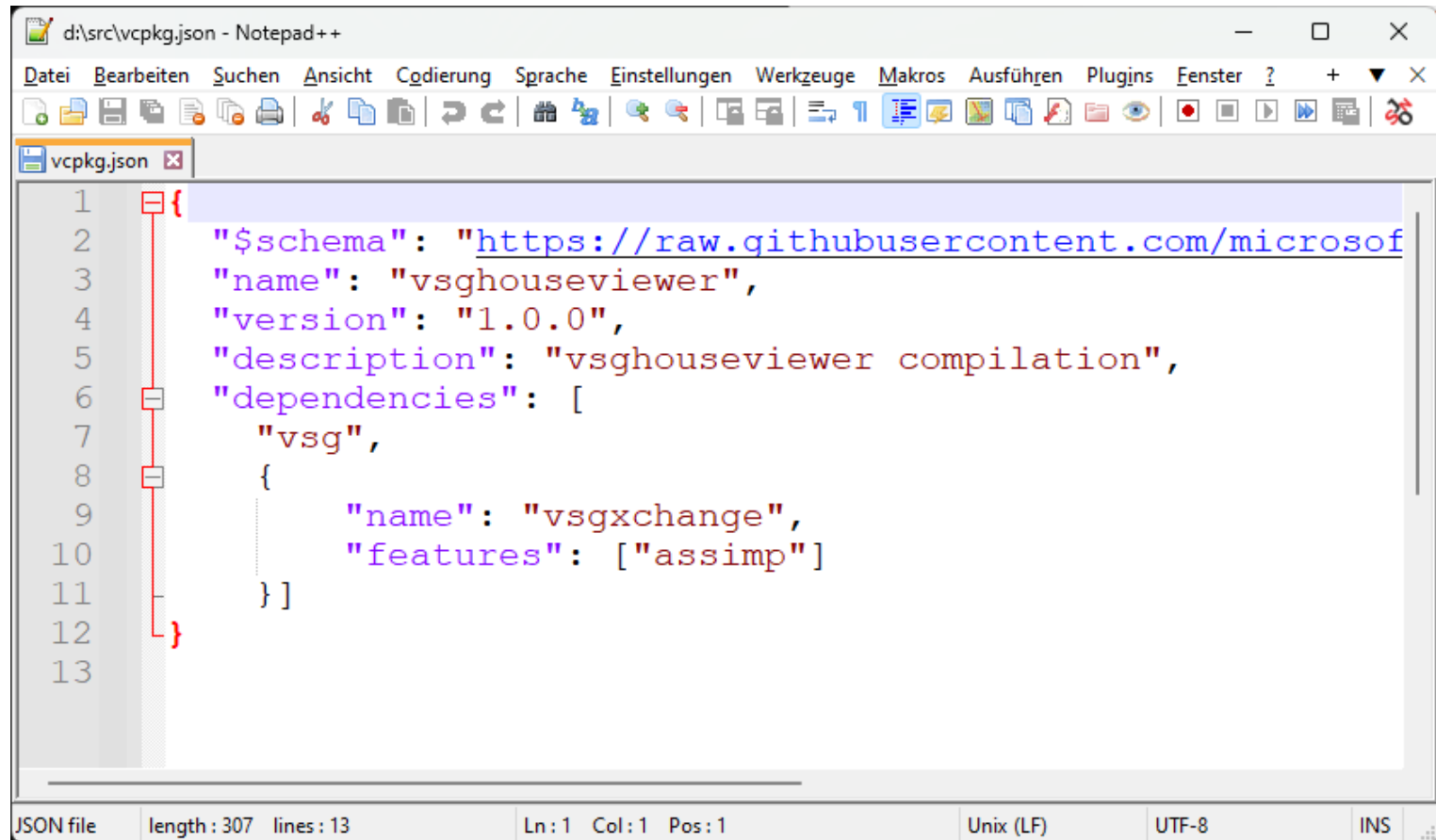
```
d:\src\CMakeLists.txt - Notepad++
Datei Bearbeiten Suchen Ansicht Codierung Sprache Einstellungen Werkzeuge Makros Ausführen Plugins Fenster ?
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.25)
2
3 project(vsghouseviewer)
4
5 find_package(vsg REQUIRED)
6 find_package(vsgXchange REQUIRED)
7
8 add_executable(${PROJECT_NAME} vsghouseviewer.cpp)
9
10 set_target_properties(${PROJECT_NAME} PROPERTIES CXX_STANDARD 17)
11
12 target_link_libraries(${PROJECT_NAME} vsg::vsg)
13 target_link_libraries(${PROJECT_NAME} vsgXchange::vsgXchange)
14

CMake file length: 371 lines: 14 Ln: 1 Col: 1 Pos: 1 Windows (CR LF) UTF-8 INS
```



