C++ AMP Mandelbrot set

Matthew Wallace - 1502616@abertay.ac.uk
Data and Structures 2 – CMP202

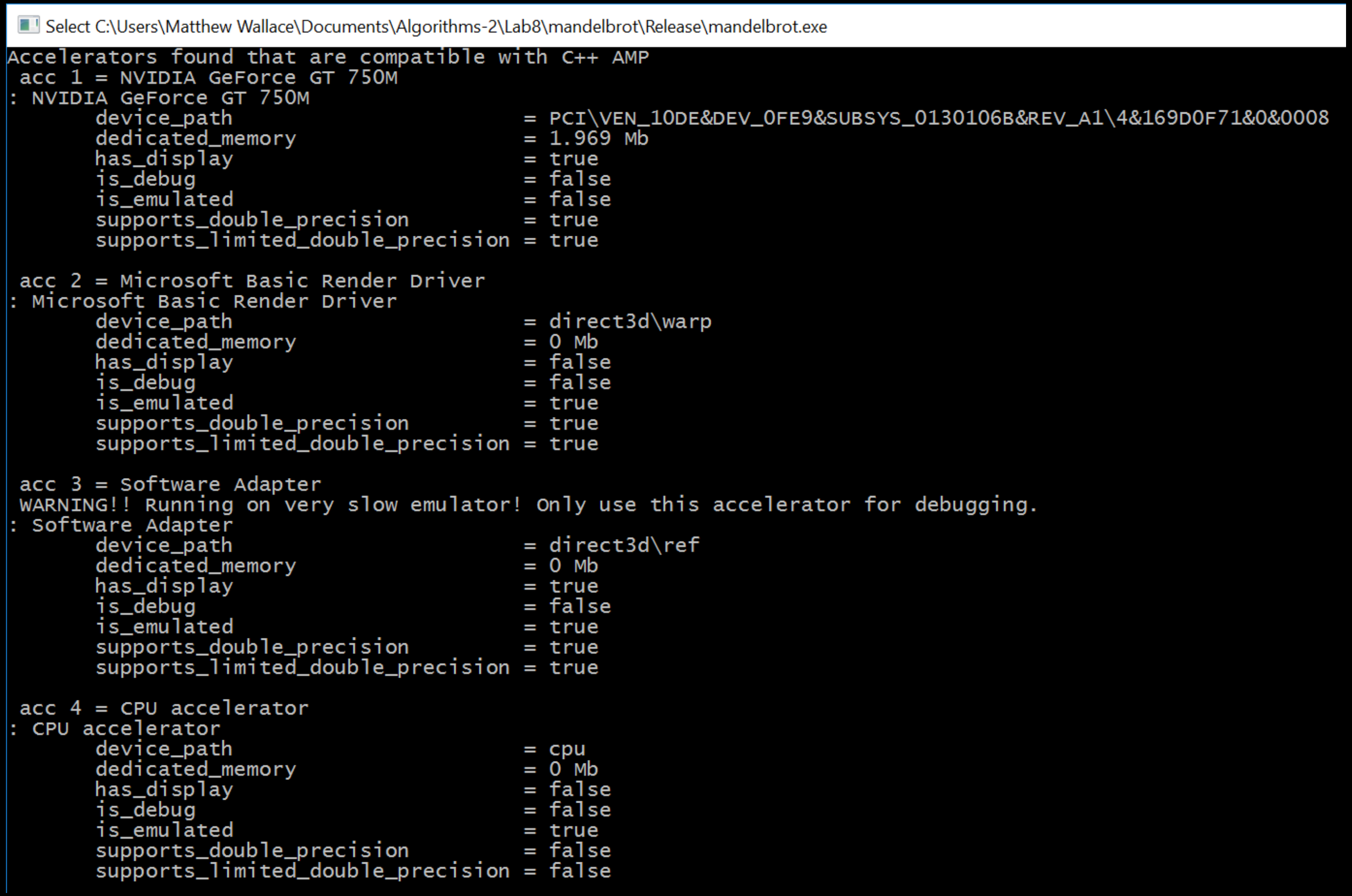
Presentation overview

- Computer and GPU specification
- Program structure
- Application build properties
- Explanation of 3 C++ AMP methods to calculate the Mandelbrot set
  - amp_mandelbrot
  - amp_pixel_mandelbrot
  - amp_barrier_mandlebrot
- Videos

Computer specification

- Windows 10 Education (BootCamp) x64
- Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz
- 16.0 GB RAM
- NVIDIA GeForce GT 750M 2GB

