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/OpenGL_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

- 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
- 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 applicat
    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
    case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR:  ELOG(" - type: GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR"); break; // Something has invoked undefined b
    case GL_DEBUG_TYPE_PORTABILITY:         ELOG(" - type: GL_DEBUG_TYPE_PORTABILITY"); break; // Some functionality the user relies upon
    case GL_DEBUG_TYPE_PERFORMANCE:         ELOG(" - type: GL_DEBUG_TYPE_PERFORMANCE"); break; // Code has triggered possible performance
    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; // foo
    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/link
    case GL_DEBUG_SEVERITY_MEDIUM:       ELOG(" - severity: GL_DEBUG_SEVERITY_MEDIUM"); break; // Major performance warnings, shader compi
    case GL_DEBUG_SEVERITY_LOW:          ELOG(" - severity: GL_DEBUG_SEVERITY_LOW"); break; // Redundant state change performance warning,
    case GL_DEBUG_SEVERITY_NOTIFICATION: ELOG(" - severity: GL_DEBUG_SEVERITY_NOTIFICATION"); break; // Anything that isn't an error or pe
    }
}
```

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 applicat
    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
    case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR:  ELOG(" - type: GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR"); break; // Something has invoked undefined b
    case GL_DEBUG_TYPE_PORTABILITY:         ELOG(" - type: GL_DEBUG_TYPE_PORTABILITY"); break; // Some functionality the user relies upon
    case GL_DEBUG_TYPE_PERFORMANCE:         ELOG(" - type: GL_DEBUG_TYPE_PERFORMANCE"); break; // Code has triggered possible performance
