Inexor Vulkan Renderer

Release v0.1-alpha.3

Inexor Collective

Jul 11, 2023

CONTENTS

Inexor is a MIT-licensed open-source project which develops a new 3D octree game engine by combining modern C++ with Vulkan API.

CHAPTER

ONE

DOCUMENTATION

Quickstart: *Building Instructions* (*Windows*/*Linux*) & *Getting started*

1.1 Development

1.1.1 Supported platforms

- Vulkan API is completely platform-agnostic, which allows it to run on various operating systems.
- The required drivers for Vulkan are usually part of your graphic card's drivers.
- Update your graphics drivers as often as possible since new drivers with Vulkan updates are released frequently.
- Driver updates contain new features, bug fixes, and performance improvements.
- Check out Khronos website for more information.

Microsoft Windows

- We support x64 Microsoft Windows 8, 8.1 and 10.
- We have *build instructions for Windows*.

Linux

- We support every x64 Linux distribution for which Vulkan drivers exist.
- We have specific *build instructions for Gentoo, Ubuntu, Debian, and Arch.*.
- If you have found a way to set it up for other Linux distributions, please open a pull request and let us know!

macOS and iOS

- We do not support macOS or iOS because it would require us to use MoltenVK to get Vulkan running on Mac OS.
- Additionally, this would require some changes in the engines as not all of Inexor's dependencies are available on macOS or iOS.

Android

- We also do not support Android because this would require some changes in the engines as not all of Inexor's dependencies are available on Android.

1.1.2 Getting started

Also see the *building instructions* (*Windows*/*Linux*).

Required Software

Git Git for cloning (downloading) the source code.

Python with pip Required for generating the documentation and the C++ package manager.

CMake The build generator which generates project files for various IDEs.

Vulkan SDK Vulkan SDK contains the libraries and tools which are necessary to work with Vulkan API.

Update your Vulkan SDK as often as possible, because new versions will be released frequently which contains new features and bug fixes.

Make sure you add the `glslangValidator` in the Vulkan SDK's bin folder to your path variable.

Optional Software

GitKraken Git GUI. A Git user interface with many features which is easy to use.

GitHub Desktop An open source Git user interface which is easy to use.

Ninja Build System Improve your build times with ninja.

RenderDoc Powerful open source graphics debugger. Inexor has full RenderDoc integration.

Doxygen Required for generating the documentation.

Notepad++ Free and open source text editor.

Atom.io Free and open source text editor.

Visual Studio Code Free and open source text editor.

Does my graphics card support Vulkan?

- You can look up your graphics card in Sascha Willem's Vulkan hardware database.
- Every new graphics card which is coming out these days supports Vulkan API.
- Vulkan is also supported on older graphics cards going back to Radeon HD 7000 series and Nvidia Geforce 6 series.

Update your graphics drivers!

- Update your graphics drivers as often as possible.
- New drivers contain new features, bug fixes, and performance improvements.

1.1.3 Building vulkan-renderer

- *Windows*
- *Linux*

Also see *Getting started*.

If you have any trouble please open a ticket or join our Discord server.

This project uses out of source builds using either gcc, clang or Microsoft Visual Studio compiler.

Generating the documentation will create two subfolders in `doc` which will be ignored by git.

The following CMake targets and options are available:

Table 1: List of CMake build targets.

build target	description	comment
inexor-vulkan-renderer	The main executable.	
inexor-vulkan-renderer-tests	Tests the renderer using Google Test.	There are no tests available yet.
inexor-vulkan-renderer-benchmark	Benchmarking of the renderer using Google Benchmark.	There are no benchmarks available yet.
inexor-vulkan-renderer-documentation	Builds the documentation with Sphinx. Enable target creation with `-DINEXOR_BUILD_DOC=ON`.	
inexor-vulkan-renderer-documentation-linkcheck	Use sphinx's linkcheck feature to search for broken links.	

Table 2: List of CMake options.

option	description	default value
INEXOR_BUILD_EXAMPLE	Builds inexor-renderer-example.	`ON`
INEXOR_BUILD_TESTS	Builds inexor-renderer-tests.	`OFF`
INEXOR_BUILD_BENCHMARKS	Builds inexor-renderer-benchmarks.	`OFF`
INEXOR_BUILD_DOC	Builds the documentation with Sphinx.	`OFF`
INEXOR_BUILD_DOCUMENTATION_USE_VENV	Generate and use a Python virtual environment for the documentation dependencies.	`ON`

Windows

Example: Create Visual Studio 2022 project map for Debug mode including docs, tests, and benchmarks:

```
cmake -G "Visual Studio 17 2022" -A x64 -B./cmake-build-debug-vs/ -DCMAKE_BUILD_
↪TYPE=Debug -DINEXOR_BUILD_DOC=ON -DINEXOR_BUILD_TESTS=ON -DINEXOR_BUILD_BENCHMARKS=ON .
↪/
```

Example: Create Visual Studio 2022 project map for Release mode but without docs, tests, and benchmarks:

```
cmake -G "Visual Studio 17 2022" -A x64 -B./cmake-build-release-vs/ -DCMAKE_BUILD_
↪TYPE=Release ./
```

If you have Ninja build system installed, you can use it like this:

```
# executing from project root assumed
# Ninja generator and Debug type
\> cmake -G Ninja -B./cmake-build-debug/ -DCMAKE_BUILD_TYPE=Debug ./
# Ninja generator and Release type
\> cmake -G Ninja -B./cmake-build-release/ -DCMAKE_BUILD_TYPE=Release ./
# Create Visual Studio Solution
\> cmake -G "Visual Studio 17 2022" -A x64 -B./cmake-build-debug-vs/ -DCMAKE_BUILD_
↪TYPE=Debug ./
# Build all targets
\> cmake --build ./cmake-build-debug/
```

- Choose any IDE that CMake can generate a project map for. If in doubt use Visual Studio 2022.
- Clone the source code. Free and good tools are GitHub Desktop or GitKraken Git GUI.
- Open CMake and select the root folder which contains `CMakeLists.txt` (not just `src` folder!).
- You can choose any location for the `build` folder.
- Click “Configure” and select your IDE (in doubt `Visual Studio 17 2022`). Click “Finish”.
- CMake will now set up dependencies automatically for you. This might take a while. If this fails, you really should open a ticket!
- Click “Generate”. You can now open the Visual Studio project file in your `build` folder.
- For debugging, please check that the root directory of the repository is set as working directory in Visual Studio. Usually, CMake should take care of this already.
- You are now ready to start debugging! Our main branch must be stable at all cost.

Linux

Install dependencies and tools:

Ubuntu	Follow the Install the SDK-instructions on the vulkan-sdk page. Install the required packages: `# apt install -y \ pkg-config \ libglm-dev \ libxcb-dri3-0 \ libxcb-present0 \ libpciaccess0 \ libpng-dev \ libxcb-keysyms1-dev \ libxcb-dri3-dev \ libx11-dev  \ libmirclient-dev \ libwayland-dev \ libxrandr-dev \ libxcb-ewmh-dev` `# apt install -y \ cmake \ ninja-build \ clang-tidy \ vulkan-sdk \ python3 \ python3-pip` `$ pip3 install \ wheel \ setuptools \`
Gentoo	`# emerge \ dev-util/cmake \ dev-util/vulkan-headers \ dev-util/vulkan-tools \ dev-vcs/git \ media-libs/vulkan-layers \ media-libs/vulkan-loader` Install ninja build tool (optional): `# emerge dev-util/ninja`
Debian	Follow the Install the SDK-instructions on the vulkan-sdk page. Install the required packages: `# apt install -y \ libvulkan-dev \ glslang-dev \ glslang-tools \ vulkan-tools \ vulkan-validationlayers-dev \ spirv-tools \ pkg-config \ libglm-dev \ libxcb-dri3-0 \ libxcb-present0 \ libpciaccess0 \ libpng-dev \ libxcb-keysyms1-dev \ libxcb-dri3-dev \ libx11-dev \`

Clone the repository:

```
$ git clone https://github.com/inexorgame/vulkan-renderer
$ cd vulkan-renderer
```

Configure cmake:

Note: Only pass `-GNinja` if the ninja build tool is installed.

```
$ cmake . \
  -Bbuild \
  -DCMAKE_BUILD_TYPE=Debug \
  -GNinja
```

Build and run:

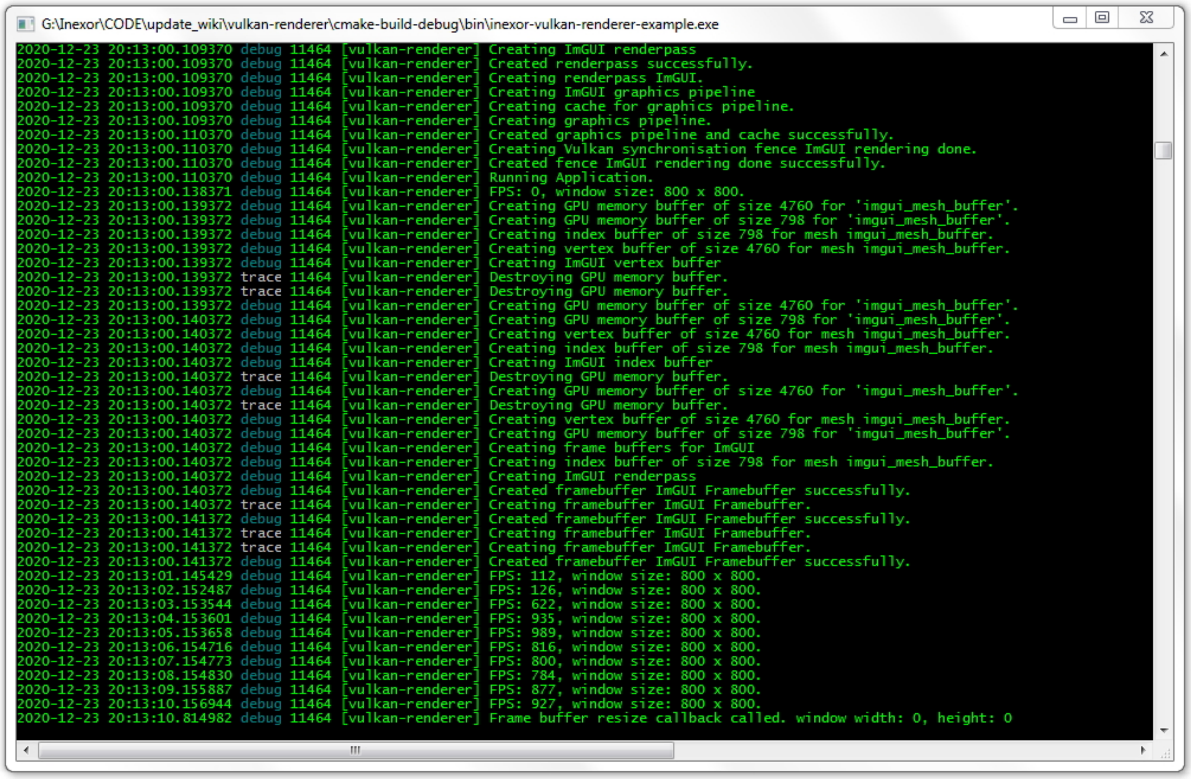
If you have any trouble please open a ticket or join our Discord server.

```
$ cmake --build build --target inexor-vulkan-renderer-example
$ ./build/bin/inexor-vulkan-renderer-example
```

1.1.4 Debugging

Logfiles

Inexor uses spdlog for both console logging and logfiles.



G:\Inexor\CODE\update_wiki\vulkan-renderer\cmake-build-debug\bin\inexor-vulkan-renderer-example.exe

```
2020-12-23 20:13:00.109370 debug 11464 [vulkan-renderer] Creating ImGUI renderpass
2020-12-23 20:13:00.109370 debug 11464 [vulkan-renderer] Created renderpass successfully.
2020-12-23 20:13:00.109370 debug 11464 [vulkan-renderer] Creating renderpass ImGUI.
2020-12-23 20:13:00.109370 debug 11464 [vulkan-renderer] Creating ImGUI graphics pipeline
2020-12-23 20:13:00.109370 debug 11464 [vulkan-renderer] Creating cache for graphics pipeline.
2020-12-23 20:13:00.109370 debug 11464 [vulkan-renderer] Creating graphics pipeline.
2020-12-23 20:13:00.110370 debug 11464 [vulkan-renderer] Created graphics pipeline and cache successfully.
2020-12-23 20:13:00.110370 debug 11464 [vulkan-renderer] Creating Vulkan synchronisation fence ImGUI rendering done.
2020-12-23 20:13:00.110370 debug 11464 [vulkan-renderer] Created fence ImGUI rendering done successfully.
2020-12-23 20:13:00.110370 debug 11464 [vulkan-renderer] Running Application.
2020-12-23 20:13:00.138371 debug 11464 [vulkan-renderer] FPS: 0, window size: 800 x 800.
2020-12-23 20:13:00.139372 debug 11464 [vulkan-renderer] Creating GPU memory buffer of size 4760 for 'imgui_mesh_buffer'.
2020-12-23 20:13:00.139372 debug 11464 [vulkan-renderer] Creating GPU memory buffer of size 798 for 'imgui_mesh_buffer'.
2020-12-23 20:13:00.139372 debug 11464 [vulkan-renderer] Creating index buffer of size 798 for mesh imgui_mesh_buffer.
2020-12-23 20:13:00.139372 debug 11464 [vulkan-renderer] Creating vertex buffer of size 4760 for mesh imgui_mesh_buffer.
2020-12-23 20:13:00.139372 debug 11464 [vulkan-renderer] Creating ImGUI vertex buffer
2020-12-23 20:13:00.139372 trace 11464 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-23 20:13:00.139372 trace 11464 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-23 20:13:00.139372 debug 11464 [vulkan-renderer] Creating GPU memory buffer of size 4760 for 'imgui_mesh_buffer'.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating GPU memory buffer of size 798 for 'imgui_mesh_buffer'.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating vertex buffer of size 4760 for mesh imgui_mesh_buffer.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating index buffer of size 798 for mesh imgui_mesh_buffer.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating ImGUI index buffer
2020-12-23 20:13:00.140372 trace 11464 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating GPU memory buffer of size 4760 for 'imgui_mesh_buffer'.
2020-12-23 20:13:00.140372 trace 11464 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating vertex buffer of size 4760 for mesh imgui_mesh_buffer.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating GPU memory buffer of size 798 for 'imgui_mesh_buffer'.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating frame buffers for ImGUI
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating index buffer of size 798 for mesh imgui_mesh_buffer.
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Creating ImGUI renderpass
2020-12-23 20:13:00.140372 debug 11464 [vulkan-renderer] Created framebuffer ImGUI Framebuffer successfully.
2020-12-23 20:13:00.140372 trace 11464 [vulkan-renderer] Creating framebuffer ImGUI Framebuffer.
2020-12-23 20:13:00.141372 debug 11464 [vulkan-renderer] Created framebuffer ImGUI Framebuffer successfully.
2020-12-23 20:13:00.141372 trace 11464 [vulkan-renderer] Creating framebuffer ImGUI Framebuffer.
2020-12-23 20:13:00.141372 trace 11464 [vulkan-renderer] Creating framebuffer ImGUI Framebuffer.
2020-12-23 20:13:00.141372 debug 11464 [vulkan-renderer] Created framebuffer ImGUI Framebuffer successfully.
2020-12-23 20:13:01.145429 debug 11464 [vulkan-renderer] FPS: 112, window size: 800 x 800.
2020-12-23 20:13:02.152487 debug 11464 [vulkan-renderer] FPS: 126, window size: 800 x 800.
2020-12-23 20:13:03.153544 debug 11464 [vulkan-renderer] FPS: 622, window size: 800 x 800.
2020-12-23 20:13:04.153601 debug 11464 [vulkan-renderer] FPS: 935, window size: 800 x 800.
2020-12-23 20:13:05.153658 debug 11464 [vulkan-renderer] FPS: 989, window size: 800 x 800.
2020-12-23 20:13:06.154716 debug 11464 [vulkan-renderer] FPS: 816, window size: 800 x 800.
2020-12-23 20:13:07.154773 debug 11464 [vulkan-renderer] FPS: 800, window size: 800 x 800.
2020-12-23 20:13:08.154830 debug 11464 [vulkan-renderer] FPS: 784, window size: 800 x 800.
2020-12-23 20:13:09.155887 debug 11464 [vulkan-renderer] FPS: 877, window size: 800 x 800.
2020-12-23 20:13:10.156944 debug 11464 [vulkan-renderer] FPS: 927, window size: 800 x 800.
2020-12-23 20:13:10.814982 debug 11464 [vulkan-renderer] Frame buffer resize callback called. window width: 0, height: 0
```

The log output which can be seen in the console will also be written to `vulkan-renderer.log` in the root directory.

You can open and read this logfile with a text editor of your choice.

We are using the following log entry format `%Y-%m-%d %T.%f %^%l%$ %5t [%-10n] %v`.

- `%Y` is the year.
- `%m` is the month (01 to 12).
- `%d` is the day of month (01 to 31).
- `%T` is ISO 8601 time format (HH:MM:SS).
- `%f` is the microsecond part of the current second.
- `%^%l%$` is the color-coded log level.
- `%5t` is the thread id formatted to a string of length 5.
- `[%-10n]` is the name of the logger, limited to 10 characters.
- `%v` is the log message.

For more information, check out spdlog's documentation about custom formatting.

Use the following rules for logging:

- Don't use `std::cout` or `printf` or similar. Just use spdlog instead.
- Place as many log messages to your code as possible.
- End log messages with a `.` to show that the message is finished.
- Use all log levels as you need it: `spdlog::trace`, `spdlog::info`, `spdlog::debug`, `spdlog::error`, `spdlog::critical`.
- You can print variables with spdlog (see this reference) because it is based on fmt library.
- Use direct API calls like `spdlog::debug("Example text here");` instead of creating your own logger name for now. We will come up with a strategy for logger hierarchy later.

Command Line Arguments

You can start vulkan-renderer with the following command line arguments:

`--gpu <index>`

Specifies which GPU to use by array index, **starting from 0**.

Note: The engine checks if this index is valid. If the index is invalid, automatic GPU selection rules apply.

`--no-separate-data-queue`

Disables the use of the special data transfer queue (forces use of the graphics queue).

Warning: Enabling this option could decrease the overall performance. Don't enable this option unless you have to.

`--no-validation`

Disables Vulkan validation layers.

Warning: You should never disable validation layers because they offer extensive error checks for debugging.

`--no-vk-debug-markers`

Disables Vulkan debug markers (even if `--renderdoc` is specified).

`--renderdoc`

Enables the RenderDoc debug layer.

`--vsync`

Warning: Vsync is currently not implemented. The command line argument will be ignored.

Enables vertical synchronization (limits FPS to monitor refresh rate).

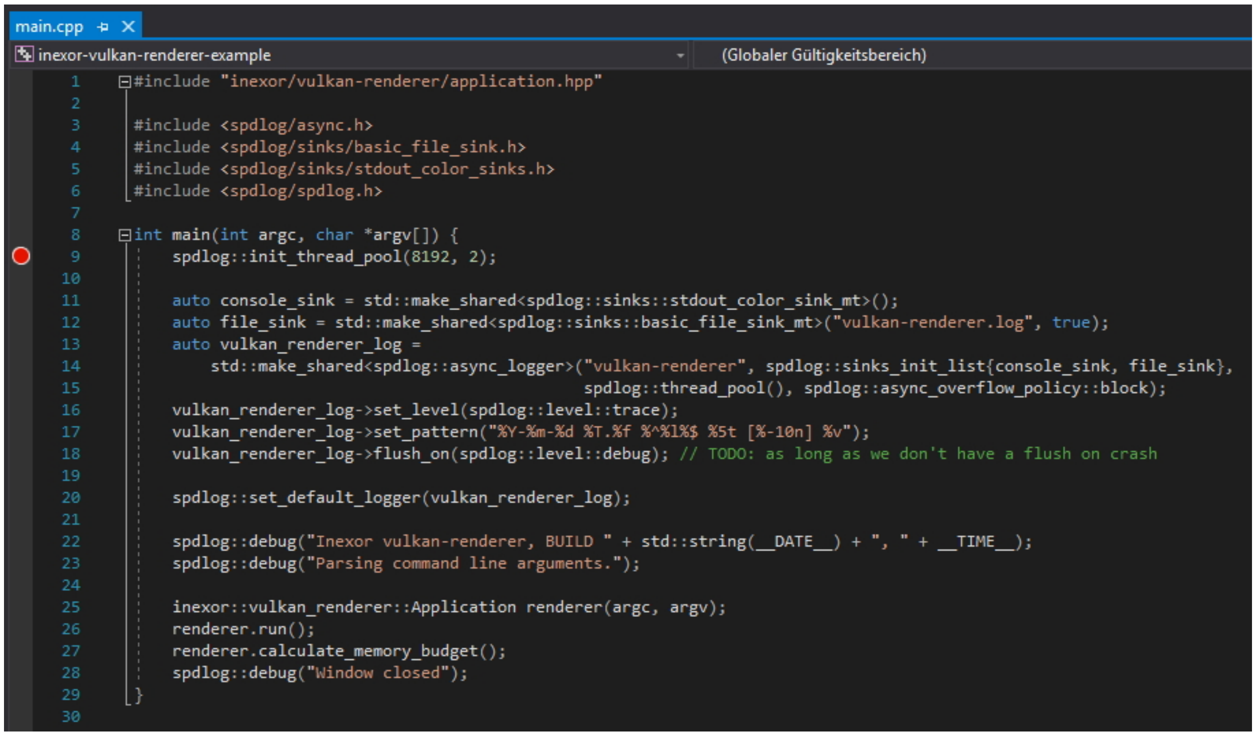
RenderDoc

- RenderDoc is a free and open source graphics debugger for Vulkan API (and other APIs) developed by Baldur Karlsson.
- It is a very powerful graphics debugging and visualization tool which makes debugging Vulkan application as easy as possible.
- Inexor has full RenderDoc integration. This includes internal resource naming using Vulkan debug markers.
- The following tutorial shows how to debug Inexor using RenderDoc.
- You can read up more details in RenderDoc's documentation.

