

# OpenGL - Debugging tools

Advanced Graphics Programming



# OpenGL - Debugging tools

## OpenGL programming can be very error prone

Most of the time, a **single failing call** to OpenGL can cause an entire portion of an application to stop working, with **nothing being drawn on the screen**.

Examples:

- An incorrect parameter was passed to an OpenGL function
- We did not send some parameters to the shader properly
- We forgot to bind some resource (VAO, program, framebuffer...)
- Long list of etcetera

Classic way: glGetError()

# OpenGL - glGetError function

Make sure no errors are being returned by OpenGL:

- After every API call
- At some points in your application

**Errors stack up**, so they have to be handled with the function `glGetError()` **using a loop**.

```
GLenum error;
do
{
    error = glGetError();
    if (error != GL_NO_ERROR)
    {
        // handle the error
    }
}
while (error != GL_NO_ERROR &&
       error != GL_CONTEXT_LOST);
```

# OpenGL - glGetError function

For a list of all errors returned by *glGetError()* you can refer to the following page:

[https://www.khronos.org/opengl/wiki/Open\\_GL\\_Error](https://www.khronos.org/opengl/wiki/Open_GL_Error)

```
GLenum error;
do
{
    error = glGetError();
    if (error != GL_NO_ERROR)
    {
        // handle the error
    }
}
while (error != GL_NO_ERROR &&
       error != GL_CONTEXT_LOST);
```

# OpenGL - glGetError function

## Error guards


- Objects to check errors within a scope
- Use of constructor and destructor

```
class OpenGLErrorGuard
{
public:

    OpenGLErrorGuard(const char *message) : msg(message) {
        checkGLError("BEGIN", msg);
    }

    ~OpenGLErrorGuard() {
        checkGLError("END", msg);
    }

    static void checkGLError(const char *around, const char *message);
    const char *msg;
};
```



glGetError() stuff...

## Usage

- Define an object at some point within a scope
- The object will check for errors when created and destroyed

```
void OpenGLWidget::blur()
{
    OpenGLErrorGuard guard("blur()");

    // Blurring OpenGL calls
}
```

GL\_KHR\_debug extension

# GL\_KHR\_debug extension

[https://www.khronos.org/registry/OpenGL/extensions/KHR/KHR\\_debug.txt](https://www.khronos.org/registry/OpenGL/extensions/KHR/KHR_debug.txt)

- First available in OpenGL 4.2
- Core since 4.3 (no need to check availability :-)

Provides several utilities. Among others, I found these two the most useful:

- Installing a custom error callback
- Inserting custom debug markers



# GL\_KHR\_debug - debug callback

During initialization, we can install a custom error callback this way:

```
if (GLVersion.major > 4 || (GLVersion.major == 4 && GLVersion.minor >= 3))  
{  
    glDebugMessageCallback(OnGLError, app);  
}
```

- First parameter is our callback function. More info here:
  - [https://www.khronos.org/opengl/wiki/Debug\\_Output](https://www.khronos.org/opengl/wiki/Debug_Output)
- Second parameter is a custom user pointer, just in case we need access to some of our data

# GL\_KHR\_debug - debug callback

```
void OnGLError(GLenum source, GLenum type, GLuint id, GLenum severity, GLsizei length, const GLchar* message, const void* userParam)
{
    if (severity == GL_DEBUG_SEVERITY_NOTIFICATION)
        return;

    ELOG("OpenGL debug message: %s", message);

    switch (source)
    {
        case GL_DEBUG_SOURCE_API:           ELOG(" - source: GL_DEBUG_SOURCE_API"); break; // Calls to the OpenGL API
        case GL_DEBUG_SOURCE_WINDOW_SYSTEM: ELOG(" - source: GL_DEBUG_SOURCE_WINDOW_SYSTEM"); break; // Calls to a window-system API
        case GL_DEBUG_SOURCE_SHADER_COMPILER: ELOG(" - source: GL_DEBUG_SOURCE_SHADER_COMPILER"); break; // A compiler for a shading language
        case GL_DEBUG_SOURCE_THIRD_PARTY:    ELOG(" - source: GL_DEBUG_SOURCE_THIRD_PARTY"); break; // An application associated with OpenGL
        case GL_DEBUG_SOURCE_APPLICATION:    ELOG(" - source: GL_DEBUG_SOURCE_APPLICATION"); break; // Generated by the user of this application
        case GL_DEBUG_SOURCE_OTHER:          ELOG(" - source: GL_DEBUG_SOURCE_OTHER"); break; // Some source that isn't one of these
    }

    switch (type)
    {
        case GL_DEBUG_TYPE_ERROR:           ELOG(" - type: GL_DEBUG_TYPE_ERROR"); break; // An error, typically from the API
        case GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR: ELOG(" - type: GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR"); break; // Some behavior marked deprecated but still usable
        case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR: ELOG(" - type: GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR"); break; // Something has invoked undefined behavior
        case GL_DEBUG_TYPE_PORTABILITY:      ELOG(" - type: GL_DEBUG_TYPE_PORTABILITY"); break; // Some functionality the user relies upon: not portable
        case GL_DEBUG_TYPE_PERFORMANCE:      ELOG(" - type: GL_DEBUG_TYPE_PERFORMANCE"); break; // Code has triggered possible performance issues
        case GL_DEBUG_TYPE_MARKER:           ELOG(" - type: GL_DEBUG_TYPE_MARKER"); break; // Command stream annotation
        case GL_DEBUG_TYPE_PUSH_GROUP:       ELOG(" - type: GL_DEBUG_TYPE_PUSH_GROUP"); break; // Group pushing
        case GL_DEBUG_TYPE_POP_GROUP:        ELOG(" - type: GL_DEBUG_TYPE_POP_GROUP"); break; // Group popping
        case GL_DEBUG_TYPE_OTHER:            ELOG(" - type: GL_DEBUG_TYPE_OTHER"); break; // Some type that isn't one of these
    }

    switch (severity)
    {
        case GL_DEBUG_SEVERITY_HIGH:         ELOG(" - severity: GL_DEBUG_SEVERITY_HIGH"); break; // All OpenGL Errors, shader compilation/linker errors
        case GL_DEBUG_SEVERITY_MEDIUM:       ELOG(" - severity: GL_DEBUG_SEVERITY_MEDIUM"); break; // Major performance warnings, shader compilation warnings
        case GL_DEBUG_SEVERITY_LOW:          ELOG(" - severity: GL_DEBUG_SEVERITY_LOW"); break; // Redundant state change performance warning, shader compilation warnings
        case GL_DEBUG_SEVERITY_NOTIFICATION: ELOG(" - severity: GL_DEBUG_SEVERITY_NOTIFICATION"); break; // Anything that isn't an error or performance warning
    }
}
```