    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; // foo
    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/link
    case GL_DEBUG_SEVERITY_MEDIUM:       ELOG(" - severity: GL_DEBUG_SEVERITY_MEDIUM"); break; // Major performance warnings, shader compi
    case GL_DEBUG_SEVERITY_LOW:          ELOG(" - severity: GL_DEBUG_SEVERITY_LOW"); break; // Redundant state change performance warning,
    case GL_DEBUG_SEVERITY_NOTIFICATION: ELOG(" - severity: GL_DEBUG_SEVERITY_NOTIFICATION"); break; // Anything that isn't an error or pe
    }
}
```

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;
}
```

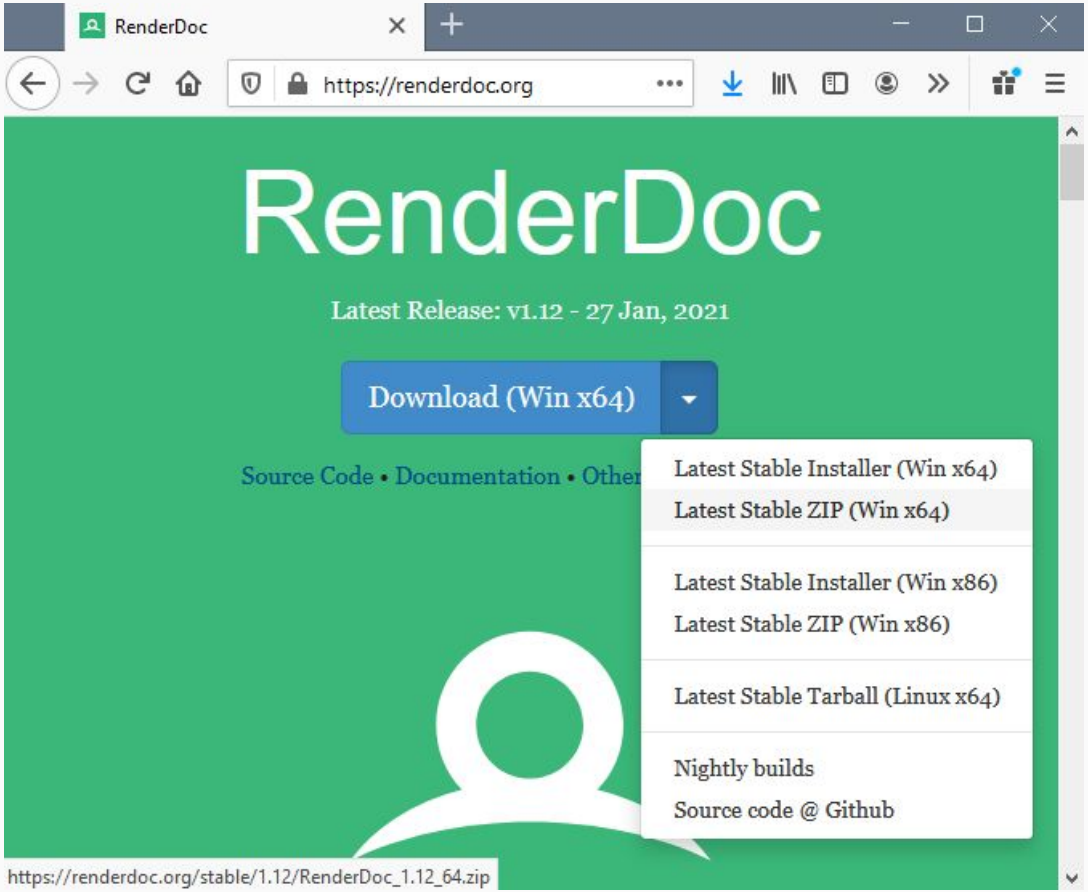
With this timestamp you can later check whether or not the file was modified since it was loaded...

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

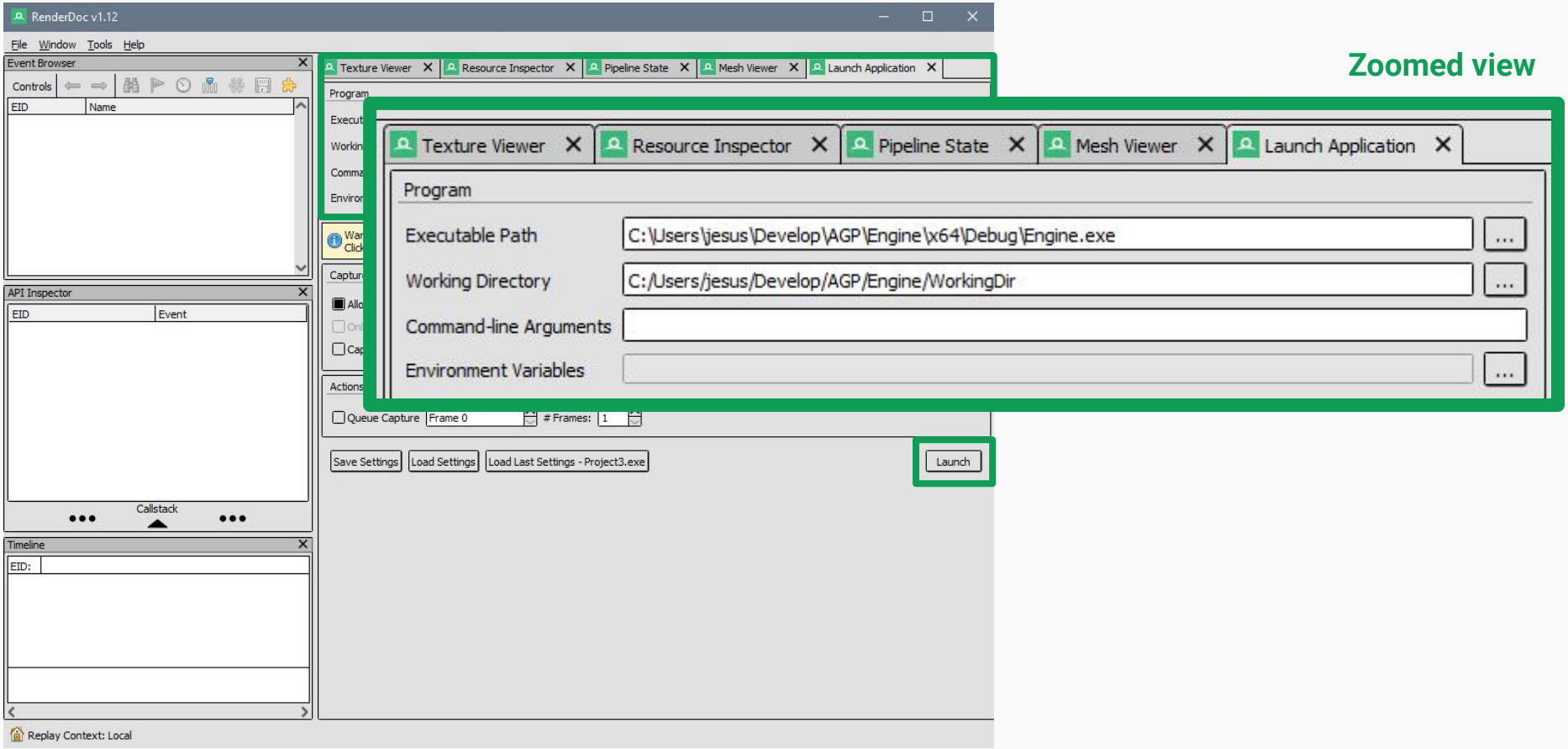
RenderDoc (will save your life ;-)



Be sure to download the appropriate version…

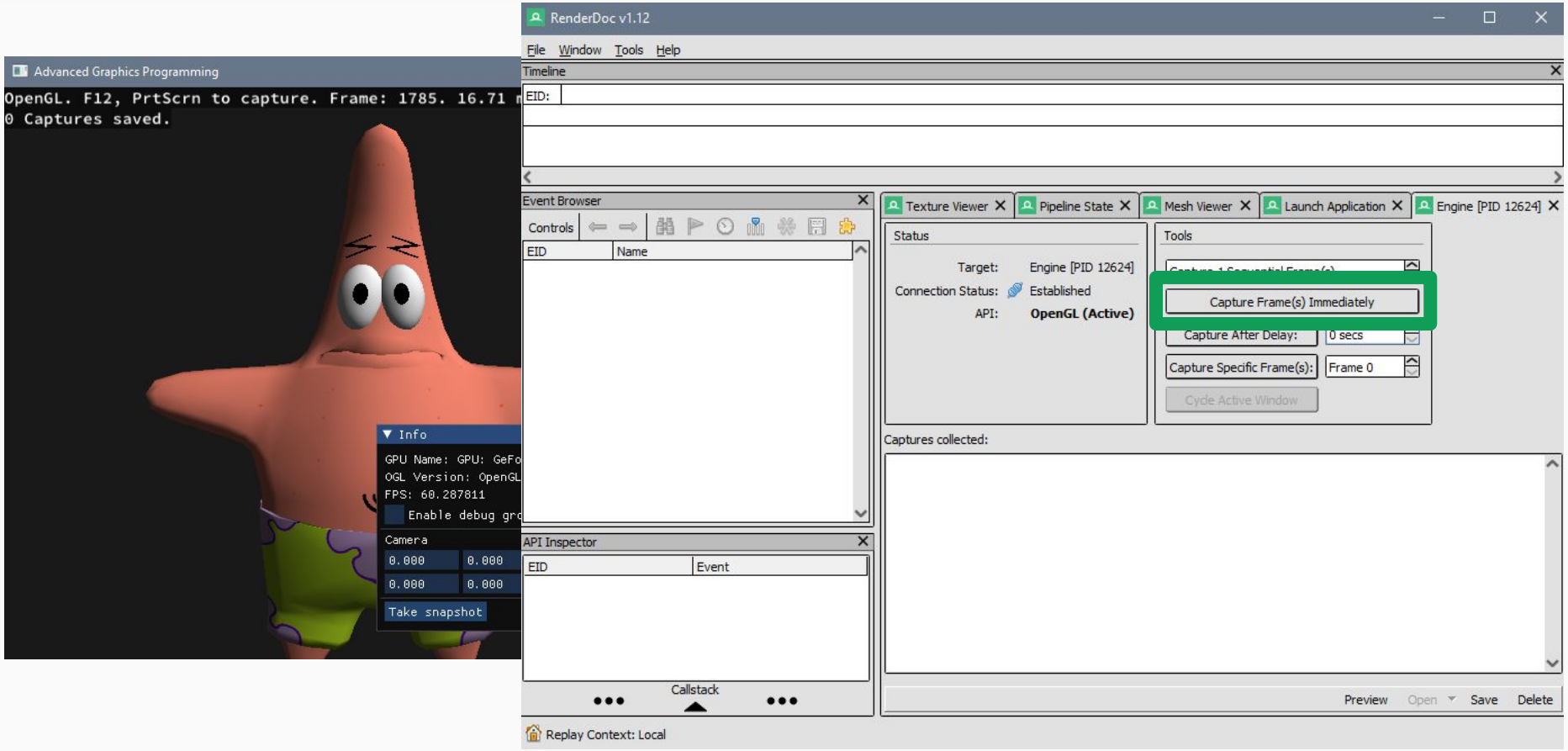
Probably Win x64 (if you are compiling your OpenGL application in for x64)

Launch your application

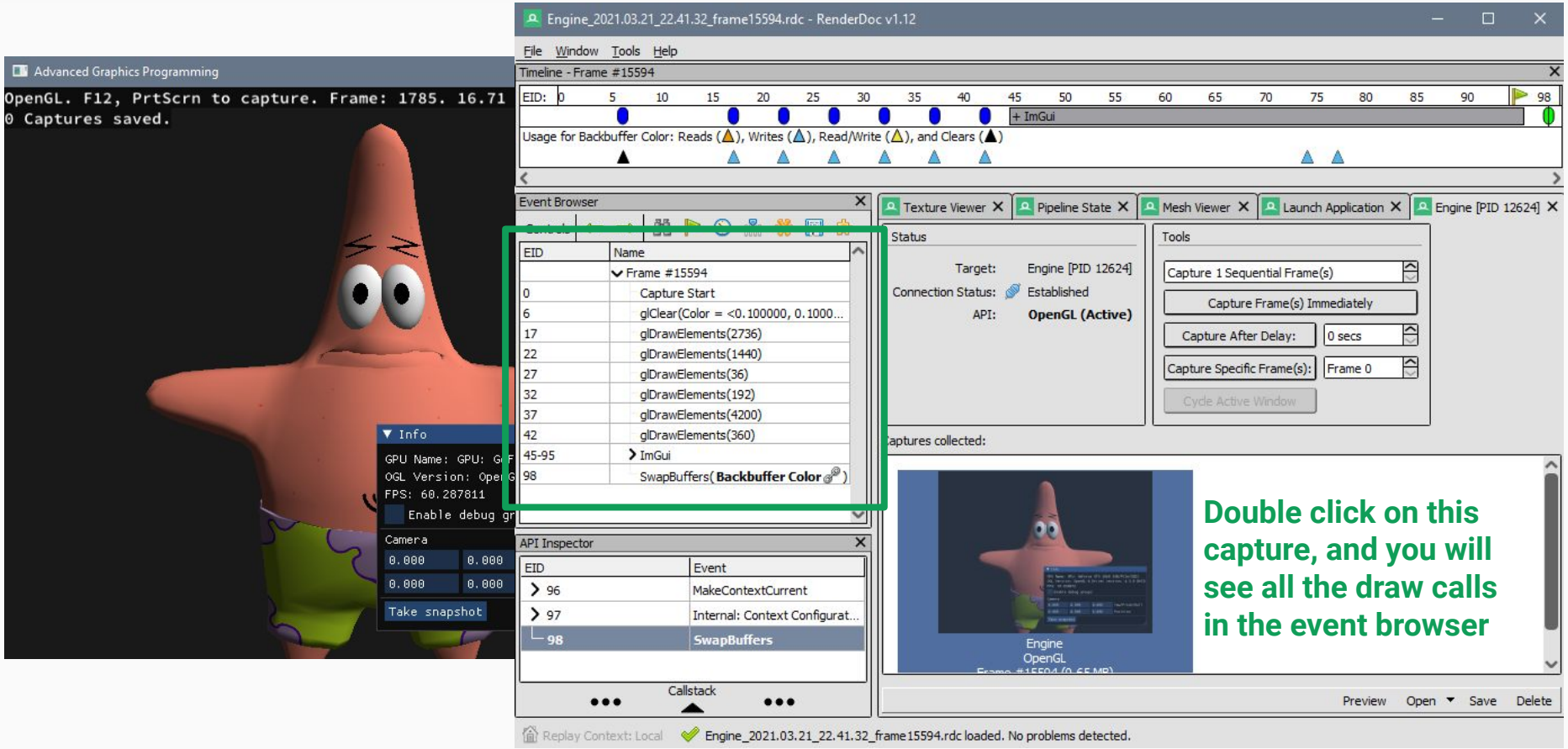


Zoomed view

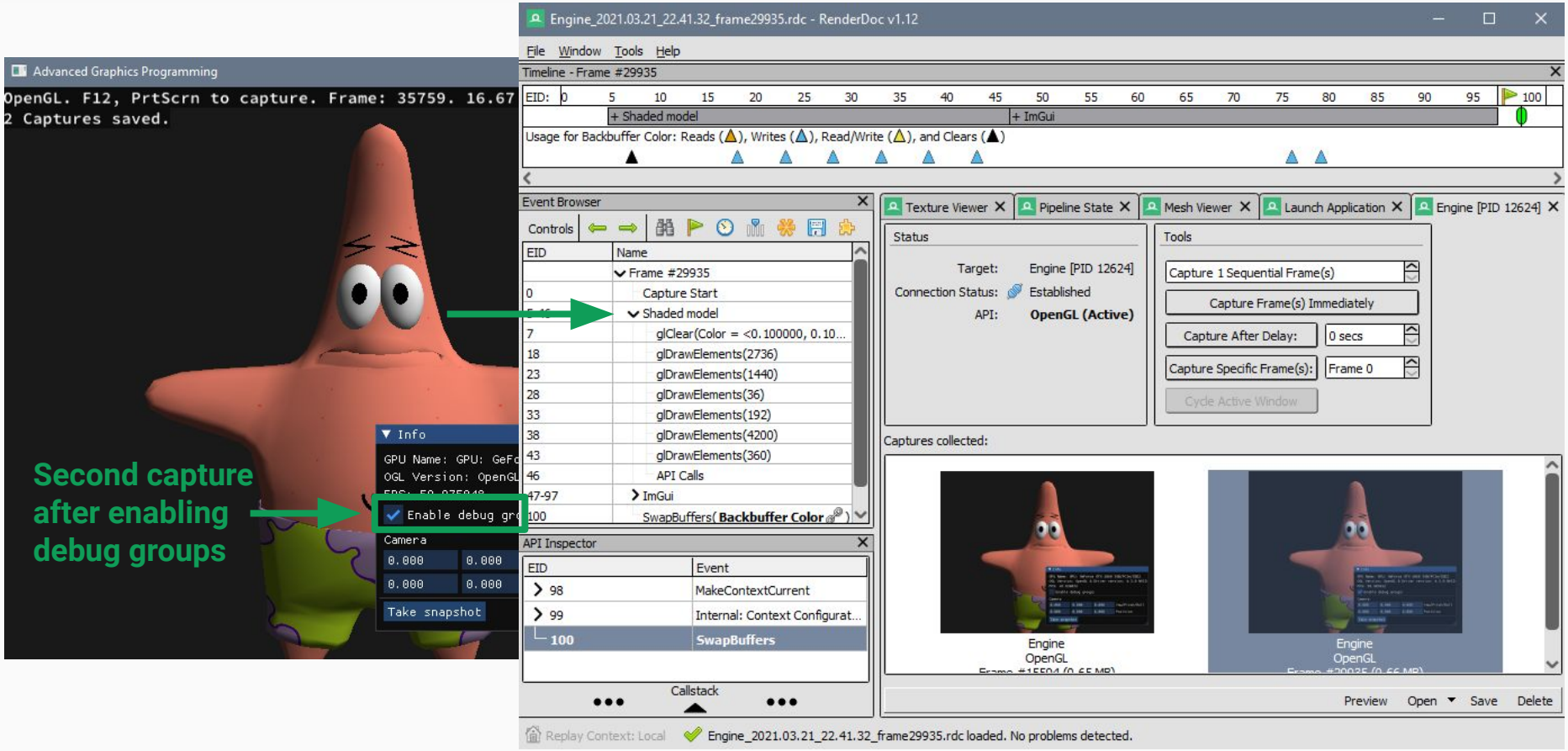
