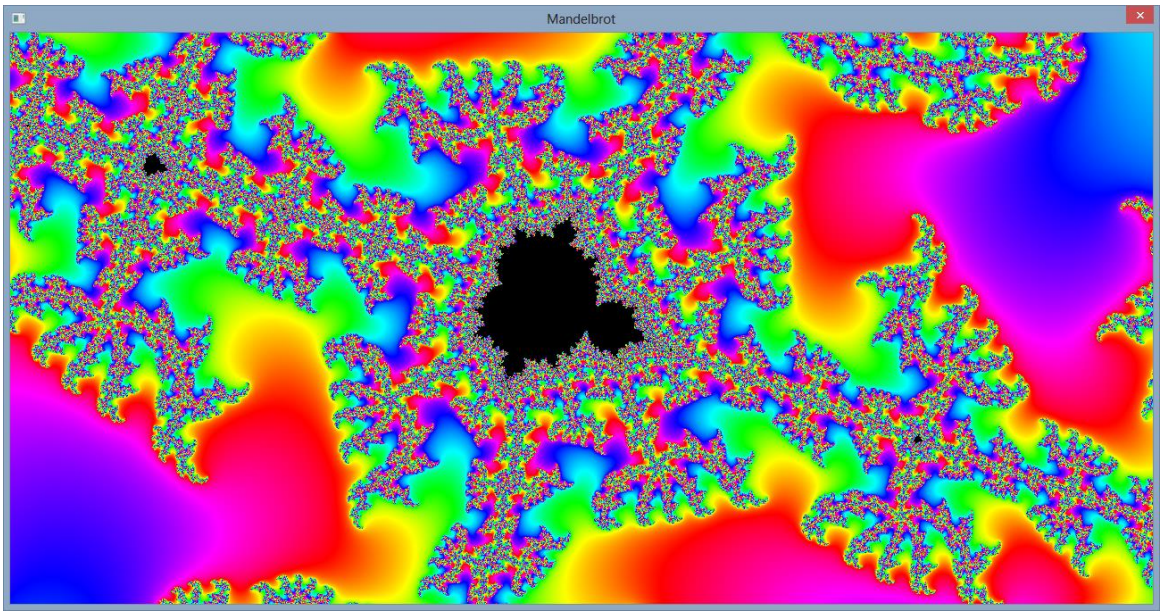
Mandelbrot Practical Assignment

Advanced Graphics course 2012-2013, Utrecht University.
Marries van de Hoef



1. Overview

The goal of this assignment is to familiarize yourself with the DirectX Graphics API and (to a lesser extent) the Windows API. Most of your time will be consumed by figuring out how these API's work, as this assignment will only give you entry points on where to start looking. Additionally, the assignment will also refresh your graphics programming skills in preparation for the second assignment.

You will create an interactive Mandelbrot renderer, which leverages the parallel processing power of the GPU. The application will be created from *scratch*, without any frameworks (or code you previously wrote). The advantage of this approach is that you will get a better understanding of the possibilities and how frameworks (such as XNA) function internally.

You will have to reuse the general parts of your application for the next assignment. With this in mind, make sure to create a nice object oriented structure.

2. Organizational Matter

Deliverables

- The source code. It should compile directly in VS2012 (no external dependencies). Do not submit intermediate build files and other temporary files.
- The executable and any dependencies. It should run directly.
- A screenshot or link to a youtube video.
- Report.

Report
The report should contain the following elements:

- The student number.
- Additional references of any type used for the assignment.
- Instructions for the application.
- Listing and a short description of the implemented additional features.
- A short description of the code structure.
- A short theoretical analysis describing the hardware performance bottleneck(s) of the application.

Grading criteria
The grading criteria are the following, roughly in order of importance:

- Compliance with the deliverable requirements and technical requirements.
- Presence of the required features.
- Quality of the report.
- Absence of bugs and glitches.
- Code structure (object orientedness, etc.).
- Code quality (e.g. the layout clarity, variable naming clarity, commenting).
- Implementation quality (polish).

Grading and additional features
The grade depends on the grading criteria mentioned above and the additional features you implemented. The additional features can be selected from the list in this document. You are also free to think of features yourself, but make sure they are relevant to the goal and focus of the assignment. Note that the difficulty of additional features is more important than the quantity.

Submission
You are encouraged to submit on Sunday the 24th of February, because the second practical assignment is already online at that time. The final submission deadline is on Saturday March 2nd at 23:55. The additional week is for students who need more time because of a deficiency.

Submission is done through the Submit system: http://www.cs.uu.nl/docs/submit/

3. General Remarks

Internet sources
You will have to use internet sources to learn how to implement most features of your application. Some of these internet sources provide nearly complete source code, so it might be tempting to copy that code. However, the goal if this assignment is to learn the API's and that is not achieved when copying the code. Additionally, copying is plagiarism which is strictly forbidden.

So, how *are* you allowed to use internet sources? You should use them to learn HOW to implement something, NOT the implementation itself NOR how the implementation is structured. You can enforce this by only consulting the internet for specific questions (for example: "which function(s) do I need to achieve X?"), and thus avoiding complete code walkthroughs or tutorials.