RenderDoc Tutorial for Windows

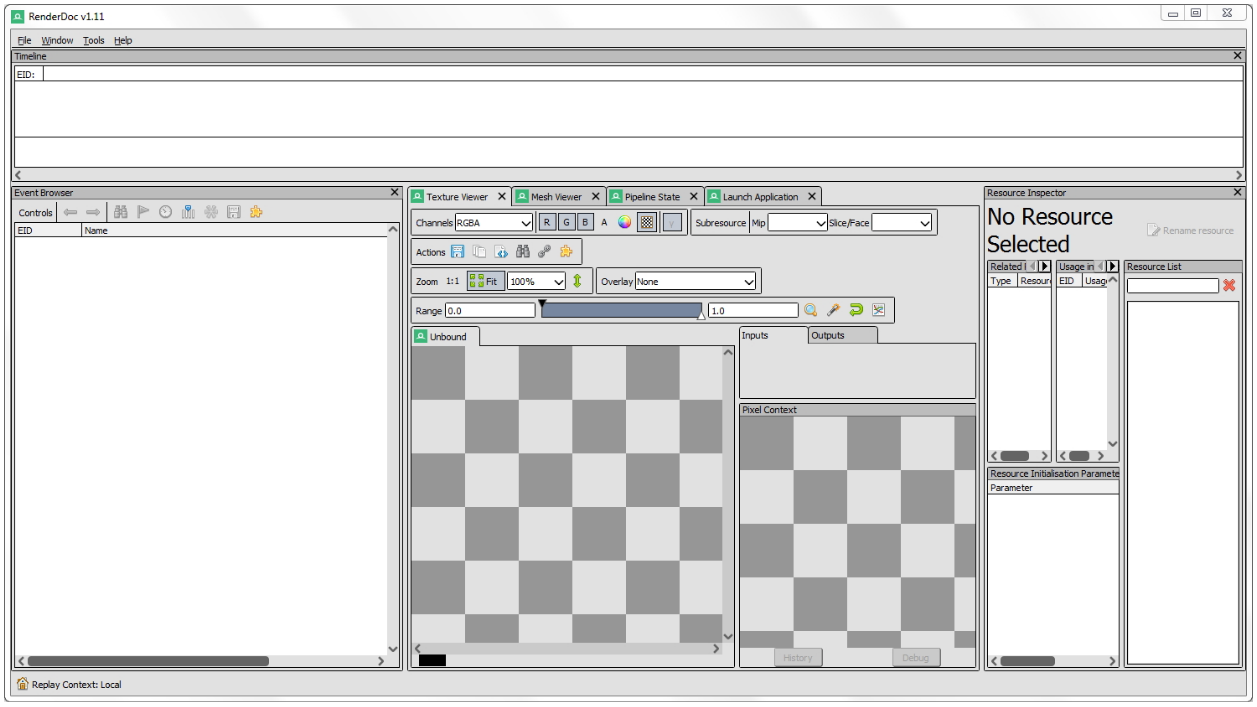
Step 1: Open Inexor in Visual Studio and add a breakpoint before Vulkan initialization

- The best spot would be right after `main()`:

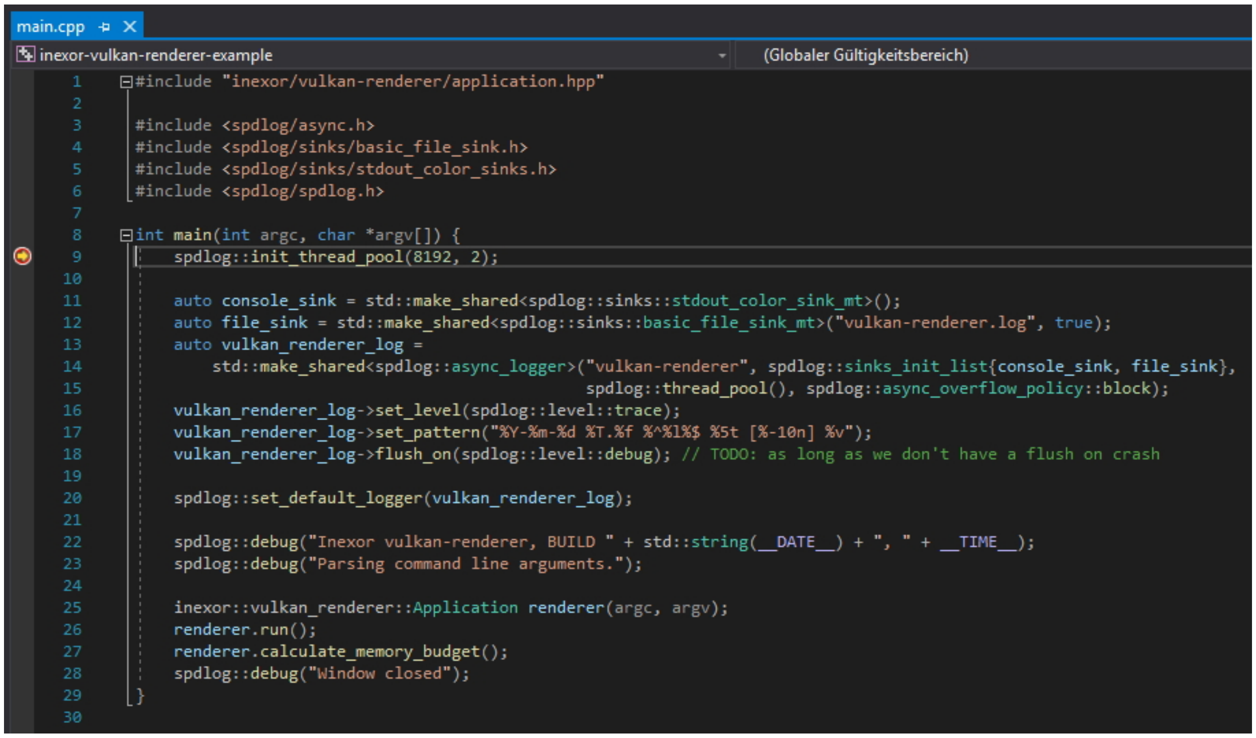


```cpp
#include "inexor/vulkan-renderer/application.hpp"

#include <spdlog/async.h>
#include <spdlog/sinks/basic_file_sink.h>
#include <spdlog/sinks/stdout_color_sinks.h>
#include <spdlog/spdlog.h>

int main(int argc, char *argv[]) {
    spdlog::init_thread_pool(8192, 2);

    auto console_sink = std::make_shared<spdlog::sinks::stdout_color_sink_mt>();
    auto file_sink = std::make_shared<spdlog::sinks::basic_file_sink_mt>("vulkan-renderer.log", true);
    auto vulkan_renderer_log =
        std::make_shared<spdlog::async_logger>("vulkan-renderer", spdlog::sinks_init_list{console_sink, file_sink},
                                               spdlog::thread_pool(), spdlog::async_overflow_policy::block);
    vulkan_renderer_log->set_level(spdlog::level::trace);
    vulkan_renderer_log->set_pattern("%Y-%m-%d %T.%f %^%l%$ %5t [%-10n] %v");
    vulkan_renderer_log->flush_on(spdlog::level::debug); // TODO: as long as we don't have a flush on crash

    spdlog::set_default_logger(vulkan_renderer_log);

    spdlog::debug("Inexor vulkan-renderer, BUILD " + std::string(__DATE__) + ", " + __TIME__);
    spdlog::debug("Parsing command line arguments.");

    inexor::vulkan_renderer::Application renderer(argc, argv);
    renderer.run();
    renderer.calculate_memory_budget();
    spdlog::debug("Window closed");
}
```

Step 2: Open RenderDoc.

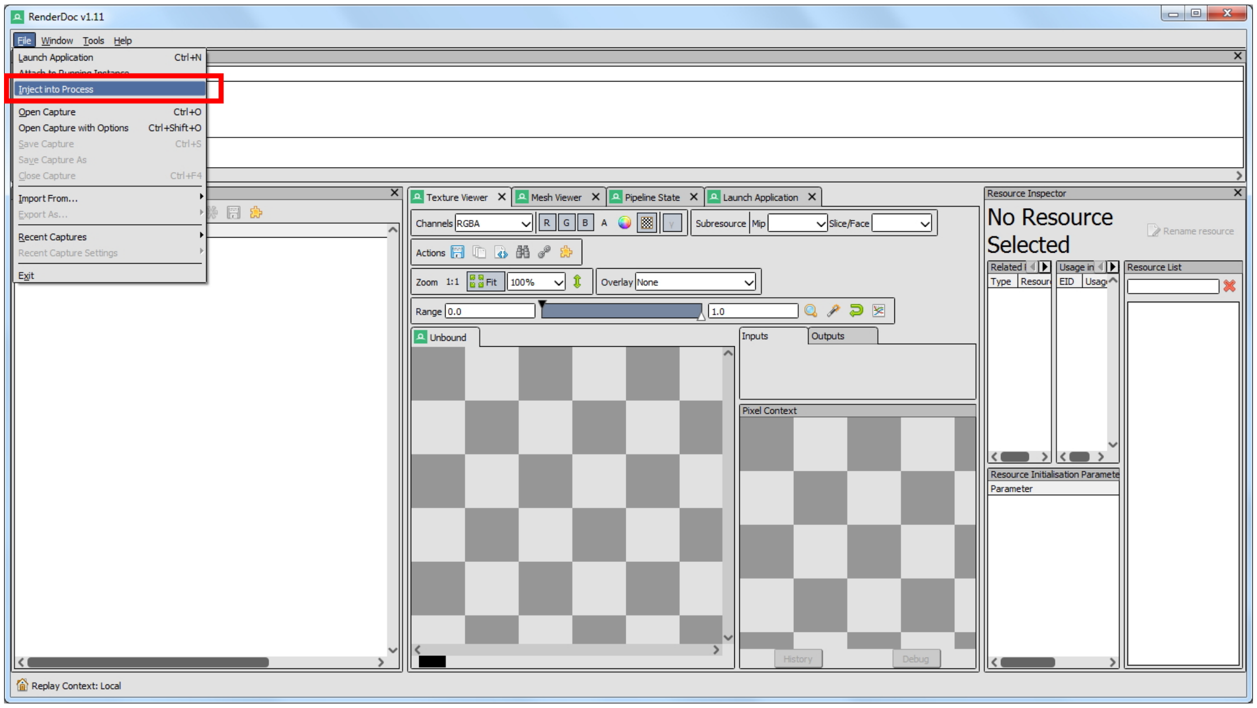


Step 3: Start debugging inexor-vulkan-renderer and halt at the breakpoint



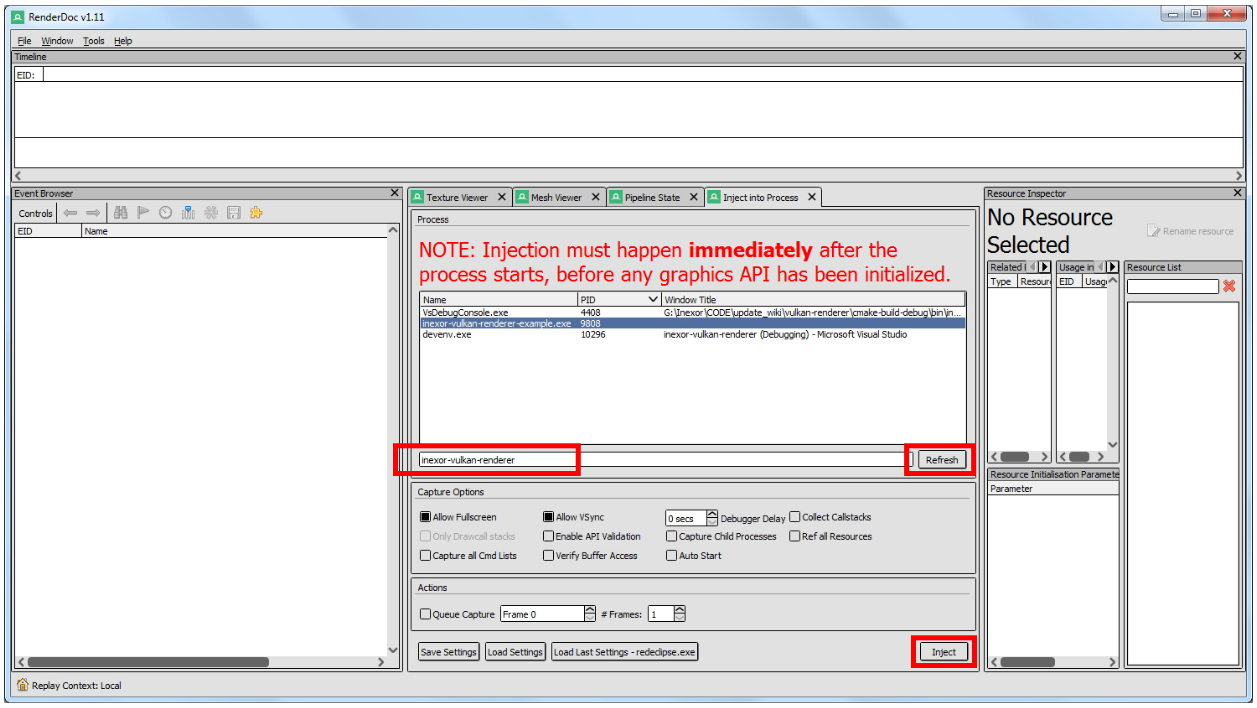
```cpp
#include "inexor/vulkan-renderer/application.hpp"

#include <spdlog/async.h>
#include <spdlog/sinks/basic_file_sink.h>
#include <spdlog/sinks/stdout_color_sinks.h>
#include <spdlog/spdlog.h>

int main(int argc, char *argv[]) {
    spdlog::init_thread_pool(8192, 2);

    auto console_sink = std::make_shared<spdlog::sinks::stdout_color_sink_mt>();
    auto file_sink = std::make_shared<spdlog::sinks::basic_file_sink_mt>("vulkan-renderer.log", true);
    auto vulkan_renderer_log =
        std::make_shared<spdlog::async_logger>("vulkan-renderer", spdlog::sinks_init_list{console_sink, file_sink},
                                               spdlog::thread_pool(), spdlog::async_overflow_policy::block);
    vulkan_renderer_log->set_level(spdlog::level::trace);
    vulkan_renderer_log->set_pattern("%Y-%m-%d %T.%f %^%l%$ %5t [%-10n] %v");
    vulkan_renderer_log->flush_on(spdlog::level::debug); // TODO: as long as we don't have a flush on crash

    spdlog::set_default_logger(vulkan_renderer_log);

    spdlog::debug("Inexor vulkan-renderer, BUILD " + std::string(__DATE__) + ", " + __TIME__);
    spdlog::debug("Parsing command line arguments.");

    inexor::vulkan_renderer::Application renderer(argc, argv);
    renderer.run();
    renderer.calculate_memory_budget();
    spdlog::debug("Window closed");
}
```

Step 4: "Inject into process" inexor-vulkan-renderer.exe using RenderDoc



Step 5: Search for "inexor-vulkan-renderer.exe" and click "inject"

- You will see a warning Windows Firewall the first time you do this.
- This is because RenderDoc is reading memory from inexor-vulkan-renderer.
- Accept the Windows Firewall warning to allow RenderDoc to read memory.



NOTE: Injection must happen **immediately** after the process starts, before any graphics API has been initialized.

Name	PID	Window Title
VsDebugConsole.exe	4408	G:\Inexor\CODE\update_wiki\vulkan-renderer\cmake-build-debug\bin\in...
inexor-vulkan-renderer-example.exe	9808	
devenv.exe	10296	inexor-vulkan-renderer (Debugging) - Microsoft Visual Studio

inexor-vulkan-renderer

Capture Options

[✓] Allow Fullscreen; [✓] Allow VSync; Debugger Delay: 0 secs; [ ] Collect Callstacks; [ ] Only Drawcall stacks; [ ] Enable API Validation; [ ] Capture Child Processes; [ ] Ref all Resources; [ ] Capture all Cmd Lists; [ ] Verify Buffer Access; [ ] Auto Start

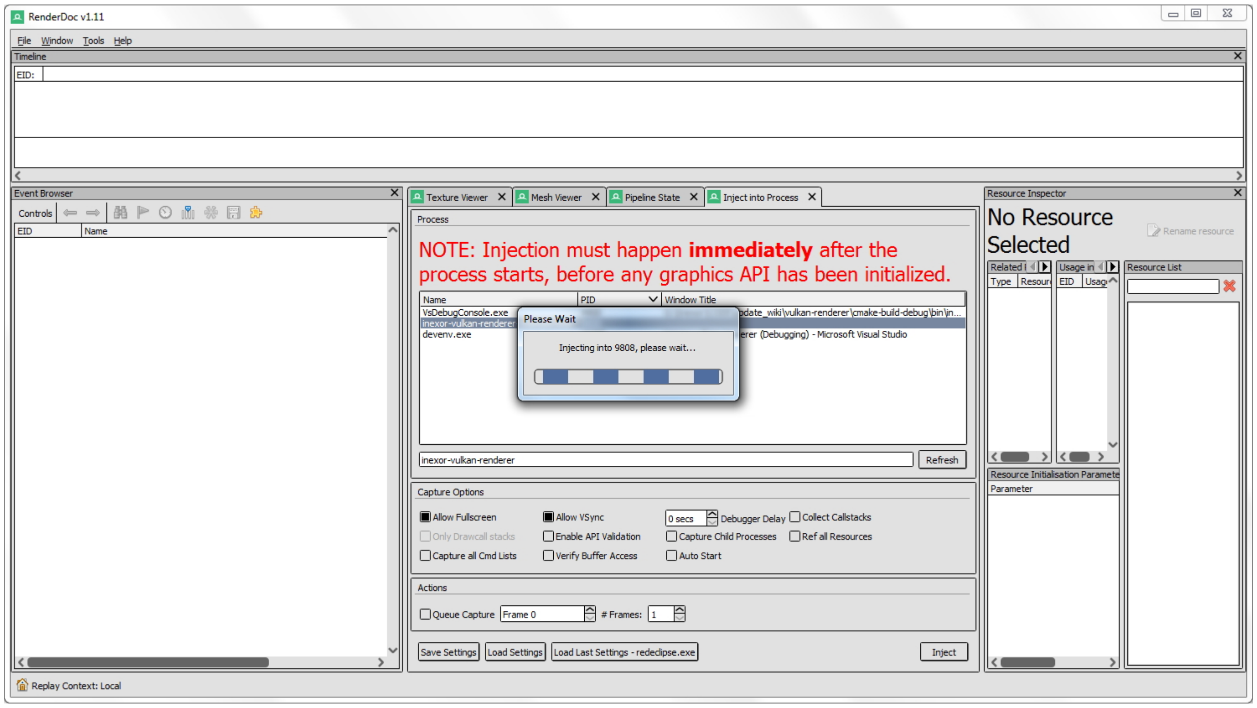
Actions

[ ] Queue Capture: Frame 0; # Frames: 1

Save Settings | Load Settings | Load Last Settings - redeclipse.exe

Step 6: Continue debugging in Visual Studio

- RenderDoc should now look like this.



NOTE: Injection must happen **immediately** after the process starts, before any graphics API has been initialized.

Name	PID	Window Title
VsDebugConsole.exe		...pdate_wiki\vulkan-renderer\cmake-build-debug\bin\in...
inexor-vulkan-renderer		
devenv.exe		...erer (Debugging) - Microsoft Visual Studio

Please Wait

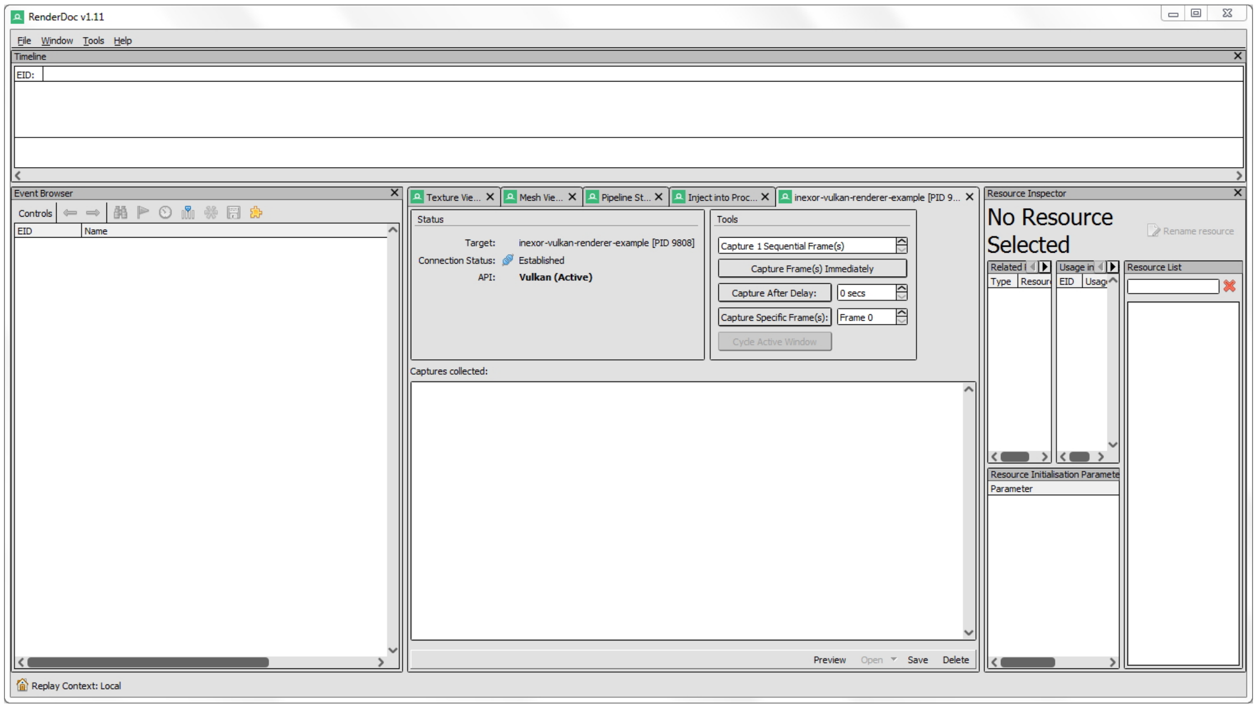
Injecting into 9808, please wait...

inexor-vulkan-renderer

Capture Options:
- [✓] Allow Fullscreen
- [ ] Only Drawcall stacks
- [ ] Capture all Cmd Lists
- [✓] Allow VSync
- [ ] Enable API Validation
- [ ] Verify Buffer Access
- 0 secs Debugger Delay
- [ ] Capture Child Processes
- [ ] Auto Start
- [ ] Collect Callstacks
- [ ] Ref all Resources

Actions: [ ] Queue Capture Frame 0 # Frames: 1

- Press F5 to continue program execution from the breakpoint.
- RenderDoc is now connected to inexor-vulkan-renderer:



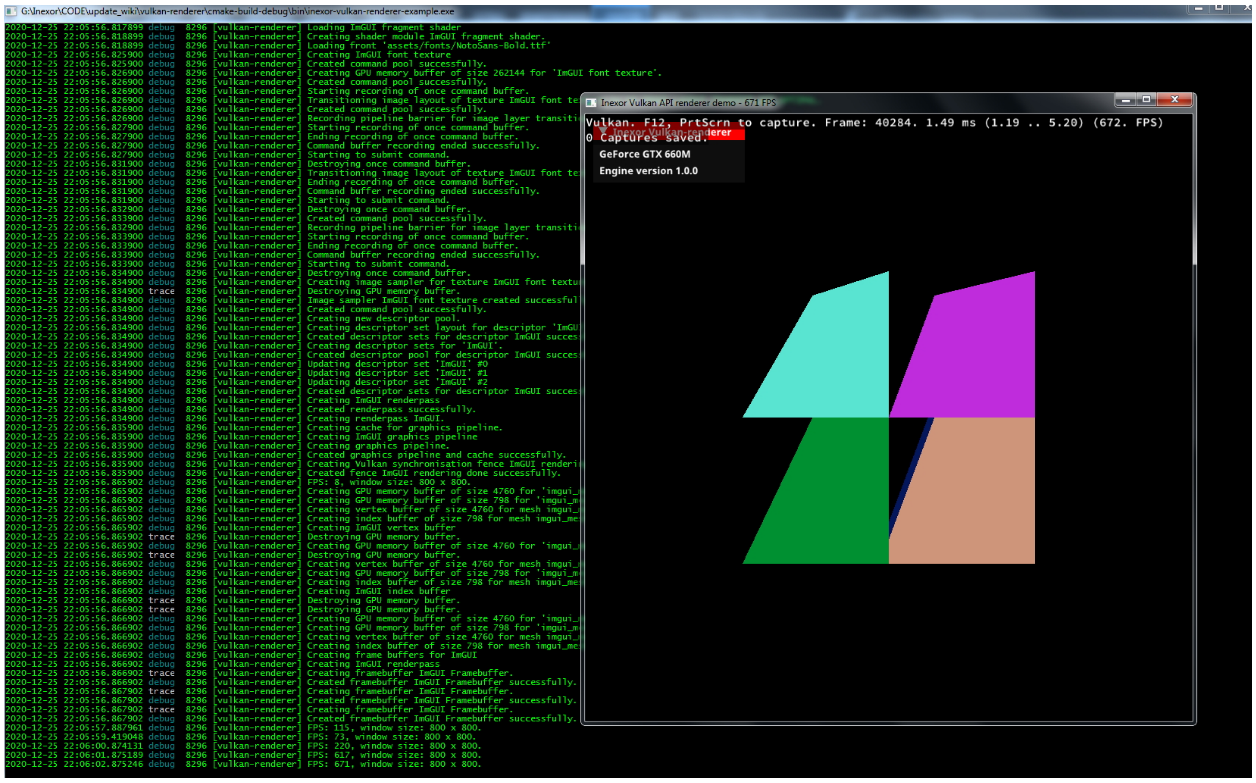
Status

Target: inexor-vulkan-renderer-example [PID 9808]
Connection Status: Established
API: **Vulkan (Active)**

