Quadratic Simplification Viewer | by MKebsi

This viewer is based on Three.js and the WASM version of the Fast Mesh Simplification function found here.

Upon the request of a protentional client, this was made to demonstrate how can it be useful. Instead of using an AWS Lambda function (probably written in Python), I went to this direction.

But before anything, check it out! Online Demo

The example uses a `MeshNormalMaterail()` to show specifically all edges of the mesh so that artifacts will be easily detected during the process. The top-right panel may also feel a bit stiff, yet that is duo to time limitations.

This was all created in two days (starting Jun 5 until Jun 7, 2024), 2-4 hours per day. A day before that was spent on researching and concepting.

Usage

Right in front of you is a 3D space divided in two sides. You can rotate, zoom, and pan around each on of them separately using your mouse.

In the left side, your initial model will be viewed once loaded (see below for how to load models). The other side will view the model after the mothed has been applied (so it will have less vertices)

In the top-right corner, you will see an opened, barely visible panel. Hover over it and it will be visible. I will be calling it Settings (as you can see its title, too)

Loading a Model

You need to have a `.obj` or `.stl` model in your device. You can load that form the button on the Settings or just drag and drop it to the browser.

You can check the logs in the Settings to know the current state of the app.

What About the Other Buttons in the Settings?

I will let you play with them and see what happens, but below are some notes.

The `Reload` button is there for re-simplifying the model if it was previously loaded and have a different simplification percentage.

Double clicking the logs panel will show it in full screen so that you can see them clearly. To go out of the full screen mode, click on the `Esc` button on your keyboard.

The last line on the Settings will tell you your current state using the app. It will also notify you if you have done something unexpected.

The `Result` will give you an approximation of the final model's size. Mostly, it lies!

Why Is This Method Better Than AWS?

1 – Cost

First and for most, it is cost efficient. Lambda cost is calculated based on the requests per month (free for the first 1 million request a month). This may seem cheap since it may not even reach a hundred thousand request a month.

The catch is that, you will also pay if the requests took a sit amount of time per request (stating that roughly about 3.2 million seconds of compute time will be free). Generally, less than ~900 hours of compute time.

Simplifying a mesh takes some time to be computed, averaging 2 seconds per request. A simple math will show that if 5,000 users used the app a day, and each one used this method three times (while most users will probably use it more than that!), you will be at ~30,000 seconds per day!

Anyhow, if the platform grew huge, there will never be any worries regarding this method. The compute time on users' machine (even on mobile phones) take an average of 0.9sec per request, so it quite efficient, thanks to web assembly and Sven Forstmann for making this possible.

2 – Compatibility

Since web assembly ends up with a JavaScript file, the integration of it to any sort of web apps is much easier and takes no time. There is no way a bug can happen where another code (run in Python) causing the issue.

It is well-known that the most used language on websites is JavaScript. Moreover, it's integration to third party libraries (like Three.js in this example) will be much easier!

3 – Simplification Functions Made with Python!

Based on my knowledge and research, I haven't found a simplification method that is as efficient as the one used in this example. This used C++ and was available since 2014 under the MIT license.

Replicating this same method on Python needs a professional Python guy (which is for sure not me!). I don't even know that this is possible, but I know one thing, "Web assembly provides performance close to the low-level programming languages like Rust and C.", which are way faster than scripting languages (Python and JavaScript).

In general, this method best suited the use case I was able to understand from my client. As long as the final product of this method is using simplified mesh, viewed in Three.js, this is the best thing that came to my mind!

Limitations

This method is a bit limited. Source files that will be simplified should be either an `.obj` file or an `.stl` file. This limitation comes down the initial method itself (the C++ version). Those files provide a clean structure to what should be simplified.

It is also stated that using multiple objects (a.k.a. meshes) or groups in a single file may result in some unwanted artifacts. So, stick to a file that has a single mesh or group for the best result.

Lastly, the smooth shading view isn't working. This is a Three.js related bug that I couldn't fix in the meantime, especially that I want to ship it ASAP!