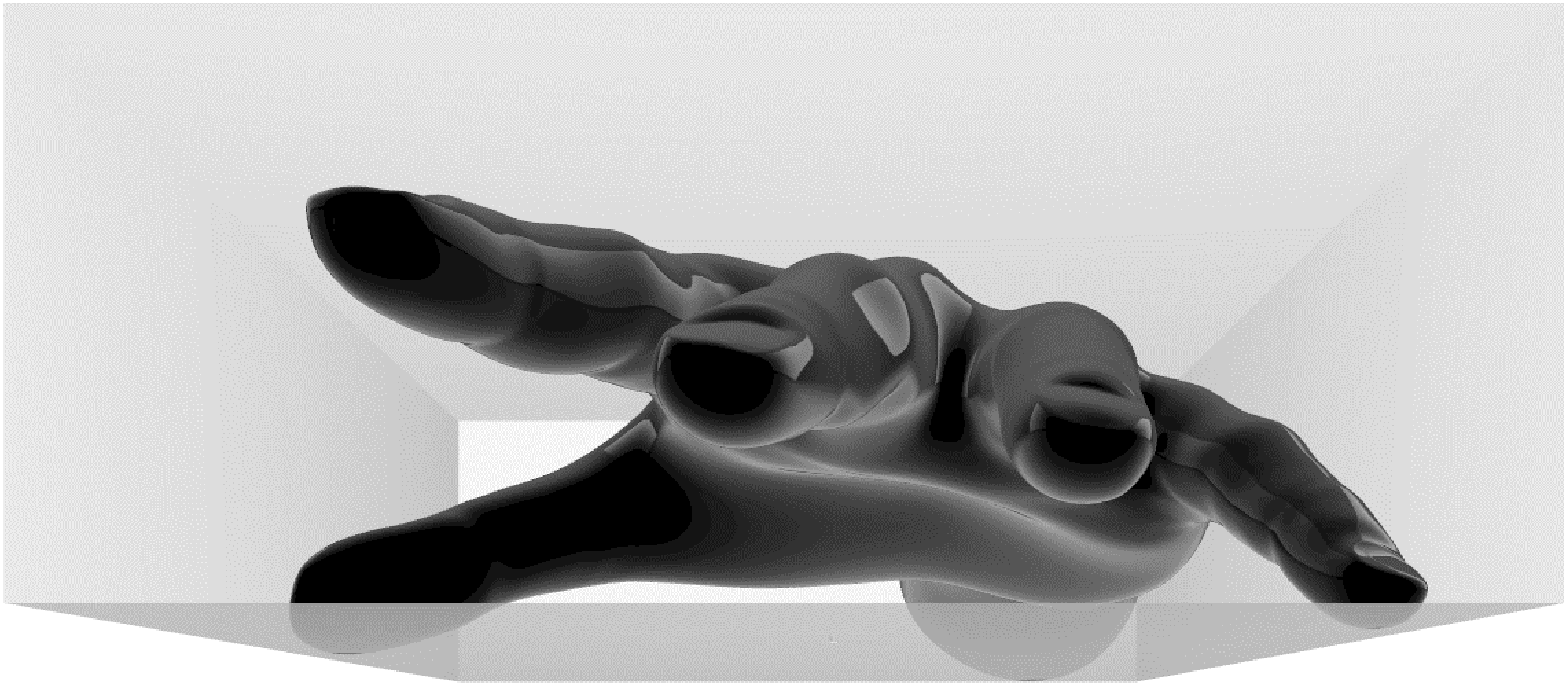


Wrapping C++ Native Code in C# with CGAL in Grasshopper as example

Spoiler 😎

The outcome of this tutorial

Citation

1. Wrapping Native Libraries by Dan Rigdon-Bel
2. Using methodgen by Giulio Piacentino
3. Cockroach by Petras Vestartas and Andrea Settimi

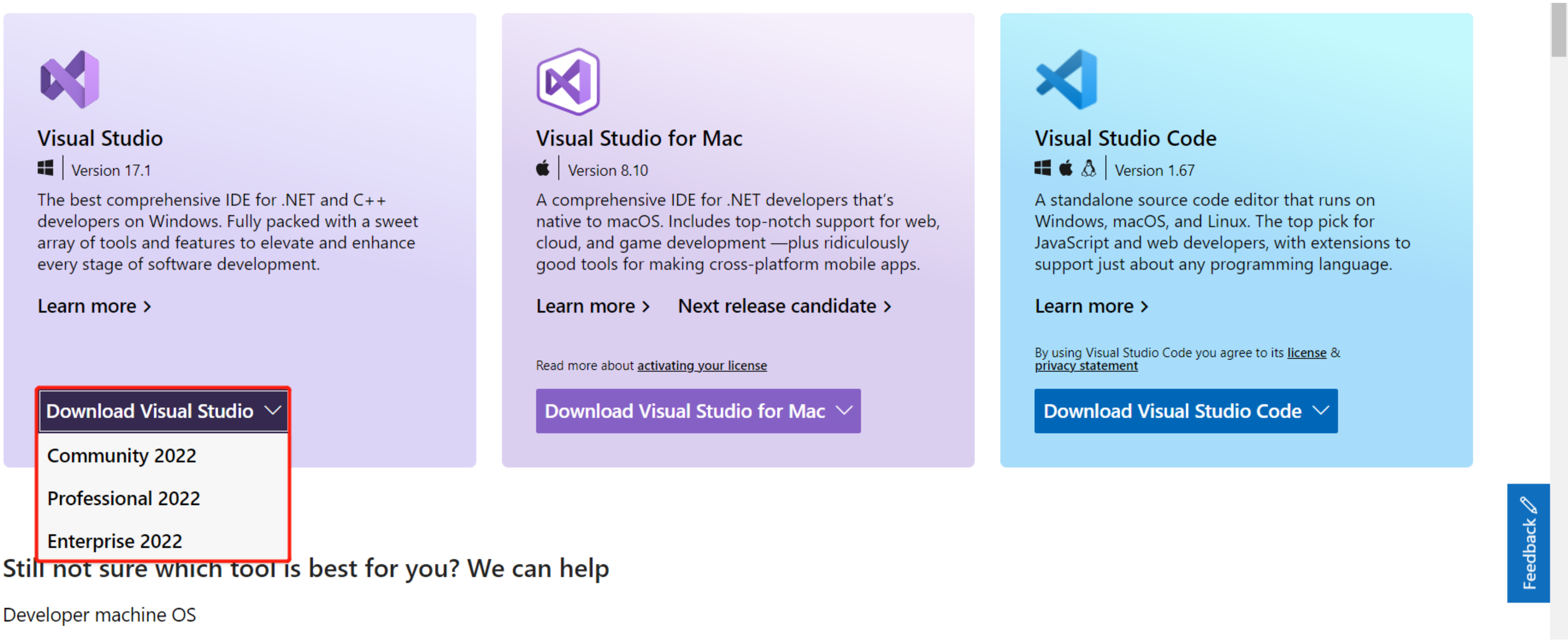
Disclaimer💬

1. I might make mistake during my sharing. 😐
2. I may try to explain the very detail. 🤐

Prerequisites ✅

Install Visual Studio

https://visualstudio.microsoft.com/



Visual Studio

Version 17.1

The best comprehensive IDE for .NET and C++ developers on Windows. Fully packed with a sweet array of tools and features to elevate and enhance every stage of software development.

Learn more ›

Visual Studio for Mac

Version 8.10

A comprehensive IDE for .NET developers that's native to macOS. Includes top-notch support for web, cloud, and game development —plus ridiculously good tools for making cross-platform mobile apps.

Learn more › Next release candidate ›

Read more about activating your license

Visual Studio Code

Version 1.67

A standalone source code editor that runs on Windows, macOS, and Linux. The top pick for JavaScript and web developers, with extensions to support just about any programming language.