Capture 1 Sequential Frame(s)
Capture After Delay: 0 secs
Capture Specific Frame(s): Frame 0

Captures collected:

- You can see RenderDoc's overlay in inexor-vulkan-renderer.exe:


G:\Inexor\CODE\update_wiki\vulkan-renderer\cmake-build-debug\bin\inexor-vulkan-renderer-example.exe

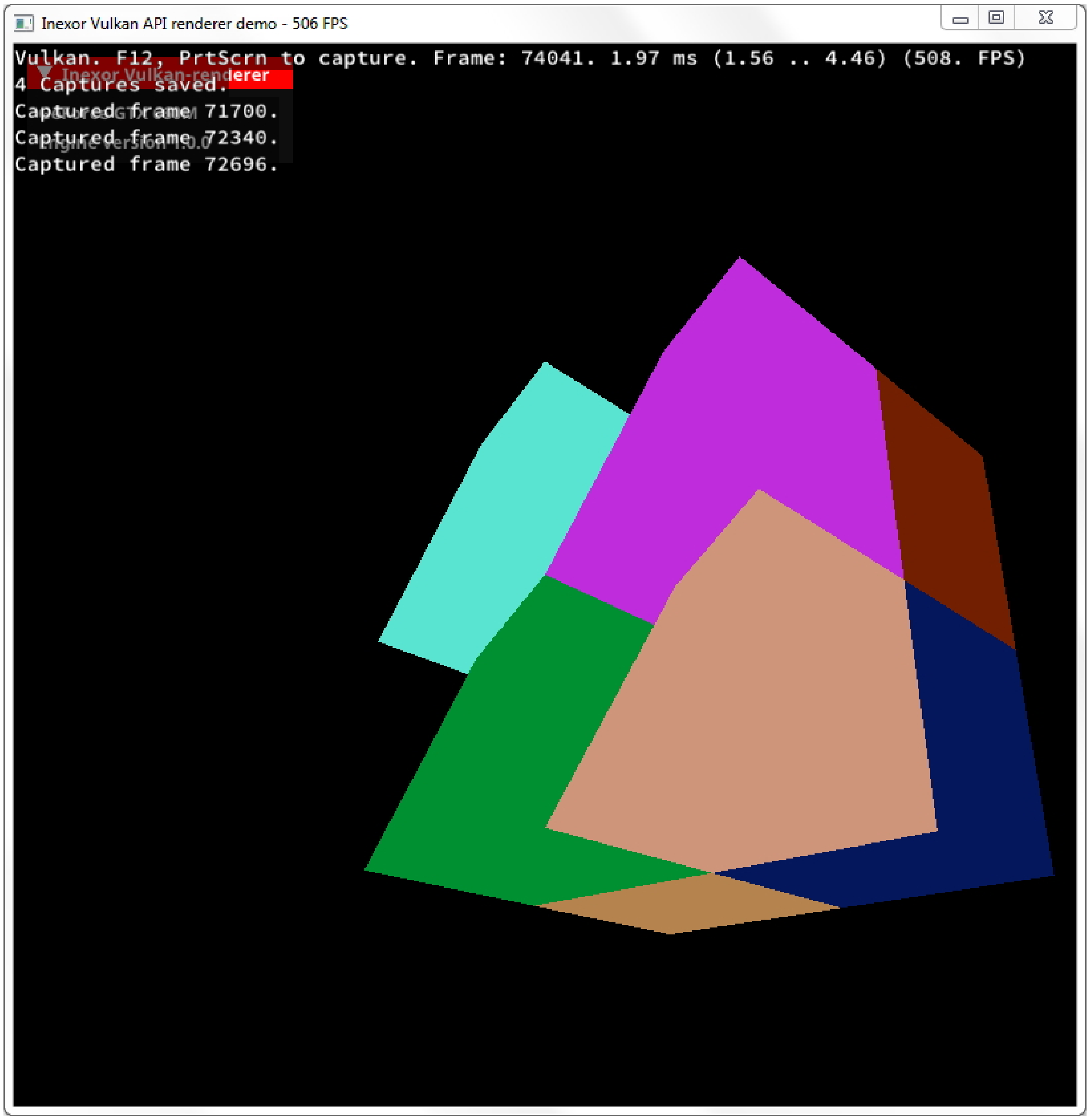
```
2020-12-25 22:05:56.817899 debug  8296 [vulkan-renderer] Loading ImGUI fragment shader
2020-12-25 22:05:56.818899 debug  8296 [vulkan-renderer] Creating shader module ImGUI fragment shader.
2020-12-25 22:05:56.818899 debug  8296 [vulkan-renderer] Loading front 'assets/fonts/NotoSans-Bold.ttf'
2020-12-25 22:05:56.825900 debug  8296 [vulkan-renderer] Creating ImGUI font texture
2020-12-25 22:05:56.825900 debug  8296 [vulkan-renderer] Created command pool successfully.
2020-12-25 22:05:56.826900 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 262144 for 'ImGUI font texture'.
2020-12-25 22:05:56.826900 debug  8296 [vulkan-renderer] Created command pool successfully.
2020-12-25 22:05:56.826900 debug  8296 [vulkan-renderer] Starting recording of once command buffer.
2020-12-25 22:05:56.826900 debug  8296 [vulkan-renderer] Transitioning image layout of texture ImGUI font te
2020-12-25 22:05:56.826900 debug  8296 [vulkan-renderer] Created command pool successfully.
2020-12-25 22:05:56.826900 debug  8296 [vulkan-renderer] Recording pipeline barrier for image layer transiti
2020-12-25 22:05:56.827900 debug  8296 [vulkan-renderer] Starting recording of once command buffer.
2020-12-25 22:05:56.827900 debug  8296 [vulkan-renderer] Ending recording of once command buffer.
2020-12-25 22:05:56.827900 debug  8296 [vulkan-renderer] Command buffer recording ended successfully.
2020-12-25 22:05:56.827900 debug  8296 [vulkan-renderer] Starting to submit command.
2020-12-25 22:05:56.831900 debug  8296 [vulkan-renderer] Destroying once command buffer.
2020-12-25 22:05:56.831900 debug  8296 [vulkan-renderer] Transitioning image layout of texture ImGUI font te
2020-12-25 22:05:56.831900 debug  8296 [vulkan-renderer] Ending recording of once command buffer.
2020-12-25 22:05:56.831900 debug  8296 [vulkan-renderer] Command buffer recording ended successfully.
2020-12-25 22:05:56.831900 debug  8296 [vulkan-renderer] Starting to submit command.
2020-12-25 22:05:56.832900 debug  8296 [vulkan-renderer] Destroying once command buffer.
2020-12-25 22:05:56.833900 debug  8296 [vulkan-renderer] Created command pool successfully.
2020-12-25 22:05:56.832900 debug  8296 [vulkan-renderer] Recording pipeline barrier for image layer transiti
2020-12-25 22:05:56.833900 debug  8296 [vulkan-renderer] Starting recording of once command buffer.
2020-12-25 22:05:56.833900 debug  8296 [vulkan-renderer] Ending recording of once command buffer.
2020-12-25 22:05:56.833900 debug  8296 [vulkan-renderer] Command buffer recording ended successfully.
2020-12-25 22:05:56.833900 debug  8296 [vulkan-renderer] Starting to submit command.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Destroying once command buffer.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Creating image sampler for texture ImGUI font textu
2020-12-25 22:05:56.834900 trace  8296 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Image sampler ImGUI font texture created successful
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Created command pool successfully.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Creating new descriptor pool.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Creating descriptor set layout for descriptor 'ImGUI
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Created descriptor sets for descriptor ImGUI succes
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Creating descriptor sets for 'ImGUI'.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Created descriptor pool for descriptor ImGUI succes
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Updating descriptor set 'ImGUI' #0
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Updating descriptor set 'ImGUI' #1
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Updating descriptor set 'ImGUI' #2
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Created descriptor sets for descriptor ImGUI succes
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Creating ImGUI renderpass
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Created renderpass successfully.
2020-12-25 22:05:56.834900 debug  8296 [vulkan-renderer] Creating renderpass ImGUI.
2020-12-25 22:05:56.835900 debug  8296 [vulkan-renderer] Creating cache for graphics pipeline.
2020-12-25 22:05:56.835900 debug  8296 [vulkan-renderer] Creating ImGUI graphics pipeline
2020-12-25 22:05:56.835900 debug  8296 [vulkan-renderer] Creating graphics pipeline.
2020-12-25 22:05:56.835900 debug  8296 [vulkan-renderer] Created graphics pipeline and cache successfully.
2020-12-25 22:05:56.835900 debug  8296 [vulkan-renderer] Creating Vulkan synchronisation fence ImGUI renderi
2020-12-25 22:05:56.835900 debug  8296 [vulkan-renderer] Created fence ImGUI rendering done successfully.
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] FPS: 8, window size: 800 x 800.
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 4760 for 'imgui_
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 798 for 'imgui_m
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] Creating vertex buffer of size 4760 for mesh imgui_
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] Creating index buffer of size 798 for mesh imgui_me
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] Creating ImGUI vertex buffer
2020-12-25 22:05:56.865902 trace  8296 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-25 22:05:56.865902 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 4760 for 'imgui_
2020-12-25 22:05:56.865902 trace  8296 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating vertex buffer of size 4760 for mesh imgui_
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 798 for 'imgui_m
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating index buffer of size 798 for mesh imgui_me
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating ImGUI index buffer
2020-12-25 22:05:56.866902 trace  8296 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-25 22:05:56.866902 trace  8296 [vulkan-renderer] Destroying GPU memory buffer.
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 4760 for 'imgui_
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating GPU memory buffer of size 798 for 'imgui_m
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating vertex buffer of size 4760 for mesh imgui_
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating index buffer of size 798 for mesh imgui_me
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating frame buffers for ImGUI
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Creating ImGUI renderpass
2020-12-25 22:05:56.866902 trace  8296 [vulkan-renderer] Creating framebuffer ImGUI Framebuffer.
2020-12-25 22:05:56.866902 debug  8296 [vulkan-renderer] Created framebuffer ImGUI Framebuffer successfully.
2020-12-25 22:05:56.867902 trace  8296 [vulkan-renderer] Creating framebuffer ImGUI Framebuffer.
2020-12-25 22:05:56.867902 debug  8296 [vulkan-renderer] Created framebuffer ImGUI Framebuffer successfully.
2020-12-25 22:05:56.867902 trace  8296 [vulkan-renderer] Creating framebuffer ImGUI Framebuffer.
2020-12-25 22:05:56.867902 debug  8296 [vulkan-renderer] Created framebuffer ImGUI Framebuffer successfully.
2020-12-25 22:05:57.887961 debug  8296 [vulkan-renderer] FPS: 115, window size: 800 x 800.
2020-12-25 22:05:59.419048 debug  8296 [vulkan-renderer] FPS: 73, window size: 800 x 800.
2020-12-25 22:06:00.874131 debug  8296 [vulkan-renderer] FPS: 220, window size: 800 x 800.
2020-12-25 22:06:01.875189 debug  8296 [vulkan-renderer] FPS: 617, window size: 800 x 800.
2020-12-25 22:06:02.875246 debug  8296 [vulkan-renderer] FPS: 671, window size: 800 x 800.
```

Inexor Vulkan API renderer demo - 671 FPS

```
Vulkan. F12, PrtScrn to capture. Frame: 40284. 1.49 ms (1.19 .. 5.20) (672. FPS)
0 Captures saved.
```

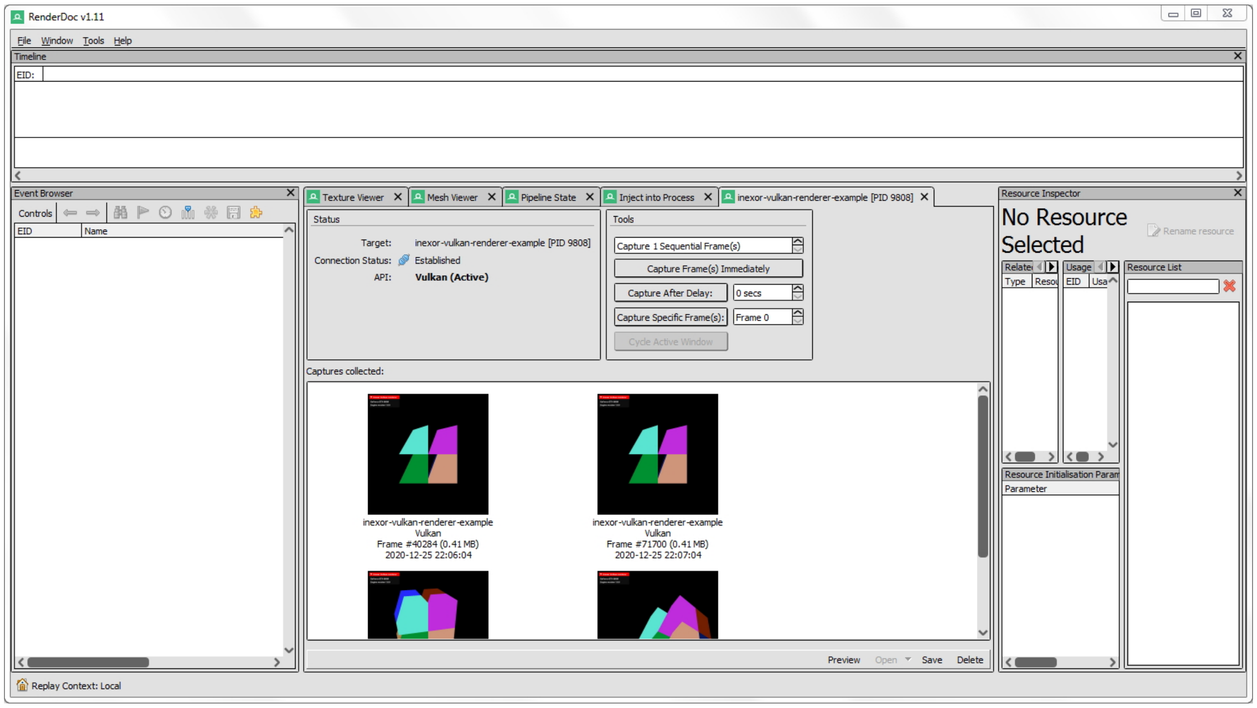
Step 7: Debug inexor-vulkan-renderer.exe as usual and press F12 to take RenderDoc snapshots

- You can take multiple snapshots with either `PRINT` or `F12` key.



```
Vulkan. F12, PrtScrn to capture. Frame: 74041. 1.97 ms (1.56 .. 4.46) (508. FPS)
4 Captures saved.
Captured frame 71700.
Captured frame 72340.
Captured frame 72696.
```

- You can see the snapshots in RenderDoc right after you took them:



EID:

Status

Target: inexor-vulkan-renderer-example [PID 9808]

Connection Status: Established

API: **Vulkan (Active)**

Tools

Capture 1 Sequential Frame(s)

Capture After Delay: 0 secs

Capture Specific Frame(s): Frame 0

Captures collected:

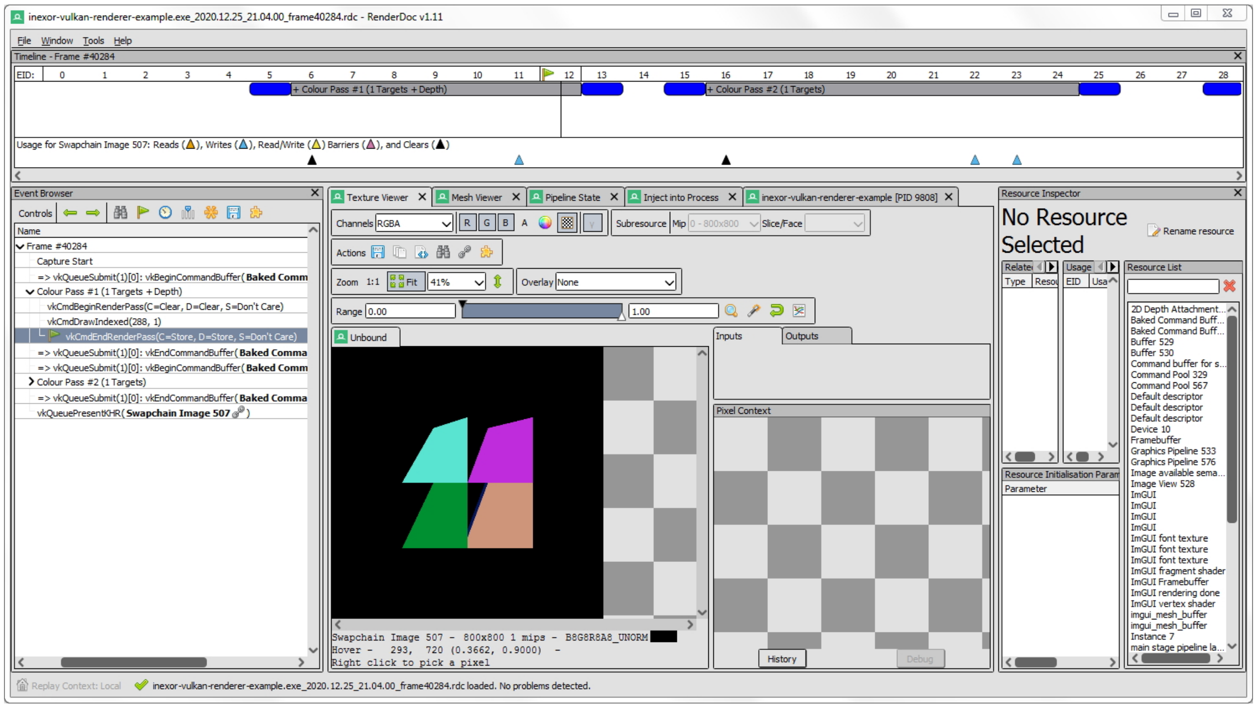
inexor-vulkan-renderer-example
Vulkan
Frame #40284 (0.41 MB)
2020-12-25 22:06:04

inexor-vulkan-renderer-example
Vulkan
Frame #71700 (0.41 MB)
2020-12-25 22:07:04

Replay Context: Local

Step 8: Open a snapshot to analyze the rendering of this frame

- Double click on a snapshot to open it:



inexor-vulkan-renderer-example.exe_2020.12.25_21.04.00_frame40284.rdc - RenderDoc v1.11

Timeline - Frame #40284

+ Colour Pass #1 (1 Targets + Depth)
+ Colour Pass #2 (1 Targets)

Usage for Swapchain Image 507: Reads (△), Writes (△), Read/Write (△) Barriers (△), and Clears (▲)

```
Frame #40284
  Capture Start
  => vkQueueSubmit(1)[0]: vkBeginCommandBuffer(Baked Comm
  Colour Pass #1 (1 Targets + Depth)
    vkCmdBeginRenderPass(C=Clear, D=Clear, S=Don't Care)
    vkCmdDrawIndexed(288, 1)
    vkCmdEndRenderPass(C=Store, D=Store, S=Don't Care)
  => vkQueueSubmit(1)[0]: vkEndCommandBuffer(Baked Comma
  => vkQueueSubmit(1)[0]: vkBeginCommandBuffer(Baked Comm
  Colour Pass #2 (1 Targets)
  => vkQueueSubmit(1)[0]: vkEndCommandBuffer(Baked Comma
  vkQueuePresentKHR(Swapchain Image 507)
```

Channels RGBA · Subresource Mip 0 - 800x800 · Slice/Face · Zoom 1:1 Fit 41% · Overlay None · Range 0.00 – 1.00 · Unbound

```
Swapchain Image 507 - 800x800 1 mips - B8G8R8A8_UNORM
Hover -   293,  720 (0.3662, 0.9000)  -
Right click to pick a pixel
```

Resource List: 2D Depth Attachment..., Baked Command Buff..., Baked Command Buff..., Buffer 529, Buffer 530, Command buffer for s..., Command Pool 329, Command Pool 567, Default descriptor, Default descriptor, Default descriptor, Device 10, Framebuffer, Graphics Pipeline 533, Graphics Pipeline 576, Image available sema..., Image View 528, ImGUI, ImGUI, ImGUI, ImGUI, ImGUI font texture, ImGUI font texture, ImGUI font texture, ImGUI fragment shader, ImGUI Framebuffer, ImGUI rendering done, ImGUI vertex shader, imgui_mesh_buffer, imgui_mesh_buffer, Instance 7, main stage pipeline la...

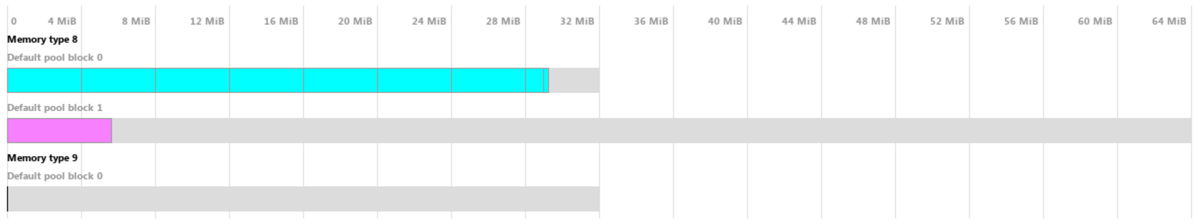
Replay Context: Local · inexor-vulkan-renderer-example.exe_2020.12.25_21.04.00_frame40284.rdc loaded. No problems detected.

- Have fun inspecting!

