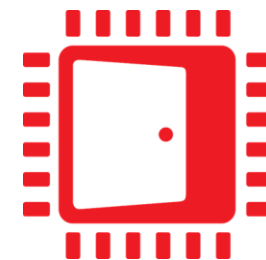


**POST-MORTEM GPU CRASH ANALYSIS  
WITH  
AMD RADEON™ GPU DETECTIVE (RGD)**

ADAM SAWICKI (AMD)

AMIT MULAY (AMD)

MARCO BOUTERSE (NIXXES SOFTWARE)




**AMD**   
GPUOpen

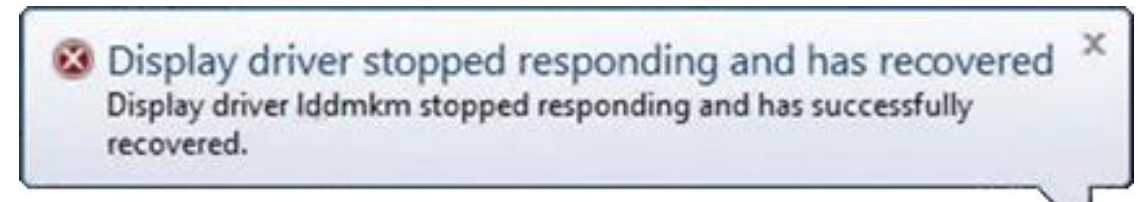
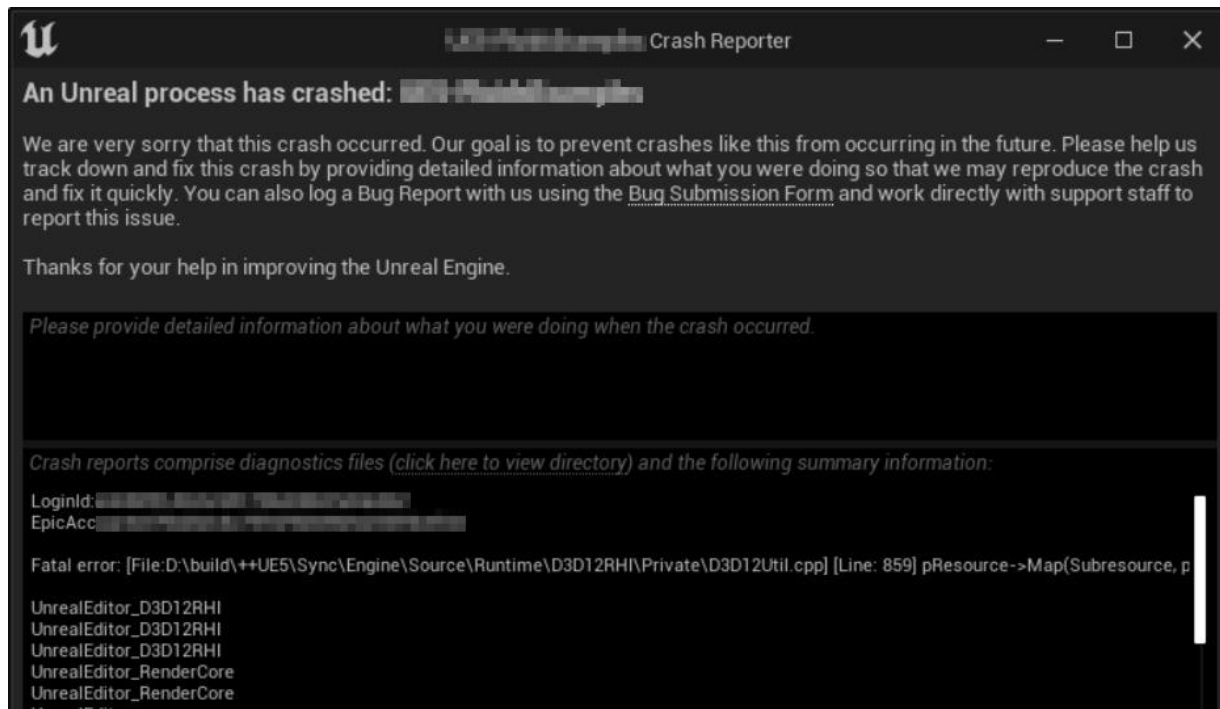
# AGENDA

1. GPU crashes
  - Possible causes
  - Why it happens so often?
  - Effects
  - Why is it so difficult to debug?
  - What can we do?
2. AMD Radeon GPU Detective
  - Introduction
  - Workflow
  - The crash analysis report
3. RGD @ Nixxes case study
4. Conclusions

# 1. GPU CRASHES

# GPU CRASH

- We talk about graphics APIs – mostly **DirectX® 12** | **Vulkan®**
- Timeout Detection and Recovery (TDR)  2 s



# POSSIBLE CAUSES

- Application bug – incorrect usage of the API ← **most likely!**
- Driver bug
- External factors: driver update, hardware failure, ...

# POSSIBLE CAUSES

- Timeout
  - Infinite loop in a shader
  - Oversized Dispatch
  - Reaching out to system memory
  - Too many commands
- Memory page fault
  - Using a resource after **Release** or **Evict**
  - Indexing out of bounds
  - Incorrect address calculation
- Invalid/missing resource binding – null, wrong type, ...
- Corrupted data (e.g., acceleration structure)
- Other...

# WHY IT HAPPENS SO OFTEN?

Old APIs (OpenGL, DX9, DX11):

- Driver is validating everything, each function returns error code
- GPU crash is likely a driver bug

# WHY IT HAPPENS SO OFTEN?

New APIs (DX12, Vulkan):

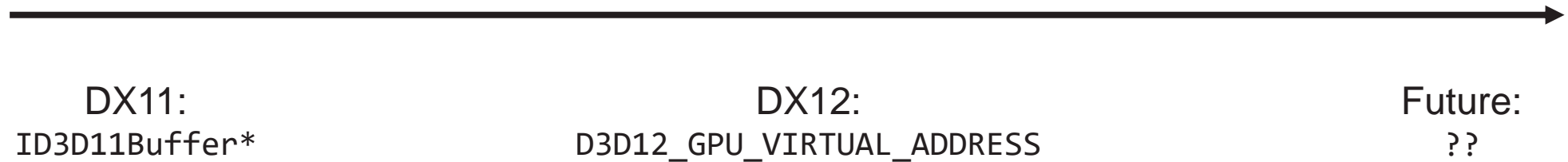
- Driver is not validating, many functions return `void`
  - Allocating functions like `CreateCommittedResource` return `HRESULT`
  - GPU commands like `DrawIndexedInstanced` return `void`
  - `Debug|validation` layers provide validation during development
- Driver is simpler and faster
- GPU crash is likely an application bug
  - Driver bugs happen but shouldn't be your first thought

*“Undefined behavior”*



# WHY IT HAPPENS SO OFTEN?

Happens more often as we use raw memory addresses, dynamic indexing, bindless, indirect, ray tracing...