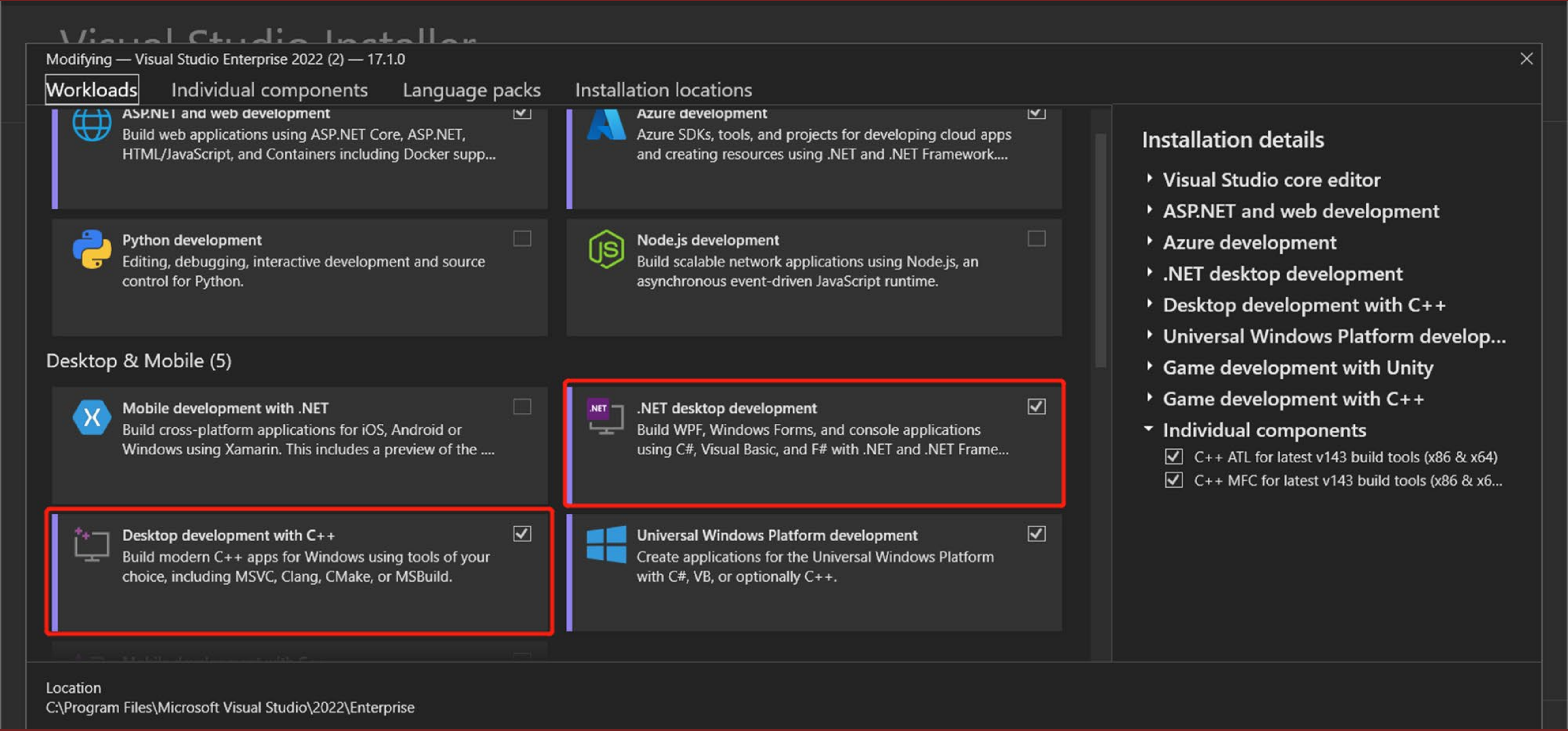
Learn more ›

By using Visual Studio Code you agree to its license & privacy statement

Still not sure which tool is best for you? We can help

Developer machine OS

Install Workloads and Individual components



Modifying — Visual Studio Enterprise 2022 (2) — 17.1.0

Workloads | Individual components | Language packs | Installation locations

[✓] ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...

[✓] Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....

[ ] Python development
Editing, debugging, interactive development and source control for Python.

[ ] Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

[ ] Mobile development with .NET
Build cross-platform applications for iOS, Android or Windows using Xamarin. This includes a preview of the ....

[✓] .NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

[✓] Desktop development with C++
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

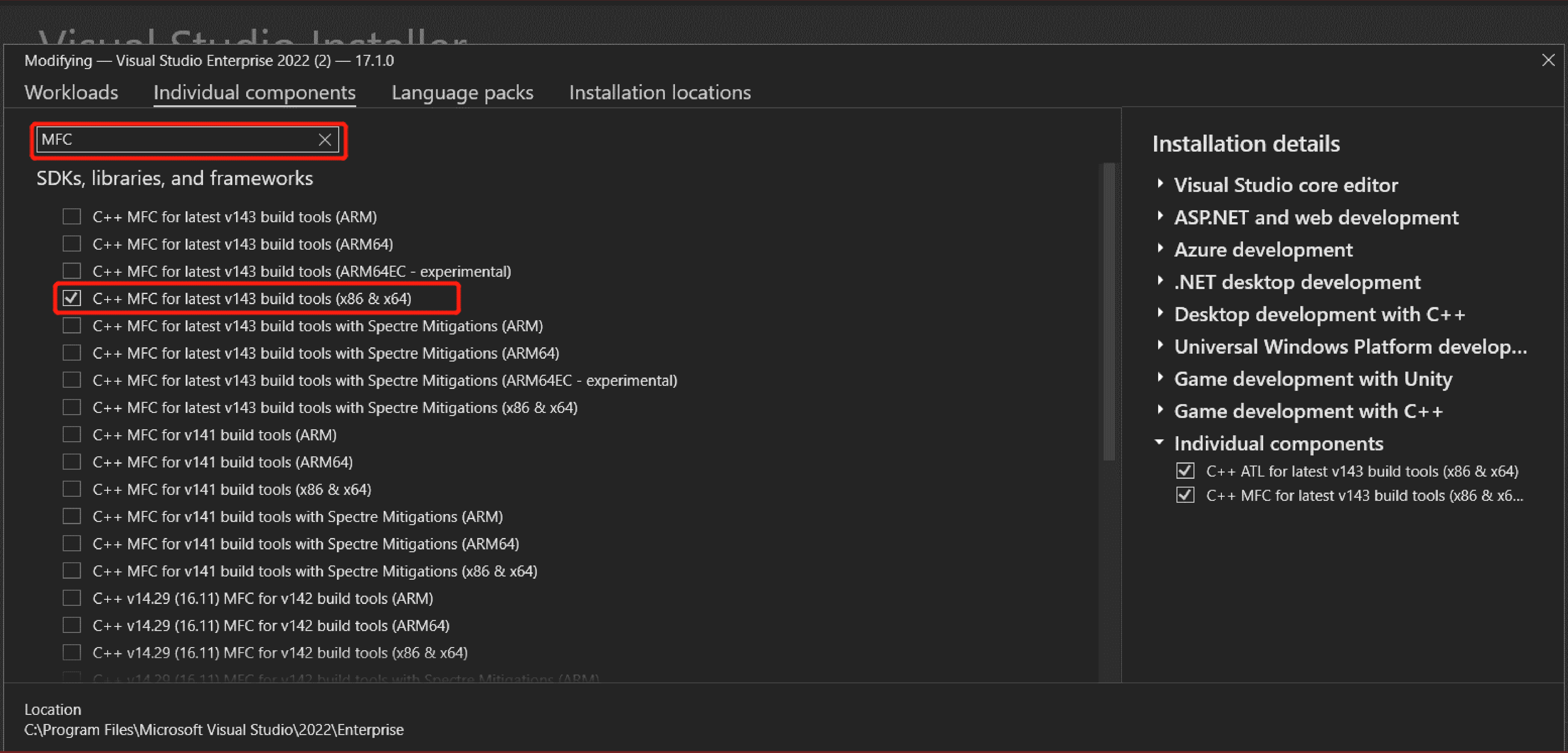
[✓] Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Installation details

- Visual Studio core editor
- ASP.NET and web development
- Azure development
- .NET desktop development
- Desktop development with C++
- Universal Windows Platform develop...
- Game development with Unity
- Game development with C++
- Individual components
  - [✓] C++ ATL for latest v143 build tools (x86 & x64)
  - [✓] C++ MFC for latest v143 build tools (x86 & x6...