# GL\_KHR\_debug - debug callback

```
void OnGLError(GLenum source, GLenum type, GLuint id, GLenum severity, GLsizei length, const GLchar* message, const void* userParam)
{
    if (severity == GL_DEBUG_SEVERITY_NOTIFICATION)
        return;

    ELOG("OpenGL debug message: %s", message);

    switch (source)
    {
        case GL_DEBUG_SOURCE_API:           ELOG(" - source: GL_DEBUG_SOURCE_API"); break; // Calls to the OpenGL API
        case GL_DEBUG_SOURCE_WINDOW_SYSTEM: ELOG(" - source: GL_DEBUG_SOURCE_WINDOW_SYSTEM"); break; // Calls to a window-system API
        case GL_DEBUG_SOURCE_SHADER_COMPILER: ELOG(" - source: GL_DEBUG_SOURCE_SHADER_COMPILER"); break; // A compiler for a shading language
        case GL_DEBUG_SOURCE_THIRD_PARTY:    ELOG(" - source: GL_DEBUG_SOURCE_THIRD_PARTY"); break; // An application associated with OpenGL
        case GL_DEBUG_SOURCE_APPLICATION:    ELOG(" - source: GL_DEBUG_SOURCE_APPLICATION"); break; // Generated by the user of this application
        case GL_DEBUG_SOURCE_OTHER:          ELOG(" - source: GL_DEBUG_SOURCE_OTHER"); break; // Some source that isn't one of these
    }

    switch (type)
    {
        case GL_DEBUG_TYPE_ERROR:           ELOG(" - type: GL_DEBUG_TYPE_ERROR"); break; // An error, typically from the API
        case GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR: ELOG(" - type: GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR"); break; // Some behavior marked deprecated but still usable
        case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR: ELOG(" - type: GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR"); break; // Something has invoked undefined behavior
        case GL_DEBUG_TYPE_PORTABILITY:      ELOG(" - type: GL_DEBUG_TYPE_PORTABILITY"); break; // Some functionality the user relies upon: not portable
        case GL_DEBUG_TYPE_PERFORMANCE:     ELOG(" - type: GL_DEBUG_TYPE_PERFORMANCE"); break; // Code has triggered possible performance issues
        case GL_DEBUG_TYPE_MARKER:          ELOG(" - type: GL_DEBUG_TYPE_MARKER"); break; // Command stream annotation
        case GL_DEBUG_TYPE_PUSH_GROUP:       ELOG(" - type: GL_DEBUG_TYPE_PUSH_GROUP"); break; // Group pushing
        case GL_DEBUG_TYPE_POP_GROUP:        ELOG(" - type: GL_DEBUG_TYPE_POP_GROUP"); break; // Group popping
        case GL_DEBUG_TYPE_OTHER:           ELOG(" - type: GL_DEBUG_TYPE_OTHER"); break; // Some type that isn't one of these
    }

    switch (severity)
    {
        case GL_DEBUG_SEVERITY_HIGH:         ELOG(" - severity: GL_DEBUG_SEVERITY_HIGH"); break; // All OpenGL Errors, shader compilation/linker errors
        case GL_DEBUG_SEVERITY_MEDIUM:       ELOG(" - severity: GL_DEBUG_SEVERITY_MEDIUM"); break; // Major performance warnings, shader compilation warnings
        case GL_DEBUG_SEVERITY_LOW:          ELOG(" - severity: GL_DEBUG_SEVERITY_LOW"); break; // Redundant state change performance warning, shader compilation warnings
        case GL_DEBUG_SEVERITY_NOTIFICATION: ELOG(" - severity: GL_DEBUG_SEVERITY_NOTIFICATION"); break; // Anything that isn't an error or performance warning
    }
}
```

You can put breakpoints in this function!!!