# “UNDEFINED BEHAVIOR”



Works fine

Visual corruption

Crash

What works on one GPU model may not work the same way on a different one.

# EFFECTS

- GPU and driver is restarted
- Application observes an error code returned from API function
  - E.g., `IDXGISwapChain4::Present` returns `DXGI_ERROR_DEVICE_HUNG`
  - E.g., `vkQueueSubmit` returns `VK_ERROR_DEVICE_LOST`
- Full machine hang or BSOD less frequent



Your device ran into a problem and needs to restart.  
We're just collecting some error info, and then we'll  
restart for you.

100% complete



For more information about this issue and possible fixes, visit  
<https://www.windows.com/help>

If you still need support, please contact the  
Windows Support team at <https://www.windows.com/help>

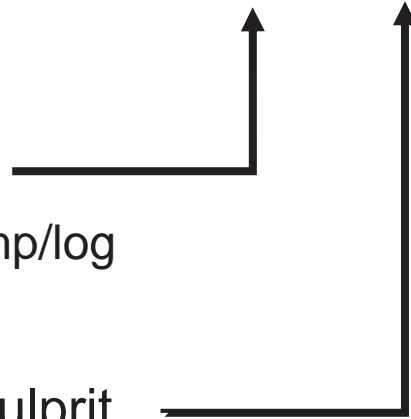
1

# EFFECTS

Note that:

`IDXGISwapChain4::Present` returns `DXGI_ERROR_DEVICE_HUNG`



- Doesn't imply our app crashing (in theory)
  - We can continue or at least save some dump/log
- Doesn't tell which pass or draw call is the culprit
  - Reported for the entire render frame



# WHY IS IT SO DIFFICULT TO DEBUG?

- “GPU Crash” can mean different things – timeout, page fault, ...
- GPUs are complex
  - Asynchronous – execute work submitted by the CPU
  - Pipelined – multiple commands in flight at various stages of the pipeline
  - Parallel – many threads, vertices, pixels processed at once
- Even if one hardware block fails, others may continue – no global STOP with break into a debugger

# WHAT CAN WE DO?

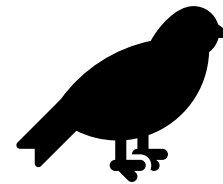
- Capture with PIX or RenderDoc? No... They need a successfully rendered frame
  - Can still help with finding some issues
- **Debug|validation** layers
  - Validate correct API usage
  - Moderate performance overhead 
  - Validates only what is known on the CPU timeline: API calls, command buffer submission, resource allocations...
  - Cannot validate what is only known on the GPU: shader-generated data, descriptors, memory contents...
- **GPU-Based Validation (GBV) | GPU-Assisted Validation**
  - Extra validation on the GPU, shader instrumentation – descriptors etc...
  - Extremely high performance overhead – makes its use impractical 

# WHAT CAN WE DO?

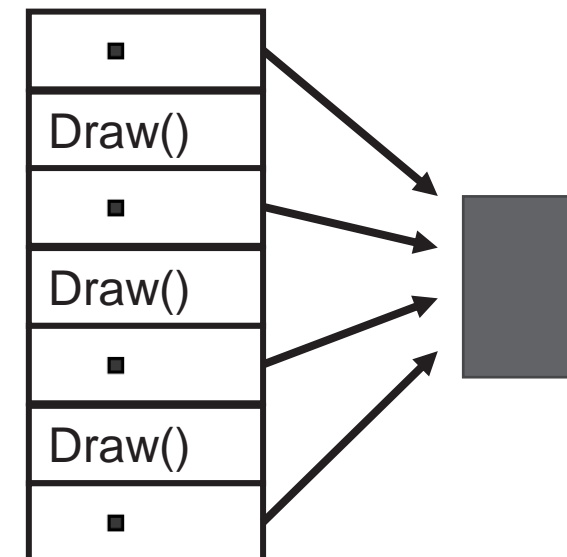
- Last resort: disable individual effects and passes, see if the bug goes away
  - Ultra → High → Medium → Low
  - Disable ray tracing
  - Lower GPU memory usage: Texture quality = Low
  - Modify/simplify shaders
- Custom breadcrumb markers
- Device Removed Extended Data (DRED)
- Vendor-specific tools



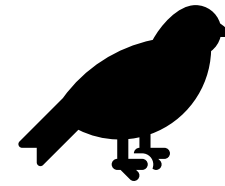
# BREADCRUMB MARKERS



1. Startup: Create a buffer in the readback CPU memory, persistently mapped
  - `VirtualAlloc + OpenExistingHeapFromAddress + CreatePlacedResource`
  - `VK_AMD_device_coherent_memory`
2. Rendering: Write numbers between passes or draw calls
  - `ID3D12GraphicsCommandList2::WriteBufferImmediate`
  - `VK_AMD_buffer_marker`
3. After crash: Inspect the buffer pointer, see which breadcrumbs were successfully written last

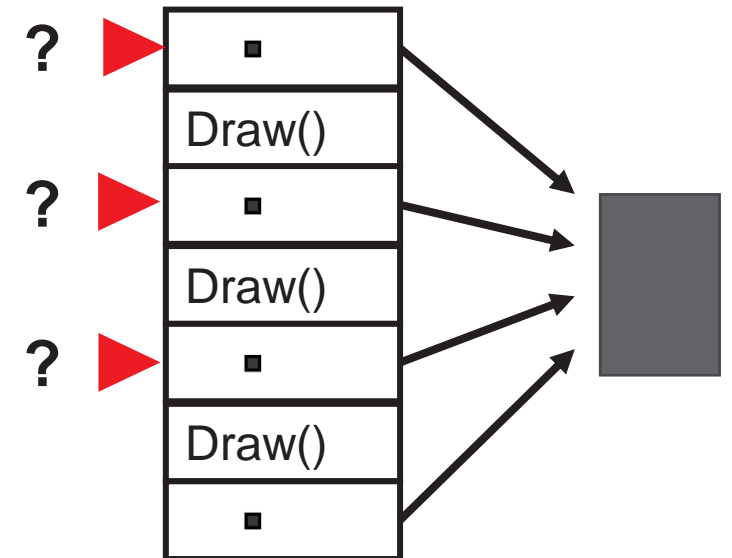


# BREADCRUMB MARKERS



Not always reliable

- Caches don't get flushed → markers too early
- GPU continues past the crashing draw call → markers too late