Location
C:\Program Files\Microsoft Visual Studio\2022\Enterprise

Install Workloads and Individual components



Visual Studio Installer

Modifying — Visual Studio Enterprise 2022 (2) — 17.1.0

Workloads | Individual components | Language packs | Installation locations

MFC

SDKs, libraries, and frameworks

- [ ] C++ MFC for latest v143 build tools (ARM)
- [ ] C++ MFC for latest v143 build tools (ARM64)
- [ ] C++ MFC for latest v143 build tools (ARM64EC - experimental)
- [✓] C++ MFC for latest v143 build tools (x86 & x64)
- [ ] C++ MFC for latest v143 build tools with Spectre Mitigations (ARM)
- [ ] C++ MFC for latest v143 build tools with Spectre Mitigations (ARM64)
- [ ] C++ MFC for latest v143 build tools with Spectre Mitigations (ARM64EC - experimental)
- [ ] C++ MFC for latest v143 build tools with Spectre Mitigations (x86 & x64)
- [ ] C++ MFC for v141 build tools (ARM)
- [ ] C++ MFC for v141 build tools (ARM64)
- [ ] C++ MFC for v141 build tools (x86 & x64)
- [ ] C++ MFC for v141 build tools with Spectre Mitigations (ARM)
- [ ] C++ MFC for v141 build tools with Spectre Mitigations (ARM64)
- [ ] C++ MFC for v141 build tools with Spectre Mitigations (x86 & x64)
- [ ] C++ v14.29 (16.11) MFC for v142 build tools (ARM)
- [ ] C++ v14.29 (16.11) MFC for v142 build tools (ARM64)
- [ ] C++ v14.29 (16.11) MFC for v142 build tools (x86 & x64)
- [ ] C++ v14.29 (16.11) MFC for v142 build tools with Spectre Mitigations (ARM)

Installation details

- Visual Studio core editor
- ASP.NET and web development
- Azure development
- .NET desktop development
- Desktop development with C++
- Universal Windows Platform develop...
- Game development with Unity
- Game development with C++
- Individual components
  - [✓] C++ ATL for latest v143 build tools (x86 & x64)
  - [✓] C++ MFC for latest v143 build tools (x86 & x6...

Location
C:\Program Files\Microsoft Visual Studio\2022\Enterprise

Install Git

https://git-scm.com/

Install vcpkg

> Open Powershell

```
PS C:\Users\Xingxin>
```

> Change to the C folder

```
cd C:\
```

> clone vcpkg

```
git clone https://github.com/Microsoft/vcpkg.git
```

> Enter vcpkg folder

```
cd vcpkg
```

> Build vcpkg

```
.\bootstrap-vcpkg.bat
```

Install CGAL

> Install yasm-tool

```
.\vcpkg.exe install yasm-tool:x86-windows
```

> Install CGAL

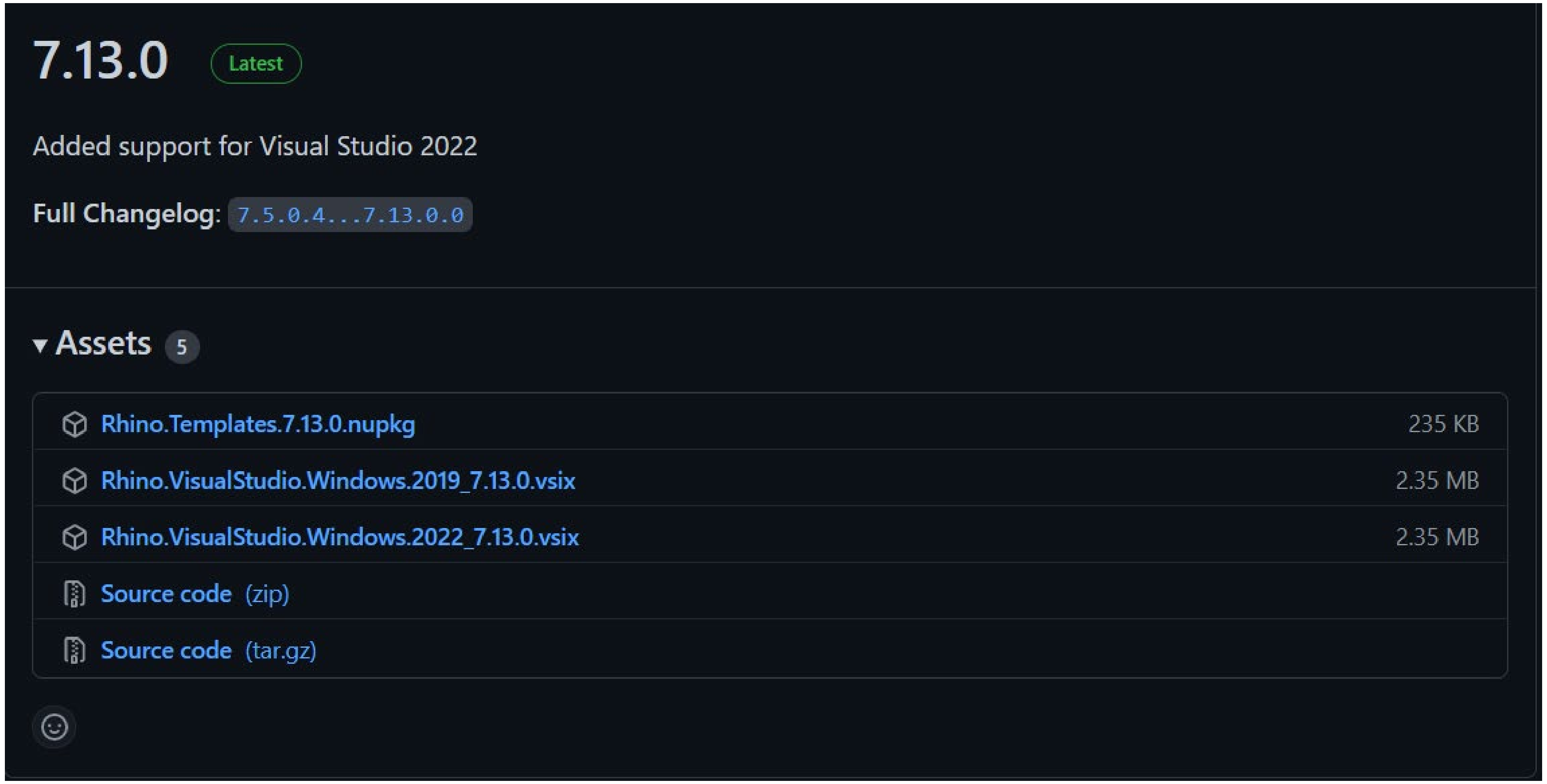
```
.\vcpkg.exe install cgal:x64-windows
```

> Using vcpkg with MSBuild

```
.\vcpkg.exe integrate install
```

Install Rhino Visual Studio Extensions

https://github.com/mcneel/RhinoVisualStudioExtensions/releases



7.13.0 Latest

Added support for Visual Studio 2022

Full Changelog: 7.5.0.4...7.13.0.0

Assets 5

Rhino.Templates.7.13.0.nupkg	235 KB
Rhino.VisualStudio.Windows.2019_7.13.0.vsix	2.35 MB
Rhino.VisualStudio.Windows.2022_7.13.0.vsix	2.35 MB
Source code (zip)	
Source code (tar.gz)	

Overview

Why Wrapping Native Code?

The perspective of developer

- DRY(Don’t Repeat Yourself!)

©CGAL

©Open3D

©libigl

©pcl

- Numeric Computation

Jargon

Unmanaged Code (unsafe/native) & Managed Code

- 🙂Managed Code - Managed by CLR(Common Language Runtime)
- 😳Unmanaged Code – not managed by CLR

📌 Platform Invoke (P/Invoke)

🚍 P/Invoke is a technology that allows you to access structs, callbacks, and functions in unmanaged libraries from your managed code.

https://docs.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke

```
[DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
```

C++ 🚎🚌 C#

Marshal

🎭Marshalling is the process of transforming types when they need to cross between managed and native code.