Shader hot reload



# Shader file timestamp

```
u32 LoadProgram(App* app, const char* filepath, const char* programName)
{
    String programSource = ReadTextFile(filepath);

    Program program = {};
    program.handle = CreateProgramFromSource(programSource, programName);
    program.filepath = filepath;
    program.programName = programName;
    program.lastWriteTimestamp = GetFileLastWriteTimestamp(filepath);
    app->programs.push_back(program);

    return app->programs.size() - 1;
}
```

With this timestamp  
you can later check  
whether or not the  
file was modified  
since it was loaded...

# In Update() -> check timestamp / reload

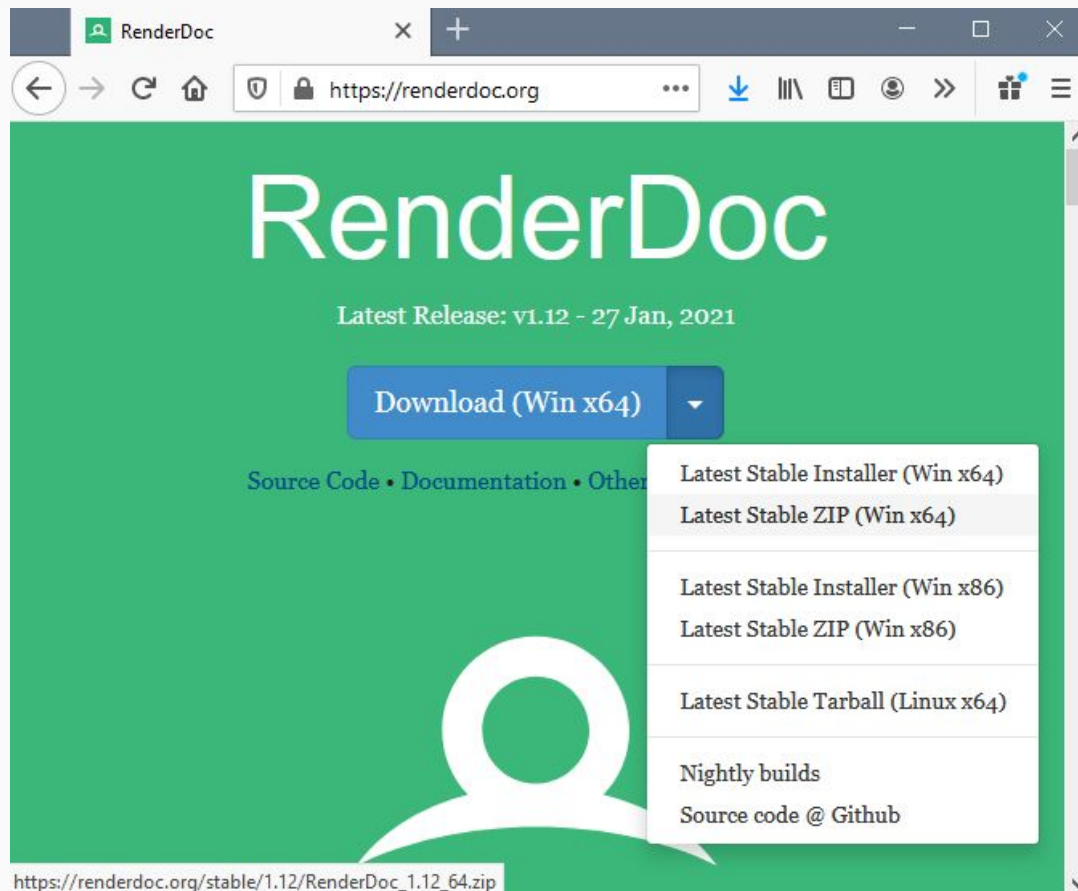
```
for (u64 i = 0; i < app->programs.size(); ++i)
{
    Program& program = app->programs[i];
    u64 currentTimeStamp = GetFileLastWriteTimestamp(program.filepath.c_str());
    if (currentTimeStamp > program.lastWriteTimestamp)
    {
        glDeleteProgram(program.handle);
        String programSource = ReadTextFile(program.filepath.c_str());
        const char* programName = program.programName.c_str();
        program.handle = CreateProgramFromSource(programSource, programName);
        program.lastWriteTimestamp = currentTimeStamp;
    }
}
```