# vcpkg-Manifest

vcpkg.json:

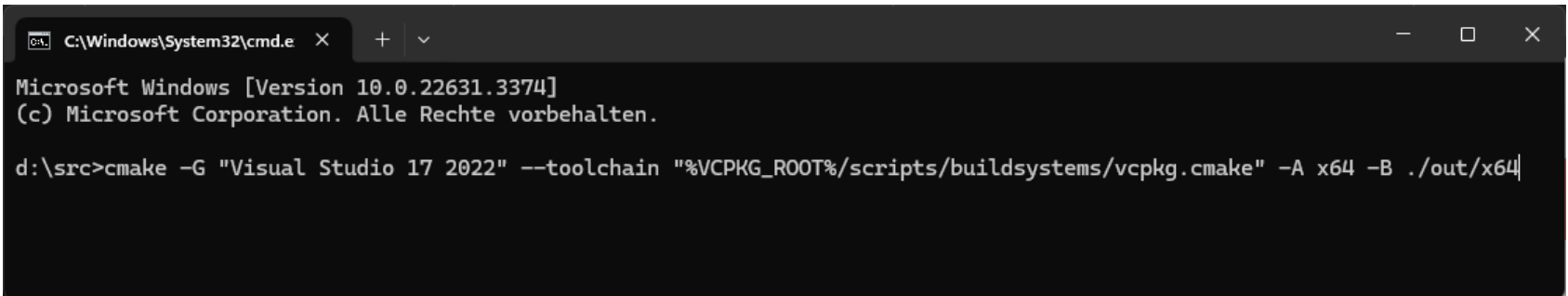


```
d:\src\vcpkg.json - Notepad++
Datei Bearbeiten Suchen Ansicht Codierung Sprache Einstellungen Werkzeuge Makros Ausführen Plugins Fenster ? + ▼ ×
vcpkg.json x
1 {
2   "$schema": "https://raw.githubusercontent.com/microsoft/vcpkg-registry/master/schemas/vcpkg.json",
3   "name": "vsghouseviewer",
4   "version": "1.0.0",
5   "description": "vsghouseviewer compilation",
6   "dependencies": [
7     "vsg",
8     {
9       "name": "vsgxchange",
10      "features": ["assimp"]
11    }
12  ]
13 }
```

JSON file   length : 307   lines : 13   Ln : 1   Col : 1   Pos : 1   Unix (LF)   UTF-8   INS

# Generierung

- nach Durchführung der vorherigen Schritte kann das Programm per CMake konfiguriert und gebildet werden
- **ACHTUNG**  
vcpkg-Toolchain muss zwingend angegeben werden



```
C:\Windows\System32\cmd.e  X  +  v
Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

d:\src>cmake -G "Visual Studio 17 2022" --toolchain "%VCPKG_ROOT%/scripts/buildsystems/vcpkg.cmake" -A x64 -B ./out/x64|
```

## Links zum Thema

- Link zu diesen Folien und dem Demo-Code inkl. weiterer Integrationen  
<https://github.com/swp-ariaci/cpp-usergroup-vcpkg>
- GitHub-Repo frei verfügbar unter  
<https://github.com/microsoft/vcpkg>
- guter Einstieg über  
<https://learn.microsoft.com/en-us/vcpkg>
- vcpkg-FAQ  
<https://learn.microsoft.com/en-us/vcpkg/about/faq>
- Package-Layout-Konventionen  
<https://learn.microsoft.com/en-us/vcpkg/reference/installation-tree-layout>
- Hauptseite  
<https://vcpkg.io/en>
- Vorstellung von 2016 unter  
<https://learn.microsoft.com/en-us/events/connect-2016/109>

## Warum nochmal vcpkg?

- <https://learn.microsoft.com/en-us/vcpkg/about/faq#why-not-conan> 😊
- vollständig self-maintained
- leichtgewichtiger Technologiestack
- Administrationsaufwand geringer
- abwärtskompatibel durch intensive Nutzung von CMake
- Package-Layout übersichtlicher
- Anbindung an verschiedenste Buildsysteme bereits integriert
- Debugging in Packages unproblematisch
- Editieren von Packages einfacher möglich

# Weitere Fragen?



C + +  
usergroup  
DRESDEN