VMA dumps

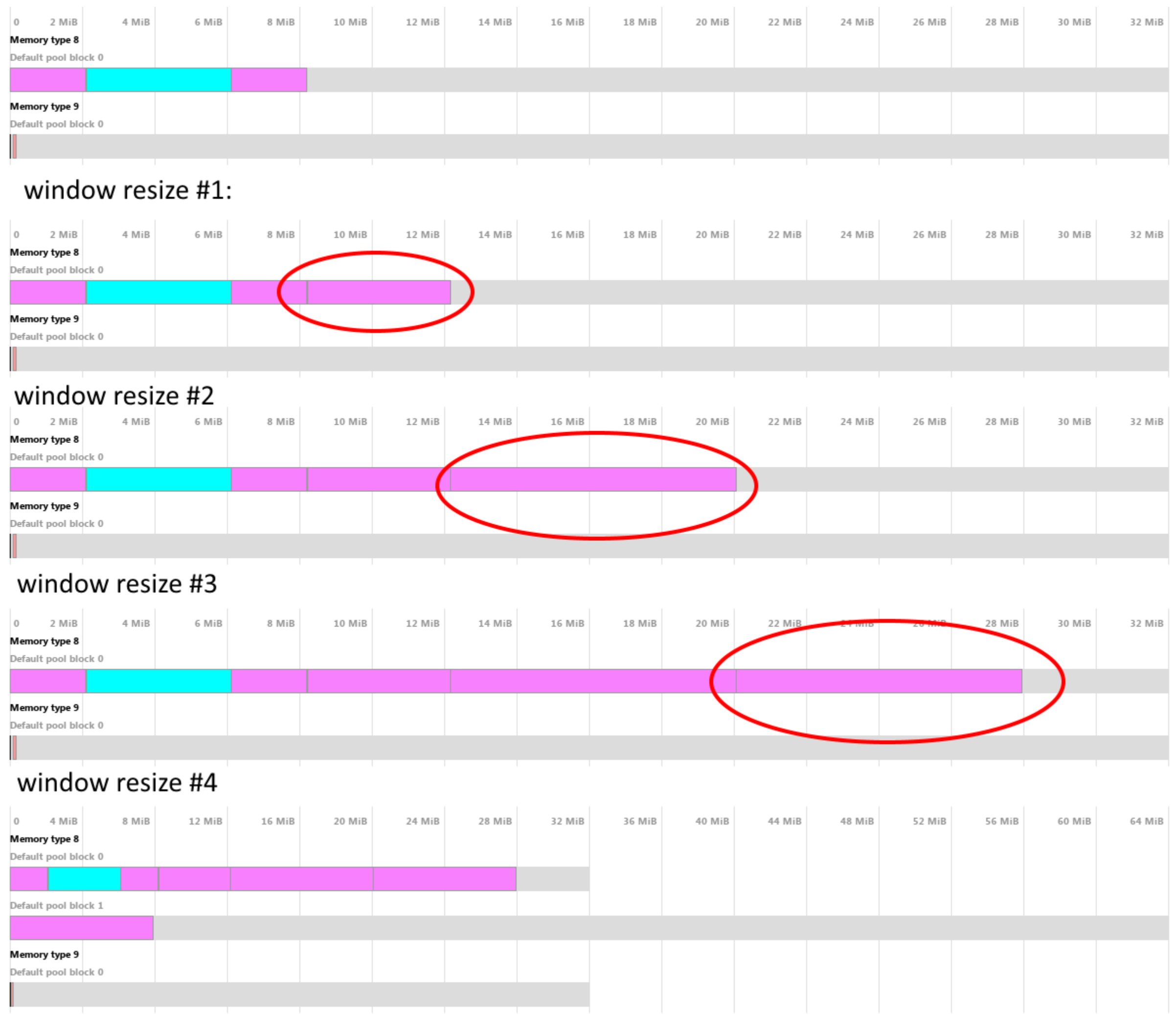
- For memory management, Inexor uses AMD's Vulkan Memory Allocator library (VMA).
- VMA can export an overview of the current GPU memory allocations into a JSON file which can be converted into an image using VMA's VmaDumpVis.
- This way we can see which memory pools exist, which memory blocks are allocated, and what they are used for.
- The example image provided here might look very simple, but with increasing complexity of our engine this will turn out very helpful.

This is a very simple example of such an image generated with VmaDumpVis:



Example: march 2020 texture memory corruption bug

- In march 2020, a bug in Inexor's early graphics memory management caused textures to corrupt on window resize.
- VmaDumpVis helped to resolve the issue by proving the memory consumption increased after each resize, which means the texture memory was simply not freed when the swapchain was recreated.
- This is an example of how VMA and VmaDumpVis make debugging graphics memory easier.



1.1.5 Engine design

Folder structure

- Use lowercase filenames
- Don't use spaces, use underscores.

Source code

```
connector/  «project root»
├── .clang-format  «Clang Format configuration»
├── .clang-tidy  «Clang Tidy configuration»
├── .git-blame-ignore-revs  «git ignore revisions»
├── .gitignore  «git ignore»
├── .readthedocs.yml  «Read The Docs configuration»
├── CHANGELOG.rst
├── CMakeLists.txt
```

(continues on next page)

(continued from previous page)

```
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── LICENSE.md
├── README.rst
├── .github/  «GitHub templates and action configurations»
├── assets/
│   ├── models/
│   └── textures/  «textures»
├── benchmarks/
├── cmake/  «CMake helpers»
├── configuration/
├── documentation/
│   ├── CMakeLists.txt  «CMake file for the documentation»
│   ├── cmake/  «documentation cmake helpers»
│   └── source/  «documentation source code»
├── example/  «example application»
├── include/  «header files»
├── shaders/
├── src/  «source code»
├── tests/
├── third_party/  «third party dependencies»
└── vma-dumps/
```

Application

```
vulkan-renderer/  «application root»
├── inexor-vulkan-renderer.exe  «executable»
├── ...
├── assets/
├── shaders/
└── ...
```

Dependency management

- In general we try to keep the number of dependencies at a minimum.
- Dependencies are downloaded directly by CMake.

Criteria for library selection

If we really need a new library, it should be selected based on the following considerations:

- Are you sure we need this library? Can't we solve the problem with C++ standard library or some other library we are already using?
- The library must have an open source license which is accepted by us (see *Licenses*).
- It must be in active development.
- It must have a good documentation.
- A sufficient number of participants must have contributed so we can be sure it is reviewed.

Coding style

The easiest way to get the right format is to use the provided clang-format file in the root directory.

Other styles which cannot be applied automatically are listed below:

- Use `#pragma once` as include guards
- Own headers are included with quotes
- **Includes are ordered as follows**
  - Own headers
  - *empty line*
  - Third Party Libraries
  - *empty line*
  - System Libraries
- Use C++17 namespace style `namespace inexor::vulkan-renderer`
- No `using <namespace>`
- For default member initialization use brace instead of equal initialization
- Prefer American English over British English
- Use spaces to indent
- Use Linux line ends (ln) in your commits
- Use `///` for multiline documentation instead of `/**/`

Naming convention

Open the `.clang-tidy` file and search for `readability-identifier-naming` to get the naming convention used by this project.

Error handling

- Use exceptions for error handling, as proposed by the C++ core guidelines.
- More information about why to use exceptions can be found here.

Get methods

- Name: **Don't** use prefix `get_`. Give the get method the same name as the resource it returns.
- For complex types (`std::vector`, `std::string`), return a const reference.
- Don't `const` the return type for simple types (`int`, `float`), because this prevents move semantics to be applied.
- For simple types (`int`, `float`), just copy the return value.
- Mark get methods as `[[nodiscard]]` in the header file only.
- Mark get methods as `const`, so they don't change members.
- Do not add documentation for get methods, since it is self-explanatory.

- Keep get methods directly in the header file.
- Do not add `inline` since get methods in header files are always inlined.
- The get method should not run any other code, like checking if the value is actually valid. Since we are using RAII, the value to return must be in a valid state anyways.
- Use operator overloading sparingly. Prefer get methods instead.

Examples:

```
[[nodiscard]] const glm::vec3& position() const {
    return m_position;
}

[[nodiscard]] float aspect_ratio() const {
    return m_aspect_ratio;
}
```

Removed clang-tidy checks

bugprone-narrowing-conversions Same as `cppcoreguidelines-narrowing-conversions`

cppcoreguidelines-avoid-magic-numbers Alias of `readability-magic-numbers`

cppcoreguidelines-c-copy-assignment-signature Alias of `misc-unconventional-assign-operator`

cppcoreguidelines-non-private-member-variables-in-classes Alias of `misc-non-private-member-variables-in-classes`

cppcoreguidelines-pro-bounds-array-to-pointer-decay Not suitable for this project.

google-readability-todo We do not care about any TODO assignments or related issues.

hicpp-explicit-conversions Alias of `google-explicit-constructor`

hicpp-move-const-arg Alias of `performance-move-const-arg`

hicpp-no-array-decay Alias of `cppcoreguidelines-pro-bounds-array-to-pointer-decay`

hicpp-uppercase-literal-suffix Alias of `readability-uppercase-literal-suffix`

llvm-header-guard `#pragma once` is used.

modernize-use-trailing-return-type Trailing return types are not used.

readability-magic-numbers Too many places where it would be useless to introduce a constexpr value.

readability-uppercase-literal-suffix Just a style preference.

Code design

Literature

The following books inspired Inexor's code design:

- Bjarne Stroustrup: The C++ Programming Language (4th Edition)
- Scott Meyers: Effective Modern C++
- Scott Meyers: Effective C++: 55 Specific Ways to Improve Your Programs and Designs, Third Edition
- Scott Meyers: Effective STL

- Nicolai M. Josuttis: C++ Move Semantics - The Complete Guide
- Nicolai M. Josuttis: C++ Templates - The Complete Guide, 2nd Edition
- Bartłomiej Filipek C++ Lambda Story
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software
- Robert C. Martin: Clean Code: A Handbook of Agile Software Craftsmanship
- Robert C. Martin: The Clean Coder: A Code of Conduct for Professional Programmers
- Robert C. Martin: Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition
- Fedor G. Pikus: Hands-On Design Patterns with C++
- Rian Quinn: Advanced C++ Programming Cookbook

General considerations

- Organize the code in components.
- Split declarations and definitions, if possible.
- Make appropriate use of the standard library.
- Avoid data redundancy in the engine. Do not keep memory copied unnecessarily.
- Do not duplicate code. Find an appropriate abstraction which accounts for the scenario.
- Try to keep dependencies between components at minimum because single components (e.g. classes) should be as recyclable as possible.
- Use spdlog instead of `printf` or `std::cout` for console output.
- Use `assert` to validate parameters or necessary resources during development (debug mode).
- Document the code using doxygen comments. Code without documentation is almost useless.
- Make sure the code is platform-independent. For now, we will support Windows and Linux but not Mac OS.
- Use Vulkan memory allocator library for Vulkan-specific memory allocations like buffers.
- Do not allocate memory manually. Use modern ++ features like smart pointers or STL containers instead.
- Don't use global variables.
- Don't use the singleton pattern as it makes thread safety and refactoring difficult.
- Don't use call-by-value for returning values from a function call.
- Don't use macros for code generation or as a replacement for enumerations.

C++ core guidelines

- The C++ code guidelines are a set of rules to use for modern C++ projects created by the C++ community.
- In the following section, we will list up the entries which are of considerable interest for the Inexor project.
- There will be some gaps in the number as we skipped some of the less importer ones.
- Also the code guidelines have gaps by default (blank space for new rules).

Philosophy

- P.1: Express ideas directly in code
- P.3: Express intent
- P.4: Ideally, a program should be statically type safe
- P.5: Prefer compile-time checking to run-time checking
- P.8: Don't leak any resources
- P.9: Don't waste time or space
- P.10: Prefer immutable data to mutable data
- P.11: Encapsulate messy constructs, rather than spreading through the code

Interfaces

- I.1: Make interfaces explicit
- I.4: Make interfaces precisely and strongly typed
- I.5: State preconditions (if any)
- I.7: State postconditions
- I.10: Use exceptions to signal a failure to perform a required task
- I.11: Never transfer ownership by a raw pointer (T*) or reference (T&)
- I.13: Do not pass an array as a single pointer
- I.23: Keep the number of function arguments low
- I.24: Avoid adjacent parameters of the same type when changing the argument order would change meaning

Functions and class methods

- F.1: "Package" meaningful operations as carefully named functions
- F.2: A function should perform a single logical operation
- F.4: If a function may have to be evaluated at compile time, declare it constexpr
- F.7: For general use, take T* or T& arguments rather than smart pointers
- F.15: Prefer simple and conventional ways of passing information
- F.16: For "in" parameters, pass cheaply-copied types by value and others by reference to const
- F.18: For "will-move-from" parameters, pass by X&& and std::move the parameter

- F.20: For “out” output values, prefer return values to output parameters
- F.21: To return multiple “out” values, prefer returning a struct or tuple
- F.26: Use a unique_ptr<T> to transfer ownership where a pointer is needed
- F.27: Use a shared_ptr<T> to share ownership
- F.43: Never (directly or indirectly) return a pointer or a reference to a local object
- F.45: Don’t return a T&&
- F.51: Where there is a choice, prefer default arguments over overloading
- F.55: Don’t use va_arg arguments

Classes

- C.2: Use class if the class has an invariant; use struct if the data members can vary independently
- C.3: Represent the distinction between an interface and an implementation using a class
- C.4: Make a function a member only if it needs direct access to the representation of a class
- C.7: Don’t define a class or enum and declare a variable of its type in the same statement
- C.8: Use class rather than struct if any member is non-public
- C.9: Minimize exposure of members

Enumerations

- Enum.1: Prefer enumerations over macros
- Enum.2: Use enumerations to represent sets of related named constants
- Enum.3: Prefer class enums over “plain” enums
- Enum.6: Avoid unnamed enumerations
- Enum.7: Specify the underlying type of an enumeration only when necessary

Resource management

- R.1: Manage resources automatically using resource handles and RAII (Resource Acquisition Is Initialization)
- R.2: In interfaces, use raw pointers to denote individual objects (only)
- R.3: A raw pointer (a T*) is non-owning
- R.4: A raw reference (a T&) is non-owning
- R.5: Prefer scoped objects, don’t heap-allocate unnecessarily
- R.10: Avoid malloc() and free()
- R.11: Avoid calling new and delete explicitly
- R.12: Immediately give the result of an explicit resource allocation to a manager object
- R.13: Perform at most one explicit resource allocation in a single expression statement

Classes

- C.30: Define a destructor if a class needs an explicit action at object destruction
- C.31: All resources acquired by a class must be released by the class's destructor
- C.35: A base class destructor should be either public and virtual, or protected and non-virtual
- C.36: A destructor must not fail
- C.40: Define a constructor if a class has an invariant
- C.41: A constructor should create a fully initialized object
- C.42: If a constructor cannot construct a valid object, throw an exception
- C.43: Ensure that a copyable (value type) class has a default constructor
- C.44: Prefer default constructors to be simple and non-throwing
- C.46: By default, declare single-argument constructors explicit
- C.47: Define and initialize member variables in the order of member declaration
- C.49: Prefer initialization to assignment in constructors
- C.62: Make copy assignment safe for self-assignment
- C.64: A move operation should move and leave its source in a valid state
- C.65: Make move assignment safe for self-assignment
- C.80: Use =default if you have to be explicit about using the default semantics
- C.81: Use =delete when you want to disable default behavior (without wanting an alternative)
- C.82: Don't call virtual functions in constructors and destructors
- C.90: Rely on constructors and assignment operators, not memset and memcpy
- C.129: When designing a class hierarchy, distinguish between implementation inheritance and interface inheritance
- C.131: Avoid trivial getters and setters
- C.132: Don't make a function virtual without reason
- C.133: Avoid protected data

Follow rule of 0 and rule of 5

- C.20 If you can avoid defining default operations, do
- C.21: If you define or =delete any copy, move, or destructor function, define or =delete them all

Performance

- Per.1: Don't optimize without reason
- Per.2: Don't optimize prematurely
- Per.3: Don't optimize something that's not performance critical
- Per.4: Don't assume that complicated code is necessarily faster than simple code
- Per.5: Don't assume that low-level code is necessarily faster than high-level code
- Per.6: Don't make claims about performance without measurements
- Per.11: Move computation from run time to compile time

Design patterns

- Check out Refactoring Guru to learn more about software design patterns.
- Don't use the singleton pattern as it makes thread safety and refactoring difficult.
- Use the builder pattern for composition of complicated data structures.
- An example of a builder pattern would be the descriptor builder.

Regressions

- If something used to work but it's broken after a certain commit, it's not just some random bug.
- It's probably an issue which was introduced by the code change which was submitted.
- It's important for us to keep working features in a stable state.
- You can use `git bisect` for tracing of the commit which introduced the bug.

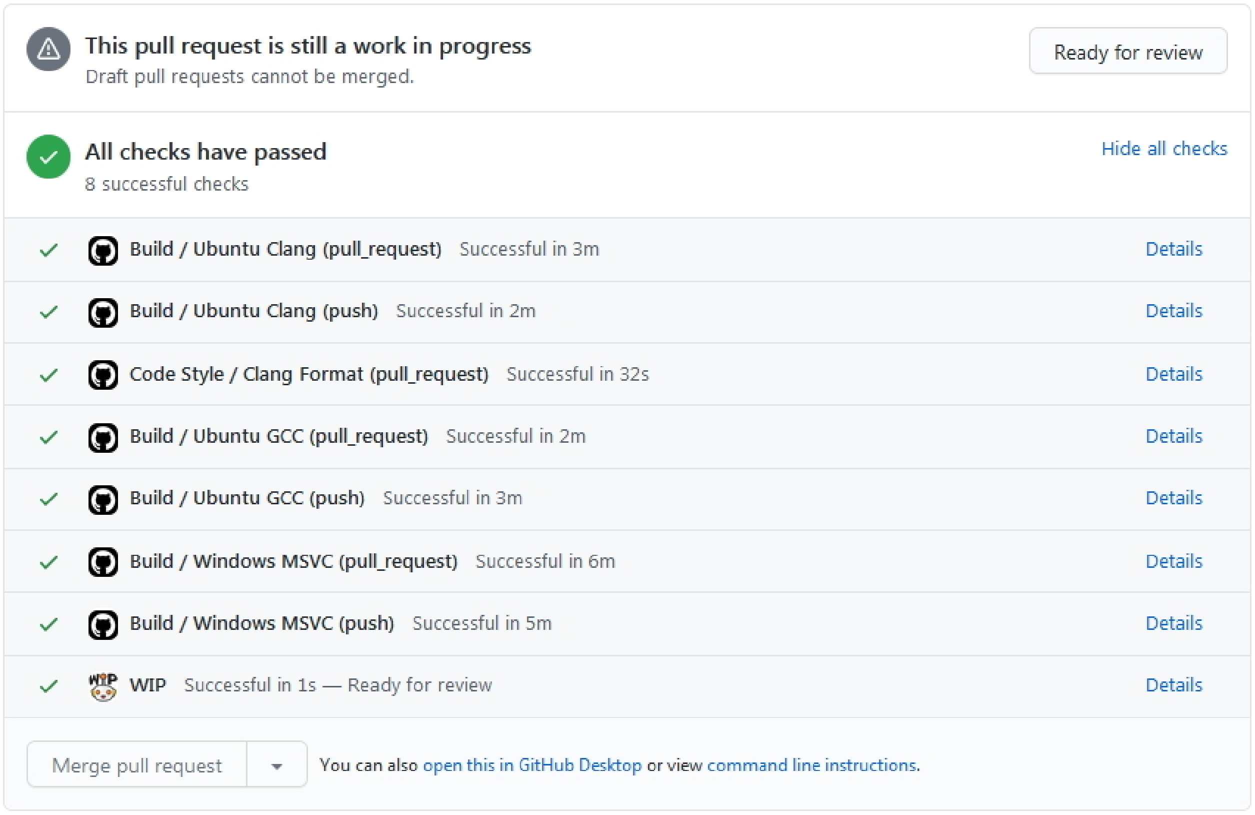
1.1.6 Continuous integration

- The main branch and the build system must stay stable at all time.
- You can see the current status of the main branch in the build batch:
- Currently we are using GitHub Actions for building with gcc, clang and Microsoft Visual Studio on every push or pull request.
- This Continuous Integration (CI) allows for automatic building and testing of our software.
- We also have a webhook which directly dispatches the build status into our Discord. This allows us to spot and fix broken code easily.
- Our CI setup is inspired by a blog entry by Adam Sawicki.

1.1.7 Clang format

- In order to have one unified code formatting style, we use clang-format.
- Clang-format automatically formats source code according to a set of rules which a project needs to agree on.
- Our current style can be found in the clang-format file file in the root folder of the repository.
- We recommend to install plugins which auto format the code when the file is being saved.
- Instructions for how to enable clang-format in Microsoft Visual Studio can be found here.
- Other editors like Visual Studio Code, Atom.io, Notepad++ and Sublime Text support this as well.
- Part of our Continuous Integration (CI) are automated clang-format checks using GitHub actions.
- Our setup of clang-format with GitHub actions can be here.
- A pull request will only be accepted if it follows those code formatting rules.

Example of clang-format checking a pull request along with gcc/clang/msvc build:


This pull request is still a work in progress
Draft pull requests cannot be merged.
Ready for review

All checks have passed
8 successful checks
Hide all checks

✓	Build / Ubuntu Clang (pull_request) Successful in 3m	Details
✓	Build / Ubuntu Clang (push) Successful in 2m	Details
✓	Code Style / Clang Format (pull_request) Successful in 32s	Details
✓	Build / Ubuntu GCC (pull_request) Successful in 2m	Details
✓	Build / Ubuntu GCC (push) Successful in 3m	Details
✓	Build / Windows MSVC (pull_request) Successful in 6m	Details
✓	Build / Windows MSVC (push) Successful in 5m	Details
✓	WIP Successful in 1s — Ready for review	Details

Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

1.1.8 Test automation

- Inexor will use Google Test for automated software testing in the future.
- Running automatic tests using GitHub actions is not possible for Vulkan features since this requires a graphics card to be present.
- There are some services which offer test automation for rendering, but they are not free.
- The tests would have to run on the developer's machine locally.

1.1.9 Benchmarking

- Inexor will use Google Benchmark in the future.
- Benchmarks can also not run in GitHub actions since testing Vulkan features would require a graphics card.
- The tests will run locally on the developer's machine along with the tests.

1.1.10 Static code analysis

We analyze our source code regularly using static code analysis.

The following tools are used:

- Clang-tidy.
- Microsoft Visual Studio code analysis.
- Valgrind, Callgrind and Cachegrind.

1.1.11 Reference

GPU selection mechanism

- In Vulkan API, the word physical device is a more general word for all types of graphics cards, integrated gpus and more
- If multiple physical devices are available on the system, Inexor engine is able to pick the most suitable one automatically
- The user can specify a preferred graphics card index with the command line argument `--gpu <index>` (starting with index `0`!)
- If a preferred index is specified, the engine will verify if the index is valid and pick the physical device if it is suitable
- If the physical device specified by the user is not suitable because of technical reasons, automatic selection rules apply
- The engine calculates a score for every available physical device based on the device type and total video memory size
- If no physical devices are available or no suitable physical device could be chosen, an exception is thrown

Binary Format Specification

Comments start with `//`, everything after for the rest of the line is ignored.

Spaces can be used to improve the readability, but are not required. Spaces are preferred over tabs.

Variable or data type names are case sensitive.