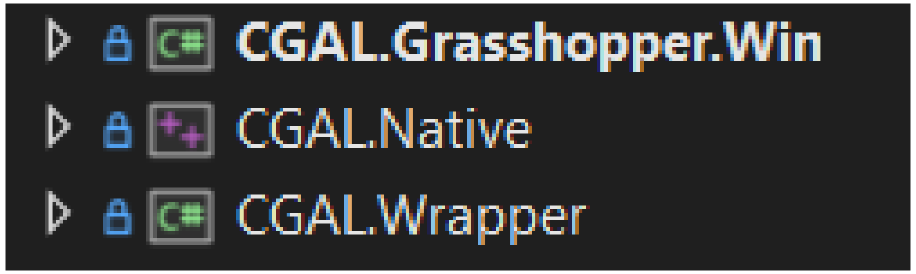
https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.marshal?view=net-6.0

```
Marshal.Copy();
```

C++ ←。。🎭。。→ C#

Fundamental

The Structure of Solution



CGAL.Grasshopper.Win 3
CGAL.Native 1
CGAL.Wrapper 2

The File Structure

.
- ├── CGAL.Grasshopper.Win.gha
- ├── **CGAL.Native.dll** ◍
- ├── CGAL.Native.exp ◍
- ├── CGAL.Native.lib ◍
- ├── CGAL.Native.pdb ◍
- ├── **CGAL.Wrapper.dll** ▨
- ├── RhinoInside.dll ▨
- └── gmp-10.dll ◍

◍ C++ ▨ C#

Windows DLL Search Path

- Application directory
  - The current directory
    - The system directory
      - The 16-bit system directory
        - The Windows directory
          - PATH environment variable

Marshal Types

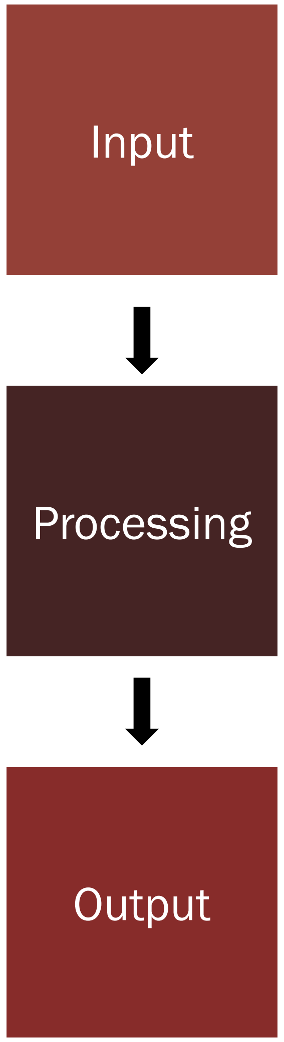
C++	C#
ON_Mesh*	IntPtr
double*	double[]
int*	int []
unsigned int	uint
…	…

https://docs.microsoft.com/en-us/dotnet/framework/interop/marshalling-data-with-platform-invoke
https://docs.microsoft.com/en-us/cpp/dotnet/calling-native-functions-from-managed-code?view=msvc-170

Case Study - CGAL::oriented_bounding_box ⌨

CGAL::bounding_box

CGAL::bounding_box

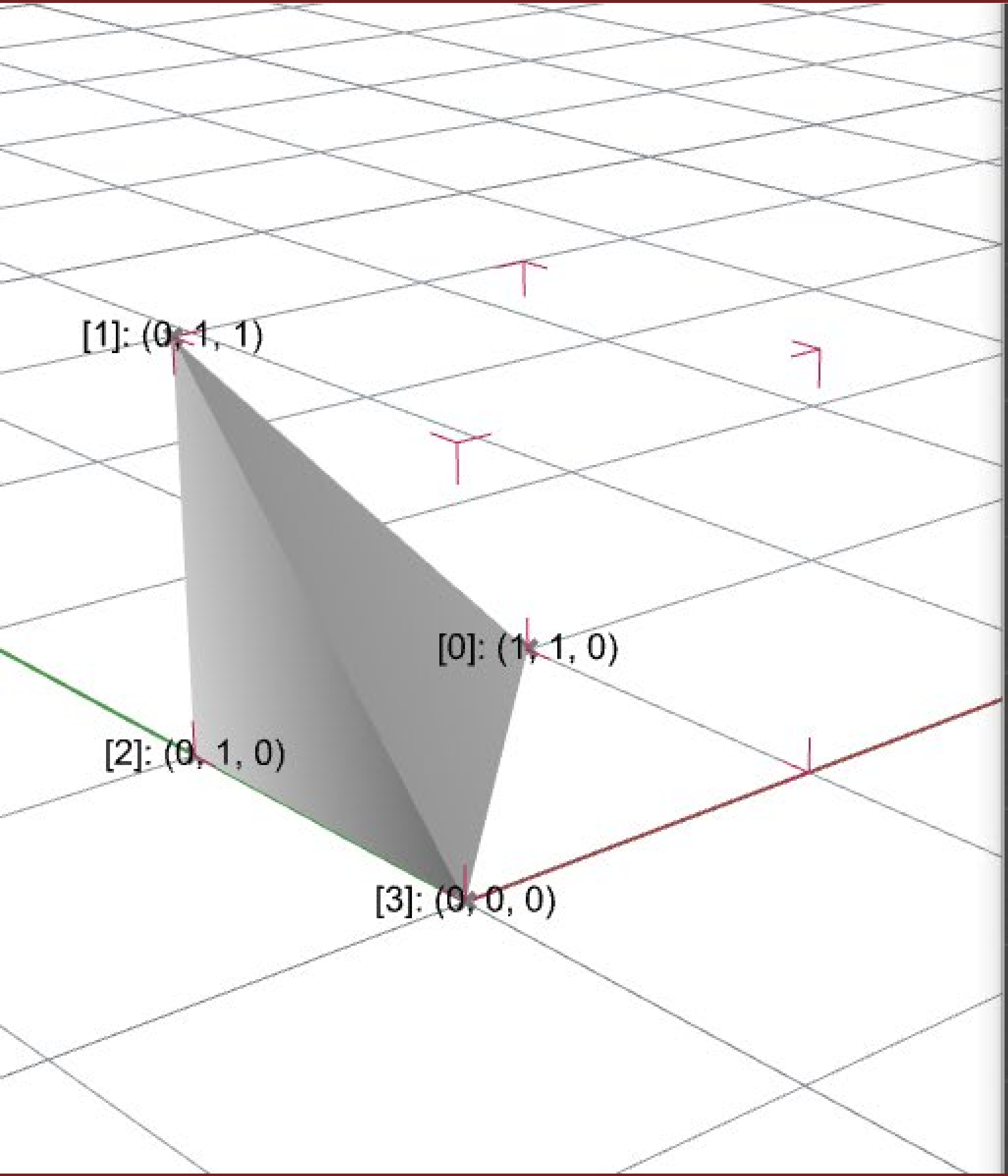


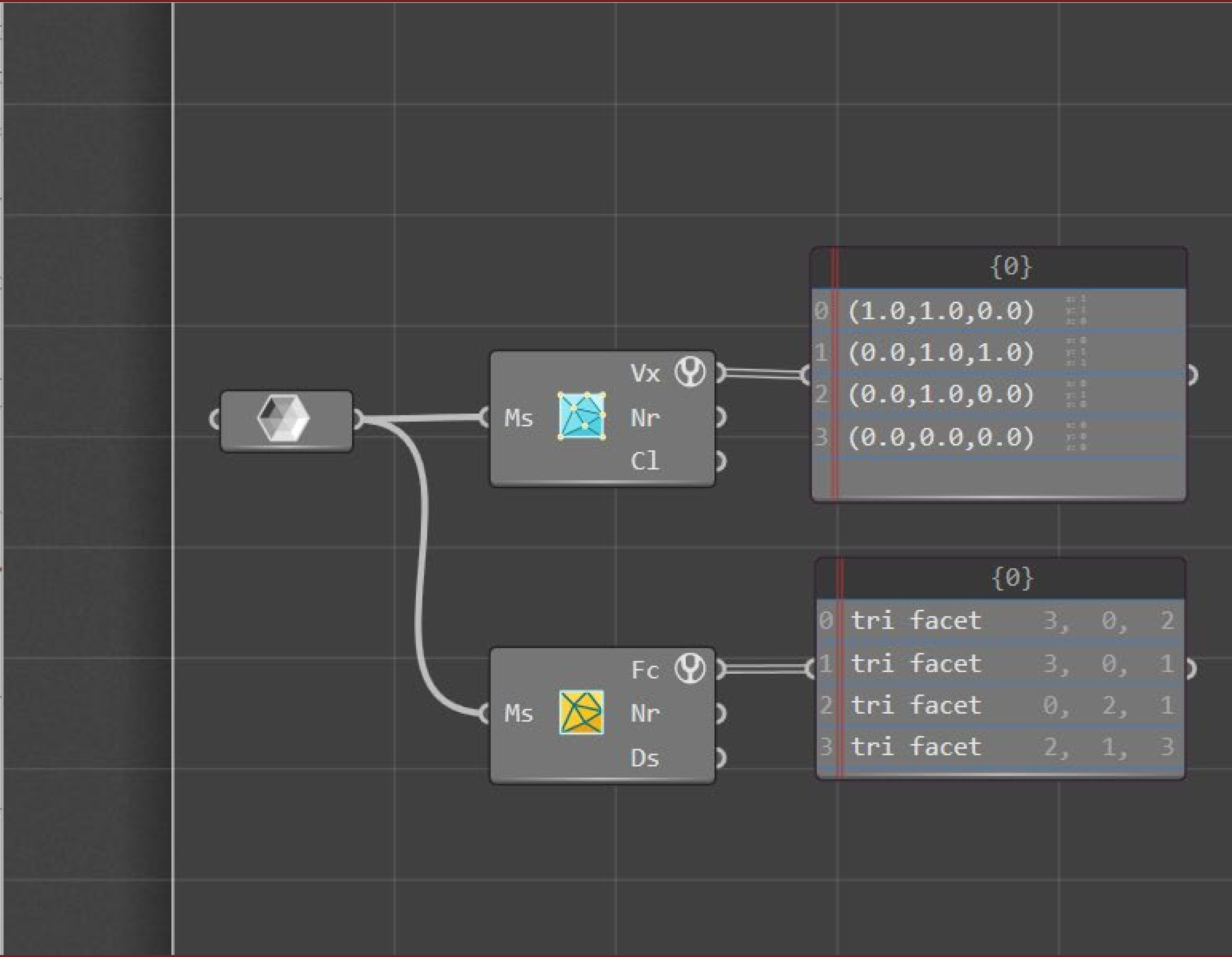
```
CGAL::Surface_mesh<K::Point_3> Mesh;
       K::Point_3
       SM_Vertex_index
```

```
CGAL::oriented_bounding_box(mesh, obb_points,
CGAL::parameters::use_convex_hull(true));
```

```
std::array<K::Point_3, 8> obb_points;
```