Capture a frame trace



You can see all the OpenGL draw calls



We can create draw call groups… how??



Thanks to glPushDebugGroup / glPopDebugGroup

They belong to GL_KHR_debug extension

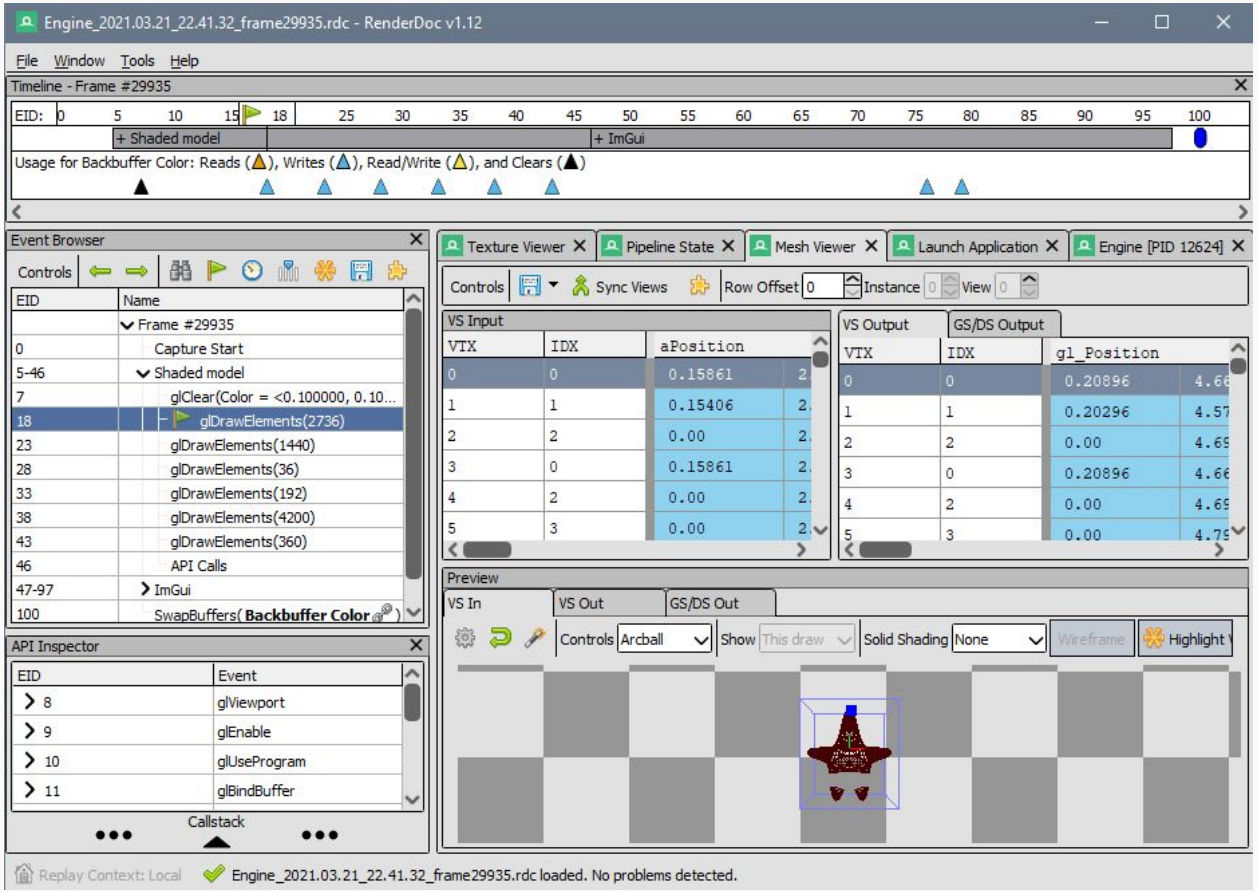
- You know, Core since OpenGL 4.3
- No need for the *enableDebugGroups* boolean (that was just to make a point)
- Between these calls one would change the OpenGL state and submit several draw calls
- Even several iterations of changing state -> draw calls

```
if (app->enableDebugGroups) glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 1, -1, "Shaded model");

// OpenGL rendering code