Endianness

Available values are `little` and `big`. When defining the reserved data type name `ENDIANNESS`, all other data types who do not define an own endianness are interpreted as this.

```
| ENDIANNESS : little // default value for all non defined
```

Data Types

Every data type name, can also be used as an variable name, they don't interfere. Data types can be defined the following way `| <name> :  <bit size> - <endianness> // description`, whereas `- <endianness>` is optional, if a default endianness is given. The bit size can be set by a variable too.

```
| Bit : 1 - little // A bit, 0 or 1.
| uByte : 4 - little // An unsigned byte.
| uInt : 8 - little // unsigned integer

// or

| ENDIANNESS : little
| Bit : 1 // A bit, 0 or 1.
| uInt : 8 // unsigned integer

uInt : mySize = 0
| myType : mySize // My special type.
```

Struct Types

If you have compressed data, it can be easily described as a struct `struct <name> {`. Members can be access with a dot.

```
struct SpecSummary {
    > uByte (1) : size // Size
    > uByte (1) : spec_a // Specification A
    > uByte (1) : spec_b // Specification B
}

> SpecSummary (1) : spec // specification summary

if (spec.size > 0) {
    read_spec()
}
```

Bit Reading & Variables

Each bit interpretation starts with >, whereas variable name and the reinterpreted type are optional. Variables are only valid in their declared scope. `<counter>` is a value, how often the data type is read, it will create a list with its content. If a reinterpreted type is given, it will be reinterpreted as if the amount of single bits was read by the reinterpretation type (`> Bit (3) : uByte` reinterprets 3 bits into one uByte). If the reinterpreted type is bigger, it will be filled with 0 regarding to the endianness, to not change the value (e.g. little endianness will fill from the left side). Same with cutting it off.

```
> <data type> (<counter>) <reinterpreted type> : <variable name> // <description>
                          ^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^ optional

// examples

> uByte (7) // string identifier: "myimage"
> uInt (1) : image_size // image size in bytes
> Bit (3) uByte : red_color // red color

> Bit (3) uByte // possible but does not make any sense, as the reinterpreted type will␣
↪not be used.

uInt : other_size = image_size // declare new variable other_size with content of image_
↪size
uInt : id = 0 // declare a new variable of type uInt and content 0
```

Conditions

`if`, `else if`, `else`. Available comparisons are ==, !=, <, >, <=, >=. Which can be combined with && (and) and `||` (or). Brackets are not required around the condition, but can improve the readability. You cannot compare a list to a number. You have to reinterpret it beforehand or cast it.

```
> Bit (2) uByte : roll_over // roll over simulation type
uInt : roll_over_int = roll_over
if (roll_over_int == 0) { // roll over specification 0
    > uInt (1) // roll over specifications
} else if (uInt(roll_over) == 1) { // roll over specification 1
    > uInt (1) // roll over specifications
    > uInt (1) // more roll over specifications
} else { // default roll over specification
    > uByte (1) // weight in kg
    > uInt (1) // roll over specifications
}
```

A short version for `else if` chaining is `switch`, `case`. `default` will be used if no other case matches. There is no default fallthrough, to get this behaviour use `fallthrough`. A `break` inside a switch case will break out of it.

```
> Bit (2) uByte : roll_over // roll over simulation type
uInt : roll_over_int = roll_over
switch (roll_over_int) {
case 0: // roll over specification 0
    > uInt (1) // roll over specifications
case 1: // roll over specification 1
```

(continues on next page)

(continued from previous page)

```
    > uInt (1) // roll over specifications
    > uInt (1) // more roll over specifications
default: // default roll over specification
    > uByte (1) // weight in kg
    > uInt (1) // roll over specifications
}
```

Loops

There are three types of loops: `for`, range-based `for` and a `while` loop. Brackets are not required around the condition, but can improve the readability. You can break out of a loop with the `break` keyword.

```
for (id; id <= 3; id++) { // 0..2
    // ...
}

for (0..2 : id) { // including both borders 0 and 2
    // ...
}

uInt : id = 0
while (id <= 3) {
    // ...
    id = id + 1
}

uInt : id = 0
while (true) {
    if (id == 3) {
        break
    }
    // ...
    id = id + 1
}
```

Functions

Functions can be used to reuse a specific block again. `def <name>(<parameter>) <return values> {}`

```
def get_cube(uInt : param) uInt, uByte {
    > uInt (1) : first
    > uByte (1) : second
    return first, second
} // get_cube

> uInt (1) : rec_first
> uByte (1) : rec_second
rec_first, rec_second = get_cube()
```

Keyboard and mouse input

Inexor engine uses glfw3 for window management and for keyboard and mouse input. Inexor does not use manual polling of keyboard and mouse input data, but uses callbacks instead. This way, we ensure we are not missing a key event. For more information check out glfw's input guide. Inexor uses a wrapper class for keyboard and mouse input data, called `KeyboardMouseInputData`. This class offers an easy-to-use interface for setting and getting keyboard and mouse input. `KeyboardMouseInputData` is thread safe since pull request 401.

Note: Inexor redirects keyboard and mouse input events to class methods which handle it. Because glfw is a C-style API, it is not possible to use class methods directly as callbacks for keyboard and mouse input data. To fix this, we set the glfw window user pointer to the class instance which contains the input callback methods. Then, we use a lambda to set up the class method as callback. All setups are done in `Application::setup_window_and_input_callbacks`. For more information about this workaround, check out this Stackoverflow issue.

Note: It's not possible handle glfw input data in a thread which is separate from the thread which created the corresponding window. For more information, check out this glfw forum post.

Keyboard input

- We store the pressed keys as a `std::array<bool, GLFW_KEY_LAST>` member in `KeyboardMouseInputData`
- The maximum number of keys is defined by `GLFW_KEY_LAST`
- If a key is pressed or released, we notify `KeyboardMouseInputData` by calling method `press_key` and `release_key`, respectively
- Check if a key is currently pressed by calling method `is_key_pressed`
- Check if a key was pressed once by calling method `was_key_pressed_once`

Mouse input

- We store the pressed mouse buttons as a `std::array<bool, GLFW_MOUSE_BUTTON_LAST>` member in `KeyboardMouseInputData`
- The maximum number of mouse buttons is defined by `GLFW_MOUSE_BUTTON_LAST`.
- If a mouse button is pressed or released, we notify `KeyboardMouseInputData` by calling method `press_mouse_button` and `release_mouse_button`, respectively
- To update the current cursor position, we call `set_cursor_pos`
- To get the current cursor position, we call `get_cursor_pos`
- The change in cursor position can be queried with `calculate_cursor_position_delta`
- Check if a mouse button is pressed by calling method `is_mouse_button_pressed`
- Check if a mouse button was pressed once by calling method `was_mouse_button_pressed_once`

Joysticks

Inexor does not support joysticks yet.

Octree File Format

The Inexor octree format describes the structure of the maps (and even models) created in Inexor.

Octree

The orange are coordinates of the corner, the pink of the block.

The position of a block is always the `(0, 0, 0)` corner coordinates.

Corner and Block Order

If blocks or corners are ordered, they use this order.

Table 3: Corner and Block Order

ID	Coordinate
0	(0, 0, 0)
1	(0, 0, 1)
2	(0, 1, 0)
3	(0, 1, 1)
4	(1, 0, 0)
5	(1, 0, 1)
6	(1, 1, 0)
7	(1, 1, 1)

Order of Faces

The following order of faces is used:

Table 4: Order of Face

ID	Name	Normal vector
0	left	(-1, 0, 0)
1	right	(1, 0, 0)
2	front	(0, 1, 0)
3	back	(0, -1, 0)
4	top	(0, 0, 1)
5	bottom	(0, 0, -1)

Order of Indices on Face

The following corner indices are associated to the faces:

Table 5: Order of Indices on Face

ID	Indices
0	0, 1, 2, 3
1	4, 5, 6, 7
2	0, 1, 4, 5
3	2, 3, 6, 7
4	1, 3, 5, 7
5	0, 2, 4, 6

Edge Order

- All edges are going into the positive direction of the axis.
- The beginning of the edge is always the smaller corner id.
- If you look at a face, the edges are always numbered with an offset of 3.
- The edges are ordered counter-clockwise starting from the axis.
- A negative ID indicates the reverse direction (-axis).

Table 6: Edge Order

ID	Corner ID Tuple
0	(0, 4)
1	(0, 2)
2	(0, 1)
3	(2, 6)
4	(1, 3)
5	(4, 5)
6	(3, 7)
7	(5, 7)
8	(6, 7)
9	(1, 5)
10	(4, 6)
11	(2, 3)

Indentation

Every cube can be indented at each corner to all axis by 8 steps. In total there are 9 position/level on each axis. The following diagram shows the indentation levels of Corner 1 on the x-axis.

Neighbors

Table 7: Neighbor Order

ID	Relative Coordinates
0	(-1, -1, -1)
1	(-1, -1, 0)
2	(-1, -1, 1)
3	(-1, 0, -1)
4	(-1, 0, 0)
5	(-1, 0, 1)
6	(-1, 1, -1)
7	(-1, 1, 0)
8	(-1, 1, 1)
9	(0, -1, -1)
10	(0, -1, 0)
11	(0, -1, 1)
12	(0, 0, -1)
13	(0, 0, 1)
14	(0, 1, -1)
15	(0, 1, 0)
16	(0, 1, 1)
17	(1, -1, -1)
18	(1, -1, 0)
19	(1, -1, 1)
20	(1, 0, -1)
21	(1, 0, 0)
22	(1, 0, 1)
23	(1, 1, -1)
24	(1, 1, 0)
25	(1, 1, 1)

Format Specification

Using this *binary format syntax*.

Cube Types

0 - EMPTY The cube does not exist, nothing to render.

1 - SOLID One solid cube.

2 - NORMAL An indented cube, with at least one intended corner.

3 - OCTANT The octree is subdivided into 8 sub cubes.

Note: The Format numbers are just to difference between the formats and not to describe an versioning.

Sauerbraten

This part only shows, how Sauerbraten saves the octrees in general. It does not show the whole format. As Sauerbraten has one cube type more, the formats can only be compared to Inexors partially.

```
| ENDIANNESS : little
| bit : 1 // A bit, 0 or 1.
| uByte : 8 // An unsigned byte.

def get_cube() {
    > uByte (1) : cube_type // cube type, actually only 3 bits are used, but only 4␣
↪types (the first two bits, can be compared to our cubes)
    switch (cube_type) {
        case 0: // octant
            // nothing
        case 1: // empty
            // nothing
        case 2: // solid
            // nothing
        case 3: // normal
            for (0..11 : corner_number) {
                > uByte (1) // edge indent
            }
    }
} // get_cube
get_cube()
```

Inexor I

File Extension: `.nxoc` - Inexor Octree

```
| ENDIANNESS : little
| bit : 1 // A bit, 0 or 1.
| uByte : 8 // An unsigned byte.
| uInt : 32 // An unsigned integer.

> uByte (13) // string identifier: "Inexor Octree"
> uInt (1) // version

def get_cube() {
    > bit (2) uByte : cube_type // cube type

    switch (cube_type) {
        case 0: // empty
            // nothing
        case 1: // fully
            // nothing
        case 2: // indented
            for (0..7 : corner_number) {
                > bit (1) : x_axis // is x axis indented
                if (x_axis == 1) { // x axis is indented
                    > bit (3) // indentation level as value + 1
```

(continues on next page)

(continued from previous page)

```
                }
                > bit (1) : y_axis // is y axis indented
                if (y_axis == 1) { // y axis is indented
                    > bit (3) // indentation level as value + 1
                }
                > bit (1) : z_axis // is z axis indented
                if (z_axis == 1) { // z axis is indented
                    > bit (3) // indentation level as value + 1
                }
            }
        case 3: // octants
            for (0..7 : sub_cube) {
                get_cube() // recurse down
            }
    }
} // get_cube
get_cube()
```

The corner position at one axis is calculated relative from the corner starting as 0 + indentation level.

Inexor II

File Extension: `.nxoc` - Inexor Octree

```
| ENDIANNESS : little
| bit : 1 // A bit, 0 or 1.
| uByte : 8 // An unsigned byte.
| uInt : 32 // An unsigned integer.

> uByte (13) // string identifier: "Inexor Octree"
> uInt (1) // version

def get_cube() {
    > bit (2) uByte : cube_type // cube type

    switch (cube_type) {
        case 0: // empty
            // nothing
        case 1: // fully
            // nothing
        case 2: // indented
            for (0..11 : edge_id) {
                > bit (2) uByte : indent // edge indentation
                switch (indent) {
                    case 0: // not indented
                        break
                    case 1: // end corner is indented
                    case 2: // start corner is indented
                        > bit (3) // indentation offset, starting from the specified␣
↪corner
                        break
```

(continues on next page)

(continued from previous page)

```
                    case 3: // both sides indented
                        > bit (5) // indentation level and offset, see below for more␣
↪information
                    }
                }
            case 3: // octants
                for (0..7 : sub_cube) {
                    get_cube() // recurse down
                }
        }
} // get_cube
get_cube()
```

Calculating edge indentation value

The indentation along the edge axis between two corners presented by a unique value. The indentation level starts with 0 at the starting corner and goes to 8 at the ending corner. We are assuming that both ends of the edge are indented by at least one. (Start at 0, is actually already indented by one).

Using i as the indentation value, s as the indentation start position and o as the offset between the start and end position. $i = 8 * s + o - \frac{s^2+s}{2}; s, o \in [0, 6]; s <= o$

Resulting into values from 0 to 27.

Inexor III

The third format takes advantage of the second format for the double-sided indentations, but makes sure that it is easy to read and write and not many bitwise operations have to be done. Also the cube type is presented by one byte, even if only the first two bits are used.

File Extension: `.nxoc` - Inexor Octree

```
| ENDIANNESS : little
| bit : 1 // A bit, 0 or 1.
| uByte : 8 // An unsigned byte.
| uInt : 32 // An unsigned integer.

> uByte (13) // string identifier: "Inexor Octree"
> uInt (1) // version

def get_cube() {
    > uByte (1) : cube_type // cube type, only the first two bits are used.

    switch (cube_type) {
        case 0: // empty
            // nothing
        case 1: // fully
            // nothing
        case 2: // indented
            for (0..11 : edge_id) {
                > bit (6) // indentation level and offset, see below for more information
            }
        case 3: // octants
```

(continues on next page)

(continued from previous page)

```
            for (0..7 : sub_cube) {
                get_cube() // recurse down
            }
    }
} // get_cube
get_cube()
```

Calculating edge indentation value

The indentation along the edge axis between two corners presented by a unique value. The indentation level starts with 0 at the starting corner and goes to 8 at the ending corner.

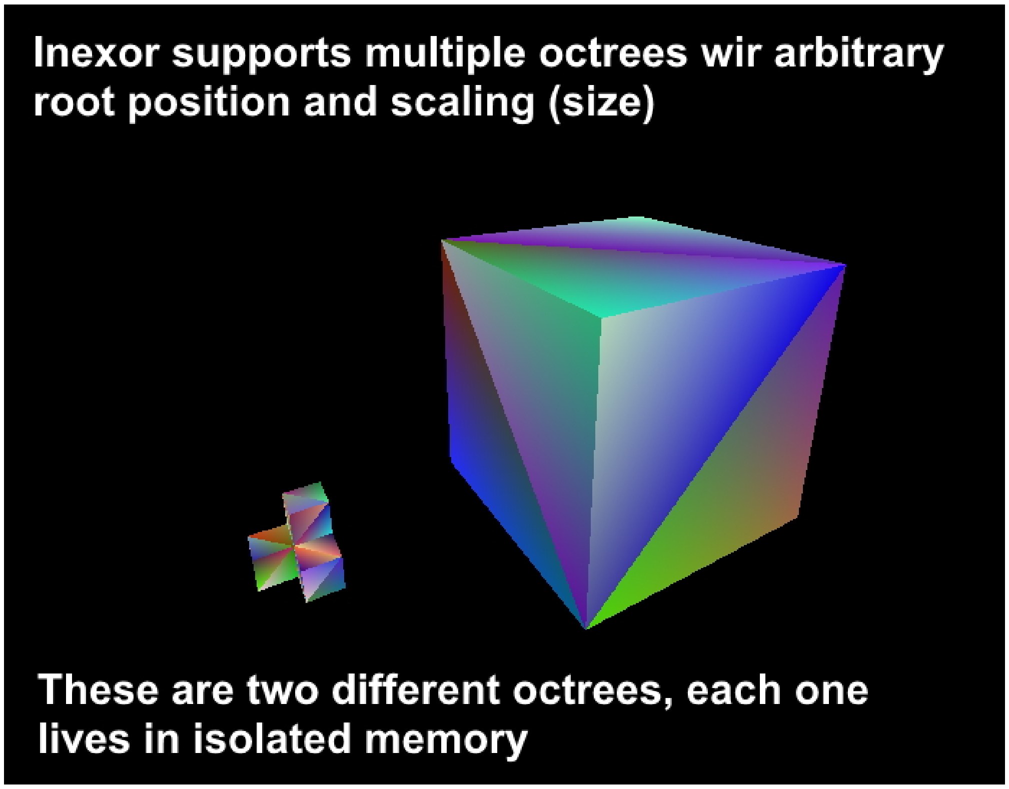
Using i as the indentation value, s as the indentation start position and o as the offset between the start and end position.

$i = 10 * s + o - \frac{s^2+s}{2}; s, o \in [0, 8]; s <= o$

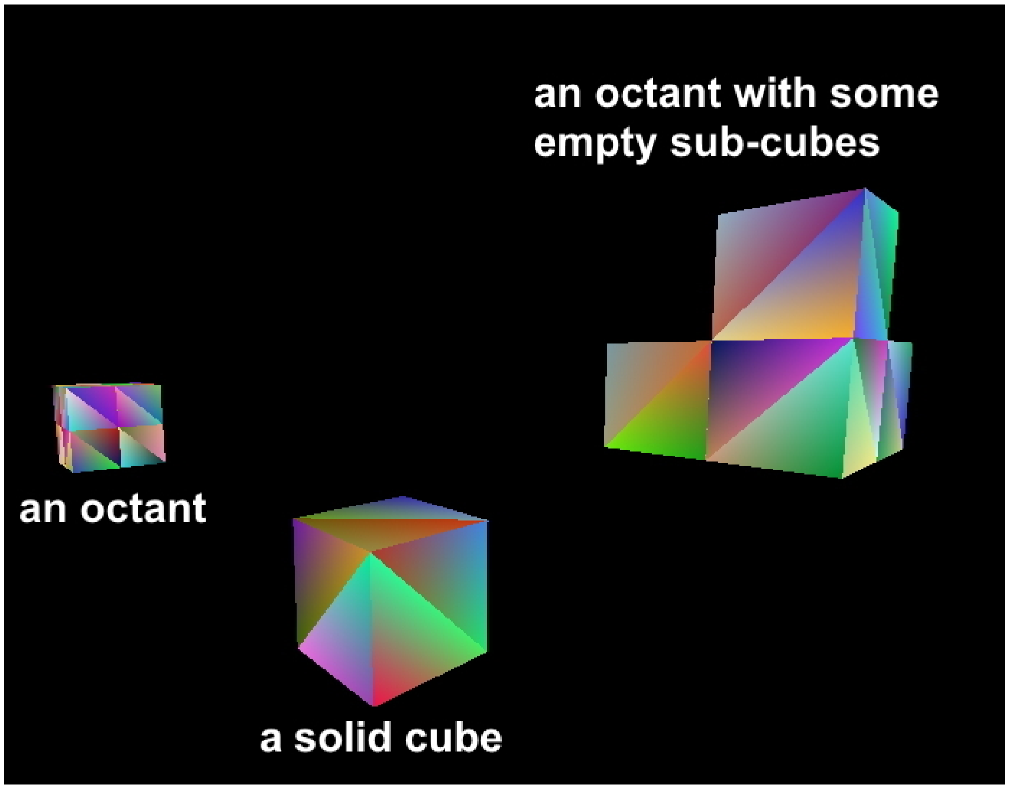
Resulting into values from 0 to 44.

Multi octree collision detection

Octree collision detection allows us to find intersections between octree geometry and a ray, for example the camera view ray. This is an essential part for the octree editor. Inexor engine allows to have multiple octrees with arbitrary position and relative size (no support for rotations yet), making collision detection significantly more complex than single octree traversal:


Inexor supports multiple octrees wir arbitrary root position and scaling (size)
These are two different octrees, each one lives in isolated memory

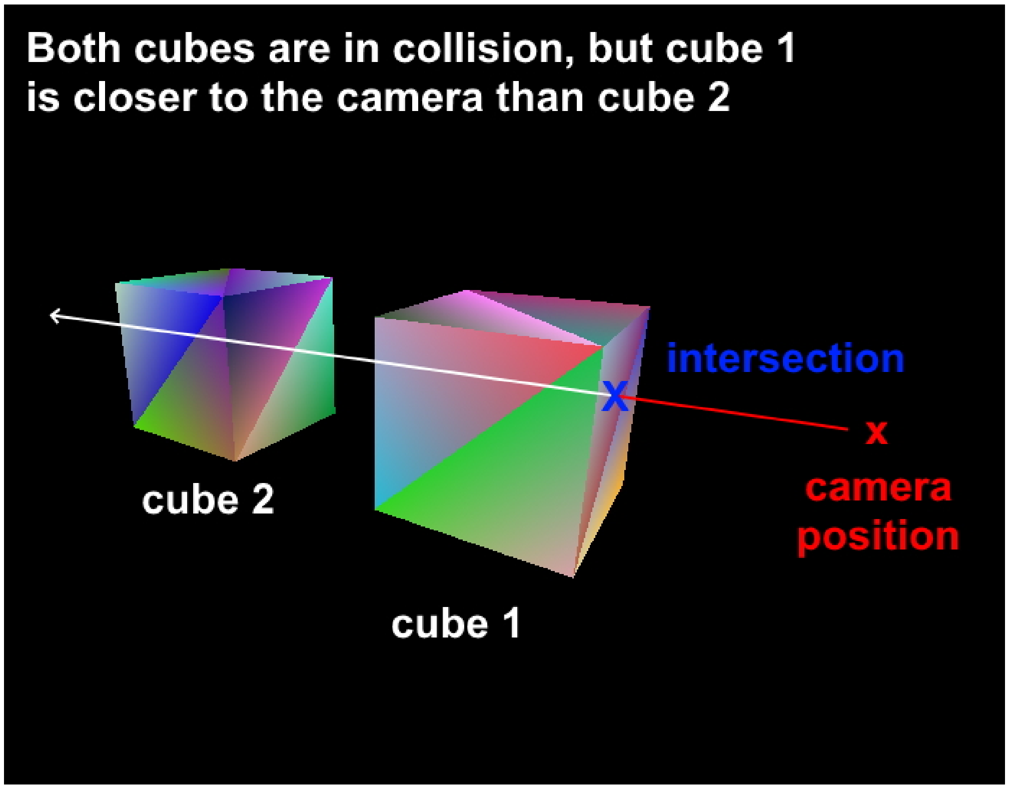
In the following screenshot, you can see three octrees of different types and different sizes. The octree in the left has 8 sub-cubes (we call a cube which has 8 children `Cube::Type::OCTANT` in the engine, even if some of them are empty). The one octree in the midle has no children, it's just one solid cube (we call it `Cube::Type::SOLID`). The one in the right has some empty and some solid sub-cubes in it (that's also an `Cube::Type::OCTANT`). You can see that these cubes are not indented at all, because indentation is not taken into account yet for octree collision.[1]



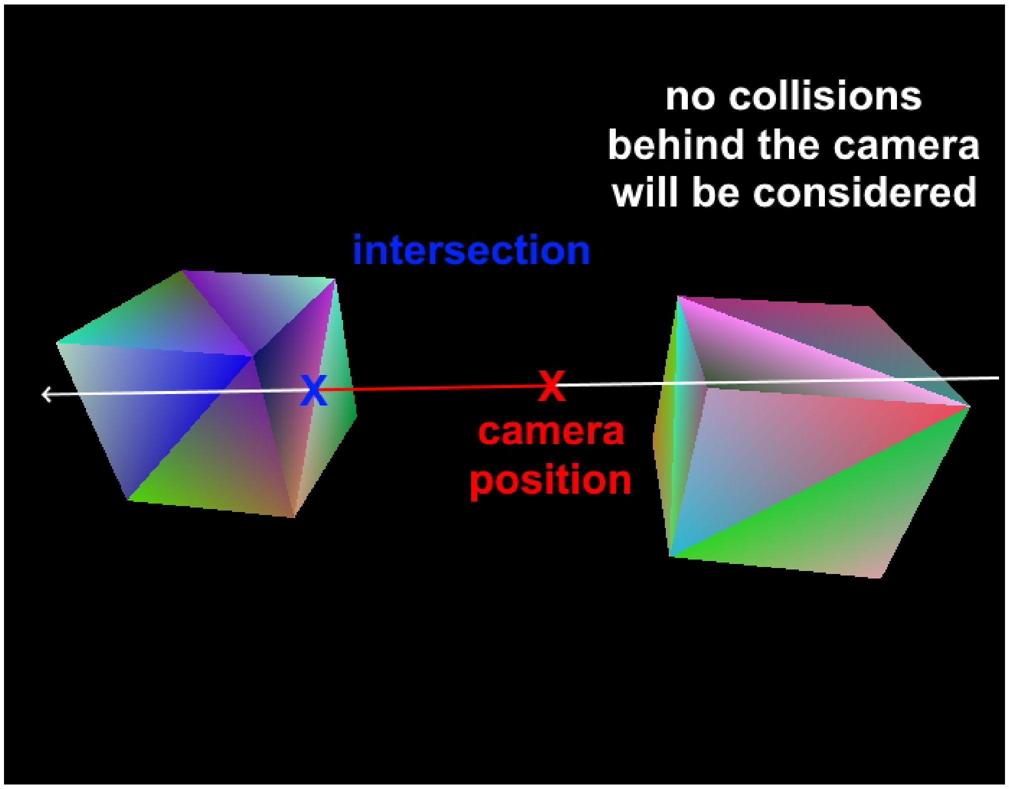
How to find collisions between octree geometry and a ray in this scene now? For simplicity, we assume that the octrees are not intersecting each other. Let's assume we want to write an octree editor. We are obviously only interested in the intersection which is closest to the camera's position: if there is another octree behind the current selection, we must move the camera to it, in order to be able to edit it:[2]

[1] The current implementation of octree-ray intersection only checks for intersections with completely filled cubes and does not take into account indentations of cubes, as this is not required for an octree editor. The bounding box of an octree is always unchanged, even if the octree geometry itself has indentations. Taking into account indentations will be required for physics calculations in the future, for example to check collisions between particles and octree.

