Metamorphic Testing of Android Graphics Drivers

Alastair F. Donaldson
Senior Software Engineer
Android Graphics Tools Team
Google UK
First paper on the approach: Alastair F. Donaldson and Andrei Lascu: “Metamorphic testing for (graphics) compilers”, MET@ICSE 2016.
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (BSOD: complete)
GraphicsFuzz

- GraphicsFuzz finds bugs in graphics drivers using **metamorphic testing**
- In particular, it finds bugs in **shader compilers**

What’s a shader?

- **OpenGL ES** and **Vulkan** are graphics programming models
- They orchestrate execution of operations and programs across GPU hardware
- A **shader** is a program that runs across GPU hardware
- A shader is like a kernel in CUDA / OpenGL

**GLSL**: the OpenGL shading language  
**SPIR-V**: the Vulkan shading language
The GraphicsFuzz testing approach
The GraphicsFuzz testing approach

- original shader
- variant shader
- semantics preserving transformations
- graphics driver (shader compiler)
- GPU
- image
The GraphicsFuzz testing approach

original shader

semantics preserving transformations

variant shader

graphics driver (shader compiler)

GPU

image

image
The GraphicsFuzz testing approach

- original shader
- semantics preserving transformations
- variant shader
- graphics driver (shader compiler)
- GPU
- image
- comparison
The GraphicsFuzz testing approach

original shader

image

diff/crash

semantics preserving transformations

variant shader

GPU

graphics driver (shader compiler)
The GraphicsFuzz testing approach

- **original shader**
- **semantics preserving transformations**
- **variant shader**
- **graphics driver (shader compiler)**

**GPU**

- **image**
- **diff/crash**

Metamorphic testing
void main(void)
{
    vec2 uv = (gl_FragCoord.xy / resolution.xy) * 2.0 - 1.0;
    uv.x *= resolution.x / resolution.y;
    if(_GLF_DEAD(_GLF_FALSE(false, (injectionSwitch.x > injectionSwitch.y))))
        return;
    vec3 finalColor = RenderScene(uv);
    if(_GLF_DEAD(_GLF_IDENTITY(false, (false) || false)))
    {
        vec3 donor_replacementp = _GLF_FUZZED(faceforward(((++ finalColor) - faceforward(vec3(4.8, 7582.5251, -3.4), vec3(-369.491, -9.0, 6172.7474), finalColor)), vec3(6108.1119, -181.078, 495.885), (finalColor).yzx));
        float donor_replacementtw = _GLF_FUZZED(sign(dot((EPS / vec3(53.44, 6.0, -752.725)), fract(finalColor))));
        float donor_replacementstrength = _GLF_FUZZED(38.04);
        float donor_replacementprev = _GLF_FUZZED(clamp((+ distance(time, -47.91)), (- finalColor.g), (mouse / EPS)[1]));
        if(_GLF_DEAD(_GLF_FALSE(false, (injectionSwitch.x > injectionSwitch.y))))
            return;
        float donor_replacementaccum = _GLF_FUZZED(distance(vec2(-349.170, -4419.3875), (- vec4(-359.006, 69.29, 96.95, -243.116)).wz));
        if(_GLF_DEAD(_GLF_IDENTITY(false, (false ? _GLF_FUZZED((-28449 < shadowType)) : false))))
            return;
        for(
            int i = 0;
            i < 16;
            ++i)
            ...
    }
}
Test-case reduction

while (diff/crash) {
    random_remove_transformation()
}
Test-case reduction

original shader

same semantics
small syntax-diff

bug-triggering shader

graphics driver (shader compiler)

GPU

image

diff/crash

isolates the bug
Reducer: small difference

```bash
$ diff reference.frag variant_reduced.frag
194a201,204
>     if (gl_FragCoord.x < -100.0) //always false
>       {
>         return;
>       }
```
GraphicsFuzz tests itself!