{ ... }

if (app->enableDebugGroups) glPopDebugGroup();
```

Texture Viewer

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

Mesh Viewer

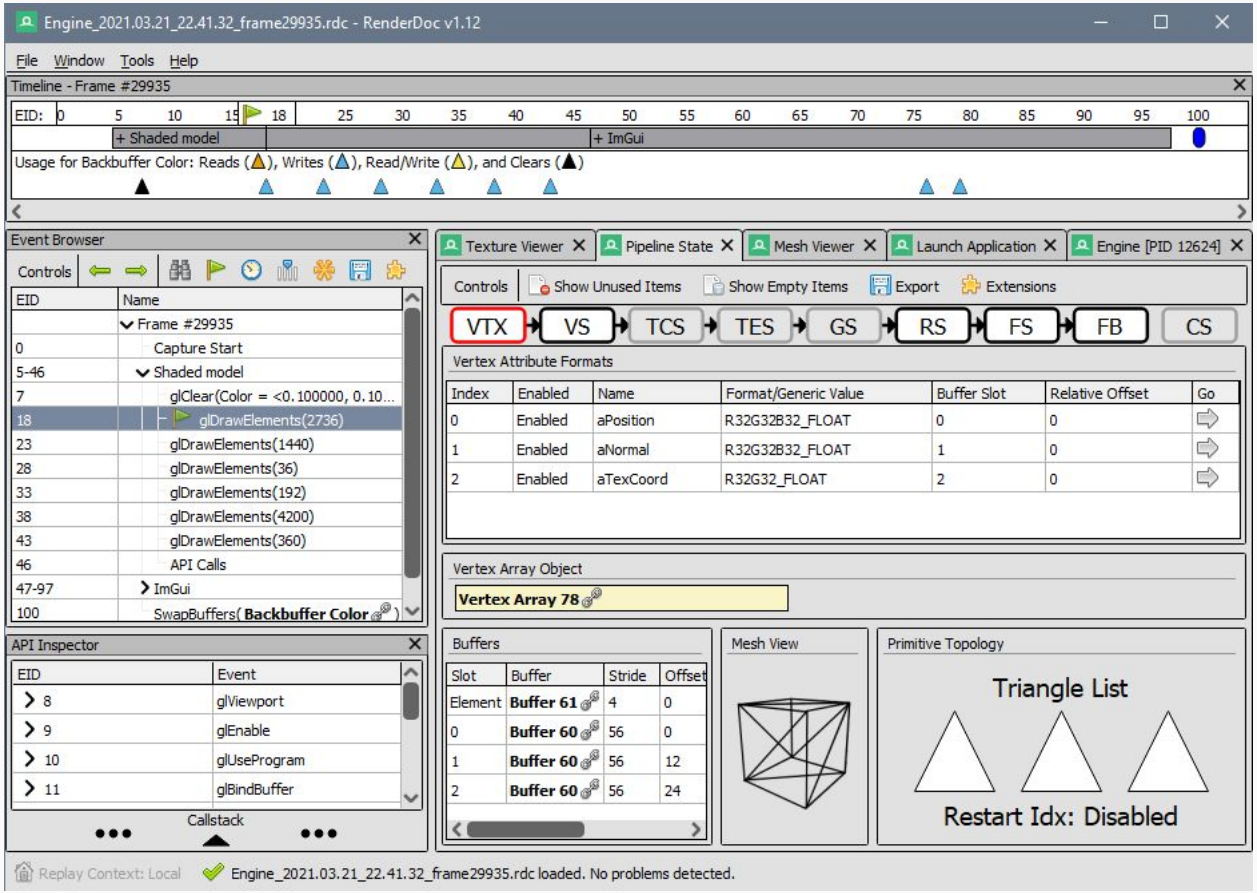


In the Mesh Viewer, we can also see (for a draw call) what the input and the output of the vertex shader is…

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



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
	Frame #29935
0	Capture Start
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

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	

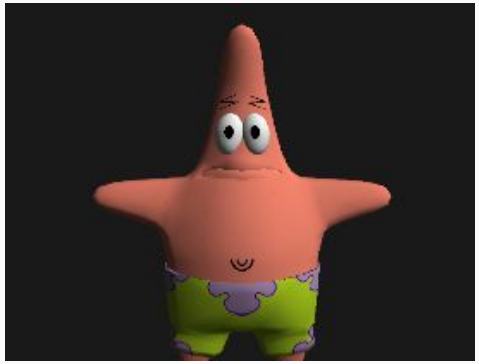
Samplers