# RenderDoc



# RenderDoc (will save your life ;-)

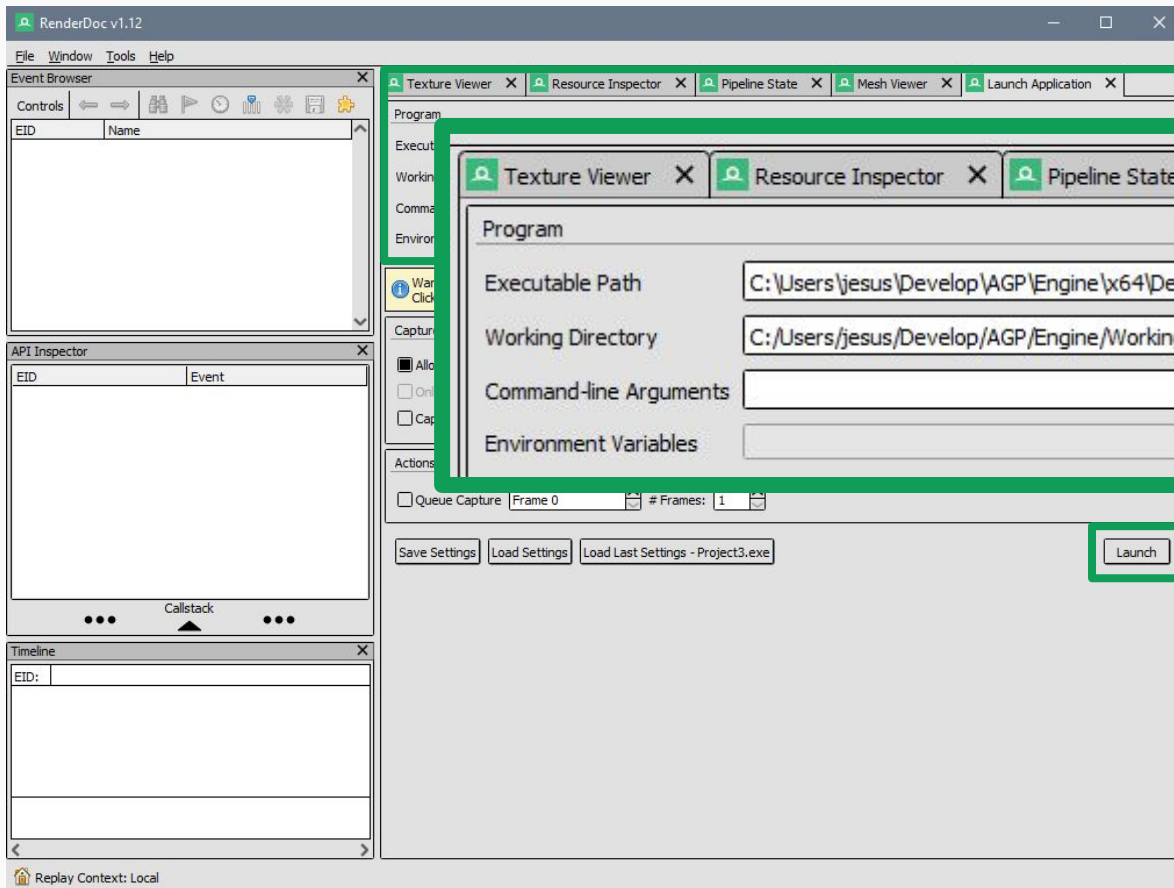


Be sure to download the appropriate version...

Probably Win x64 (if you are compiling your OpenGL application in for x64)