## 2. AMD RADEON™ GPU DETECTIVE (RGD)

<https://gpuopen.com/rgd/>

# AMD RADEON™ GPU DETECTIVE (RGD)

- Newest member of AMD Radeon™ Developer Tool Suite (<https://gpuopen.com/tools/>)

## Overview:

- Tool for post-mortem analysis of GPU crashes
- Sets driver to Crash Analysis mode before reproducing crash
- Developers capture AMD GPU Crash Dump files (.rgd) upon crash
- Produces concise crash analysis report in Text/JSON formats
- Report helps narrow down the search for the crash root cause

# AMD RADEON™ GPU DETECTIVE (RGD)

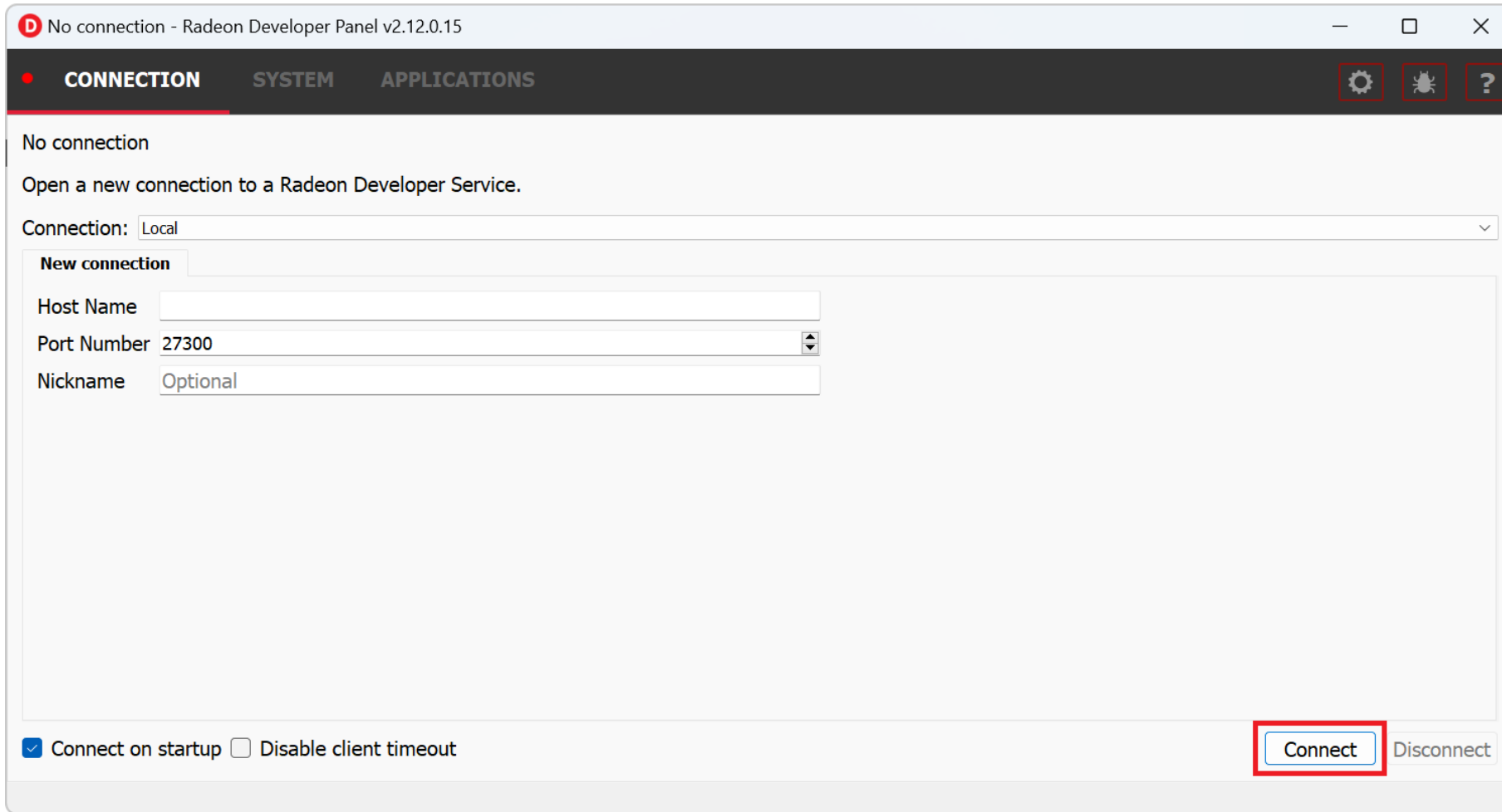
- Newest member of AMD Radeon™ Developer Tool Suite (<https://gpuopen.com/tools/>).

## Requirements:

- OS: Windows 10 or 11
- GPU: AMD Radeon™ RX 7000 Series (AMD RDNA™ 3 architecture) or AMD Radeon™ RX 6000 Series (AMD RDNA™ 2 architecture)
- Driver: AMD Software: Adrenalin Edition 23.12.1 or newer
- Graphics API used by the crashing application: **DirectX 12** or **Vulkan**