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

Controls

EID	Name
	Frame #29935
0	Capture Start
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... | Pipeli... | Mesh ... | Launch... | Engine [... | * Editing Shader 69 - main....

Find | Refresh

main.glsl

```
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
206      oColor *= vec4(1.0, 0.0, 0.0, 1.0); // <-- I've added this
207  }
208
209  #endif
210  #endif
```

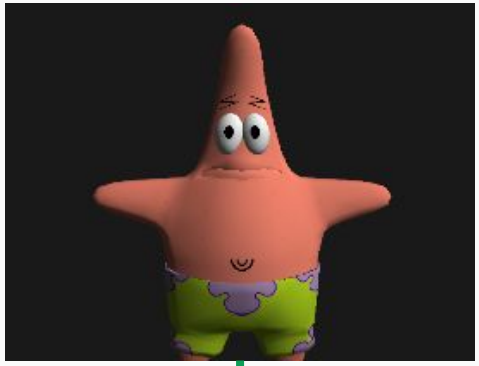
Compilation Settings

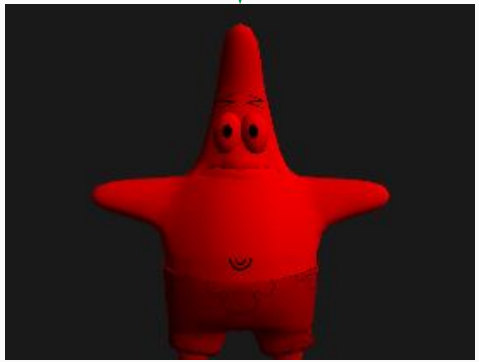
Entry Point: main Source Type: GLSL

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!!!