# Launch your application



Zoomed view

# Capture a frame trace

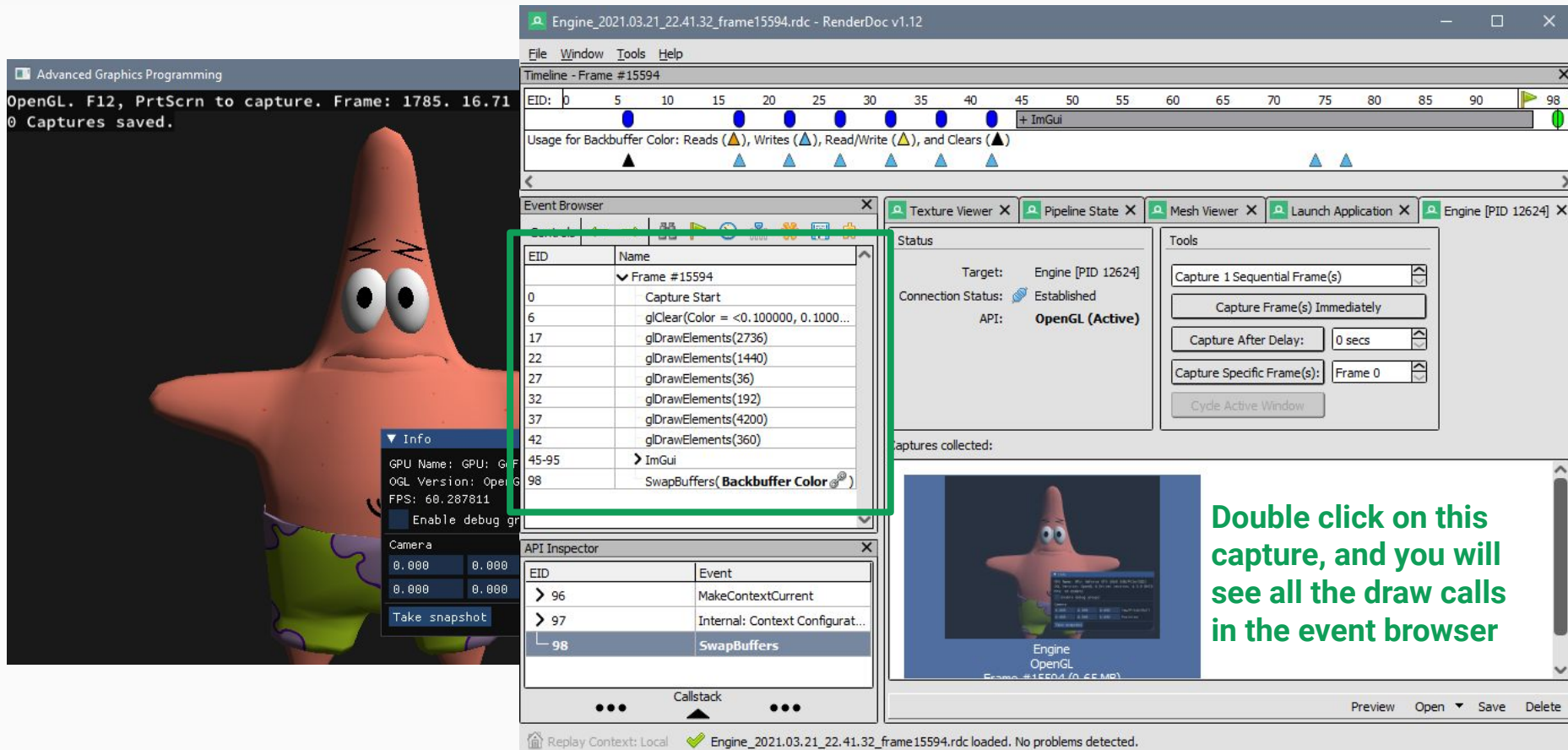
The screenshot displays the RenderDoc v1.12 application interface. On the left, a 3D scene features a pink starfish character with large eyes and a green and purple patterned garment. A text overlay in the top-left corner of the scene reads: "OpenGL. F12, PrtScrn to capture. Frame: 1785. 16.71 FPS. 0 Captures saved." Below the scene, an "Info" panel shows GPU details: "GPU Name: GPU: GeForce 1080 Ti", "OpenGL Version: OpenGL 4.5", and "FPS: 60.287811". It also includes a "Take snapshot" button.

The main interface on the right is divided into several panels:

- Timeline:** A panel at the top right for recording and reviewing captures.
- Event Browser:** A panel below the timeline for viewing captured events.
- Texture Viewer, Pipeline State, Mesh Viewer, Launch Application, Engine [PID 12624]:** A row of panels for inspecting different aspects of the captured frame.
- Tools:** A panel on the right side of the main interface containing capture controls. The "Capture Frame(s) Immediately" button is highlighted with a green box. Other options include "Capture After Delay: 0 secs" and "Capture Specific Frame(s): Frame 0".
- API Inspector:** A panel at the bottom for inspecting API calls.
- Callstack:** A panel at the very bottom for viewing the callstack.

The bottom status bar indicates "Replay Context: Local".

# You can see all the OpenGL draw calls



The screenshot displays the RenderDoc v1.12 interface, which is used for capturing and analyzing OpenGL draw calls. The main window shows a 3D scene with a starfish character. The interface includes several panels:

- Timeline - Frame #15594:** Shows a sequence of events with markers for reads, writes, read/writes, and clears. A green box highlights the 'SwapBuffers' event at EID 98.
- Event Browser:** A table listing events with their EIDs and names. The 'SwapBuffers' event is highlighted.
- Texture Viewer, Pipeline State, Mesh Viewer, Launch Application, and Engine [PID 12624]:** Panels for viewing specific data and managing the application.
- Tools:** A panel with options to capture frames, including 'Capture 1 Sequential Frame(s)', 'Capture Frame(s) Immediately', 'Capture After Delay', and 'Capture Specific Frame(s)'. A green box highlights the 'Capture 1 Sequential Frame(s)' option.
- API Inspector:** A panel showing the current API state, including the 'SwapBuffers' event.
- Info:** A panel showing GPU information, including GPU Name, Version, FPS, and a 'Take snapshot' button.

A green box highlights the 'SwapBuffers' event in the Event Browser, which is the event that triggers the capture of the frame. A green box also highlights the 'Capture 1 Sequential Frame(s)' option in the Tools panel, indicating that this is the method used to capture the frame.