Bug in GraphicsFuzz

<table>
<thead>
<tr>
<th></th>
<th>reference</th>
<th>variant 1</th>
<th>variant 2</th>
<th>variant 3</th>
<th>variant 4</th>
<th>variant 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM GPU</td>
<td>![ARM GPU](ARM GPU)</td>
<td>![variant 1](variant 1)</td>
<td>![variant 2](variant 2)</td>
<td>![variant 3](variant 3)</td>
<td>![variant 4](variant 4)</td>
<td>![variant 5](variant 5)</td>
</tr>
<tr>
<td>Intel GPU</td>
<td>![Intel GPU](Intel GPU)</td>
<td>![variant 1](variant 1)</td>
<td>![variant 2](variant 2)</td>
<td>![variant 3](variant 3)</td>
<td>![variant 4](variant 4)</td>
<td>![variant 5](variant 5)</td>
</tr>
<tr>
<td>NVIDIA GPU</td>
<td>![NVIDIA GPU](NVIDIA GPU)</td>
<td>![variant 1](variant 1)</td>
<td>![variant 2](variant 2)</td>
<td>![variant 3](variant 3)</td>
<td>![variant 4](variant 4)</td>
<td>![variant 5](variant 5)</td>
</tr>
<tr>
<td>Qualcomm GPU</td>
<td>![Qualcomm GPU](Qualcomm GPU)</td>
<td>![variant 1](variant 1)</td>
<td>![variant 2](variant 2)</td>
<td>![variant 3](variant 3)</td>
<td>![variant 4](variant 4)</td>
<td>![variant 5](variant 5)</td>
</tr>
</tbody>
</table>
Testing Android graphics drivers

- Gaming on mobile becoming increasingly in demand
- Mobile graphics drivers need to be reliable
- OpenGL ES: legacy API for graphics
- Vulkan: modern graphics

GraphicsFuzz at Google: make Vulkan as reliable as possible

But GraphicsFuzz works on the OpenGL shading language...
Testing Vulkan via translation

- OpenGL ES reference shader
- glslang compiler
- Vulkan reference shader
- Vulkan driver
- Reference image
- Vulkan driver
- Vulkan variant shader
- glslang compiler
- OpenGL ES variant shader
- Compare
Android Compatibility Test Suite (CTS)

- Each Android dessert release (.... Marshmallow, Nougat, Oreo, Pie ...) has associated *compatibility test suite* (CTS)
- Original equipment manufacturers (OEMs) must pass CTS for their devices to be official Google Android
- New, more rigorous tests can be added to CTS for future Android versions
- Tests for bugs found by GraphicsFuzz are being added to CTS for Android Q and R
- Ensures that associated bugs will be fixed and will not return
Metamorphic testing for finding vulnerabilities

- Chrome web browser: billions of users
- Lots of people trying to attack Chrome
- **ClusterFuzz**: continuous fuzzing of Chromium via:
  - Santizers (address sanitizer, memory sanitizer, thread sanitizer …)
  - Coverage-guided mutation-based fuzzing (libFuzzer, similar to AFL)
- WebGL - OpenGL for the web - is implemented in Chrome
- WebGL vulnerabilities are thus a concern
- Chrome security do not care about wrong images (not exploitable!)

How does metamorphic testing fit in?
Mutation-based fuzzing (AFL)

- Most mutated inputs **invalid**
- Great for finding vulnerabilities in parsers
- Parsers are a first point of attack
- Not good for finding bugs deeper in system under test
Metamorphic testing

- Original valid => variants valid
- Finds deep vulnerabilities
- Does not find bugs triggered by malformed inputs

Metamorphic testing conveniently produces well-formed inputs

(Other well-formed input generation methods have same benefits.)
GraphicsFuzz + ClusterFuzz finds WebGL vulnerabilities

The metamorphic approach complements mutation-based fuzzing

We are trying out the combination of metamorphic + coverage-guided
GraphicsFuzz auto

- CTS patch
- Convert to bug report
- Generate one shader
- Run it
- De-duplicate it
- Reduce it
Fuzzing shaders from games
Summary

- GraphicsFuzz: metamorphic testing for shader compilers
- Families of equivalent shaders via semantics-preserving transformations
- Finds bugs in Android Vulkan drivers
- Tests exposing bugs are added to the Compatibility Test Suite
- Surprisingly, GraphicsFuzz finds WebGL vulnerabilities
  - Metamorphic approach keeps shaders valid
  - Valid shaders can go deep
- The future:
  - Direct fuzzing for Vulkan
  - Better automation
  - Fuzzing at the level of graphics APIs

Thank you!