
Documentation: Creating a `.vtk` File Using VTK and CMake

Objective:

To create a `.vtk` file containing a 3D structured grid with a time-evolving vector field. The `.vtk` file is generated using the Visualization Toolkit (VTK) and compiled using a CMake-based build system.

Steps to Create the `.vtk` File:

1. Folder Structure:

Both the `CMakeLists.txt` file and the C++ source code file (`VectorFieldGenerator.cxx`) must be located in the same folder for the CMake configuration to work seamlessly. For example:

```
/ProjectFolder
│
├── CMakeLists.txt
├── VectorFieldGenerator.cxx
```

This ensures that CMake can correctly locate the source file (`VectorFieldGenerator.cxx`) specified in the `CMakeLists.txt` file.

---

2. C++ Code Implementation:

The vector field generation and `.vtk` file creation were implemented in the `VectorFieldGenerator.cxx` file.

Key Functionality:

- Constructs a 3D structured grid with dimensions 21×21×2021 \times 21 \times 20 (x, y, and time steps).
- Computes a vector field with components (u,v,w)(u, v, w) using sinusoidal and trigonometric functions.

- Adds random noise to simulate variability in the field.
- Writes the grid and vector field data to a `.vtk` file (`vector_field.vtk`) using the `vtkStructuredGridWriter` class.

---

3. CMake Configuration:

The `CMakeLists.txt` file specifies the build system configuration. Below is the exact `CMakeLists.txt` file used:

```
cmake_minimum_required(VERSION 3.12 FATAL_ERROR)

project(VectorFieldGenerator)

find_package(VTK COMPONENTS
  CommonCore
  CommonDataModel
  IOLegacy
)

if (NOT VTK_FOUND)
  message(FATAL_ERROR "Unable to find the VTK build folder.")
endif()

add_executable(VectorFieldGenerator VectorFieldGenerator.cxx)
target_link_libraries(VectorFieldGenerator PRIVATE ${VTK_LIBRARIES})

# vtk_module_autoinit is needed
vtk_module_autoinit(
  TARGETS VectorFieldGenerator
  MODULES ${VTK_LIBRARIES}
)
```

Explanation:

1. **`find_package(VTK COMPONENTS ...)`**: Locates the required VTK modules:
    - `CommonCore`: Core data structures and algorithms.
    - `CommonDataModel`: For structured grid representation.
    - `IOLegacy`: For writing legacy `.vtk` files.
2. **Source and Executable**:
    - The executable `VectorFieldGenerator` is built from the `VectorFieldGenerator.cxx` file.

3. **Library Linking**:
    - Links the required VTK libraries.
4. **Initialization**:
    - `vtk_module_autoinit` ensures proper module initialization at runtime.

---

4. Building the Project with CMake GUI:

To build the project, the **CMake GUI** was used:

1. Open the **CMake GUI**.
2. Set the source directory to the folder containing both `CMakeLists.txt` and `VectorFieldGenerator.cxx`.
3. Set the build directory (e.g., `C:/build/VectorFieldGenerator`).
4. Click **Configure** and select a compiler (e.g., Visual Studio, GCC).
5. Click **Generate** to produce build system files.
6. Open the generated files in the chosen IDE (e.g., Visual Studio) and build the `VectorFieldGenerator` executable.

---

5. Running the Program:

Then we should run the generated executable (`VectorFieldGenerator.exe`) , and it produces the output file `vector_field.vtk` in the working directory. This `.vtk` file contains:

- A structured grid with dimensions 21×21×2021 \times 21 \times 20.
- A vector field named `VectorField` with three components (u,v,wu, v, w) at each grid point.

Output:

The generated `.vtk` file (`vector_field.vtk`) can be visualized using tools like **ParaView** to inspect the structured grid and vector field.

Notes:

1. **Folder Organization**:
    - Ensure that the `CMakeLists.txt` and the C++ code file are in the same folder to avoid configuration issues with CMake.

2. **CMake GUI Advantage**:
    - The GUI simplifies configuration and build processes, especially for users new to CMake.
3. **Visualization**:
    - After generating the `.vtk` file, tools like ParaView can be used for 3D visualization of the grid and vector field.

Conclusion:

By placing the `CMakeLists.txt` and `VectorFieldGenerator.cxx` in the same folder and using CMake GUI, a structured workflow is achieved for generating `.vtk` files efficiently. This process combines the power of VTK and CMake to produce high-quality visualization data.