Double click on this capture, and you will see all the draw calls in the event browser

Replay Context: Local Engine\_2021.03.21\_22.41.32\_frame15594.rdc loaded. No problems detected.

# We can create draw call groups... how??

The screenshot displays the RenderDoc v1.12 interface, which is used for capturing and analyzing GPU activity. The main window shows a 3D scene with a star character. A green arrow points to the 'Info' panel, which contains the following information:

- GPU Name: GPU: GeForce 1080 Ti
- OpenGL Version: OpenGL 4.5.0 NVIDIA 418.87
- Device: GeForce 1080 Ti
- Driver: NVIDIA GeForce 418.87
- API: OpenGL 4.5
- API Calls: 100
- SwapBuffers (Backbuffer Color)

A second green arrow points to the 'Enable debug groups' checkbox, which is checked. Below this, the 'Camera' section shows the following values:

- 0.000 0.000
- 0.000 0.000

The 'Take snapshot' button is also visible. The 'Event Browser' panel on the right shows a list of events, including 'Capture Start', 'Shaded model', and 'ImGui'. The 'Texture Viewer' panel shows the 'Status' of the capture, including 'Target: Engine [PID 12624]', 'Connection Status: Established', and 'API: OpenGL (Active)'. The 'Mesh Viewer' panel shows the 'Tools' section, which includes 'Capture 1 Sequential Frame(s)', 'Capture Frame(s) Immediately', 'Capture After Delay: 0 secs', and 'Capture Specific Frame(s): Frame 0'. The 'Launch Application' panel shows the 'Captures collected' section, which displays two captured frames of the star character. The bottom status bar indicates that the engine is loaded and no problems were detected.

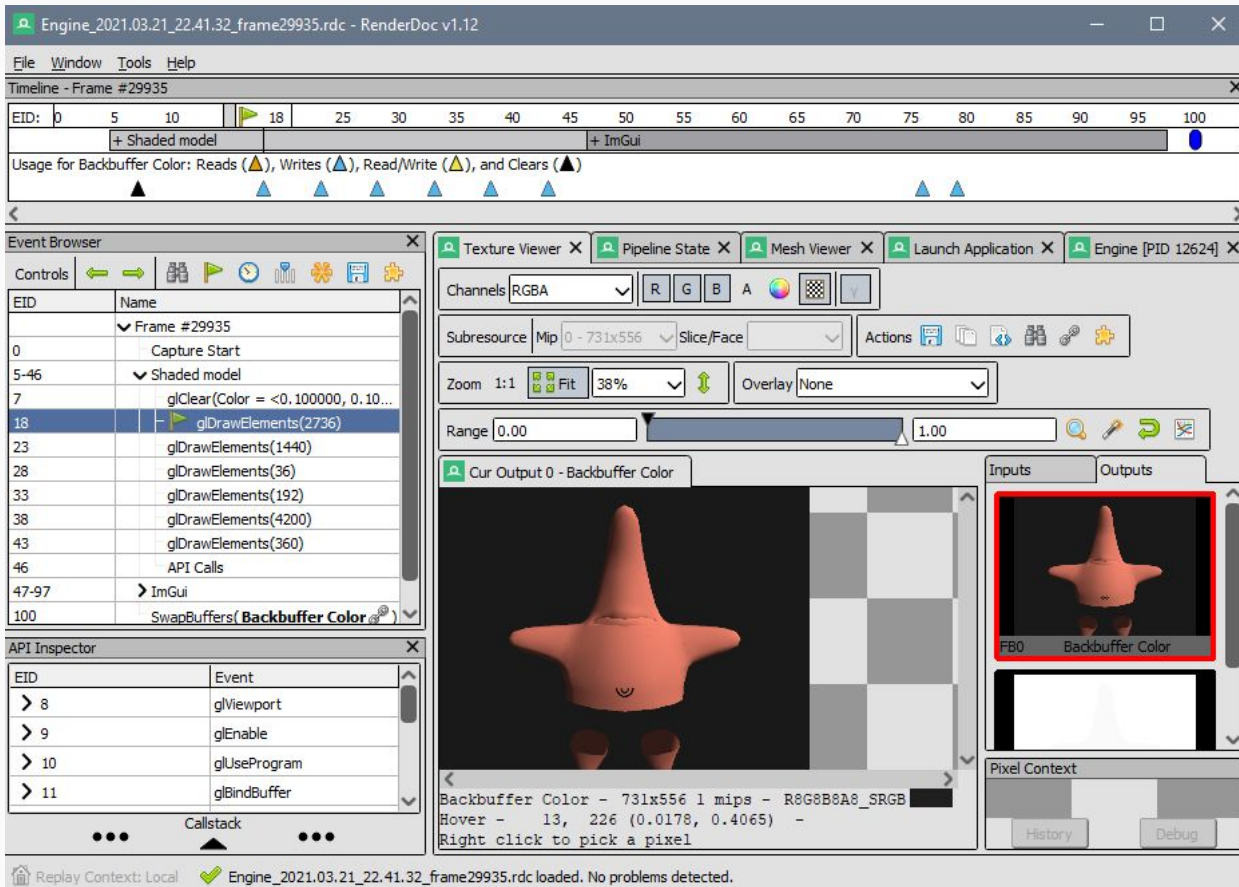
# Thanks to glPushDebugGroup / glPopDebugGroup