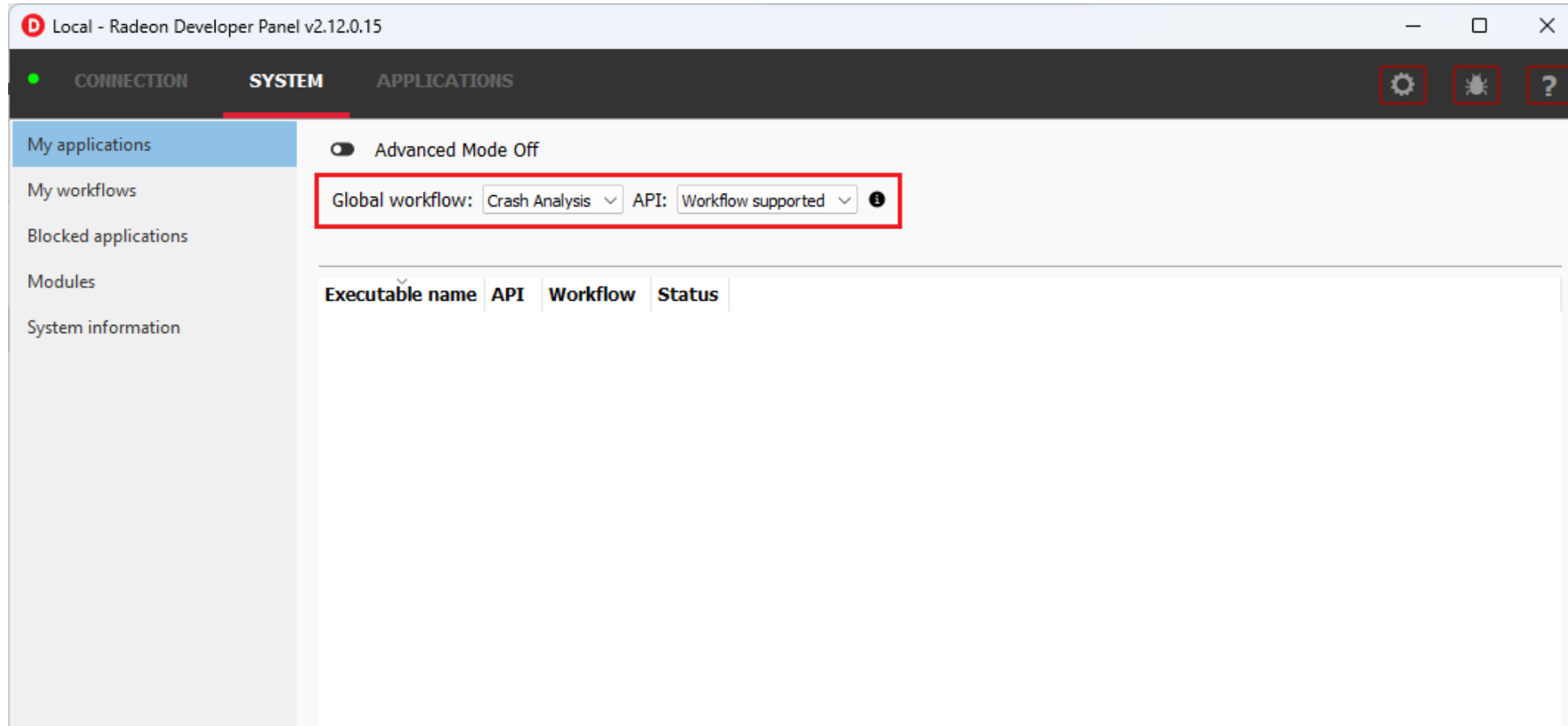
# WORKFLOW

RadeonDeveloperPanel.exe > Connect > Workflow: Crash Analysis



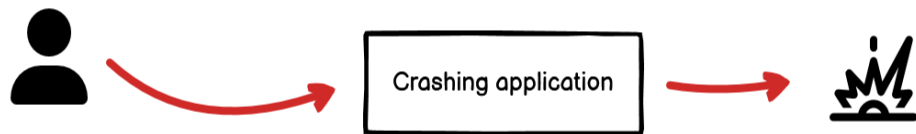
# WORKFLOW

RadeonDeveloperPanel.exe > Connect > Workflow: Crash Analysis



# WORKFLOW

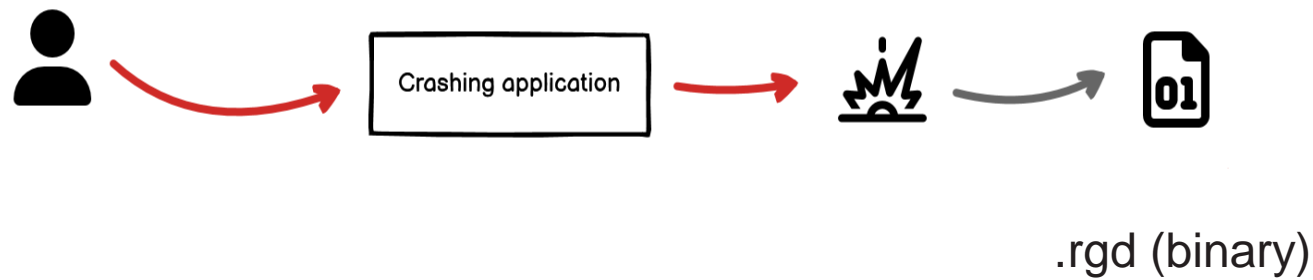
- User launches crashing app, reproduces the crash
- Driver tracks crashing app's behavior from startup