Rhino.Geometry.Mesh





```
{0}
0 (1.0,1.0,0.0)
1 (0.0,1.0,1.0)
2 (0.0,1.0,0.0)
3 (0.0,0.0,0.0)
```

```
{0}
0 tri facet    3,  0,  2
1 tri facet    3,  0,  1
2 tri facet    0,  2,  1
3 tri facet    2,  1,  3
```

Strategy

Step 1 Rhino.Geometry.Mesh ♻

- info of vertices
- info of faces

Step 2 Info from Rhino ♻

- vertices of mesh in CGAL
- faces of mesh in CGAL

Step 3 Run bounding_box CGAL🔨

- 8 * Points of bounding box

Step 4 Info of CGAL::Point ♻

- info of xyz coordinates

Step 5 Info from CGAL ♻

- Convert to Rhino.Geometry.Point3d

TODO List

Step 1 Rhino.Geometry.Mesh ♻
- info of vertices
- info of faces

Step 2 Info from Rhino ♻
- vertices of mesh in CGAL
- faces of mesh in CGAL

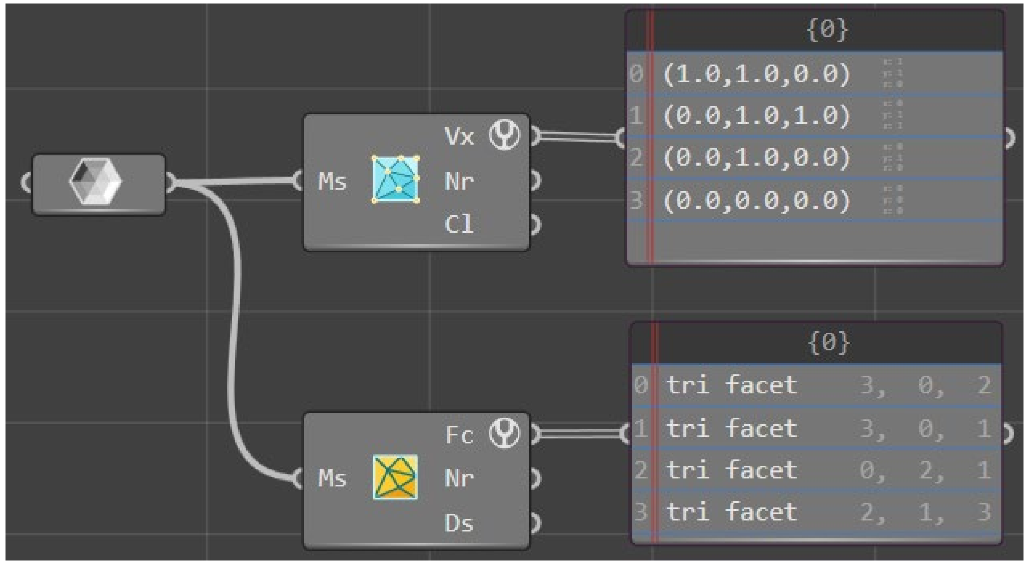
Step 3 Run bounding_box CGAL
- 8 * Points of bounding box

Step 4 Info of CGAL::Point ♻
- info of xyz coordinates

Step 5 Info from CGAL ♻
- Convert to Rhino.Geometry.Point3d

📌 TODO List - Step 1 Rhino.Geometry.Mesh ♻️

Rhino.Geometry.Mesh



```
{0}
0 (1.0,1.0,0.0)
1 (0.0,1.0,1.0)
2 (0.0,1.0,0.0)
3 (0.0,0.0,0.0)
```

```
{0}
0 tri facet   3, 0, 2
1 tri facet   3, 0, 1
2 tri facet   0, 2, 1
3 tri facet   2, 1, 3
```

→

Marshal Types

```
IntPtr
double[]
int []
uint
…
```

📌 TODO List - Step 1 Rhino.Geometry.Mesh ♻️

TODO List

~~Step 1 Rhino.Geometry.Mesh~~ ♻ ✓

~~- info of vertices~~

~~- info of faces~~

Step 2 Info from Rhino ♻

- vertices of mesh in CGAL
- faces of mesh in CGAL

Step 3 Run bounding_box CGAL

- 8 * Points of bounding box

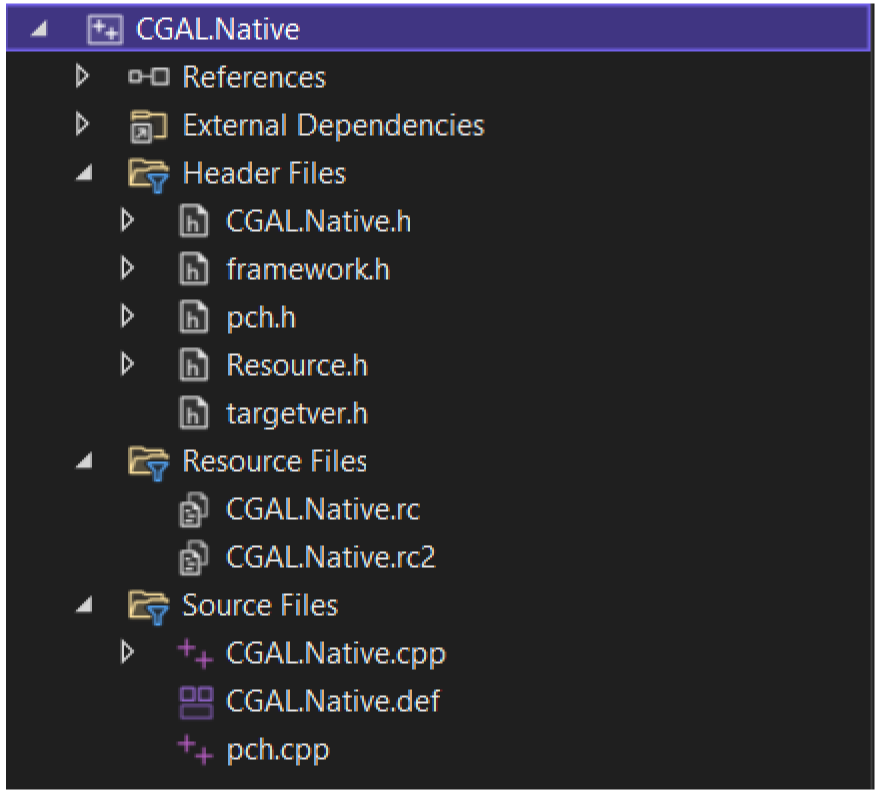
Step 4 Info of CGAL::Point ♻

- info of xyz coordinates

Step 5 Info from CGAL ♻

- Convert to Rhino.Geometry.Point3d

C++ Boilerplate



```
◢ CGAL.Native
    ▷ References
    ▷ External Dependencies
    ◢ Header Files
        ▷ CGAL.Native.h
        ▷ framework.h
        ▷ pch.h
        ▷ Resource.h
          targetver.h
    ◢ Resource Files
          CGAL.Native.rc
          CGAL.Native.rc2
    ◢ Source Files
        ▷ CGAL.Native.cpp
          CGAL.Native.def
          pch.cpp
```