They belong to GL\_KHR\_debug extension

- You know, Core since OpenGL 4.3
- No need for the *enableDebugGroups* boolean (that was just to make a point)
- Between these calls one would change the OpenGL state and submit several draw calls
- Even several iterations of changing state -> draw calls

```
if (app->enableDebugGroups) glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 1, -1, "Shaded model");  
  
// OpenGL rendering code  
{ ... }  
  
if (app->enableDebugGroups) glPopDebugGroup();
```

# Texture Viewer



Engine\_2021.03.21\_22.41.32\_frame29935.rdc - RenderDoc v1.12

File Window Tools Help

Timeline - Frame #29935

EID: 0 5 10 18 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

+ Shaded model + ImGui

Usage for Backbuffer Color: Reads (▲), Writes (▲), Read/Write (▲), and Clears (▲)

Event Browser

Controls

EID Name

- 0 Frame #29935
  - 5-46 Capture Start
  - 7 Shaded model
    - 18 glClear(Color = <0.100000, 0.10...
    - 23 glDrawElements(1440)
    - 28 glDrawElements(36)
    - 33 glDrawElements(192)
    - 38 glDrawElements(4200)
    - 43 glDrawElements(360)
    - 46 API Calls
    - 47-97 ImGui
    - 100 SwapBuffers(Backbuffer Color)

API Inspector

EID Event

- > 8 glViewport
- > 9 glEnable
- > 10 glUseProgram
- > 11 glBindBuffer

Texture Viewer

Channels: RGBA R G B A

Subresource: Mip 0 - 731x556 Slice/Face

Actions

Zoom: 1:1 Fit 38% Overlay: None

Range: 0.00 1.00

Cur Output 0 - Backbuffer Color

Inputs Outputs

FB0 Backbuffer Color

Pixel Context

History Debug

Backbuffer Color - 731x556 1 mips - R8G8B8A8\_SRGB

Hover - 13, 226 (0.0178, 0.4065) -

Right click to pick a pixel

Replay Context: Local Engine\_2021.03.21\_22.41.32\_frame29935.rdc loaded. No problems detected.

Selecting a draw call, we can see what was rendered in the output viewport



# Mesh Viewer

The screenshot displays the Mesh Viewer application interface, which is part of a larger engine window titled "Engine\_2021.03.21\_22.41.32\_frame29935.rdc - RenderDoc v1.12". The interface is divided into several panels:

- Timeline:** Shows a sequence of events for frame #29935. The timeline ranges from EID 0 to 100. A "Shaded model" event is highlighted, and a "Usage for Backbuffer Color" bar indicates reads, writes, and clears.
- Event Browser:** A list of events for frame #29935. The event "glDrawElements(2736)" is selected, showing its details in the "API Inspector" panel.
- API Inspector:** Displays the details of the selected event, including the event name, EID, and a callstack.
- Mesh Viewer:** The main panel showing the mesh data. It includes a "VS Input" table, a "VS Output" table, and a "GS/DS Output" table. The "VS Input" table shows vertex indices and positions. The "VS Output" table shows the transformed positions. The "GS/DS Output" table shows the final output positions.
- Preview:** A 3D preview window showing the rendered mesh. The mesh is a red, star-like object with a blue outline, positioned on a checkered floor.

The bottom status bar indicates: "Replay Context: Local Engine\_2021.03.21\_22.41.32\_frame29935.rdc loaded. No problems detected."

In the Mesh Viewer, we can also see (for a draw call) what the input and the output of the vertex shader is...

Useful to see if you have configured your VBO/EBO/VAOs correctly (input)

Useful to see if your vertex shader transforms are correct (output)

# Pipeline state

Engine\_2021.03.21\_22.41.32\_frame29935.rdc - RenderDoc v1.12

File Window Tools Help

Timeline - Frame #29935

EID: 0 5 10 15 18 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

+ Shaded model + ImGui

Usage for Backbuffer Color: Reads (▲), Writes (▲), Read/Write (▲), and Clears (▲)

Event Browser

Controls

EID Name

0 Frame #29935

5-46 Capture Start

7 glClear(Color = <0.100000, 0.10...