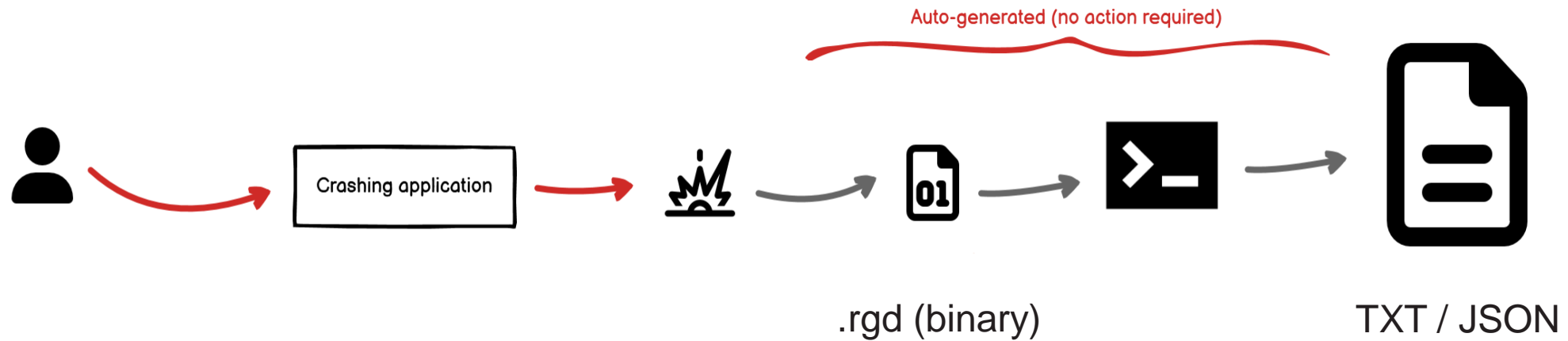
# WORKFLOW

- Upon crash the AMD GPU Crash Dump file (.rgd) is generated



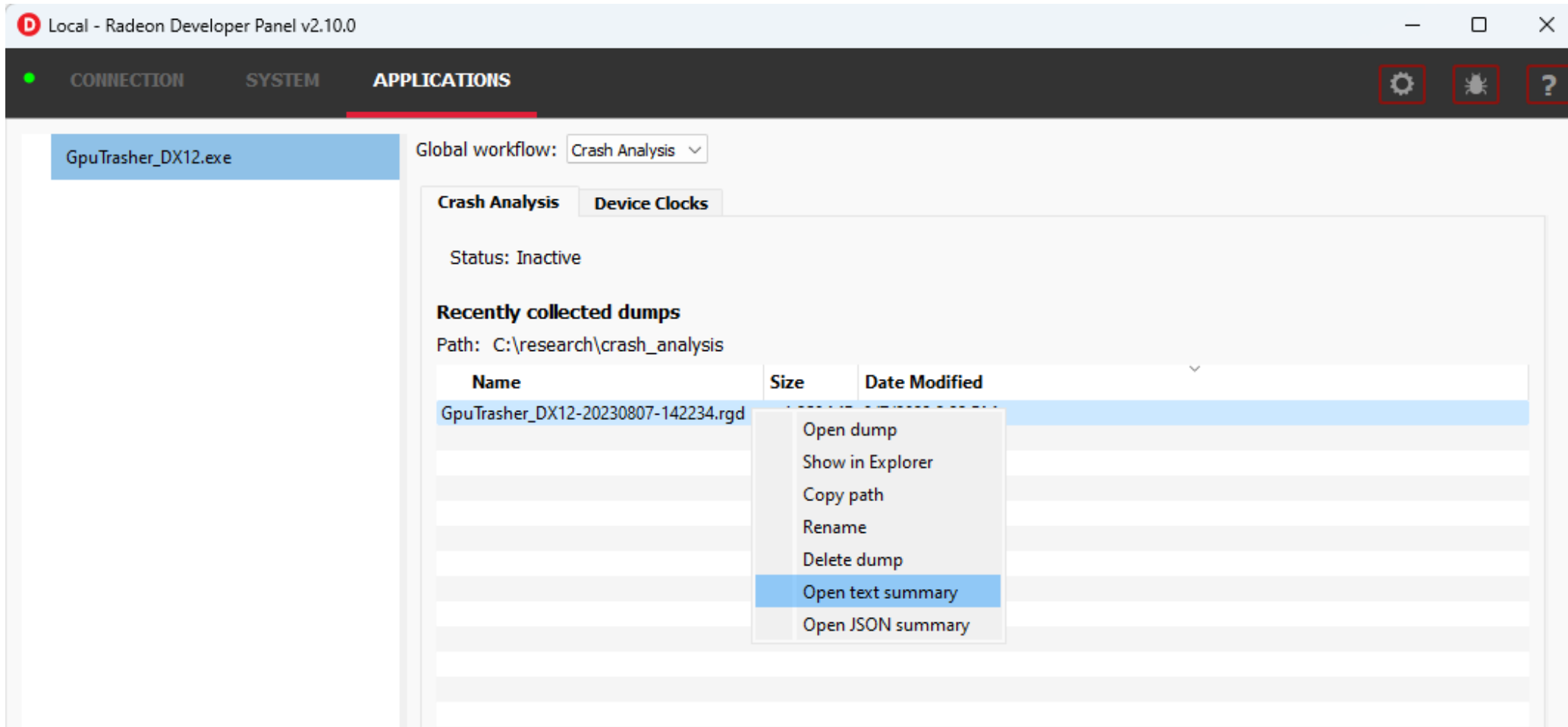
# WORKFLOW

- Crash analysis summary auto-generated by RGD CLI (launched by RDP)



# WORKFLOW

RDP: New crash dump appears > double-click or right-click and select “Open text summary”



# WORKFLOW

Concise TXT (or optionally JSON) file opens with crash report

Sections:

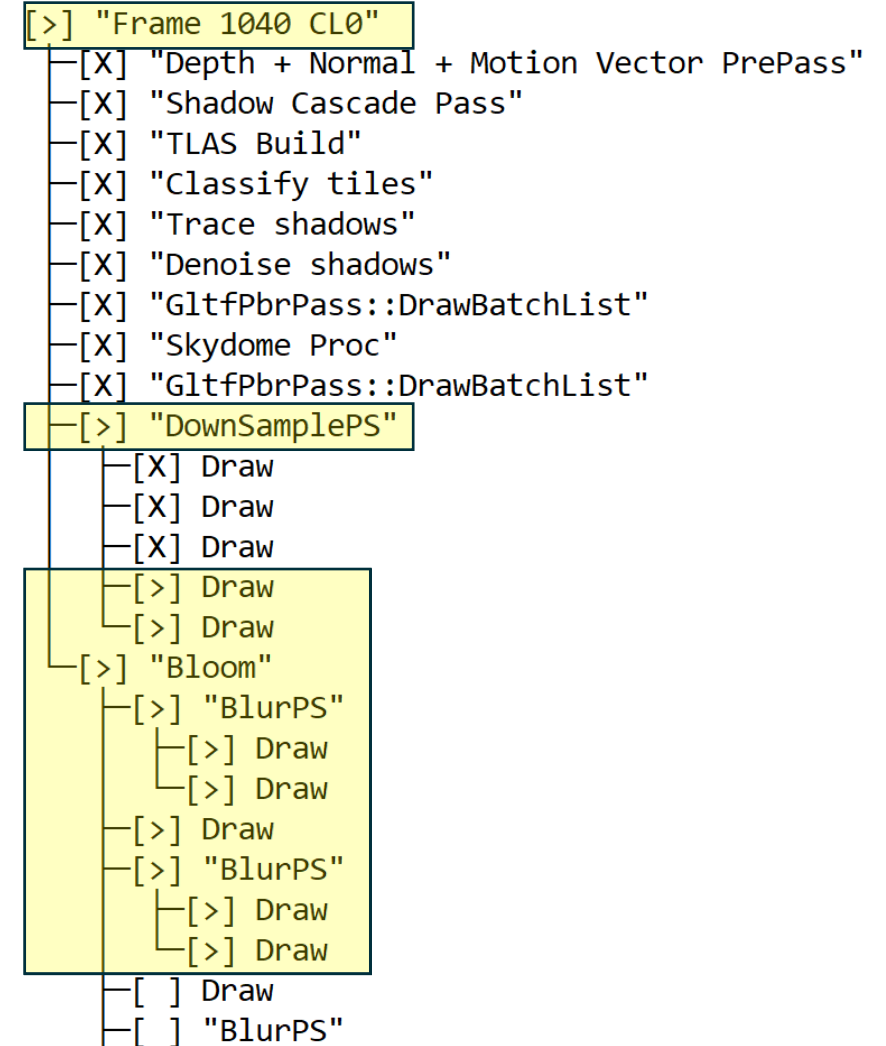
- Execution marker tree
- Markers in progress
- Page fault summary
- System info