Header Files

Function Prototypes

Source Files

Function Implementation

“pch.h” – precompiled header file

C++ Exported Functions

```
// Windows build
extern "C" __declspec(dllexport)
void some_function(/* arguments */);
```

```
// Apple build
extern "C" __attribute__ ((visibility ("default")))
void some_function(/* arguments */);
```

Precompile MACRO

```
// Windows build
#if defined (_WIN32)
#if defined (CGALNATIVE_DLL_EXPORTS)
#define CGALNATIVE_CPP_CLASS __declspec(dllexport)
#define CGALNATIVE_CPP_FUNCTION __declspec(dllexport)
#define CGALNATIVE_C_FUNCTION extern "C" __declspec(dllexport)
#else
#define CGALNATIVE_CPP_CLASS __declspec(dllimport)
#define CGALNATIVE_CPP_FUNCTION __declspec(dllimport)
#define CGALNATIVE_C_FUNCTION extern "C" __declspec(dllimport)
#endif // CGALNATIVE_DLL_EXPORTS
#endif // _WIN32

// Apple build
#if defined(__APPLE__)
#define CGALNATIVE_CPP_CLASS __attribute__ ((visibility ("default")))
#define CGALNATIVE_CPP_FUNCTION __attribute__ ((visibility ("default")))
#define CGALNATIVE_C_FUNCTION extern "C" __attribute__ ((visibility ("default")))
#endif // __APPLE__
```

Bounding Box Function Prototype

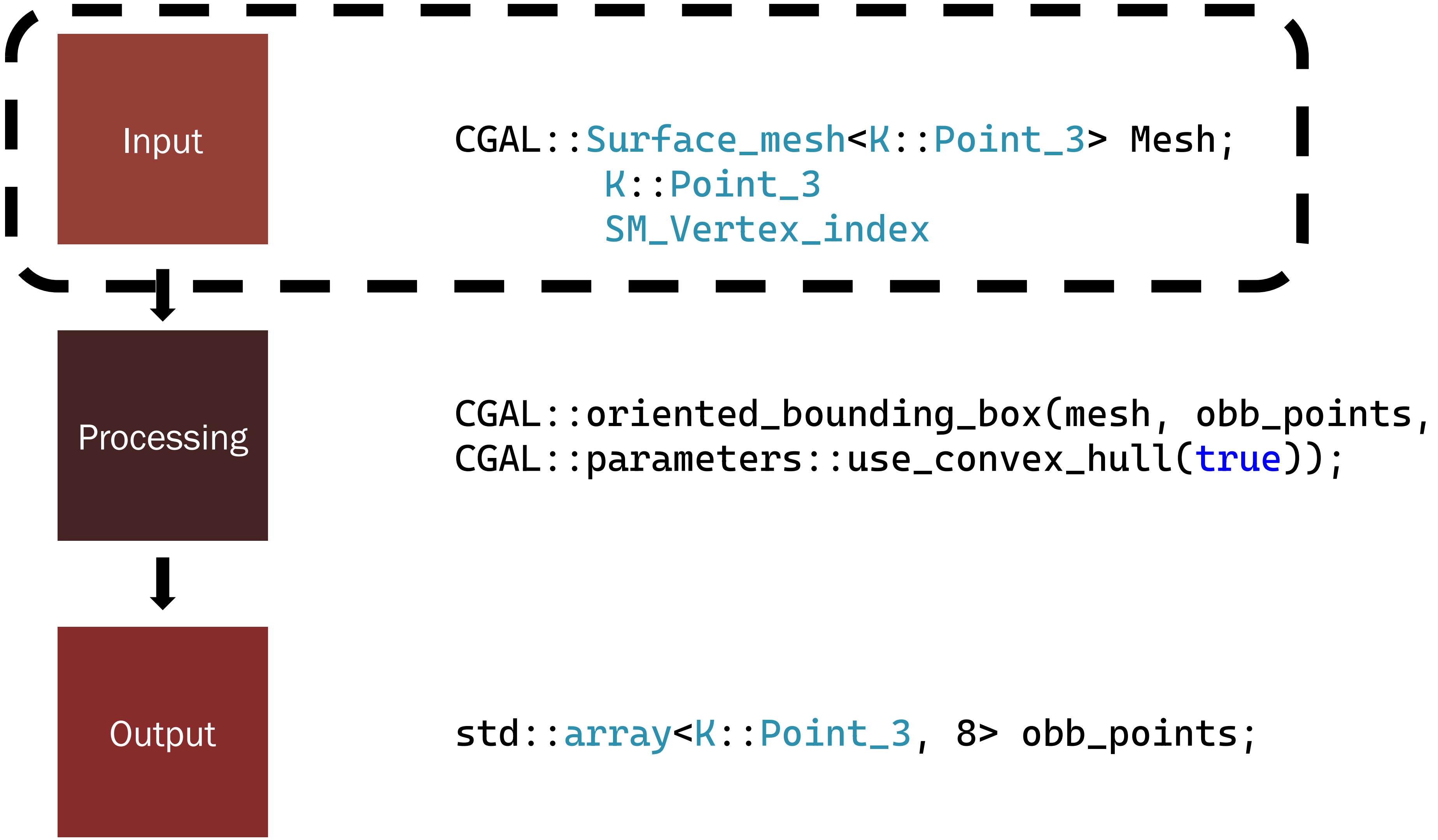
▨

```
// info of vertices
double[] vertXyzArray = new double[m.Vertices.Count * 3];
var vertCount = (ulong)m.Vertices.Count;
// info of faces
int[] faceIndexArray = m.Faces.ToIntArray(true);
var facesCount = (ulong)m.Faces.Count;
```

◍

```
CGALNATIVE_C_FUNCTION
void OrientedBoundingBoxBySurfaceMesh(
        double* vert_xyz_array,  size_t vert_count,
        int* face_index_array,   size_t faces_count);
```

CGAL::bounding_box



```
CGAL::Surface_mesh<K::Point_3> Mesh;
       K::Point_3
       SM_Vertex_index
```

```
CGAL::oriented_bounding_box(mesh, obb_points,
CGAL::parameters::use_convex_hull(true));
```

```
std::array<K::Point_3, 8> obb_points;
```

