COMP220: Graphics & Simulation

09: Debugging & Optimisation for Graphics

Learning outcomes

By the end of this week, you should be able to:

- **Understand** how to approach problems with OpenGL applications.
- **Utilize** a selection of tools and techniques to fix and enhance your GPU code.

Agenda

Agenda

- Lecture (async):
  - **Introduce** a variety of methods for inspecting OpenGL code.
  - **Suggest** tools that might be useful for debugging and profiling GPU applications.

FALMOUTH
UNIVERSITY

Writing Good Code

Programming Aims

Programming Aims

1. Make it work

Programming Aims

1. Make it work
2. Make it not break - **debugging**

Programming Aims

1. Make it work
2. Make it not break - **debugging**
3. Make it work quickly - **profiling & optimisation**

Programming Aims

1. Make it work
2. Make it not break - **debugging**
3. Make it work quickly - **profiling & optimisation**
4. Make it readable and maintainable - **code design**

Other People's Code

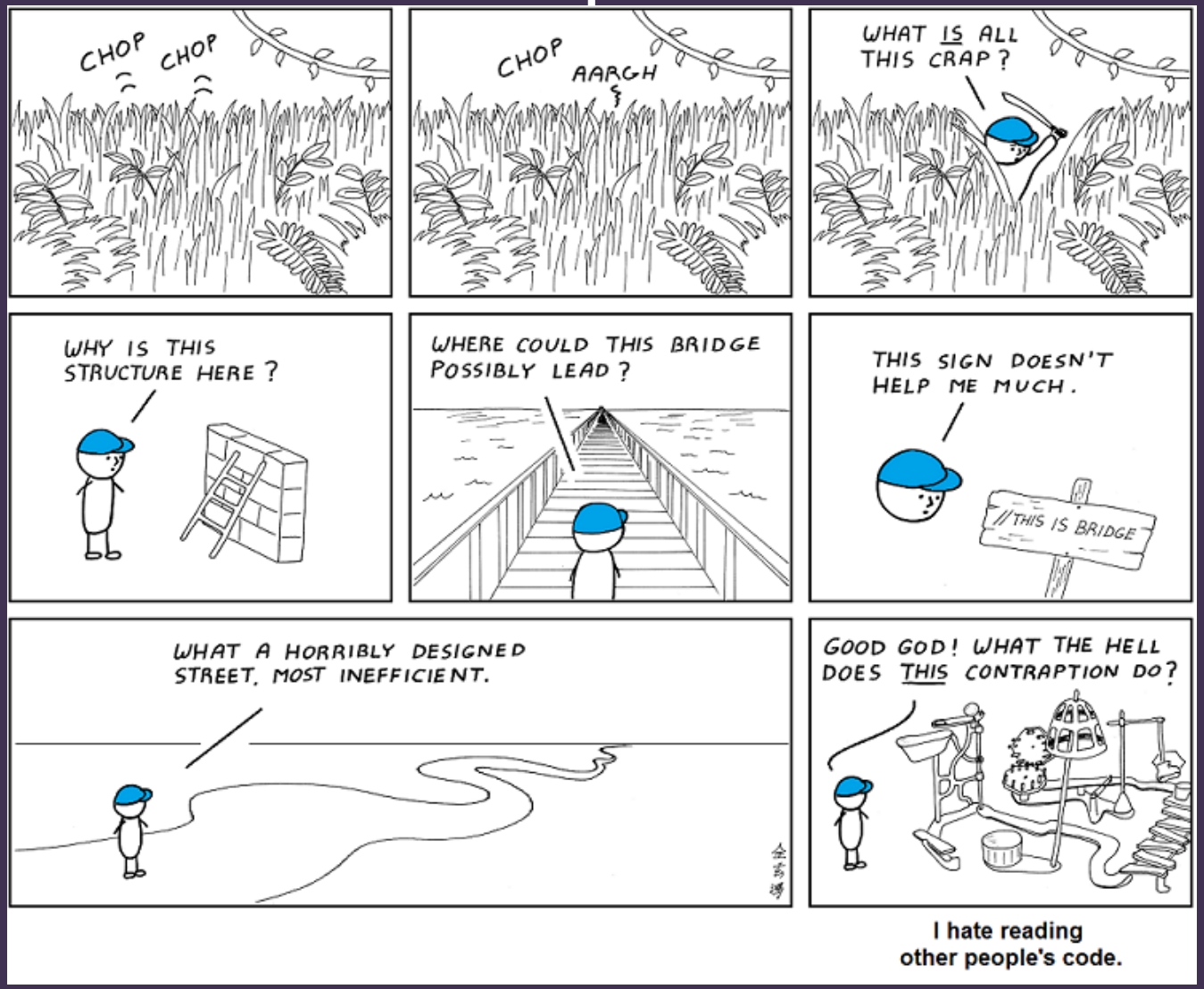

Image source: `https://abstrusegoose.com/432`

FALMOUTH
UNIVERSITY

Debugging

OpenGL Error States

OpenGL Error States

- `glGetError()` queries behind-the-scenes error flags to check state.

OpenGL Error States

- `glGetError()` queries behind-the-scenes error flags to check state.
- Possible `glEnum` error codes for each function are listed in the documentation, e.g. for glBindTexture:

Errors

`GL_INVALID_ENUM` is generated if *`target`* is not one of the allowable values.

`GL_INVALID_VALUE` is generated if *`target`* is not a name returned from a previous call to glGenTextures.

`GL_INVALID_OPERATION` is generated if *`texture`* was previously created with a target that doesn't match that of *`target`*.

Debug Output

Debug Output

- Extension made core feature from v4.3

Debug Output

- Extension made core feature from v4.3
- Includes information about the cause and severity.

Debug Output

- Extension made core feature from v4.3
- Includes information about the cause and severity.

```
SDL_GL_SetAttribute(SDL_GL_CONTEXT_FLAGS,
        SDL_GL_CONTEXT_DEBUG_FLAG); // Set up debug context
glEnable(GL_DEBUG_OUTPUT);
glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS);
glDebugMessageCallback(debugMessage, NULL);
glDebugMessageControl(GL_DONT_CARE, GL_DONT_CARE,
        GL_DONT_CARE, 0, NULL, GL_TRUE);    // Filter errors

// Callback
void APIENTRY debugMessage(GLenum source, GLenum type,
