Providing a better error message when someone tries to use std::vector as a buffer

devblogs.microsoft.com/oldnewthing/20200313-00

March 13, 2020

Raymond Chen

Last time, we looked at how we could generate a useful error message if somebody tried to pass `std::vector<bool>` to our `buffer_view` class. The `std::vector<bool>` is unlike all the other `std::vector<T>` types because it does not require its storage to be in the form of a traditional C array, which means that it is not possible to obtain direct access to the underlying storage.

Last time, we struggled with this buffer type:

```
struct buffer_view
{
  template<typename C>
  buffer_view(std::vector<C> const& v) :
    data(v.data()), size(v.size() * sizeof(C)) { }

  // Imagine other constructors for std::array, etc.

  void const* data;
  std::size_t size;
};
```

If somebody tries to create a `buffer_view` from a `std::vector<bool>`, they get an incomprehensible error message because there is no `v.data()` method. (For some reason, gcc and clang do have a `data()` method, but it doesn't return anything interesting, so the error message is *even more* incomprehensible.)

We addressed the problem last time by introducing an overload of the constructor that is active only for `std::vector<bool>`, and putting a `static_assert` in the body with a deceptively type-dependent expression so that the assertion wasn't raised until the overload was invoked.

I noted that Kenny Kerr came up with a simpler solution: Move the call to `data()` to a helper function, and templatize that helper.

```
struct buffer_view
{
  template<typename C>
  buffer_view(std::vector<C> const& v) :
    data(get_data(v)), size(v.size() * sizeof(C)) { }

  // Imagine other constructors for std::array, etc.

  void const* data;
  std::size_t size;

private:
  void const* get_data(std::vector<C> const& v)
  {
    static_assert(!is_same_v<C, bool>,
      "Can't use std::vector<bool>. Try std::array instead.");
    return v.data();
  }
};
```

The `static_assert` comes ahead of the call to `v.data()`, so it becomes the first error message.

You could go even further and make it the *only* error message by adding some `if constexpr`:

```
  void const* get_data(std::vector<C> const& v)
  {
    static_assert(!is_same_v<C, bool>,
      "Can't use std::vector<bool>. Try std::array instead.");
    if constexpr (!is_same_v<C, bool>) {
      return v.data();
    } else {
      return nullptr;
    }
  }
```

Raymond Chen

Follow