📌 TODO List - Step 2 Info from Rhino ♻️

TODO List

~~Step 1 Rhino.Geometry.Mesh ♻~~ ✓

~~- info of vertices~~

~~- info of faces~~

~~Step 2 Info from Rhino ♻~~ ✓

~~- vertices of mesh in CGAL~~

~~- faces of mesh in CGAL~~

Step 3 Run bounding_box CGAL🔨

- 8 * Points of bounding box

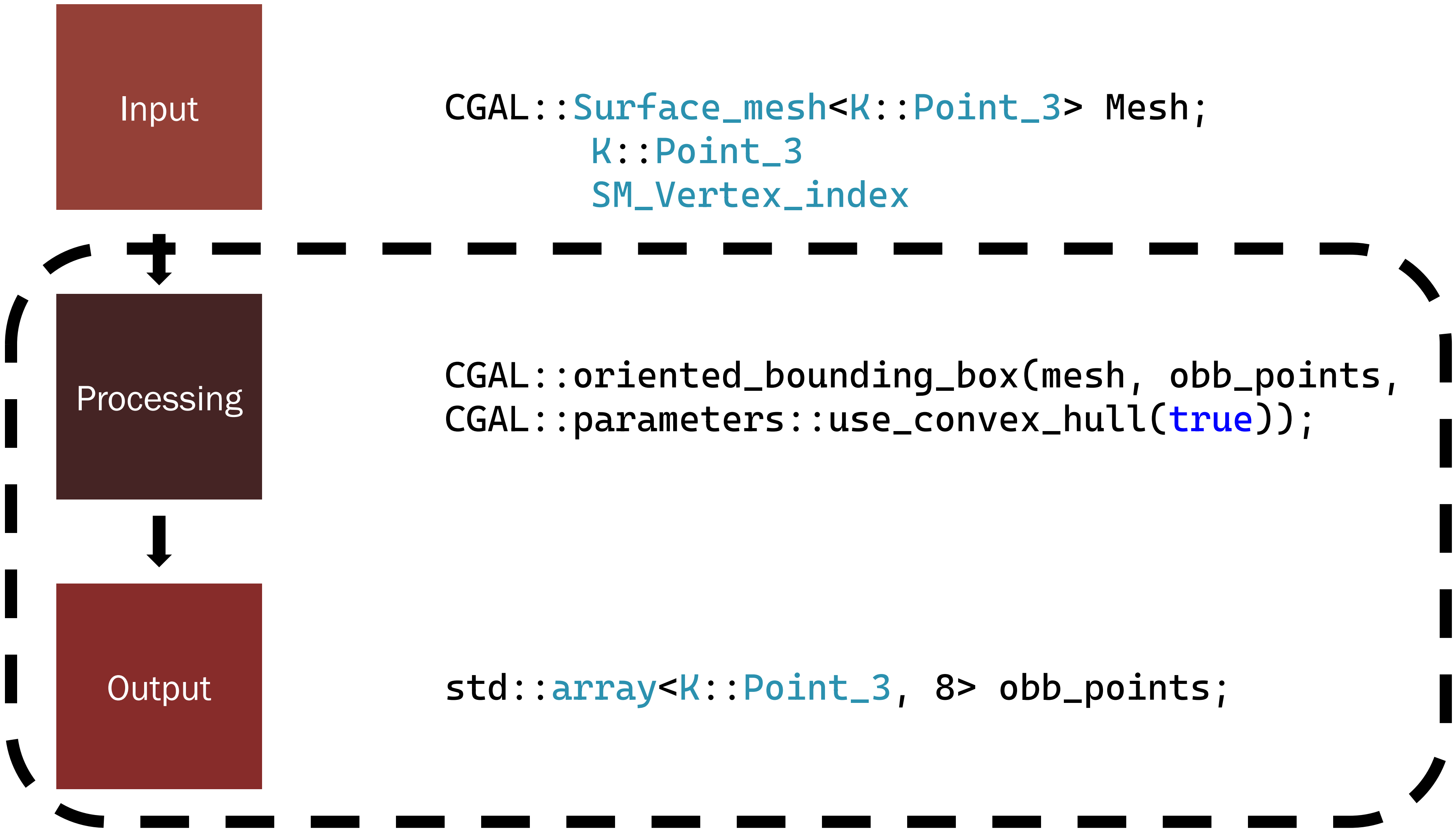
Step 4 Info of CGAL::Point ♻

- info of xyz coordinates

Step 5 Info from CGAL ♻

- Convert to Rhino.Geometry.Point3d

CGAL::bounding_box



```
CGAL::Surface_mesh<K::Point_3> Mesh;
       K::Point_3
       SM_Vertex_index
```

```
CGAL::oriented_bounding_box(mesh, obb_points,
CGAL::parameters::use_convex_hull(true));
```

```
std::array<K::Point_3, 8> obb_points;
```

TODO List - Step 3 Run bounding_box CGAL

TODO List

~~Step 1 Rhino.Geometry.Mesh ♻~~ ✓

~~- info of vertices~~

~~- info of faces~~

~~Step 2 Info from Rhino ♻~~ ✓

~~- vertices of mesh in CGAL~~

~~- faces of mesh in CGAL~~

~~Step 3 Run bounding_box CGAL 🔨~~ ✓

~~- 8 * Points of bounding box~~

Step 4 Info of CGAL::Point ♻

- info of xyz coordinates

Step 5 Info from CGAL ♻

- Convert to Rhino.Geometry.Point3d

📌 TODO List - Step 4 Info of CGAL::Point ♻️

Function Prototype C++

```
CGALNATIVE_C_FUNCTION
void OrientedBoundingBoxBySurfaceMesh(
      double* vert_xyz_array,  size_t vert_count,
      int* face_index_array,   size_t faces_count
);
```

Function Prototype C++

input

output

```
CGALNATIVE_C_FUNCTION
void OrientedBoundingBoxBySurfaceMesh(
    double* vert_xyz_array,   size_t vert_count,
    int* face_index_array,    size_t faces_count,
    double*& obb_xyz_array,   int& obb_pts_count
);
```

Memory Management

manage memory
<u>manually</u>

CLR manage memory
<u>automatically</u>

Memory Management

```
void ReleaseDoubleArray(double* arr)
{
        delete[] arr;
}
```