# REPORT – EXECUTION MARKER TREE

We worked really hard to make sure that you can narrow down the search for the culprit as quickly as possible

[X] finished  
[>] in progress  
[ ] not started

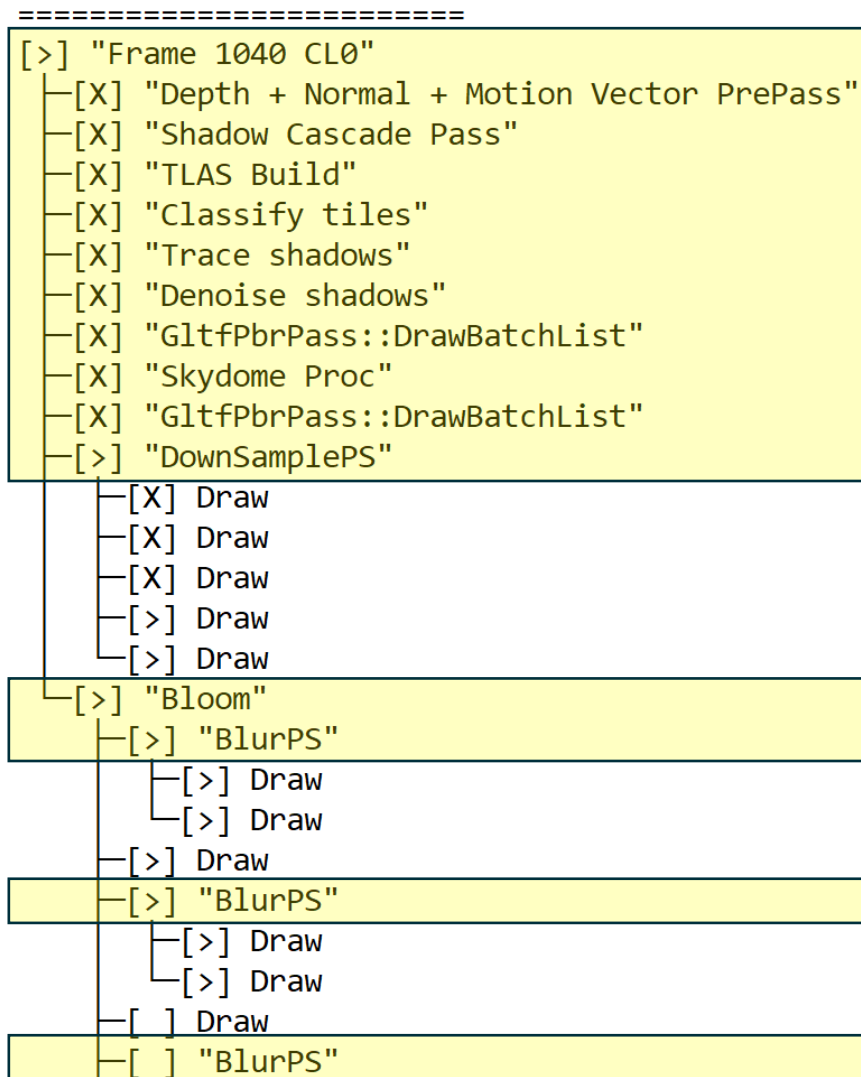
Command Buffer ID: 0x107c



# REPORT – EXECUTION MARKER TREE

User markers:

Command Buffer ID: 0x107c

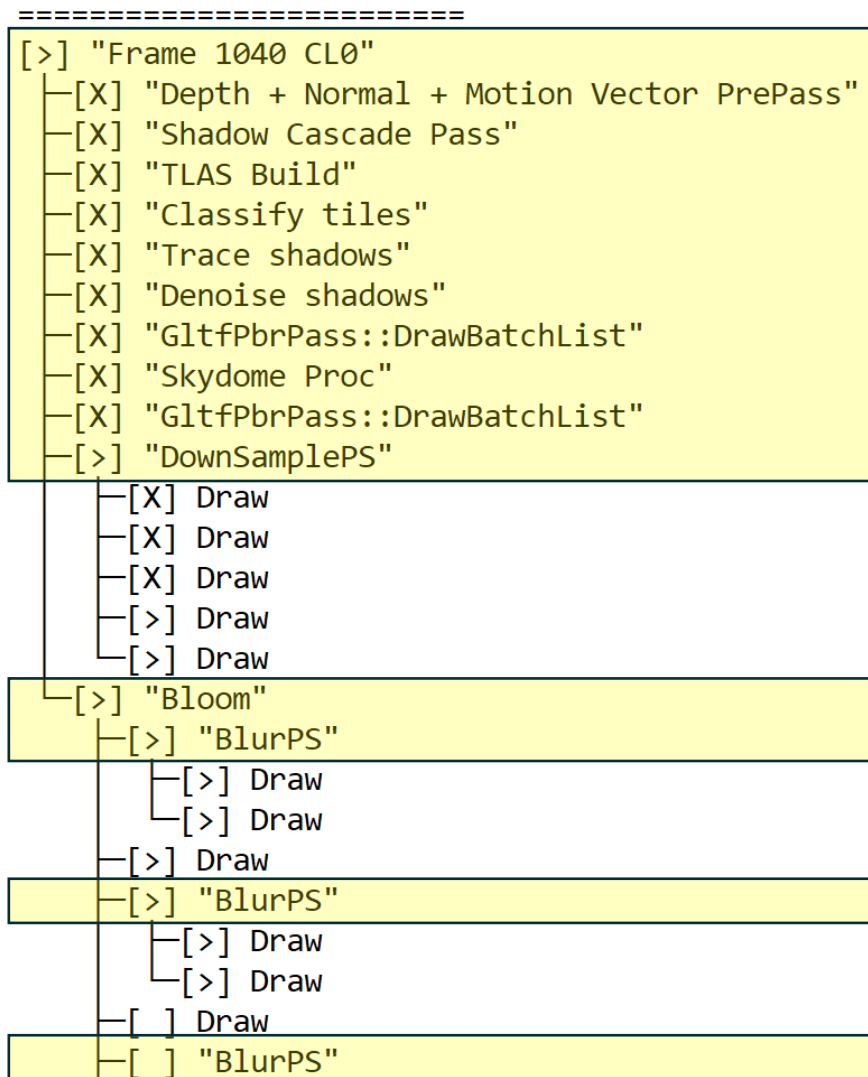


# REPORT – EXECUTION MARKER TREE

User markers:

- DX12:
  - AMD GPU Services (AGS) library
  - PIX markers: Use our replacement header provided with RDTs
- Same as for Radeon GPU Profiler
- Vulkan: `VK_EXT_debug_utils`
- Unreal Engine: `D3D12.EmitRgpFrameMarkers=1`

Command Buffer ID: 0x107c



# REPORT – MARKERS IN PROGRESS

A summary of markers:

- Only those in progress
- In form of paths with '/' separator

Command Buffer ID: 0x107c

=====

```
Frame 1040 CL0/DownSamplePS/Draw [2 repeating occurrences]
Frame 1040 CL0/Bloom/BlurPS/Draw [2 repeating occurrences]
Frame 1040 CL0/Bloom/Draw
Frame 1040 CL0/Bloom/BlurPS/Draw [2 repeating occurrences]
```

=

Command Buffer ID: 0x107c

=====

```
[>] "Frame 1040 CL0"
  |-- [X] "Depth + Normal + Motion Vector PrePass"
  |-- [X] "Shadow Cascade Pass"
  |-- [X] "TLAS Build"
  |-- [X] "Classify tiles"
  |-- [X] "Trace shadows"
  |-- [X] "Denoise shadows"
  |-- [X] "GltfPbrPass::DrawBatchList"
  |-- [X] "Skydome Proc"
  |-- [X] "GltfPbrPass::DrawBatchList"
  |-- [>] "DownSamplePS"
      |-- [X] Draw
      |-- [X] Draw
      |-- [X] Draw
      |-- [>] Draw
      |-- [>] Draw
      |-- [>] "Bloom"
          |-- [>] "BlurPS"
              |-- [>] Draw
              |-- [>] Draw
          |-- [>] Draw
          |-- [>] "BlurPS"
              |-- [>] Draw
              |-- [>] Draw
          |-- [ ] Draw
          |-- [ ] "BlurPS"
```



# REPORT – PAGE FAULT SUMMARY

- Offending virtual address
- Resources that resided in this address during the crashing app's lifetime

Resource id: 0x5a49f060000a7f

Type: Image

Name: Postprocessing render target 4

Virtual address:

0x236c0000 [size: 16810352 (16.03 MB), parent address + offset: 0x236c0000 + 0x0]

Commit type: COMMITTED

Attributes:

Create flags: PREFER\_SWIZZLE\_EQUATIONS | FIXED\_TILE\_SWIZZLE (24576)

Usage flags: SHADER\_READ | SHADER\_WRITE | RESOLVE\_DESTINATION | COLOR\_TARGET (27)

Image type: 2D

Dimensions <x, y, z>: 1920 x 1080 x 1

Swizzle pattern: XYZW

Image format: X16Y16Z16W16\_FLOAT

Mip levels: 1

Slices: 1

Sample count: 1

Fragment count: 1

Tiling type: Optimal

Resource timeline:

00:00:09.4618368 : Create

00:00:09.4622336 : Bind into 0x236c0000

00:00:09.4622336 : Make Resident into 0x236c0000

00:00:09.4634816 : Destroy

=====  
PAGE FAULT SUMMARY  
=====

Offending VA: 0x236c04000

# REPORT – PAGE FAULT SUMMARY

## Resource parameters

Resource id: 0x5a49f0600000a7f

Type: Image

Name: Postprocessing render target 4

Virtual address:

0x236c0000 [size: 16810352 (16.03 MB), parent address + offset: 0x236c0000 + 0x0]

Commit type: COMMITTED

Attributes:

```
Create flags: PREFER_SWIZZLE_EQUATIONS | FIXED_TILE_SWIZZLE (24576)
Usage flags: SHADER_READ | SHADER_WRITE | RESOLVE_DESTINATION | COLOR_TARGET (27)
Image type: 2D
Dimensions <x, y, z>: 1920 x 1080 x 1
Swizzle pattern: XYZW
Image format: X16Y16Z16W16_FLOAT
Mip levels: 1
Slices: 1
Sample count: 1
Fragment count: 1
Tiling type: Optimal
```

Resource timeline:

```
00:00:09.4618368 : Create
00:00:09.4622336 : Bind into 0x236c0000
00:00:09.4622336 : Make Resident into 0x236c0000
00:00:09.4634816 : Destroy
```

```
=====
PAGE FAULT SUMMARY
=====
```

Offending VA: 0x236c04000

# REPORT – PAGE FAULT SUMMARY

Assign resource names:

- DX12: ID3D12Object::SetName
- Vulkan: VK\_EXT\_debug\_utils

Resource id: 0x5a49f060000a7f

Type: Image

Name: Postprocessing render target 4

Virtual address:

0x236c00000 [size: 16810352 (16.03 MB), parent address + off

Commit type: COMMITTED

Attributes:

Create flags: PREFER\_SWIZZLE\_EQUATIONS | FIXED\_TILE\_SWIZZLE (

Usage flags: SHADER\_READ | SHADER\_WRITE | RESOLVE\_DESTINATION

Image type: 2D

Dimensions <x, y, z>: 1920 x 1080 x 1

Swizzle pattern: XYZW

Image format: X16Y16Z16W16\_FLOAT

Mip levels: 1

Slices: 1

Sample count: 1

Fragment count: 1

Tiling type: Optimal

Resource timeline:

00:00:09.4618368 : Create

00:00:09.4622336 : Bind into 0x236c00000

00:00:09.4622336 : Make Resident into 0x236c00000

00:00:09.4634816 : Destroy

# REPORT – PAGE FAULT SUMMARY

## Interpreting the results:

- Page fault & resources found at offending VA, resource has been destroyed  
→ Likely use-after-free bug

Resource id: 0x5a49f060000a7f

Type: Image

Name: Postprocessing render target 4

Virtual address:

0x236c00000 [size: 16810352 (16.03 MB), parent address + offset]

Commit type: COMMITTED

Attributes:

Create flags: PREFER\_SWIZZLE\_EQUATIONS | FIXED\_TILE\_SWIZZLE

Usage flags: SHADER\_READ | SHADER\_WRITE | RESOLVE\_DESTINATION

Image type: 2D

Dimensions <x, y, z>: 1920 x 1080 x 1

Swizzle pattern: XYZW

Image format: X16Y16Z16W16\_FLOAT

Mip levels: 1

Slices: 1

Sample count: 1

Fragment count: 1

Tiling type: Optimal

Resource timeline:

00:00:09.4618368 : Create

00:00:09.4622336 : Bind into 0x236c00000

00:00:09.4622336 : Make Resident into 0x236c00000



00:00:09.4634816 : Destroy

# QUICK INTERPRETATION OF CRASH TYPE

Page fault detected?	VA has associated resources?	Meaning
Yes	Yes	Attempt to access a resource that is already released.
Yes	No (means no resource ever resided in this VA)	Out-of-bounds access.
No	No	Hang (e.g., an infinite loop in a shader). Use markers to narrow down.

# REPORT – SYSTEM INFO

## System information:

- Current date and time
- Hostname
- Crashing app .exe file path, PID
- Windows version, graphics driver version
- CPU, GPU
- RAM, VRAM
- ...

CPU info

=====

CPU count: 1

CPU #1:

Name: AMD Ryzen 7 5800X 8-Core Processor

Architecture: x64

CPU ID: AMD64 Family 25 Model 33 Stepping 0

Virtualization: disabled

GPU info

=====

GPU count: 1

GPU #1:

Name: AMD Radeon(TM) RX 6500 XT

Device ID: 0x743f

Device revision ID: 0x46

Device family ID: 0x8f

Device graphics engine ID: 0xd

Device PCI revision ID: 0xc1

Big SW version: 2021.1.1

Memory type: Gddr6

Memory heap count: 2

Memory heap #1:

Heap type: invisible

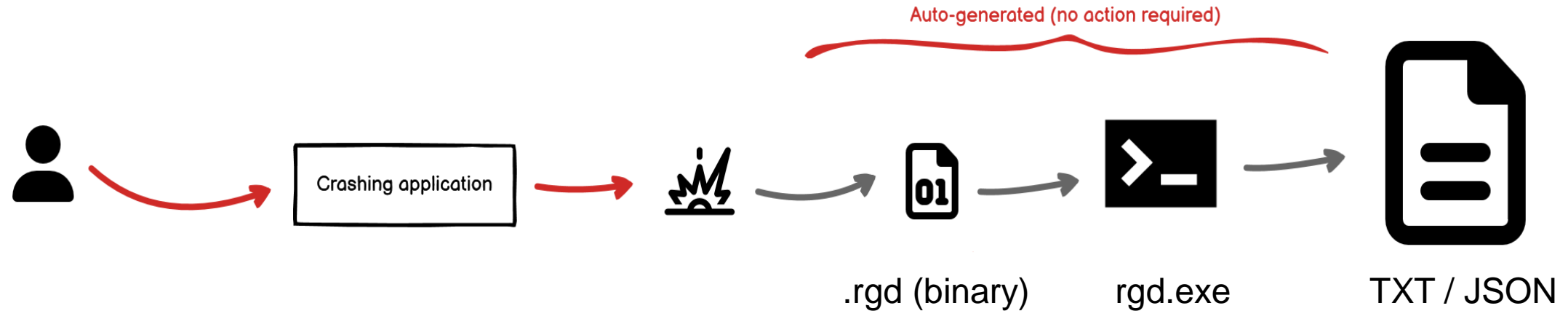
Heap size: 8304721920 (7.73 GB)

Memory heap #2:

Heap type: local

Heap size: 268435456 (256.00 MB)

# RGD UNDER THE HOOD



rgd.exe – a command-line app

- Additional parameters available (e.g., to output JSON format)
  - Great for making RGD part of your automated crash reporting pipeline!
- Open-source, MIT license
  - [https://github.com/GPUOpen-Tools/radeon\\_gpu\\_detective](https://github.com/GPUOpen-Tools/radeon_gpu_detective)
- Feedback welcome! Contact us or create Issue# on GitHub.

# CONCLUSIONS

- Nice to have another tool for investigating GPU hangs on PC
  - Especially when they are AMD specific
- AMD Radeon™ GPU Detective is easy to use
- Just enabling crash analysis improves DRED output
- RGD can provide extra information on page faults
  - Recently destroyed resources timeline
  - Show associated resources
- RGD has minimal overhead
  - Can have it running for normal development flow
  - May want to turn it off for CPU profiling



# THANKS

Amit Ben-Moshe

Budirijanto Purnomo

Christopher Cenotti

Gordon Selley

Jonas Gustavsson

Ken Lagos

Marek Machliński

Nicolas Thibieroz

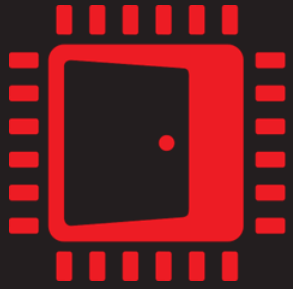
Yana Mateeva

# DISCLAIMER & ATTRIBUTIONS

## DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© 2024 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective owners.



**AMD**   
GPUOpen

**Visit our website**

<https://gpuopen.com>



**Follow us on X**

<https://twitter.com/GPUOpen>



**Follow us on Mastodon**

<https://mastodon.gamedev.place/@gpuopen>



**Follow us on Zhihu**

<https://www.zhihu.com/org/gpuopen-7>