[2] We could also make the layer which is blocking view invisible for a moment in the future.


Both cubes are in collision, but cube 1 is closer to the camera than cube 2
intersection
x
camera position
cube 2
cube 1

We are also only interested in collisions which are in front of our camera view:



Let's imagine we now have N octrees, and we want to find all those which collide with the ray and we want to know the one which is closest to the camera. Furthermore, since we want to write an octree editor, we want not only the cube which is in selection, but we also want to know which one of the 6 faces of the cube is in selection. In addition, we want to know the coordinates or the intersection between camera ray and the plane of the selected face. We also need the closest corner on the selected face and the closest edge, just so we have all the data we could possibly need for implementing the editor. This leaves us the following questions:

- How do we even determine if there are any collisions occuring at all?
- How do we now find out which of the N octrees is in selection?
- How do we determine the octree which is closest to our camera's position?

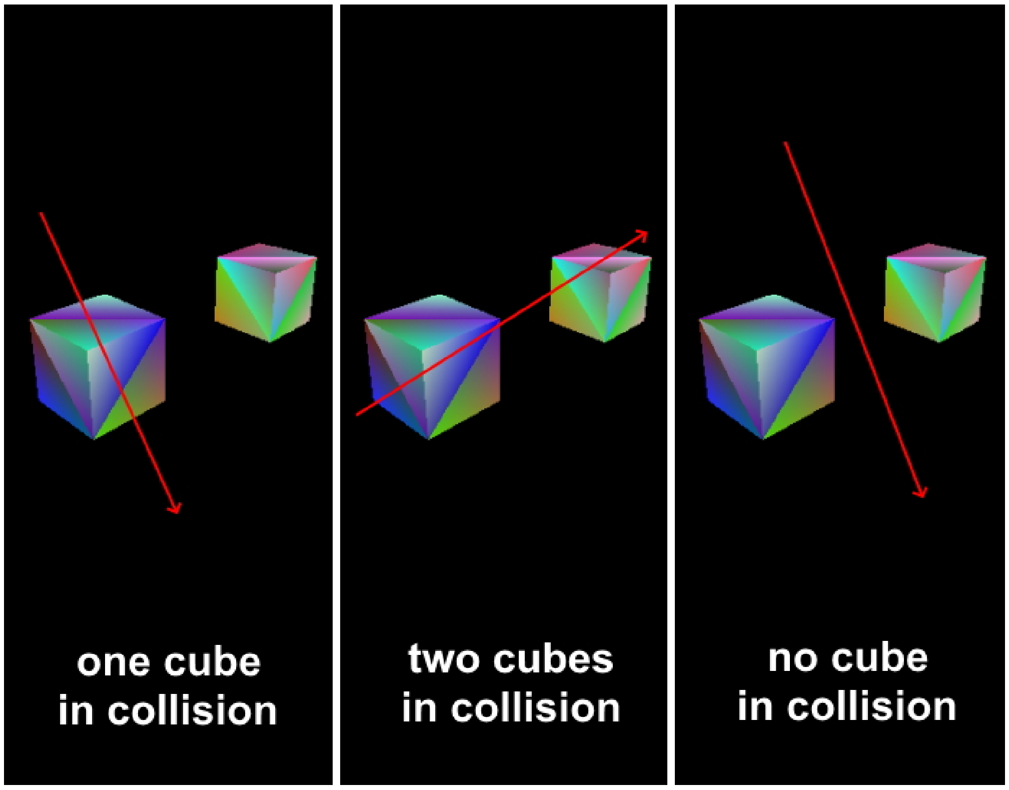
Finding the octree closest to the camera

Assuming we have N octrees, the first thing we do is to iterate through every one of the N octrees and to check for collision of the camera ray with the bounding sphere of the octree. This is a quick way to optimize the collision in the beginning and to save a lot of computation time. It is a common trick in computer graphics. If we would check for every possible collision without this step, the algorithm would be way too slow. So we need to iterate through the N octrees we have and calculate the distance d between the ray and the center of the octree's bounding sphere. In our engine, the center of the octree is also the center of the bounding sphere. We are using glm::intersectRaySphere to check if a collision is happening. If the bounding sphere check was successful, we also check collision of the ray with the axis aligned bounding box (aabb). This check is more expensive but also more precise than the bounding sphere check. However it is only used if the bounding sphere check previously was successful to save performance.

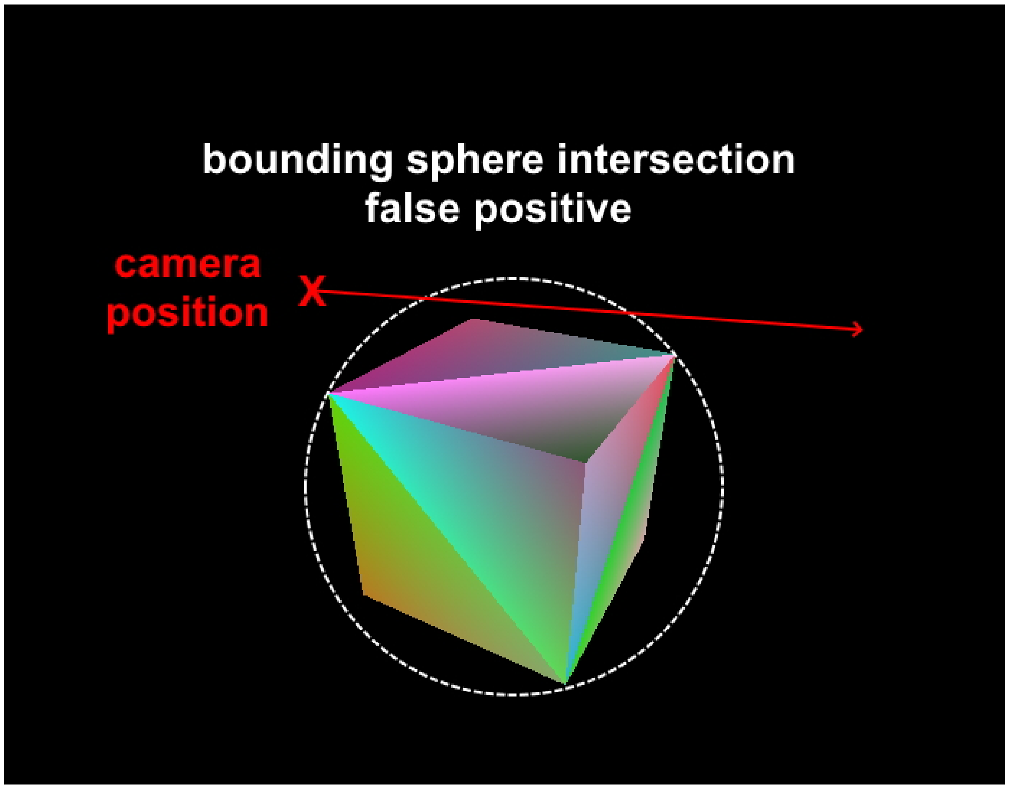
Note: Simply iterating through all N octrees is a naive approach. This only works for small number of octrees. Much better would be to use a hierarchical data structure like a bounding volume hierarchy, which groups objects which are close to each other into a unified bounding sphere. This hierarchical bounding sphere check is much faster than iterating through all N octrees. There are libraries which could help implement this for Inexor in the future.

Note: Currently we use the entire octree as axis axis aligned bounding box (aabb). However, we could optimize this: We could fit the bounding box to only the filled cubes of that octree. For example if one half side of the octree is empty, we could adjust the bounding box to the other half. If a camera ray now collides with the empty part of the octree, this could give us improved performance, as the bounding box is not hit. Otherwise the subcube iteration would be executed and come to the same conclusion: only empty subcubes are hit and therefore no collision takes place.

After this step, we have 0 to N octrees which collide with the ray. The following screenshot shows the possible situations for $N = 2$:



If we have 0 collisions, we can already stop collision detection here because there are no collisions occuring: if a camera ray intersects an octree, it must also intersect the bounding sphere. The reverse statement is not true: if a ray collides with a bounding sphere, that does not mean it collides with the octree. It could be a false positive:



We now need to find the octree which is closest the camera. Even if the camera is inside of an octree, there could be multiple octrees which have bounding spheres that intersect the camera ray. The first thing which comes to our mind is sorting the octrees by distance to the camera: we could calculate the distance d between camera's position and bounding sphere's center (= the octree's center) for every one octree which intersects with the camera ray and order them by distance:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

The one with the lowest distance will be the one which is closest to the camera. This should be the octree we will perform any further detailed collision checks on. However, there are two things we can already optimize here.

The square of the distance

First, we do not need to sort the octrees by distance. Sorting would mean we need all of the data sorted by distance. We are only interested in the octree with the smallest distance though. Since we iterate through all of them, we check if the calculated distance d between bounding sphere's center and camera position is smaller than the stored value, and if that is the case, store it as the new closest octree.[3] This is significantly faster than sorting all octrees. We also lose information about the distance to all the other octrees in selection, but that's not important at the moment (at least for the octree editor that is irrelevant for now). As a second optimization, we should not calculate the distance d between the bounding sphere's center and the camera's center, as we are not interested in the exact value of the distance. The reason we should avoid this is because distance calculation using glm::distance makes an expensive sqrt call, as it needs to calculate the distance like this:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

If we take this equation and square both sides, we obtain d^2, the squared distance:

$$d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

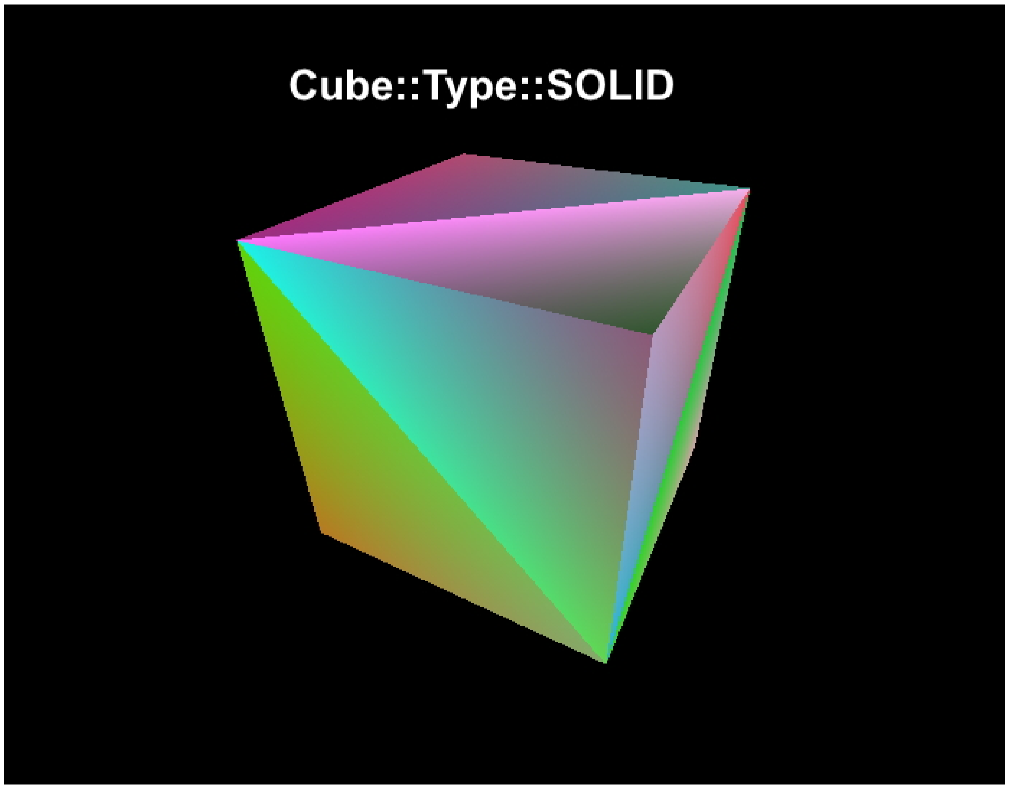
This way, we **perform no square root calculation**. The squared distance d^2 will serve as our value for determination of the closest octree. Think about it: if the distance d is the value which allows us to find the closest octree, the square of the distance d^2 will work as well. If you take N octrees, each one having a distance d to the camera's position, the order will not change if we square the distance.

Note: For simplicity, we assume that the octrees have a variable position and size, but are **not intersecting each other**. If that is the case, the determination of the octree which is closest to the camera would be more complicated. For example if there would be two octrees, one being closer to the camera than the other, but the one further away has a bigger size, maybe resulting in faces which are closer to the camera than the other cube. We will implement support for this in the future.

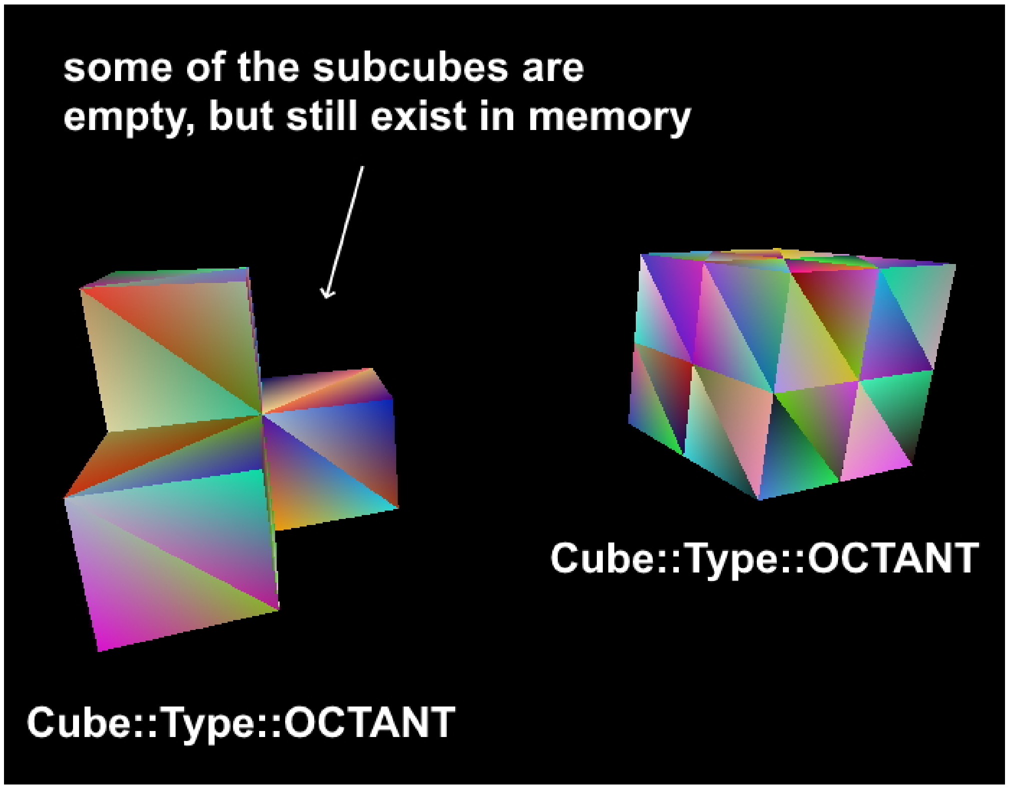
Finding the leaf node

Now that we have found the octree which is closest to the camera, we need to find a leaf node in the octree which is being intersected. The most simple case would be if the octree's root is of type `Cube::Type::SOLID`, as completely filled octrees are leaf nodes by definition:

[3] To do so, we need to set the initial value of the distance to a maximum value. We use `std::numeric_limits<float>::max()`



If the octree's root is of type `Cube::Type::OCTANT`, we need to iterate through all 8 sub-cubes. This is described in the next section.



Please note that every octant has 8 sub-cubes, even if some (or even all) of them are of type `Cube::Type::EMPTY`.[4]

Note: Technically, the octree's root could also be of type `Cube::Type::EMPTY`. In this case, there also no collision possible. However, such octrees will be skipped when iterating through all possible sub-cubes which could possibly collide with the ray.

Subcube iteration

So we found the octree which is closest to the camera, but it's neither completely empty (`Cube::Type::EMPTY`) nor completely filled (`Cube::Type::OCTANT`). We now simply iterate through all 8 sub-cubes and repeat the bounding sphere and axis aligned bounding box collision checks for every subcube. If a subcube is empty, no collision with it is possible and it will be excluded from detailed collision checks. We now need to find the sub-cube which is closest to the camera again. We therefore perform the same search by squared distance as we already did for the octree octrees. We simply calculate the squared distance from the center of the sub-cube to the camera and if the distance is lower than the one which is currently stored, we accept it as new closest sub-cube. Imagine a cube is an octant and it has 8 sub-cubes which are all not empty. If a ray goes through that cube, no more than 4 sub-cubes can be intersected. Therefore we abort the hit collection after 4 hits. Once we determined the sub-cube which is closest to the camera, we recursively perform this algorithm. The iteration depth can be limited in the engine. A common example of this is the grid size of the octree editor. So a leaf node is either found if the current subcube is of type `Cube::Type::SOLID` or if the iteration depth has been reached. Once a leaf cube was found, we proceed to calculate the selected face, as described in the following section.

Note: Every cube of type `Cube::Type::OCTANT` has 8 subcubes. Iterating through all subcubes from index 0 to 7 is a naive approach as well. Inexor should use a fast octree traversal algorithm in the future. For more information, check out this paper. Also check out the hero algorithm.

Determination of selected face

Now that we have found the selected cube, we need to determine on which one of the 6 faces (left, right, top, bottom, front, back) the collision takes place. We are only interested in the intersection which is facing the camera. That is also the intersection which is closer to the camera position. There is also a backside intersection from the outgoing ray, but we are not interested in this for now. There are several ways how to determine which face is in collision. We decided to use the following approach: first we filter out all sides of the cube which are not facing the camera. In order to do so, let's take a look at the following equation which describes the angle α of two vectors $\vec{a}$ and $\vec{a}$:

$$cos(\alpha) = \frac{\vec{a}\cdot\vec{b}}{|a|\cdot|b|}$$

If we define $\vec{a}$ as the normal vector on the face and $\vec{b}$ as the camera direction vector, we realize that the normal vector on the cube's face is no longer facing the camera if the angle α becomes greater than `90 degrees`. We now think we should rearrange for the angle:

$$\alpha = cos^{-1}\left(\frac{\vec{a}\cdot\vec{b}}{|a|\cdot|b|}\right)$$

However, we can simplify this: If the angle is slightly greater than `90 degrees`, the value of $cos(\alpha)$ becomes smaller than `0`. If the angle is a little less than `90 degrees`, $cos(\alpha)$ becomes greater than `0`. If we take a look at the right side of the equation we started with, we can see that the dot product of $\vec{a}$ and $\vec{b}$ is in the nominator while the product of the magnitudes is in the denominator. Since the magnitude of a vector is never negative, the product of two magnitudes

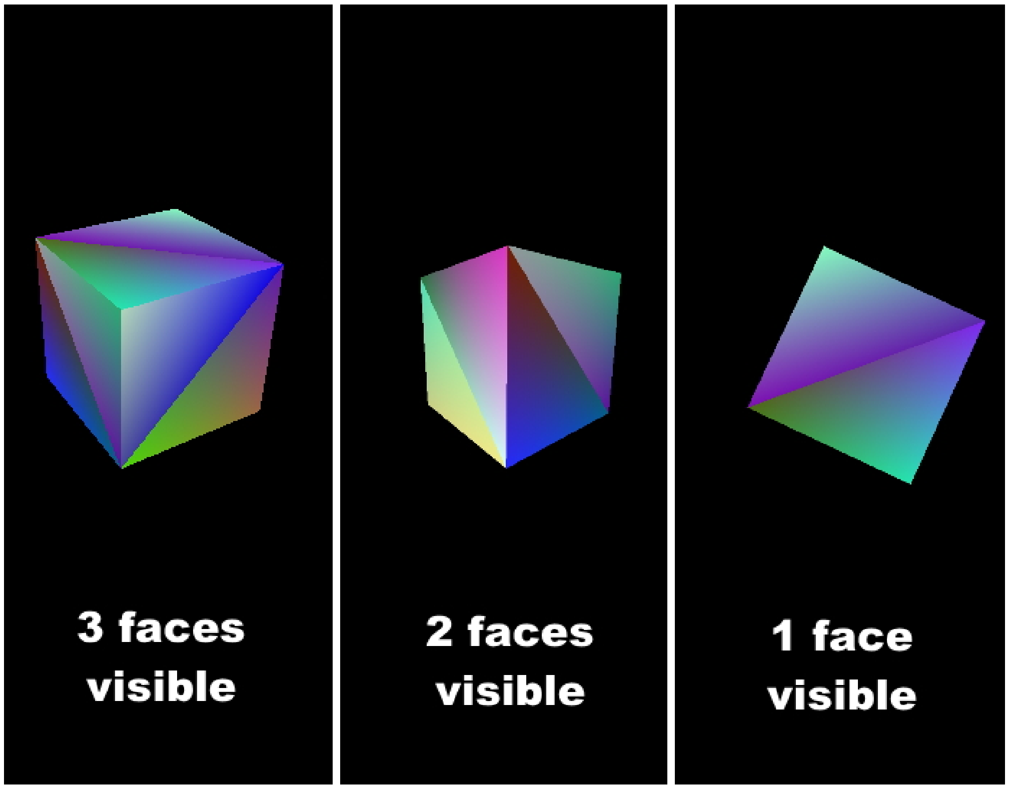
[4] This has to do with the way the engine lays out memory for the octree data structure. The engine will allocate memory for the empty sub-cube because it's faster to change the sub-cube's data if it gets modified. However, empty sub-cubes will not result in additional vertex or index data being generated.

will always be positive. We now see that the sign change is entirely dictated by the nominator. Furthermore, we already elaborated that it's comparably expensive to calculate the square root. We can simplify all this to the following condition: the face on a cube is visible, if the dot product of the two vectors $\vec{a}$ and $\vec{b}$ is smaller than zero:

$\alpha < 0$ for $\vec{a} \cdot \vec{b} < 0$

This is quite nice, because the dot product of $\vec{a}$ and $\vec{b}$ is a cheap calculation. This is another very popular trick in computer graphics.[5]

We now simply iterate through all 6 faces of the cube, take the normal vector on that cube face and check if it's facing the camera. We are only interested in the planes which are facing the camera.[6] If you look at a cube, no more than 3 sides can be visible at the same time. This means we can stop after we found 3 cube sides which are facing the camera. It could be less than 3 sides though. Imagine you are right on top of a solid cube and your look down on it, only the top side is visible. If you look from a certain position, only 2 sides are visible.

