Mandel (Vulkan version)

Introduction

Mandel is a GPU demonstration program based on the Mandelbrot Set. It is an example of a program that uses the GPU for both computation (the Mandelbrot set is computationally intensive) and display, programmed using the Vulkan framework.

Building

This should build on any recent Linux system that has the usual C++ development tools (a C++17 compliant c++ compiler, and make). It also needs to have Vulkan installed, together with a couple of smaller packages - the GLFW windowing system and the GLM linear algebra library. Finally, it needs the glslc shader compiler. The program is built from the command line using make, and the directory includes a Makefile that controls the build.

If you don't have these already, instructions for installing them for Ubuntu and Fedora systems can be found at the end of this document. If they've been installed somewhere non-standard, you may need to modify the Makefile so it can find them.

If you have the necessary libraries, you should be able to build this program from the terminal just by typing:

```
make
```

Running Mandel

The Makefile builds an executable called 'Mandel', and you can run it by typing:

```
./Mandel &
```

There are some command line parameters accepted by the program, and these are described in detail in a later section.

What you see

Once Mandel starts, you should see a window with a display of the Mandelbrot set. If you're not familiar with this, you can look it up in Google. Wikipedia has a good article about it. The important point is that generating this image requires a lot of calculation, and it used to be a popular benchmark for comparing the speeds of computers. The shape has some fascinating properties. The boundary of the Mandelbrot set is infinitely long, and the more you zoom in on it the more complex it gets.

This isn't the place to explain the Mandelbrot set in detail. When you start up the program, you're seeing a 2D coordinate space. Each point in that continuous coordinate space has an X and Y value, and there is a calculation that can be carried out at each point based on those X,Y values. That generates a new pair of values and the same calculation can be repeated. And so on, and so on. If the sequence of values does not converge, then that original X,Y position is part of the Mandelbrot Set. In the diagram you see, anything black is a position in the set, anything coloured is outside it. The colour depends on the number of iterations it took before it was clear the value was not going to converge.

Obviously, the program could loop indefinitely repeating the calculation for a point that isn't ever going to diverge. Instead, it sets a limit on the number of iterations it will try. By default, this is 1024. So for each point in the image that's black, the program has repeated the calculation it does 1024 times. The program is displaying a 1024 by 1024 image, so it's sampling the coordinate space 1024*1024 times. And it will have repeated its calculation up to 1024 times for each point. That's why this is computationally intensive.

Interacting with the program

You can put the cursor at any point in the image - try somewhere near the edge of the black area - and zoom in and out by scrolling up or down. When you zoom in, the window is covering a smaller part of the coordinate space, each of those 1024 by 1024 points now has new coordinates, and the program must repeat the calculation, up to 1024 times for each point. Then it redisplays the resulting image. The fact that it can do this quickly enough shows the power of the GPU in your laptop.

You can change the window size by dragging the edges or corners, and you can put it into full screen mode. You can close it using the window's red close button, with command-q or using the menu's 'Quit' option. You can drag the image by holding down the left mouse button while moving the cursor.

All other interaction with the program is by pressing one of a limited set of keyboard keys. (The point of the program is to show what a GPU can do, not to provide a fancy user interface. Keys are easy to program.)

Some points in the set are more interesting than others. If you press any of the numeric keys '0' through '9', you get taken to a preset visually interesting centre position and zoom level. Try them. Go to one – try them all! - and then scroll in or out.

If you press 'I' the program zooms in for as long as you hold the key down. When you release it, it tells you how many frames the program calculated during that zoom, and gives a figure for frames/sec. This lets you see how fast the program can calculate and display a new frame. Areas with a lot of black need longer to calculate than lighter coloured areas - the program has to go through the full 1024 iterations for every one of those black pixels. Pressing 'O' and holding it down zooms out. 'Z' zooms in for 10 seconds and then out for 10 seconds.

The 'D' key displays the path that the Mandelbrot calculation takes for the coordinate point under the cursor. A point inside the set will have a path that converges and remains confined to the image, while points outside the set have paths that diverge and escape the image. Some points, particularly those within the black area but close to the boundary, can generate very complex shapes. The 'E' key toggles a mode where the program continually displays the path for the point under the cursor as it moves within the image. See what shapes you can generate by moving the cursor around.