    GLuint id, GLenum severity, GLsizei length,
    const GLchar *message, const void *userParam) {
    // Do something with the error info
    // (print, write to file etc.)
}
```

Debugging Shaders

Debugging Shaders

- Basic information from **compilation error reports**.

Debugging Shaders

- Basic information from **compilation error reports**.
- OpenGL GLSL **reference compiler** tests shader code against OpenGL specification.

Debugging Shaders

- Basic information from **compilation error reports**.
- OpenGL GLSL **reference compiler** tests shader code against OpenGL specification.
- Can use **colour channels** to display values.

Debugging Shaders

- Basic information from **compilation error reports**.
- OpenGL GLSL **reference compiler** tests shader code against OpenGL specification.
- Can use **colour channels** to display values.
- Display **framebuffer contents** in the corner of the screen (similar to post-processing setup).

Debugging Shaders

- Basic information from **compilation error reports**.
- OpenGL GLSL **reference compiler** tests shader code against OpenGL specification.
- Can use **colour channels** to display values.
- Display **framebuffer contents** in the corner of the screen (similar to post-processing setup).
- More detailed inspection requires using a **3rd party tool** (depending on GPU vendor etc.).

FALMOUTH
UNIVERSITY

Profiling & Optimisation

Writing Efficient GPU Code

Writing Efficient GPU Code

- Same key principles apply as for CPU code!

Writing Efficient GPU Code

- Same key principles apply as for CPU code!
- Certain operations are optimised in GLSL, eg. swizzling, built-in functions for linear interpolation, dot product etc.

Writing Efficient GPU Code

- Same key principles apply as for CPU code!
- Certain operations are optimised in GLSL, eg. swizzling, built-in functions for linear interpolation, dot product etc.
- Write as much as possible in the vertex shader - remember there are fewer vertices than pixels!

Writing Efficient GPU Code

- Same key principles apply as for CPU code!
- Certain operations are optimised in GLSL, eg. swizzling, built-in functions for linear interpolation, dot product etc.
- Write as much as possible in the vertex shader - remember there are fewer vertices than pixels!
- As always: base your changes on profiling results...

Next steps

- **Review** the additional asynchronous material for more information and resources on code design, debugging and profiling.