Metamorphic Testing for (Graphics) Compilers

Alastair Donaldson and Andrei Lascu
Imperial College London

Some screenshots from Alastair’s talk at MET 2016
Still from an attractive 3D animation, achieved via an OpenGL fragment shader.
The effect of a shader compiler bug
Another example of compiler bug impact on this shader program – less recognisable!
How do we find these bugs?
Metamorphic relations

Original shader

```c
void shader_func(...) {
    ...
    ...  // some statements
    ...
    x = y + z; // an assignment
    ...
}
```
How do we find these bugs?
Metamorphic relations

Add new parameter. We will set it to false at runtime.
The shader compiler doesn’t know that!

```cpp
void shader_func(..., bool FF) {
    ...
    ...
    ...
    x = y + z;
    ...
}
```
How do we find these bugs?
Metamorphic relations

Now we can inject new code – will not be executed at runtime, but the compiler doesn’t know that

```c
void shader_func(..., bool FF) {
    ...
    if(FF) {
        ...
    }
    ...
    x = y + z;
    ...
}
```
How do we find these bugs? Metamorphic relations

Now we can inject new code – will not be executed at runtime, but the compiler doesn’t know this.

```c
void shader_func(..., bool FF) {
    ...
    if(FF) {
        ...
    }
    ...
    x = y + z;
    ...
}
```

```c
void another_shader(...) {
    ...
    for(...) {
        ...
        if(...) {
            ...
        }
        ...
    }
    ...
}
```

Variable substitution makes this work
How do we find these bugs?

Metamorphic relations

We can also apply **identity functions**, e.g.

```c
void shader_func(..., bool FF) {
    ...
    if(FF) {
        ...
    }
    ...
    x = y + (FF ? ... : 1)*z;
    ...
}
```

Arbitrarily complex expression

Guaranteed to evaluate to 1

```c
void another_shader(...) {
    ...
    for(...) {
        if(...) {
            ...
        }
    }
    ...
}
```
How do we find these bugs?
Metamorphic relations

Dead code injection and identity transformations are metamorphic relations – they should not change program semantics
Example

An original shader
(only partly shown – the entire shader is much larger, and we inject into many places all over it)

```cpp
vec3 tunnelEnd(float z, float radius, vec3 camera) {
    float angle = atan(camera.x, camera.y);
    float dist = length(camera.xy);
    return vec3(angle * radius / (3.14159265359 / 2.0),
                dist * (z - camera.z), z - camera.z);
}
```
Example

Image looks pretty
Example

An transformed version
of the shader

```c
vec3 tunnelEnd(float z, float radius, vec3 camera) {
  float angle = atan(camera.x, (vec3(1.0, 1.0, 1.0) * (vec3(1.0, 1.0, 1.0) * (camera + vec3(0.0, 0.0, 0.0)))).y);
  float dist = length(camera.xy);
  return vec3((angle * (radius + (0.0 * 1.0)) + 0.0) / (3.14159265359 / 2.0), dist * (z - camera.z), z - (vec3(1.0, 1.0, 1.0) * camera).z);
}
```
Example

The image is wrong!
**Example**

We iteratively and automatically undo MRs to reach minimal diff:

```cpp
vec3 tunnelEnd(float z, float radius, vec3 camera) {
    float angle = atan(camera.x, camera.y);
    float dist = length(camera.xy);
    return vec3(angle * radius / (3.14159265359 / 2.0),
               dist * (z - camera.z), z - camera.z);
}
```

The small, bug-triggering transformation makes for a good bug report to compiler developers.
Intel have now fixed this shader compiler bug!
Summary

• Metamorphic transformations are effective in triggering compiler bugs
• Undoing metamorphic transformations allows us to find minimal, bug-triggering transformations
• The resulting bugs are taken seriously
• The metamorphic approach is also good at coping with floating-point variability (see MET 2016 paper)
Related work on metamorphic compiler testing

• Qiuming Tao, Wei Wu, Chen Zhao, Wuwei Shen: An Automatic Testing Approach for Compiler Based on Metamorphic Testing Technique. ASPEC 2010.

• Vu Le, Mehrdad Afshari, Zhendong Su: Compiler validation via equivalence modulo inputs. PLDI 2014. [The inspiration for our dead code injection method]