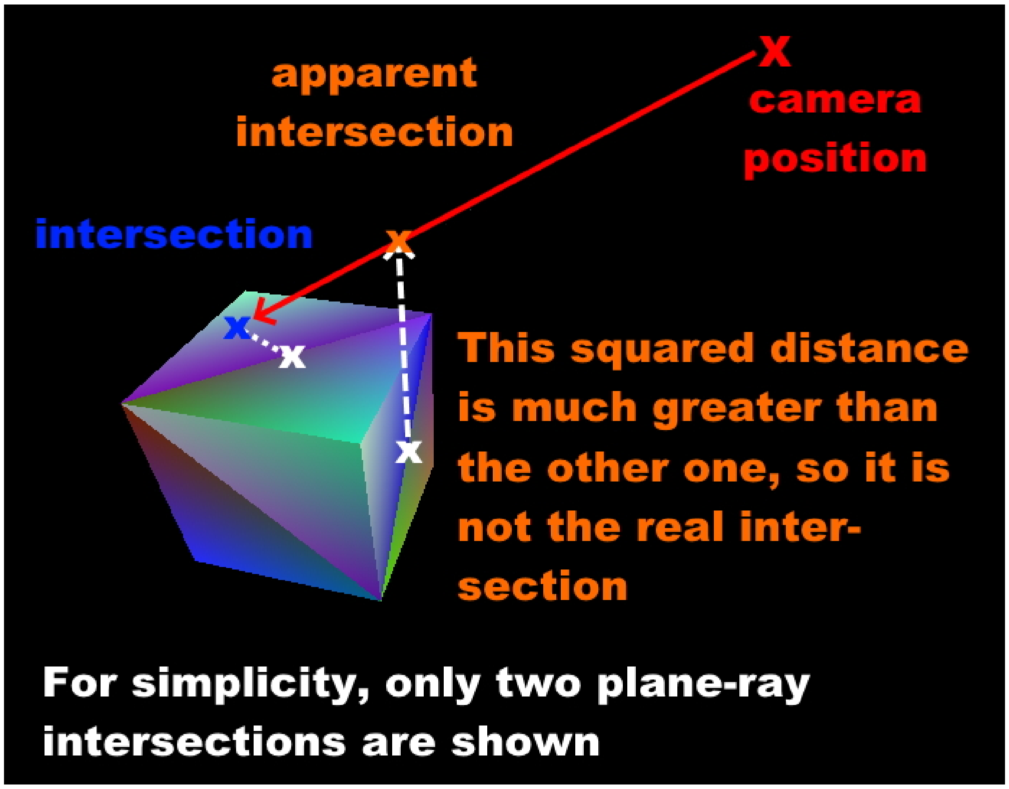


Note: We could optimize this in the future by doing some coordinate checks of the camera and the octree. For example if the `x` and `y` coordinates are inside the square of the cube, we could only see top or bottom of the cube. However, since Inexor wants to account for arbitrary rotations around all 3 axis, this is more complex than for unrotated octrees. We think our current solution is sufficiently performant.

We now have 3 or less sides of the cube facing the camera. We calculate the intersection point between the ray and every plane which represents a cube face. In order to determine the real intersection, we come back to searching the lowest squared distance again. However, it is important to state that we can't use the squared distance to the camera position in this case. We must calculate the squared distance between the intersection point on every plane and the center of the cube's face which is associated to this plane. This way, we find the real intersection point and the selected corner:

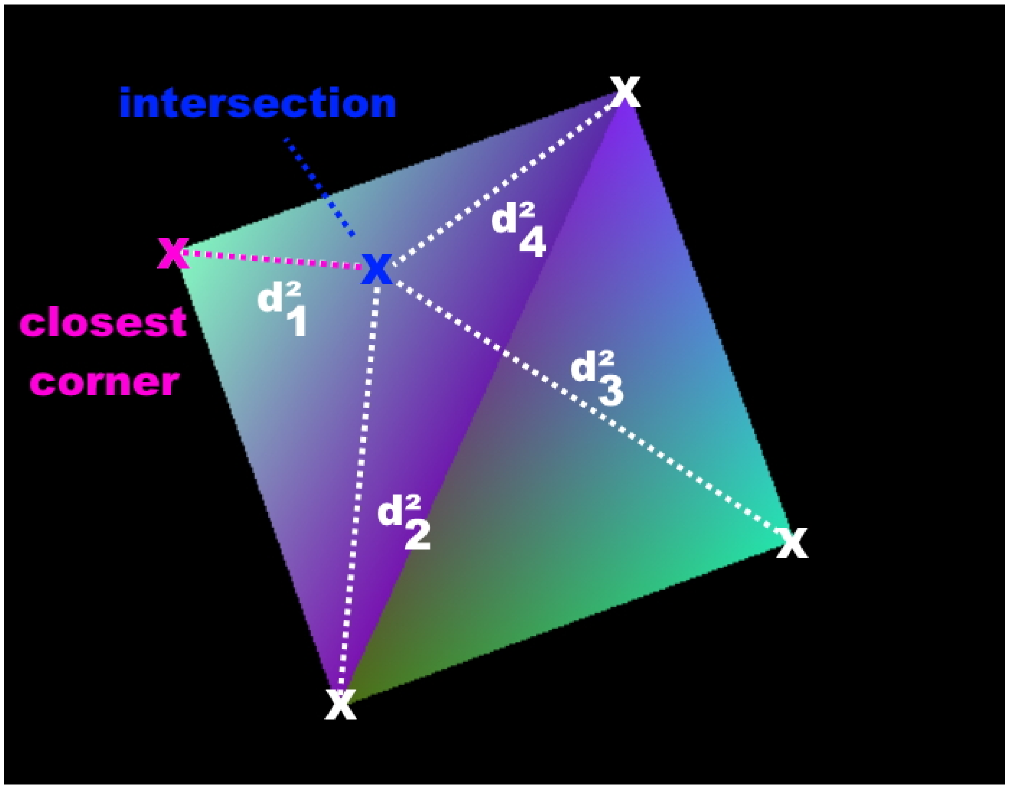
[5] In fact this is used during the rasterization step in rendering to discard all triangles which are not facing the camera.

[6] For some reasons we might be interested in those sides of a cube which are not facing the camera in the future?


apparent intersection
camera position
intersection
This squared distance is much greater than the other one, so it is not the real inter-section
For simplicity, only two plane-ray intersections are shown

Calculation of closest corner

We now successfully determined the selected face and the intersection point. We already know the coordinates of every one of the 4 corners on that face. In order to determine the nearest corner, we come back to calculating the squared distance between the intersection point and every corner point. The corner with the lowest squared distance is the nearest.



Calculation of closest edge

The determination of the closest edge works the same way as the determination of the closest corner: searching the lowest squared distance between intersection point and center of the four edges on the selected face. To find the edges which are associated to the selected size, the following array is used. The indices of edges are the same as in the octree documentation:

```
using edge_on_face_index = std::array<std::size_t, 4>;

// These indices specify which 4 edges are associated with a given face of the bounding␣
↪box.
static constexpr std::array BBOX_EDGE_ON_FACE_INDICES{
    edge_on_face_index{0, 1, 2, 3},   // left
    edge_on_face_index{4, 5, 6, 7},   // right
    edge_on_face_index{1, 5, 8, 11},  // front
    edge_on_face_index{3, 7, 9, 11},  // back
    edge_on_face_index{0, 4, 10, 11}, // top
    edge_on_face_index{2, 6, 8, 9}    // bottom
};
```

Closing remarks

With this algorithm, we have a good starting point writing an octree editor. However, we know that this is not the fastest solution possible. Nevertheless, it is a solution which is easy to understand, easy to improve and easy to optimize for sure. Furthermore, it will be easy to parallelize it. All the aspects which could be improved have been listed on this page.

Management of command pools and command bufers

Inexor engine's command pool and command buffer management code is based on the following resources:

- Tips and Tricks: Vulkan Dos and Don'ts
- Writing an efficient Vulkan renderer
- Themaister's Granite renderer
- Multi-Threading in Vulkan

From that we concluded our solution should look like this:

- The device wrapper should be exclusively responsible for the management of command pools and command buffers
- The command pool and command buffer management system must be thread safe
- We should have only one command pool per thread per queue (as is recommended)
- Each command buffer should be allocated from the command pool which is associated to the current thread
- Command buffers must be reused instead of being allocated and destroyed every time
- We should abstract command buffers as much as possible
- The solution must support an arbitrary number of threads and an arbitrary number of command buffers per thread
- Command buffer recording should be done with the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` flag
- We should start with graphics command pools only and work on other pool types (transfer, compute..) later (GitHub issue https://github.com/inexorgame/vulkan-renderer/issues/486)

Note: Do not forget that we need to specify on which queue the command pool will operate on. We are currently using the graphics queue for copy operations, but there are other queue types such as transfer queues or compute queues. We will need to create separate command pools on a per-thread basis once we want to use those queues.

Thread-local command pools

The command pools are managed exclusively by the device wrapper class. Inside of it, there is a std::vector of std::unique_ptr of the CommandPool wrapper class:

```
std::vector<std::unique_ptr<CommandPool>> m_cmd_pools;
std::mutex m_mutex;
```

Note: It must be a `std::vector<std::unique_ptr<CommandPool>>` and not a `std::vector<CommandPool>` because std::vector is allowed to reorder the memory and we want to store a pointer into the std::vector!

Since the command pools are owned by a thread-global device object, the vector must be protected by a mutex. But how do we set up a command pool on a per-thread basis and how do we get the command pool for the current thread? To do this, we have a non-owning `thread_local` pointer to the thread's command pool which points to an entry inside the device's command pool vector, and is initialized the first time the getter is called.

```
CommandPool &Device::thread_graphics_pool() {
    // Note that thread_graphics_pool is implicitely static!
    thread_local CommandPool *thread_graphics_pool = nullptr; // NOLINT
    if (thread_graphics_pool == nullptr) {
        auto cmd_pool = std::make_unique<CommandPool>(*this, "graphics pool");
        std::scoped_lock locker(m_mutex);
        thread_graphics_pool = m_cmd_pools.emplace_back(std::move(cmd_pool)).get();
    }
    return *thread_graphics_pool;
}
```

This get method is private. It is used only internally for the command buffer request system, as explained in the next section.

Command buffer request system

Inexor engine uses a command buffer request system. If you need to record and submit a command buffer in any place in the engine code, you can call `m_device.request_command_buffer()`. You should have a reference to the device wrapper `m_device` available in the part of the code you want to use the command buffer. The command buffers are managed by the command pool wrapper and can only be accessed through the the wrapper. However, the command pools themselves are managed by the device wrapper. This means command pools are never directly exposed in the rest of the engine code. You request a command buffer from the device wrapper, and the request will be redirected internally to the thread local command pool:

```
const CommandBuffer &Device::request_command_buffer(const std::string &name) {
    return thread_graphics_pool().request_command_buffer(name);
}
```

The request method of the command pool wrapper tries to find a command buffer which is currently not used anywhere else. It does so by testing the state of the command buffer's fence. If no free command buffer is found, a new one is simply allocated. Note that this is thread local, so we need no synchronization here. Note that the command buffer request method resets the command buffer's fence. The request method will call `begin_command_buffer` before returning the requested command buffer.

```
const CommandBuffer &CommandPool::request_command_buffer(const std::string &name) {
   // Try to find a command buffer which is currently not used
   for (const auto &cmd_buf : m_cmd_bufs) {
       if (cmd_buf->fence_status() == VK_SUCCESS) {
            // Reset the command buffer's fence to make it usable again
            cmd_buf->reset_fence();
            m_device.set_debug_marker_name(*cmd_buf->ptr(), VK_DEBUG_REPORT_OBJECT_TYPE_
↪COMMAND_BUFFER_EXT, name);
            cmd_buf->begin_command_buffer();
            return *cmd_buf;
        }
   }
   // We need to create a new command buffer because no free one was found
   // Note that there is currently no method for shrinking m_cmd_bufs, but this should␣
↪not be a problem
```

(continues on next page)

(continued from previous page)

```
    m_cmd_bufs.emplace_back(std::make_unique<CommandBuffer>(m_device, m_cmd_pool,
↪"command buffer"));
    return *m_cmd_bufs.back();
}
```

After this, you can use it to record and submit your command buffer. You can also use the `execute` method, as explained in the next section.

Device wrapper's execute method

To automate beginning and ending of command buffer recording and submission, we created the execute method of the device wrapper. This is quire helpful and it is recommended to use it instead of requesting command buffer handles manually. The execute method takes a lambda as argument and requests a command buffer. After execution, it calls `submit_and_wait`. For debugging purposes, it also assigns a debug name to the command buffer which executes your lambda:

```
void Device::execute(const std::string &name, const std::function<void(const␣
↪CommandBuffer &cmd_buf)> &cmd_lambda) {
    // TODO: Support other queues (not just graphics)
    const auto &cmd_buf = thread_graphics_pool().request_command_buffer(name);
    // Execute the lambda
    cmd_lambda(cmd_buf);
    cmd_buf.submit_and_wait();
}
```

Note: Note that `execute` will wait for the command buffer submission and execution to complete using a fence, meaning it's a blocking operation. In case you don't want this, you should be experienced enough to use the `request_command_buffer` method manually and to do your synchronization yourself.

Here is an example for an image copy operation which uses the execute method:

```
m_device.execute(m_name, [&](const CommandBuffer &cmd_buf) {
    cmd_buf.change_image_layout(m_texture_image->get(), VK_IMAGE_LAYOUT_UNDEFINED, VK_
↪IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL)
            .copy_buffer_to_image(texture_data, static_cast<VkDeviceSize>(texture_size), m_
↪texture_image->get(), copy_region, m_name)
            .change_image_layout(m_texture_image->get(), VK_IMAGE_LAYOUT_TRANSFER_DST_
↪OPTIMAL, VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL);
});
```

Note: Inexor engine's command buffer request system does not differentiate between a normal command buffer and a command buffer which is used for single submission. In fact, all command buffers have the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` when calling `begin_command_buffer`.

1.1.12 GitHub Issue Labeling

The labels are written in two parts, separated by a colon. The first part is the label category, the second the name.

Table 8: Categories

Abbreviation	Name	Description
cat	category	main category
feat	feature	specific feature
prio	priority	priority
org	organization	organization/ whats the state of this issue
diff	difficulty	which skill is required to work on this
plat	platform	platform specific issue

Some labels are mutual to each other like `org:in progress` and `org:on hold`. Also some labels should not be specified on closed issues like `org:in progress`.

Note: Github will use black as font color if the background is too light.

Preview	Category	Name	Description	Co
	cat	benchmark	testing code performance with automated benchmarking	#C
	cat	bug	bug/error/mistake which limits the program	#F
	cat	dependency	dependency management	#B
	cat	dev tools	building/ compiling the program, cmake configuration and general development tools	#7
	cat	documentation	documentation	#D
	cat	enhancement	enhancement/requested feature/update of existing features	#0
	cat	performance	performance	#F
	cat	refactor	refactor/clean up/simplifications/etc.	#F
	cat	review	review	#6
	cat	security	security/ privacy issues	#B
	cat	testing	testing	#C
	diff	first issue	good first issue to start contributing	#F
	diff	beginner	beginner skills required	#F
	diff	intermediate	intermediate skills required	#F
	diff	advanced	advanced skills required	#F
	diff	expert	expert skills required	#F
	org	discussion	needs further discussion with others	#D
	org	duplicate	duplicate issue	#D
	org	help wanted	help wanted	#D
	org	idea	needs further elaboration before it is possible to continue	#D
	org	in progress	somebody is working on this	#D
	org	invalid	invalid issue	#D
	org	on hold	on hold, until …	#D
	org	planned	planned/ prepared issue	#D
	org	third party	this issue depends on a third party project and is out of our hands	#D
	org	triage	labels have to be specified	#D
	org	wontfix	wont be fixed, ever	#D
	feat	ci	continuous integration	#D
	feat	concurrency	multithreading, asynchronous events, concurrency	#D
	feat	render graph	render graph	#D

continues on n

Table 9 – continued from previous page

Preview	Category	Name	Description	Co
	feat	gui	graphical user interface	#D
	feat	input	keybord/mouse input	#D
	feat	lightning	light system	#D
	feat	logging	logging system	#D
	feat	octree	octree, cube computations	#D
	feat	rendering	rendering	#D
	feat	settings	settings	#D
	feat	shader	shaders	#D
	feat	texture	textures	#D
	plat	linux	Linux specific issue	#F
	plat	macos	MacOS specific issue	#F
	plat	windows	Windows specific issue	#F
	prio	blocker	this issue cannot be moved to a later milestone, also this label cannot be removed	#B
	prio	high	high priority	#F
	prio	low	low priority	#A

1.2 Source Code

1.3 How to contribute

1.3.1 Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission

- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at `info@inexor.org`. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

1.3.2 Licenses

In general we accept every widely acknowledged license which allows to share, modify, share the modified work, make commercial use or any combination of these. The licenses we accept are listed here, and below are the ones we do not. If you are not sure about something or the license doesn't appear in these lists, don't hesitate to ask us about it on our Discord.

List of accepted licenses

- CC0 1.0 or any later version
- CC BY 3.0 or any later version
- CC BY-SA 3.0 or any later version
- ZLIB
- MIT

List of licenses we don’t accept:

The following licenses either have a non-derivative or non-commercial section:

- CC BY-ND any version
- CC BY-NC any version
- CC BY-NC-ND any version
- CC BY-NC-SA any version
- No license at all

Special notes

We reject content which is licensed under the terms of a license which aims at one special jurisdiction. We consider ourselves as an international community and we define freedom as the freedom for everyone, everywhere. Please, don’t use licenses which could cause trouble, e.g. all Creative Commons licenses before the 4.0 versions have local derivations for a lot of countries (e.g. CC BY 3.0 DE). Starting with CC v4.0 there is only one international license to address this issue.

We also don’t accept Public Domain. The problem with Public Domain is that the definition is different from jurisdiction to jurisdiction, there isn’t an international declaration. Also in some jurisdiction it’s very complicated till impossible to make your work Public Domain before your rights expire after the ordinary time.

Instead of Public Domain we recommend using CC0 (Creative Commons Zero). CC0 is basically a license which gives the creator a way to waive all their copyright and related rights in their works to the fullest extent allowed by law + a Public Domain fallback if the waiving of the rights isn’t possible under special circumstances.

Common mistakes:

- Missing version number of the license (if exists), e.g. “Creative Commons BY”.
- Missing full name of the license, e.g. “licensed under a Creative Commons license”.
- Mix between the short form and the complete name, e.g. “Creative Commons BY Attribution 4.0”.

1.3.3 Contribute code

- If you want to contribute code, join our Discord and have a look at our vulkan-renderer repository.
- Try to find a first good issue in our issue tracker.
- Open a pull request.
- We want to keep commit as small in size as possible.

Signing commits

- We encourage you to sign your commits, although this is not a strict requirement from our side.
- You can find help on how to sign commits in GitHub's docs.

Commit naming convention

The commit naming convention will be checked by our continuous integration. In order to be valid, the followin rules must all be fulfilled:

- The commit message must begin with a category in square brackets which describes the code part that was changed
- The commit category must not be empty
- A commit category must contain at least two characters. Allowed characters are: letters `a-z`, numbers `0-9`, and `-`. For example `[gpu-texture]`
- If two code parts are affected, separate the categories using `|`, for example `[gpu-texture|gpu-info]`
- If three or more code parts are affected, use `[*]` as category
- Leave one space after the commit category
- The commit message itself must consist of the following characters: letters `a-z`, numbers `0-9`
- The commit message must begin with a capital letter
- The commit message must exist and it must not be empty
- The commit message must not end with `.`, `!` or `?`

Examples

- `[docs] Explain commit naming convention`
- `[gpu-info] Don't display empty gpu info queries as error`
- `[gpu-info|representation] Cleanup`
- `[ci] Add commit naming check`
- `[*] Move GPU info to vk tools`

Additional information

The regex pattern for the commit category is `\*|(?:[a-z0-9]{2,}[\s|-]?)+`, the pattern for the message is `[A-Z0-9].+[^.!?,\s]`

1.3.4 Contribute art

If you want to contribute textures, sounds or models, check out our website or join our Discord server to find more information about supported formats etc.

1.3.5 List of Contributors

We are always open for suggestions or pull requests. You can search after `diff:first issue` for good issues to get in touch with this project.

Join our Discord server.

IAmNotHanni

- Project and community management.
- Large portion of Vulkan renderer.

Iceflower

- Threadpool improvements.
- Advice on thread safety and code design.
- CMake improvements and feedback.
- Feedback on Travis CI and Github actions.
- Support for Exhale.
- Overall contributions to CI (Travis/Github actions).
- Replaced unscoped enums with scoped enums.
- Replaced macros with constexpr's.
- Improvement of overall project structure.
- Overall code cleanup.
- Read the docs support.

movabo

- First generation of octree engine code.

authenticate

- Refactoring of texture loading code.
- Error texture generation.

Shikijo

- Pointed out broken glsl compiler batch script on Windows.

Croydon

- Advice on conan package manager.

westernheld

- Testing and debugging.

uilianries

- Travis CI setup

yeetari

- Building and testing vulkan-renderer on Linux.
- Added USE_VMA_RECORDING CMake option so vulkan-renderer can be build on Linux until VMA recording has been ported.
- Overall contributions to CI (Travis/Github actions).
- Add build instructions for gentoo.
- Replace `_DEBUG` with NDEBUG.
- std::vector size check improvement.
- Help with git.
- Overall code cleanup.
- Getting vulkan-renderer compile and run on Linux.
- Overall CMake expertise.
- Compiling SPIR-V shaders through CMake setup.

azkoon

- Help with Discord server.
- Help with wiki.

joetoth

- Help with Discord server.