The window's title bar shows the current zoom level. It also shows whether the computation is being done using the GPU or the CPU. If the GPU supports double precision (Mac GPUs don't) it shows if this is being used. If the zoom level is too high for the precision being used, this is indicated using asterisks. The CPU always uses double precision (single gives no speed gain on most modern CPUs). By default, the program uses the GPU unless it needs double precision and the GPU doesn't support it, in which case it switches to the CPU. You can force CPU mode using the 'C' key, can force GPU mode using the 'G' key, and go back to the automatic CPU/GPU switching using the 'A' key.

The 'H' key gives a list of the various key options.

By default, the program computes an image with 1024 by 1024 resolution. The 'L','M','S' and 'T' keys will select different resolutions - large, medium, small and tiny (by default, 2048x2048, 1024x1024,512x512, and 128x128 respectively). Higher resolutions involve more computation, obviously.

The '9' key selects a centre position and zoom value quite close to the point where the program will switch automatically from GPU to CPU (unless the GPU supports double precision). Try pressing '9' and then hold 'I' down until the window title shows the program has switched to the CPU. You'll see a summary of the compute times from the GPU and from the CPU. This should show how much faster the GPU is that the CPU (and the program uses all the CPU cores the machine has available). This will be less obvious in the display, because the program compensates for the time the CPU takes by increasing the amount each frame is zoomed. The result is a zoom that seems as fast as that with the GPU, but jerkier. This compensation can be toggled on and off with the 'W' key. Turning it off makes the difference

between CPU and GPU more obvious. Going to a position with more black pixels also makes it more obvious.

Key Effects

'0'..'9'	select pre-determined settings for centre point and magnification.
'r'	resets the display to its starting point
'i'	hold down the 'i' key to zoom in
'o'	hold down the 'o' key to zoom out
'j'	centers the image on the cursor position.
'p'	outputs the current image centre and magnification on the terminal.
'd'	displays the Mandelbrot path for the point under the cursor.
'e'	toggles a continuous display of the Mandelbrot path as the cursor moves.
'x'	clears any Mandelbrot path from the display
'z'	does a zoom test. It zooms in for 5 seconds, then out for 5 seconds
'a'	sets auto mode - the program switches automatically between CPU and GPU.
'c'	forces the program to use the CPU - all available cores.
'g'	forces the program to use the GPU at all magnifications.
'w'	toggles magnification rate compensation for slow compute times during zoom
'l'	sets size of images to large (twice the default size in X and Y)
'm'	sets size of images to medium (the default size in X and Y)
's'	sets size of images to small (half default size in X and Y)
't'	sets size of images to tiny (quarter the default size in X and Y)

Command line parameters

The program accepts a set of command line parameters, using a flexible (if quirky) command line processor which accepts specifications in a variety of formats. Parameters can be specified by name and value, with or without an equals sign, eg “Nx 1024” or “Nx = 1024” or “Validate = true” or “Validate yes” or (for boolean parameters) “validate” or “novalidate”.

The parameters specific to the program are:

Nx is an integer giving the initial size of the calculated image in X. Normally 1024.
Ny is an integer giving the initial size of the calculated image in Y. Normally 1024.

Iter is an integer giving the maximum number of iterations used. Normally 1024.
Validate is a Boolean that enables the Vulcan validation layers. Normally true.
Debug is a comma-separated list of hierarchical debugging options. Normally "".

Changing Nx and Ny will change the resolution of the computed image, with a consequent change in the computation time required. Changing Iter will change the number of iterations performed by the calculation. A high value for Iter will significantly increase the time needed to compute an image where most of the pixels are black (within the Mandelbrot set), as these all require the maximum number of iterations.

Nx, Ny, and Iter can also be specified positionally on the command line. For example:

```
./Mandel 256 512 200
```

will set Nx to 256, Ny to 512 and Iter to 200.

Validate and Debug need to be specified by name and value.

Enabling the Vulkan validation layers switches on a set of internal Vulkan tests and diagnostics. This is mainly important when testing changes to the program - normally none of these tests should fail. Enabling validation does add to the Vulkan processing, but its effects are usually slight, which is why the program leaves it on by default.