By following the above approach, you will ensure that you have a good understanding of the choices you made in the code. If there are doubts that you may have plagiarized, you will be requested to explain the choices made in your implementation in an interview.

Additional sources

There are several internet sources provided in this assignment, but you are also allowed to find additional sources yourself. If you use an additional source for more than answering one small question, you have to reference the source in your report.

Be aware that a lot of "tutorials" are written by people who are learning the subject themselves and commonly contain wrong or suboptimal information. In my experience, this is very often the case with graphics programming tutorials.

Books are a more reliable source of additional information. There are several introductory DirectX 11 books, but most feel like rebranded DirectX 10 books. The only real Direct3D 11 book is "Practical Rendering and Computation with Direct3D 11". This book is recommended for a thorough understanding of Direct3D 11, but it probably is too elaborate for this assignment. Alternatively, there is a free online Direct3D 10 book. It is slightly outdated but it can provide useful background information for this assignment, especially the introductory section:
http://content.gpwiki.org/index.php/D3DBook:Table_Of_Contents

Academic dishonesty

Fraud or plagiarism has to be reported to the Board of Examiners, as required by the Education and Examination Regulations. This includes for example copying code from the internet or other students, letting other students create (parts of) your assignment, letting other students copy code from your assignment, and reusing previously written code (without the teacher's permission).

Code quality

The grading criteria include the code structure and code quality. The main reason is because it shows that you understand what you are doing and that you care enough to express it.

Quality in code is not expressed by using difficult or obscure syntax constructions to show your skill, but by creating code that is easy to understand and maintain while still performing (near) optimally. You can use any coding *style*, as long as it does not conflict with the previous properties.

Support

You are encouraged to post questions on the forum when (for example) you are stuck, the assignment is unclear, or for advice on how to solve a specific problem. This includes questions from people who are solving deficiencies. You are also encouraged to answer the questions of other students.

4. Technical Requirements

You have to use C++ and the Direct3D 11 API. The Direct3D 11 API can also be used on older hardware. For the assignment, you will have to use Shader Model 4.0 and therefore you need at least DirectX 10 compatible hardware.

Note that Direct3D 11.1 features may not be used (Direct3D 11.0 is fine), and neither can double precision be used. (My GPU does not support it, so your submission can't be graded then.) Additionally, you should not use the WinRT API for Windows 8 "metro" style applications.

Visual Studio 2012

You have to use Visual Studio 2012 as programming environment. There have been significant changes in this last iteration, which are favorable for this assignment. The DirectX SDK has been integrated in the Windows 8 SDK, which is part of Visual Studio 2012. There have been some significant modernizations in the integrated DirectX SDK. Additionally, VS2012 has an integrated Direct3D debugger which can be helpful. The Windows 8 SDK and VS2012 run fine on Windows 7.

Visual Studio 2012 can be obtained through DreamSpark: https://ict.science.uu.nl/index.php/MSDN

Libraries

The only additional library you might need is a math library. Consult me first if you want to use other libraries (for example for additional features). Make sure that the library dependencies are stored locally in the submission, so the project can be compiled directly.

Submission

Before submitting your code, delete any intermediate build files and other temporary files. If your code is larger than 1MB, you probably forgot to delete something.

Assignment Description

You will create a Windows application which calculates the Mandelbrot fractal function for each pixel in a shader. The user should be able to move around and zoom in and out using the mouse and/or keyboard. The fractal should be visualized using smooth colors.

The assignment is broken down into several parts which are discussed in the next sections:

- Creating a window
- Initializing Direct3D
- Render using a shader
- Rendering the Mandelbrot fractal
- User input
- Additional features

5. Creating a Window

This link gives a good introduction: http://msdn.microsoft.com/en-us/library/bb384843.aspx

The introduction is for general applications and not for specifically for usage with Direct3D, so you will need to handle some things slightly differently.

Read the documentation of the used functions, and try to change the default behavior: Make a non-resizable window without minimize or maximize buttons. Always position the window in the center of the screen, and correct the used size to include the borders. (Note that you will need to use additional functions for this.)

Don't forget to restructure your code to something object-oriented which can be reused for the second practical assignment.

6. Initializing Direct3D

Follow this link: http://msdn.microsoft.com/en-us/library/windows/desktop/ff476879.aspx

Don't forget to include <d3d11.h> and to add the library (Go to project properties -> Linker -> Input, and add d3d11.lib to the additional dependencies for both Debug and Release mode).

Remember not to copy the code directly, but to use the link as a reference for the functions you need.

The objects you just created can be used to perform actions similar to what you might know from using for example XNA. This is how the functionality is divided in general:

- ID3D11Device is used to create objects.
- ID3D11DeviceContext is used for setting states and drawing.
- IDXGISwapChain is used for back buffer related things, such as presenting the rendered image on the back buffer to the user.

Finally, clear the back buffer and present the result to the user each frame.

7. Render using a Shader

In short, you have to perform these steps:

- Create a vertex buffer and input layout for your full screen quad.
- Create and compile your (very simple) shader. (Don't use effect files, but use separate shaders instead.)
- Set the shaders, vertex buffer, input layout and primitive topology.
- Draw the full screen quad.