1.4 Frequently asked questions

Please visit inexor.org and join our Discord server.

1.4.1 What is Inexor?

images/inexor2.png

Inexor is a MIT-licensed open-source project which develops a new 3D octree game engine by combining modern C++ with Vulkan Vulkan API.

We have the following goals for the Inexor engine:

- Combine modern C++ with Vulkan API.
- Task-based parallelization using a threadpool and a work stealing queue.
- Generic rendering architecture using a rendergraph.
- Create a Vulkan API codebase which can be used in production.

We are using good software engineering practices:

- Resource acquisition is initialization (RAII).
- Software design patterns.
- Continuous integration (CI) using GitHub actions.
- Code design by strict compliance with the C++ core guidelines and Scott Meyers books.
- Use of the new C++ standard library (C++11, C++14, and C++17).
- Code documentation using doxygen.
- Automatic unit testing and benchmarking.
- Static code analysis with clang-tidy.
- Automatic code formatting using clang-format.
- CMake project setup.

You can find Vulkan example code online which follows the mantra "don't use this in production - it's tutorial code". Inexor disagrees with this as we believe that defeats its own purpose. If example code is not meant to be used in some other projects then there's something wrong with that example code. Many projects don't use a proper memory management library like VMA or they do not abstract their code using RAII, but use a lot of code duplication instead. Inexor is designed to be used in production. Bear in mind however that Inexor is also still far away from being a finished game engine.

1.4.2 Which platforms are supported?

- We support x64 Microsoft Windows 8, 8.1, and 10.
- We support every x64 Linux distribution for which Vulkan drivers exist.
- We have specific build instructions for Gentoo and Ubuntu. If you have found a way to set it up for other Linux distributions, please open a pull request and let us know!
- We do not support macOS or iOS because it would require us to use MoltenVK to get Vulkan running on Mac OS. Additionally, this would require some changes in the engines as not all of Inexor's dependencies are available on macOS or iOS.
- We also do not support Android because this would require some changes in the engines as not all of Inexor's dependencies are available on Android.

1.4.3 What is the current state of the project?

We are still in very early development, but this project can already offer:

- A modern C++20 codebase with CMake setup.
- Stable builds for Windows and Linux using Continuous Integration (CI).
- A rendergraph in early development.
- ImGui integration using separate renderpasses.
- RAII wrappers for various Vulkan resources.
- Extensive logging with spdlog.
- Vulkan Memory Allocator for graphics memory management.
- VMA memory replays for debugging are already working.
- Full RenderDoc integration with internal resource naming.

1.4.4 How is Inexor organized?

- Inexor has no central authority.
- It's a headless collective which makes decisions through creative discussions.
- We are welcoming new contributors to our team.

1.4.5 How to contact us?

Please visit inexor.org and join our Discord server.

1.4.6 How to build?

If you have any trouble building please open a ticket or join our Discord.

How to build vulkan-renderer?

1.4.7 Where to find Inexor's documentation?

- Read our docs here.

1.4.8 What is Vulkan API?

Inexor uses Vulkan API as rendering backend. Vulkan is a new, multi platform low level API (application programming interface) for high-performance graphics programming and computing. It is the successor to OpenGL, and it is important to state that is is very different from it. Vulkan is not just a new version of OpenGL or an extension of it. Instead, Vulkan is a very low level API which allows for much deeper control over the graphics card and the driver, like DirectX 12 or Apple's Metal. Unlike OpenGL, Vulkan API is build in a way it fits the architecture of modern graphics cards. This offers better performance due to reduction of overhead and driver guesswork during runtime. This results in higher frame rate, more predictable CPU workload and a lower memory usage. The most important benefit of Vulkan is the fact that it allows for multithreaded rendering, which is not possible in OpenGL at all. In general, Vulkan does a lot of work during the initialization of the application but therefore reduces work during rendering. Since Vulkan is much more explicit in terms of code, it foces you to think about the structure and architecture of your code. Both Vulkan and OpenGL are being developed by the Khronos Group. Vulkan is being developed through an unprecedented collaboration of major industry-leading companies (Google, Intel, AMD, NVidia, Sony, Samsung, Huawei, Qualcomm, Valve Software and many more). Vulkan is the only multi platform low level graphics API.

1.4.9 Why is Vulkan API the future?

Performance

- Lower and more predictable CPU load which results in better performance and a reduction of driver guesswork.
- Vulkan API is asynchronous and encourages multithreaded rendering. This is not possible with OpenGL!
- The low level API design of Vulkan allows for advanced optimizations such as rendergraphs for generic rendering architectures.

- It also wants you to use the GPU asynchronously, sometimes referred to as GPU multithreading.
- Vulkan allows the use of multiple GPUs, even if they are not physically linked via crossfire bridge.
- The reduction of CPU workload and it's improved predictability can enforce the GPU to be the limiting factor of performance, as it should be.

Memory efficiency

- Vulkan gives much deeper control and better interfaces over graphics and system memory.
- Vulkan API enforces memory management to be done by the application rather than the driver.
- Since the application knows best about the importance of every resource it uses, Vulkan API allows for a better memory usage.

Architecture

- Unlike OpenGL, Vulkan fits the design of modern GPUs as it is not just one single state machine. This means Vulkan API was designed from the beginning to match the architecture of modern graphics cards. OpenGL however still matches the design of graphics cards from the time it was invented in the 1990s.
- Vulkan is a fresh start, whereas OpenGL contains a myriad of hacks to support very rare use cases.
- Vulkan has layers and extensions as part of its design. You can check for supported GPU plugins on the target machine and enable them as needed.
- Vulkan API itself is completely platform agnostic.
- Available on a variety of platforms: Windows, Linux, mobile devices and much more!
- The ending of the OpenGL era has begun.
- Vulkan is being developed through an unprecedented collaboration of major industry-leading companies. It is not being developed by one company only (like Microsoft's DirectX for example).
- As Vulkan's motto states, it really is *industry-forged*.

Consistency and standardization

- Vulkan precompiles shaders to a standardized bytecode format called SPIR-V. This also reduces driver guesswork during runtime.
- The explicit design of Vulkan gives much deeper control and avoids driver guesswork and undefined behavior of graphics drivers.

Debugging tools

- Validation layers and diagnostics can be independently activated during development, allowing better error handling and debugging compared with OpenGL or DirectX.
- Upon release builds, the validation layers can be turned off easily.
- Vulkan API applications can be debugged with RenderDoc.
- The Vulkan specification is very easy to read and it is the central guideline for how to use the API.

Open Source

- Vulkan API and some Vulkan graphics card drivers are open source.

1.4.10 Can you explain Vulkan API in simple terms?

- Vulkan API gives programmers much deeper control over the gamer's hardware.
- If applied correctly, Vulkan can result in a significant performance boost.
- The API encourages the programmers to think in detail about graphics cards and their game engine.
- It offers advanced optimization techniques which can result in a lower RAM and video memory usage.
- Using Vulkan can yield in lower and more predictable CPU usage.
- Vulkan allows programmers to make more effective use of multiple CPU cores.

1.4.11 How difficult is development with Vulkan API?

- This API does a lot of initialization during the loading phase of the application.
- The key to success is a good abstraction of Vulkan API based on the needs of the application/game.
- Vulkan is a C-style API. In simplified terms you fill out structures which start with `Vk..` and submit them together with other parameters to `vk...` functions. That's it. No complex interfaces.
- Vulkan API has a very good documentation.
- The challenges of Vulkan game/engine development boil down to basic programming challenges: abstraction, resource management and parallelization.
- You may want to read Vulkan in 30 minutes by Baldur Karlsson.

1.4.12 Does my graphics card support Vulkan API?

- You can look up your graphics card in the Vulkan hardware database by Sascha Willems.
- Every new graphics card which is coming out these days supports Vulkan API.
- Vulkan is also supported on older graphics cards going back to Radeon HD 7000 series and Nvidia Geforce 6 series.

1.4.13 Will you support other rendering APIs?

- No, because testing for Vulkan already takes a lot of time and there is no sense in supporting deprecated technology.
- Some studios like id-software also dropped OpenGL entirely.
- Vulkan API is the only low level multi platform graphics and compute API.

1.4.14 Which topics are currently not in focus of development?

- We are currently focusing on the renderer and Vulkan API. When the time has come, we will take parallelization into account.
- A game engine needs other components besides rendering of course. However, we are currently not focusing on the following topics: networking, sound, physics, packaging of game engine resources and everything else which is not related to rendering.
- We will not begin to support additional platforms besides Linux and Windows in the near future.

1.5 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

You can find all releases in our GitHub repository.

1.5.1 v0.1-alpha.3 (17 May 2020)

Changed

- RAII in shader code
- RAII in shaders, gpu memory buffers, staging buffers, mesh buffers and textures
- RAII in descriptors
- RAII VkInstance
- RAII Swapchain
- Removed manager classes entirely.

1.5.2 v0.1-alpha.2 (26 Apr 2020)

Added

- Create a threadpool using C++17.
- Added a simple C++17 implementation of an octree.
- Added event system using boost::signals2.
- Use boost::bitstream for data processing.
- Convert octree data structure to vertex geometry (a mesh buffer).
- Support arbitrary indentations of octree geometry.
- Added a descriptor set layout for simple octree geometry.
- Ported Vulkan Memory Allocator library (VMA) to Linux.
- Added a simple camera movement class.
- Write spdlog console output to a logfile.

Changed

- Improvements considering C++ code quality standards.
- Logging format and logger usage.

Fixed

- Fixed a bug that would render every model twice.

1.5.3 v0.1-alpha.1 (12 Apr 2020)

Added

- Create a CMake file with conan package manager setup.
- Integrate Vulkan Memory Allocator library (VMA).
- Integrate RenderDoc support.
- Use spdlog as logger library.
- Integrate tiny_gltf library.
- Mesh buffer manager for vertex and index buffers based on VMA.
- Texture manager based on stb_image and VMA.
- Uniform buffer manager based on VMA.
- Shader manager for loading SPIR-V shaders.
- Load TOML configuration files using toml11. We deliberately won't use JSON for this.
- Vulkan fence manager.
- Vulkan semaphore manager.
- GPU info query functions.
- Vulkan debug callbacks.
- Vulkan standard validation layers.
- C++11 std::chrono class.
- Use glm.
- Depth buffer.
- Let VMA generate memory debug logs.
- Associate internal resource names with memory regions to improve debugging.
- Use separate data transfer queue for cpu to gpu memory copies if available.
- Availability checks for Vulkan features.
- Settings decision maker for Vulkan initialization.
- Simple command line argument parser.
- Automatic GPU selection mechanism and -gpu <N> command line argument for preferential GPU.
- Create windows using glfw3.

- Keyboard input based on glfw3.
- Load geometry of glTF 2.0 files using tiny_gltf library.
- Basic camera class.

1.6 Helpful Links

- Khronos' website about Vulkan API
- Vulkan API 1.1 specification

1.6.1 Advantages of Vulkan

- NVidia developer: What Is Vulkan?
- Stackoverflow: What can Vulkan do specifically that OpenGL 4.6+ cannot?
- Stackexchange gamedev: What is Vulkan and how does it differ from OpenGL?
- Stuck on OpenGL ES? Time to move on! Why Vulkan is the future of graphics
- A Brief Overview Of Vulkan API
- What advantages does Vulkan have over already established graphics APIs?

1.6.2 Getting started with Vulkan

GDC 2018 - Getting explicit: How Hard is Vulkan really? Dustin Land, Software engineer, id-Software.

Porting your engine to Vulkan or DX12 Adam Sawicki, Developer Software Engineer, AMD.

DevU 2017: Getting Started with Vulkan Developers from Imagination, Google and LunarG.

Vulkan Best Practices Roundtable discussion NVidia, Imagination, Qualcomm, id-Software, EPIC-games and Google.

Vulkan Memory Management Jordan Logan, Developer technology engineer, AMD.

Vulkan Memory Management Steven Tovey, Developer technology engineer, AMD.

Vulkan: State of the Union 2019 Developers from ARM, LunarG, NVidia.

1.6.3 Beginner Vulkan API tutorials

- Click here to check if your graphics card supports Vulkan API.
- How to learn Vulkan?
- The Vulkan tutorial
- The Vulkan guide
- Sascha Willems' Vulkan tutorials
- Intel's Vulkan tutorial: API without secrets
- Qualcomm developer network's Vulkan tutorials
- Introduction To Vulkan
- Raw Vulkan: An overview on how to program a Hello Triangle Vulkan application from the ground up.
- Vulkan Tutorial - 101
- Mike Bailey's Vulkan Page
- Using the Vulkan Validation Layers
- Sascha Willems' tutorial about Vulkan debug markers
- Awesome Vulkan list

1.6.4 Migrating from OpenGL to Vulkan API

- Dustin Land's blog series 'I Am Graphics And So Can You'
- Transitioning from OpenGL to Vulkan
- Porting a Graphics Engine to the Vulkan API

1.6.5 Advanced Vulkan API tutorials and articles

- Sascha Willems' glTF2 + PBR (physically based rendering) demo
- NVidia Developer Blog: Tips and Tricks: Vulkan Dos and Don'ts
- Vulkan in 30 minutes
- Writing an efficient Vulkan renderer
- A simple Vulkan Compute example
- Getting started with Vulkan API on Linux
- Get started with Vulkan on Android
- Use Vulkan API on macOS and iOS using MoltenVK
- Multithreading in Vulkan, where should I start?
- Multi-Threading in Vulkan
- Vulkan Device Generated Commands
- NVidia: Vulkan Memory Management
- Using Vulkan Device Memory

- Vulkan Barriers Explained
- GPUOpen performance guide

1.6.6 Vulkan API presentations

- NVidia: GPU-driven rendering
- NVidia: Vulkan: the essentials
- The most common Vulkan mistakes
- High-performance, Low-Overhead Rendering with OpenGL and Vulkan
- Vulkan on NVidia GPUs
- Get Your Engine Ready for Vulkan on Mobile
- Vulkan's Key Features on ARM Architecture

1.6.7 Vulkan API example projects

- The official Khronos Vulkan samples
- Sascha Willems' Vulkan tutorials
- Sascha Willems' tutorial about Vulkan debug markers
- VFPR - a Vulkan Forward Plus Renderer

1.6.8 Drivers

- NVidia drivers
- Open source AMD Vulkan driver
- Open source Intel Vulkan driver

1.6.9 Debuggers

- RenderDoc
- NVidia NSight
- AMD's GPU profiler

1.6.10 Modern C++

- Awesome modern C++
- C++ Core Guidelines

1.7 Source Code License

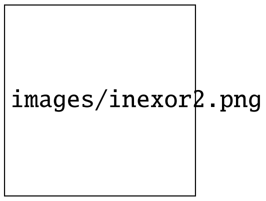
1.8 Contact us

Please join our Discord server and visit our website at www.inexor.org.

1.9 Frequently asked questions

Please visit inexor.org and join our Discord server.

1.9.1 What is Inexor?



Inexor is a MIT-licensed open-source project which develops a new 3D octree game engine by combining modern C++ with Vulkan Vulkan API.

We have the following goals for the Inexor engine:

- Combine modern C++ with Vulkan API.
- Task-based parallelization using a threadpool and a work stealing queue.
- Generic rendering architecture using a rendergraph.
- Create a Vulkan API codebase which can be used in production.

We are using good software engineering practices:

- Resource acquisition is initialization (RAII).
- Software design patterns.
- Continuous integration (CI) using GitHub actions.
- Code design by strict compliance with the C++ core guidelines and Scott Meyers books.
- Use of the new C++ standard library (C++11, C++14, and C++17).
- Code documentation using doxygen.
- Automatic unit testing and benchmarking.
- Static code analysis with clang-tidy.
- Automatic code formatting using clang-format.
- CMake project setup.

You can find Vulkan example code online which follows the mantra “don’t use this in production - it’s tutorial code”. Inexor disagrees with this as we believe that defeats its own purpose. If example code is not meant to be used in some other projects then there’s something wrong with that example code. Many projects don’t use a proper memory management library like VMA or they do not abstract their code using RAII, but use a lot of code duplication instead. Inexor is designed to be used in production. Bear in mind however that Inexor is also still far away from being a finished game engine.

1.9.2 Which platforms are supported?

- We support x64 Microsoft Windows 8, 8.1, and 10.
- We support every x64 Linux distribution for which Vulkan drivers exist.
- We have specific build instructions for Gentoo and Ubuntu. If you have found a way to set it up for other Linux distributions, please open a pull request and let us know!
- We do not support macOS or iOS because it would require us to use MoltenVK to get Vulkan running on Mac OS. Additionally, this would require some changes in the engines as not all of Inexor's dependencies are available on macOS or iOS.
- We also do not support Android because this would require some changes in the engines as not all of Inexor's dependencies are available on Android.

1.9.3 What is the current state of the project?

We are still in very early development, but this project can already offer:

- A modern C++20 codebase with CMake setup.
- Stable builds for Windows and Linux using Continuous Integration (CI).
- A rendergraph in early development.
- ImGui integration using separate renderpasses.
- RAII wrappers for various Vulkan resources.
- Extensive logging with spdlog.
- Vulkan Memory Allocator for graphics memory management.
- VMA memory replays for debugging are already working.
- Full RenderDoc integration with internal resource naming.

1.9.4 How is Inexor organized?

- Inexor has no central authority.
- It's a headless collective which makes decisions through creative discussions.
- We are welcoming new contributors to our team.

1.9.5 How to contact us?

Please visit inexor.org and join our Discord server.

1.9.6 How to build?

If you have any trouble building please open a ticket or join our Discord.

How to build vulkan-renderer?

1.9.7 Where to find Inexor's documentation?

- Read our docs here.

1.9.8 What is Vulkan API?

Inexor uses Vulkan API as rendering backend. Vulkan is a new, multi platform low level API (application programming interface) for high-performance graphics programming and computing. It is the successor to OpenGL, and it is important to state that is is very different from it. Vulkan is not just a new version of OpenGL or an extension of it. Instead, Vulkan is a very low level API which allows for much deeper control over the graphics card and the driver, like DirectX 12 or Apple's Metal. Unlike OpenGL, Vulkan API is build in a way it fits the architecture of modern graphics cards. This offers better performance due to reduction of overhead and driver guesswork during runtime. This results in higher frame rate, more predictable CPU workload and a lower memory usage. The most important benefit of Vulkan is the fact that it allows for multithreaded rendering, which is not possible in OpenGL at all. In general, Vulkan does a lot of work during the initialization of the application but therefore reduces work during rendering. Since Vulkan is much more explicit in terms of code, it foces you to think about the structure and architecture of your code. Both Vulkan and OpenGL are being developed by the Khronos Group. Vulkan is being developed through an unprecedented collaboration of major industry-leading companies (Google, Intel, AMD, NVidia, Sony, Samsung, Huawei, Qualcomm, Valve Software and many more). Vulkan is the only multi platform low level graphics API.

1.9.9 Why is Vulkan API the future?

Performance

- Lower and more predictable CPU load which results in better performance and a reduction of driver guesswork.
- Vulkan API is asynchronous and encourages multithreaded rendering. This is not possible with OpenGL!
- The low level API design of Vulkan allows for advanced optimizations such as rendergraphs for generic rendering architectures.

- It also wants you to use the GPU asynchronously, sometimes referred to as GPU multithreading.
- Vulkan allows the use of multiple GPUs, even if they are not physically linked via crossfire bridge.
- The reduction of CPU workload and it's improved predictability can enforce the GPU to be the limiting factor of performance, as it should be.

Memory efficiency

- Vulkan gives much deeper control and better interfaces over graphics and system memory.
- Vulkan API enforces memory management to be done by the application rather than the driver.
- Since the application knows best about the importance of every resource it uses, Vulkan API allows for a better memory usage.

Architecture

- Unlike OpenGL, Vulkan fits the design of modern GPUs as it is not just one single state machine. This means Vulkan API was designed from the beginning to match the architecture of modern graphics cards. OpenGL however still matches the design of graphics cards from the time it was invented in the 1990s.
- Vulkan is a fresh start, whereas OpenGL contains a myriad of hacks to support very rare use cases.
- Vulkan has layers and extensions as part of its design. You can check for supported GPU plugins on the target machine and enable them as needed.
- Vulkan API itself is completely platform agnostic.
- Available on a variety of platforms: Windows, Linux, mobile devices and much more!
- The ending of the OpenGL era has begun.
- Vulkan is being developed through an unprecedented collaboration of major industry-leading companies. It is not being developed by one company only (like Microsoft's DirectX for example).
- As Vulkan's motto states, it really is *industry-forged*.

Consistency and standardization

- Vulkan precompiles shaders to a standardized bytecode format called SPIR-V. This also reduces driver guesswork during runtime.
- The explicit design of Vulkan gives much deeper control and avoids driver guesswork and undefined behavior of graphics drivers.

Debugging tools

- Validation layers and diagnostics can be independently activated during development, allowing better error handling and debugging compared with OpenGL or DirectX.
- Upon release builds, the validation layers can be turned off easily.
- Vulkan API applications can be debugged with RenderDoc.
- The Vulkan specification is very easy to read and it is the central guideline for how to use the API.

Open Source

- Vulkan API and some Vulkan graphics card drivers are open source.

1.9.10 Can you explain Vulkan API in simple terms?

- Vulkan API gives programmers much deeper control over the gamer's hardware.
- If applied correctly, Vulkan can result in a significant performance boost.
- The API encourages the programmers to think in detail about graphics cards and their game engine.
- It offers advanced optimization techniques which can result in a lower RAM and video memory usage.
- Using Vulkan can yield in lower and more predictable CPU usage.
- Vulkan allows programmers to make more effective use of multiple CPU cores.

1.9.11 How difficult is development with Vulkan API?

- This API does a lot of initialization during the loading phase of the application.
- The key to success is a good abstraction of Vulkan API based on the needs of the application/game.
- Vulkan is a C-style API. In simplified terms you fill out structures which start with `Vk..` and submit them together with other parameters to `vk...` functions. That's it. No complex interfaces.
- Vulkan API has a very good documentation.
- The challenges of Vulkan game/engine development boil down to basic programming challenges: abstraction, resource management and parallelization.
- You may want to read Vulkan in 30 minutes by Baldur Karlsson.

1.9.12 Does my graphics card support Vulkan API?

- You can look up your graphics card in the Vulkan hardware database by Sascha Willems.
- Every new graphics card which is coming out these days supports Vulkan API.
- Vulkan is also supported on older graphics cards going back to Radeon HD 7000 series and Nvidia Geforce 6 series.

1.9.13 Will you support other rendering APIs?

- No, because testing for Vulkan already takes a lot of time and there is no sense in supporting deprecated technology.
- Some studios like id-software also dropped OpenGL entirely.
- Vulkan API is the only low level multi platform graphics and compute API.

1.9.14 Which topics are currently not in focus of development?

- We are currently focusing on the renderer and Vulkan API. When the time has come, we will take parallelization into account.
- A game engine needs other components besides rendering of course. However, we are currently not focusing on the following topics: networking, sound, physics, packaging of game engine resources and everything else which is not related to rendering.
- We will not begin to support additional platforms besides Linux and Windows in the near future.

INDEX