18 glDrawElements(2736)

23 glDrawElements(1440)

28 glDrawElements(36)

33 glDrawElements(192)

38 glDrawElements(4200)

43 glDrawElements(360)

46 API Calls

47-97 ImGui

100 SwapBuffers( Backbuffer Color )

API Inspector

EID Event

> 8 glViewport

> 9 glEnable

> 10 glUseProgram

> 11 glBindBuffer

Callstack

Texture Viewer Pipeline State Mesh Viewer Launch Application Engine [PID 12624]

Controls Show Unused Items Show Empty Items Export Extensions

VTX VS TCS TES GS RS FS FB CS

Vertex Attribute Formats

Index	Enabled	Name	Format/Generic Value	Buffer Slot	Relative Offset	Go
0	Enabled	aPosition	R32G32B32_FLOAT	0	0	→
1	Enabled	aNormal	R32G32B32_FLOAT	1	0	→
2	Enabled	aTexCoord	R32G32_FLOAT	2	0	→

Vertex Array Object

Vertex Array 78

Buffers

Slot	Buffer	Stride	Offset
Element	Buffer 61	4	0
0	Buffer 60	56	0
1	Buffer 60	56	12
2	Buffer 60	56	24

Mesh View

Primitive Topology

Triangle List

Restart Idx: Disabled

Replay Context: Local Engine\_2021.03.21\_22.41.32\_frame29935.rdc loaded. No problems detected.

By clicking on each one of the stages, we can see exactly how they are configured... and detect if something is not the way we intended.

- Vertex input buffers
- Shader uniform inputs
  - VS / FS
  - Buffers and textures
- Rasterization state
  - Backface culling
  - Fill mode
- Framebuffer state
  - Blending
  - Depth test



Now a magic trick...



# You can edit->apply changes in shaders!!!

Engine\_2021.03.21\_22.41.32\_frame29935.rdc - RenderDoc v1.12

File Window Tools Help

Timeline - Frame #29935

EID: 0 5 10 15 18 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

+ Shaded model + ImGui

Usage for Backbuffer Color: Reads (▲), Writes (▲), Read/Write (▲), and Clears (▲)

Event Browser

Controls

EID Name

0 ✓ Frame #29935

5-46 Capture Start

7 ✓ Shaded model

18 glClear(Color = <0.100000, 0.10...

23 glDrawElements(1440)

28 glDrawElements(36)

33 glDrawElements(192)

38 glDrawElements(4200)

43 glDrawElements(360)

46 API Calls

47-97 > ImGui

100 SwapBuffers( Backbuffer Color )

API Inspector

EID Event

> 8 glViewport

> 9 glEnable

> 10 glUseProgram

> 11 glBindBuffer

Callstack

Texture Viewer Pipeline State Mesh Viewer Launch Application Engine [PID 12624]

Controls Show Unused Items Show Empty Items Export Extensions

VTX VS TCS TES GS RS FS FB CS

Shader

Program 70 > Shader 69 View Edit Save

Textures

Slot	Resource	Type	Width	Height	Depth	Array Size	Format	Go
0: uTexture	Texture 57	Texture 2D	900	900	1	1	R8G8B8_UNORM	Go

Samplers

Slot	Object	Wrap Mode	Filter	LOD Clamp	LOD Bias
0: uTexture	Texture 57	STR: ClampEdge	Min&Mag: Linear, Mip: None	-1000 - 1000	0

Uniforms and UBOs

Slot	Buffer	Byte Range	Size	Go
------	--------	------------	------	----

Replay Context: Local Engine\_2021.03.21\_22.41.32\_frame29935.rdc loaded. No problems detected.



# You can edit->apply changes in shaders!!!

Engine\_2021.03.21\_22.41.32\_frame29935.rdc - RenderDoc v1.12

File Window Tools Help

Timeline - Frame #29935

EID: 0 5 10 15 18 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

+ Shaded model + ImGui

Usage for Backbuffer Color: Reads (▲), Writes (▲), Read/Write (▲), and Clears (▲)

Event Browser

EID	Name
0	Frame #29935
5-46	Shaded model
7	glClear(Color = <0.100000, 0.10...
18	glDrawElements(2736)
23	glDrawElements(1440)
28	glDrawElements(36)
33	glDrawElements(192)
38	glDrawElements(4200)
43	glDrawElements(360)
46	API Calls
47-97	ImGui
100	SwapBuffers( Backbuffer Color )

API Inspector

EID	Event
> 8	glViewport
> 9	glEnable
> 10	glUseProgram
> 11	glBindBuffer

Callstack

Textu... X Pipeli... X Mesh ... X Launch... X Engine [... X \*Editing Shader 69 - main... X

Find Refresh

```
main.gsl
195 layout(location = 0) out vec4 oColor;
196
197 void main()
198 {
199     vec4 albedo = texture(uTexture, vTexCoord);
200     vec3 N = normalize(vNormal);
201     vec3 L = normalize(vec3(1.0));
202     float ambientFactor = 0.2;
203     float diffuseFactor = 0.8 * max(0.0, dot(L,N));
204     oColor = ambientFactor * albedo +
205             diffuseFactor * albedo;
206     oColor *= vec4(1.0, 0.0, 0.0, 1.0); // <-- I've added this
207 }
208
209 #endif
210 #endif
```

Compilation Settings

Entry Point: main Source Type: GLSL

Compiler: BuiltIn

Errors

Replay Context: Local Engine\_2021.03.21\_22.41.32\_frame29935.rdc loaded. No problems detected.



Now get your weapons ready...



Here we go OpenGL!!!