Search on MSDN on how to perform these steps. This page is a nice start:
http://msdn.microsoft.com/en-us/library/windows/desktop/hh404569.aspx

The HLSL shader programming documentation can be found here:
http://msdn.microsoft.com/en-us/library/windows/desktop/bb509561.aspx

Shader compilation

There are two methods for shader compilation: compiling during runtime (easier), and compiling as part of the build process (better).

During runtime: http://msdn.microsoft.com/en-us/library/windows/desktop/hh968107.aspx
Don't forget to include <d3dcompiler.h>, add d3dcompiler.lib to your project, and add d3dcompiler_46.dll in the same folder when distributing the final executable.

At compile time: http://msdn.microsoft.com/en-us/library/windows/desktop/bb509633.aspx
The link only gives a start, search for additional information if necessary.

Math library

You might want to use a math library for the vertex attribute types (although this is not necessary). You can use the DirectXMath library, it is a very good free SSE accelerated math library. It is part of the Windows 8 SDK which is included in VS2012, so you only have to include <DirectXMath.h> and you're done. You don't need to add a library because everything is compiled directly into your application. You can also include <DirectXColors.h> and <DirectXPackedVector.h> if you like. The latter is especially useful for vertex attributes. Because the math library is in the DirectX namespace, you might want to add "using namespace DirectX". You are advised to compile exclusively for x64 to prevent SSE alignment issues when allocating on the heap.

The documentation can be found here:
http://msdn.microsoft.com/en-us/library/windows/desktop/hh437833.aspx

A note of warning: if you have deficiencies for this course or you have found this assignment to be hard so far, it might be best to postpone using this math library. This library can be hard to learn, mostly because of the SSE-style programming.

Debugging

Graphics debugging is integrated in VS2012. It allows you to view your API calls and the state in all stages of the graphics pipeline. This can be especially useful if you don't see anything drawn on the screen. It also features a shader debugger but it can be unreliable, especially with complex shaders.

An introduction to the graphics debugger can be found here:
http://blogs.msdn.com/b/vcblog/archive/2011/11/08/10235150.aspx

The introduction seems to be slightly outdated. Run the application using Alt+F5 to enable the graphics debugger.

If you get stuck, don't forget to ask your questions on the forum!

8. Rendering the Mandelbrot Fractal

You are not expected to understand the Mandelbrot function itself for this assignment, but you do have to translate it to the GPU.

This page gives a very nice introduction to Mandelbrot, including an explanation on how to calculate it programmatically: http://warp.povusers.org/Mandelbrot/

You have to add a smooth coloring to your rendering, preferably using multiple colors. Achieving a smooth result is commonly done by normalizing the iteration count, as explained here:
http://linas.org/art-gallery/escape/escape.html

Remember: when programming shaders, use very short programming iterations. If you write all code first and then start testing it, it will be very hard to debug. Do not rely on the VS2012 shader debugger, because it is common that it doesn't work.

Note: do not use double precision values in the shader (at least for the submitted version). My GPU doesn't support it, so your assignment can't be graded then.

9. User Input

The easiest way is to capture keyboard/mouse input messages in your window message handler.

The documentation: http://msdn.microsoft.com/en-us/library/windows/desktop/ms632585.aspx

At the least, you have to handle movement in the complex plane and zooming in and out. Try to achieve a smooth user experience.

To transfer shader variables to the shader, use a constant buffer. To update a constant buffer, use the Map and Unmap functions on the device context. Search for the relevant pages on MSDN.

10. Additional Features

Implement additional features to enhance your Mandelbrot renderer. You are free to choose the features you want to implement from the list below. You can also think of features yourself, but make sure that they are relevant to the goal and focus of this assignment. If it is not clear, consult me.

In general, difficulty is preferred over quantity. Original additional features are encouraged, as long as they meet the above requirements.

Don't forget to mention and shortly describe your implemented additional features in the report. Add a short motivation if you implemented a feature which is not in this list.

Ideas

This is a list of ideas for additional features, roughly in order of increasing difficulty.

Interactive fractal parameters

Implement the Julia fractal and interactively change the parameter.

Interactive coloring

Let the user modify the color scheme interactively.

Dynamic number of iterations

Adjust the maximum number of iterations based on the frame rate, in order to keep the frame rate constant.

Compute Shader

Execute the Mandelbrot function using a compute shader. (This is also possible on DirectX 10 hardware.)

Anti-aliasing

Super sample each pixel using different subsample offsets over the course of multiple frames (while averaging the result).

Reuse data while moving

Copy the visible data from the previous frame, and only evaluate the Mandelbrot function for the newly visible pixels.

Incremental Mandelbrot calculation

Continue iterating the Mandelbrot function in the next frame. To achieve this, write to multiple render targets from your shader to output both the intermediate results and the visible image.

Bloom

Pimp your visualization by adding bloom as a post processing effect.

Enhanced precision (using floats)

There are methods to get a higher (or even infinite) precision using multiple floats for each variable.

3D fractal

For example render the Mandelbulb.