Debug is a comma-separated list of named hierarchical debug levels which are enabled by being included in the Debug string. The '*' character acts as a wildcard. If you specify Debug as '*' (you'll need to quote this, or escape the '*' character as "*" or the shell will interpret it as a filename wildcard), you will enable every single debug level and will be swamped by output. However, you should be able to see what levels are supported. Or you can use the 'help' option described below.

If you specify an invalid value for a parameter, you will be prompted for it. Some parameters, like Debug) will provide more detailed help if you respond to a prompt with '?'.

The command line handler has some built-in boolean options of its own.

'Help'	outputs a full list of the accepted command line parameters.
'List'	lists all the values being used for all the parameters.
'Prompt'	forces prompting for all parameters not explicitly specified.
'Reset'	all unspecified parameters default to a set of standard values.

Normally, the command handler remembers the values used the last time the program was run; often this is handy, but sometimes you really want to revert to the default values. (The value for Debug is unusual as it is not remembered and always defaults to no diagnostic output.)

The program remembers the most recently used values for most parameters, and uses these as the default values. 'Reset' makes it forget these, and revert to the set of standard values that are built into the code. If you're seeing what seems to be unusual behaviour, try using 'reset'.

Help and support

Any problems, e-mail keith@knaveandvarlet.com.au

Appendix: Installing Vulkan etc on Ubuntu

These instructions are a simplified version of more detailed instructions from the excellent vulkan-tutorial site. They should work on Ubuntu with the apt package manager. A version for Fedora and dnf can be found in the following appendix.. For more details:

https://vulkan-tutorial.com/Development_environment

o Make sure things are up to date:

```
sudo apt update
```

o Install the standard C++ programming tools, if you need them:

```
sudo apt install build-essential
```

o Install Vulkan

```
sudo apt install vulkan-tools
```

(at this point, it might be as well to see if Vulkan is supported on your machine - try the 'vulkaninfo' command - you should see a lot of detail you don't care about, but it should work - and then try the 'vkcube' command to see if you see a spinning cube).

o Install Vulkan development tools:

```
sudo apt install libvulkan-dev

sudo apt install vulkan-validationlayers-dev
```

o Install GLFW and GLM:

```
sudo apt install libglfw3-dev

sudo apt install libglm-dev
```

o Install Google's glslc shader compiler. Go to:

```
https://github.com/google/shaderc/blob/main/downloads.md
```

and download the gcc binary version for Linux. (Or the clang version, if that's your preferred C++ compiler.) Uncompress it and copy over the compiler. Assuming your downloads go into ~/Downloads:

```
cd ~/Downloads

tar -xvf install.tgz

sudo cp install/bin/glslc /usr/local/bin
```

Appendix: Installing Vulkan etc on Ubuntu

These instructions are a simplified version of more detailed instructions from the excellent vulkan-tutorial site. They should work on Fedora with the dnf package manager. For more details:

https://vulkan-tutorial.com/Development_environment

o Make sure things are up to date:

```
sudo dnf check-update

sudo dnf upgrade
```

o Install the standard C++ programming tools, if you need them:

```
sudo dnf install gcc-c++

sudo dnf install make
```

o Install Vulkan

```
sudo dnf install vulkan-tools
```

(at this point, it might be as well to see if Vulkan is supported on your machine - try the 'vulkaninfo' command - you should see a lot of detail you don't care about, but it should work - and then try the 'vkcube' command to see if you see a spinning cube).

o Install Vulkan development tools:

```
sudo dnf install vulkan-loader-devel

sudo dnf install mesa-vulkan-devel

sudo dnf install vulkan-validation-layers-devel
```

o Install GLFW and GLM:

```
sudo dnf install glfw-devel

sudo dnf install glm-devel
```

o Install Google's glslc shader compiler. Go to:

```
https://github.com/google/shaderc/blob/main/downloads.md
```

and download the gcc binary version for Linux. (Or the clang version, if that's your preferred C++ compiler.) Uncompress it and copy over the compiler. Assuming your downloads go into ~/Downloads:

```
cd ~/Downloads

tar -xvf install.tgz

sudo cp install/bin/glslc /usr/local/bin
```