TODO List

~~Step 1 Rhino.Geometry.Mesh~~ ♻ ✓

~~- info of vertices~~

~~- info of faces~~

~~Step 2 Info from Rhino~~ ♻ ✓

~~- vertices of mesh in CGAL~~

~~- faces of mesh in CGAL~~

~~Step 3 Run bounding_box CGAL~~ 🔨 ✓

~~- 8 * Points of bounding box~~

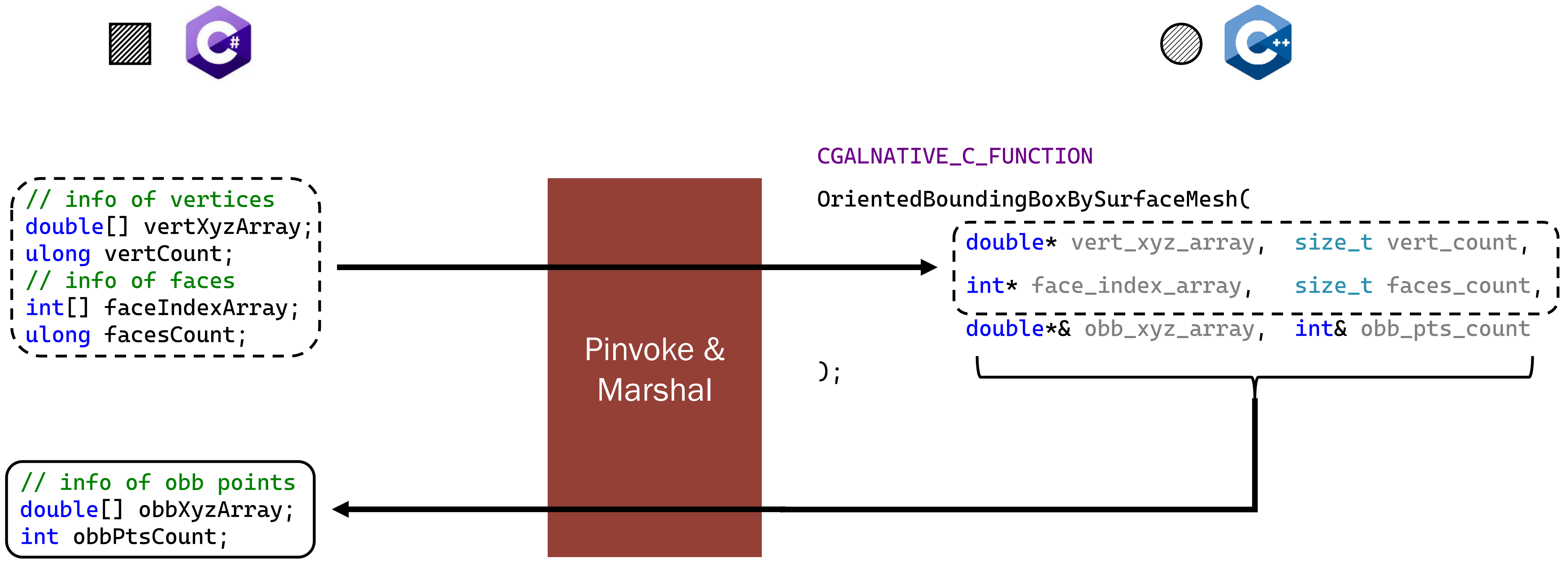
~~Step 4 Info of CGAL::Point~~ ♻ ✓

~~- info of xyz coordinates~~

Step 5 Info from CGAL ♻

- Convert to Rhino.Geometry.Point3d

Summary so far

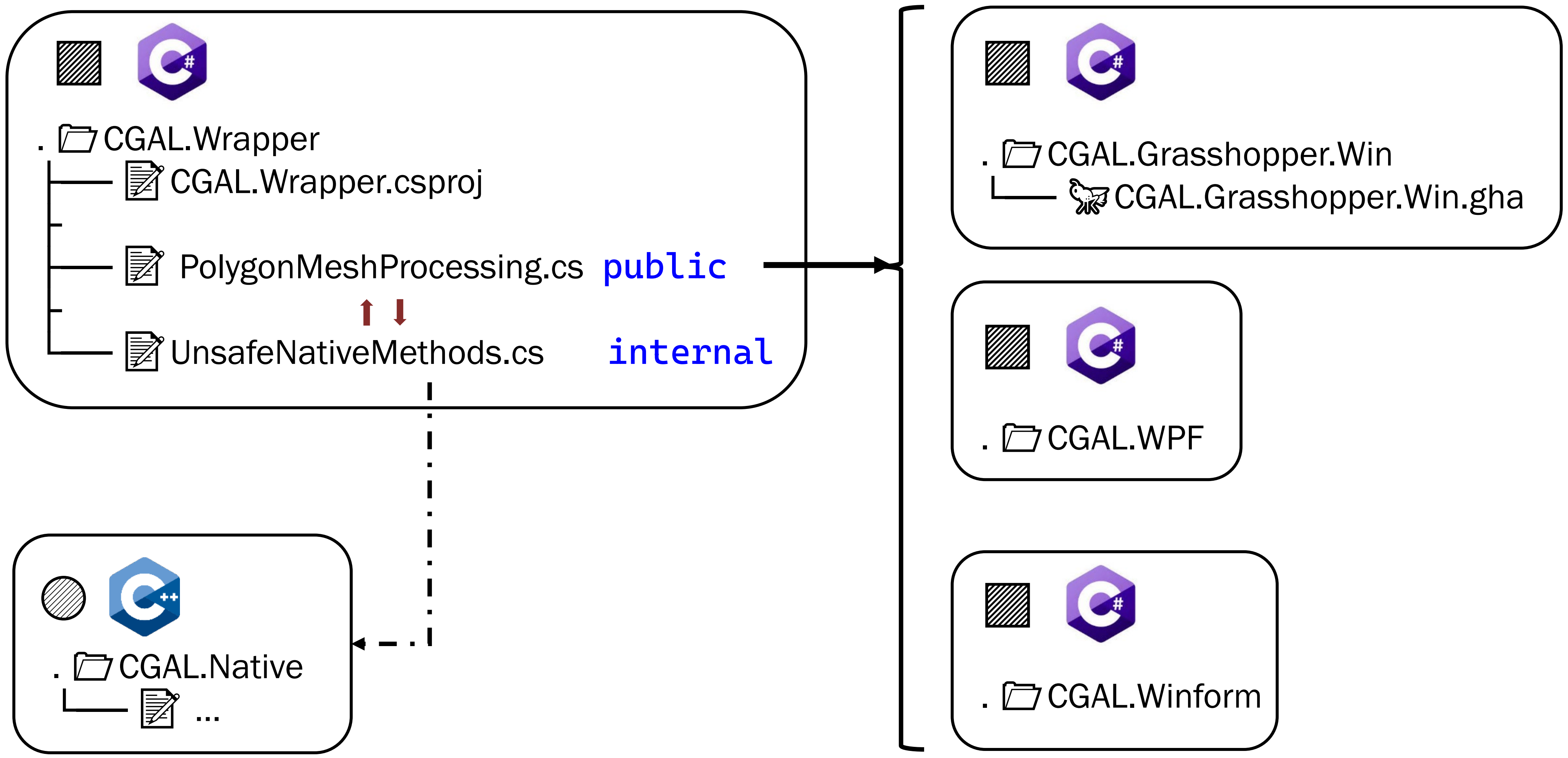


```csharp
// info of vertices
double[] vertXyzArray;
ulong vertCount;
// info of faces
int[] faceIndexArray;
ulong facesCount;
```

```csharp
// info of obb points
double[] obbXyzArray;
int obbPtsCount;
```

```cpp
CGALNATIVE_C_FUNCTION
OrientedBoundingBoxBySurfaceMesh(
    double* vert_xyz_array,  size_t vert_count,
    int* face_index_array,   size_t faces_count,
    double*& obb_xyz_array,  int& obb_pts_count
);
```

Convention of Wrapper Project



Unsafe Native Methods

```
// Windows build
extern "C" __declspec(dllexport)
void some_function(/* arguments
*/);
```

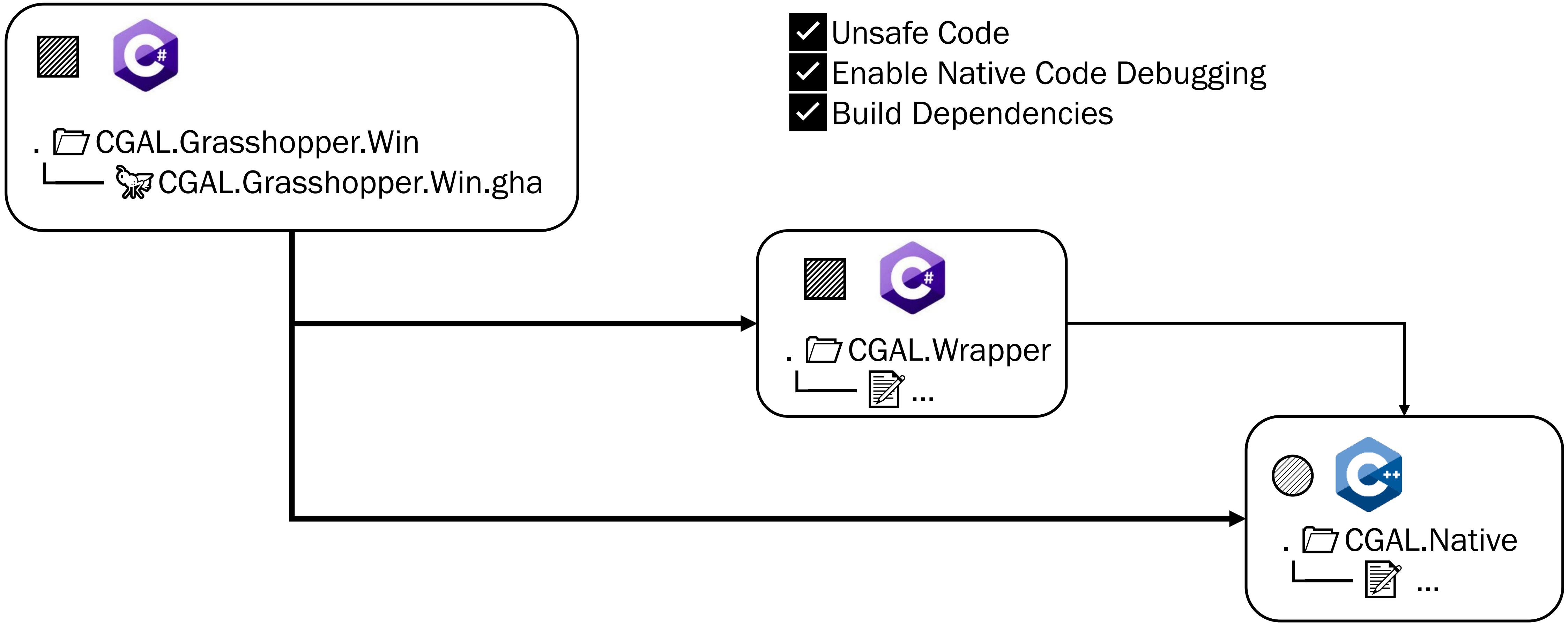
```
private const string DLL_NAME = "CGAL.Native.dll";

[DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
internal static extern void OrientedBoundingBoxBySurfaceMesh(
    [MarshalAs(UnmanagedType.LPArray)] double[] mesh_vert_xyz, ulong mesh_vert_count,
    [MarshalAs(UnmanagedType.LPArray)] int[] mesh_face_vertIndex, ulong mesh_face_count,

    ref IntPtr obb_pts_xyz, ref int obb_pts_count
);
```

Prebuilt Event

```
SolutionDirectory/
├── deps/
│   ├── CGAL.Native.dll ✓
├── CGAL.Grasshopper/
│   ├── CGAL.Wrapper.dll ✓
│   ├── CGAL.Native.dll ✓
│   ├── CGAL.Grasshopper.gha ✓
├── CGAL.Wrapper/
│   ├── CGAL.Native.dll ✓
│   ├── CGAL.Wrapper.dll ✓
├── CGAL.Native/
│   ├── CGAL.Native.dll ✓
```

Debug C++ Code in C#

☑ Unsafe Code
☑ Enable Native Code Debugging
☑ Build Dependencies



📌 TODO List - Step 5 Info from CGAL ♻️

Summary📝

Prerequisites☑

Overview🔭

Fundamental📦

Case Study⌨

CGAL