GPU specification



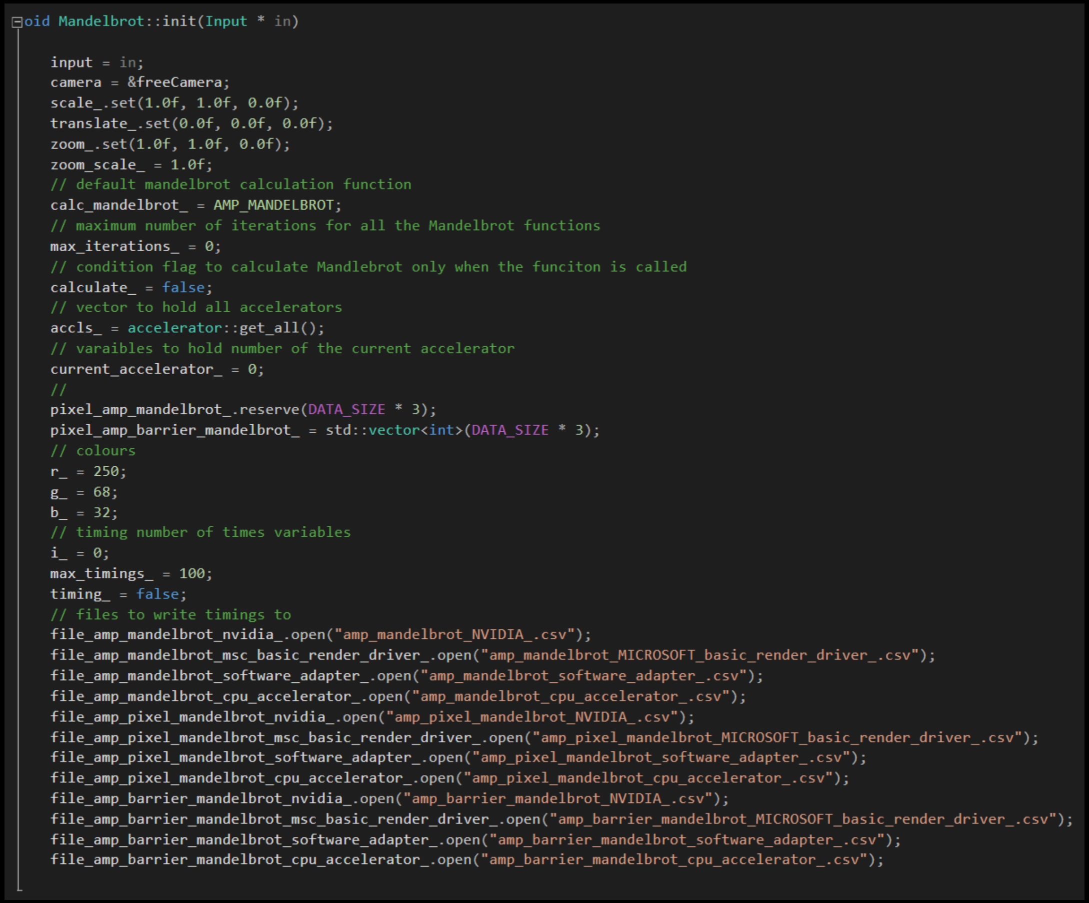
Select C:\Users\Matthew Wallace\Documents\Algorithms-2\Lab8\mandelbrot\Release\mandelbrot.exe

```
Accelerators found that are compatible with C++ AMP
 acc 1 = NVIDIA GeForce GT 750M
: NVIDIA GeForce GT 750M
       device_path                       = PCI\VEN_10DE&DEV_0FE9&SUBSYS_0130106B&REV_A1\4&169D0F71&0&0008
       dedicated_memory                  = 1.969 Mb
       has_display                       = true
       is_debug                          = false
       is_emulated                       = false
       supports_double_precision         = true
       supports_limited_double_precision = true

 acc 2 = Microsoft Basic Render Driver
: Microsoft Basic Render Driver
       device_path                       = direct3d\warp
       dedicated_memory                  = 0 Mb
       has_display                       = false
       is_debug                          = false
       is_emulated                       = true
       supports_double_precision         = true
       supports_limited_double_precision = true

 acc 3 = Software Adapter
 WARNING!! Running on very slow emulator! Only use this accelerator for debugging.
: Software Adapter
       device_path                       = direct3d\ref
       dedicated_memory                  = 0 Mb
       has_display                       = true
       is_debug                          = false
       is_emulated                       = true
       supports_double_precision         = true
       supports_limited_double_precision = true

 acc 4 = CPU accelerator
: CPU accelerator
       device_path                       = cpu
       dedicated_memory                  = 0 Mb
       has_display                       = false
       is_debug                          = false
       is_emulated                       = true
       supports_double_precision         = false
       supports_limited_double_precision = false
```

Program structure

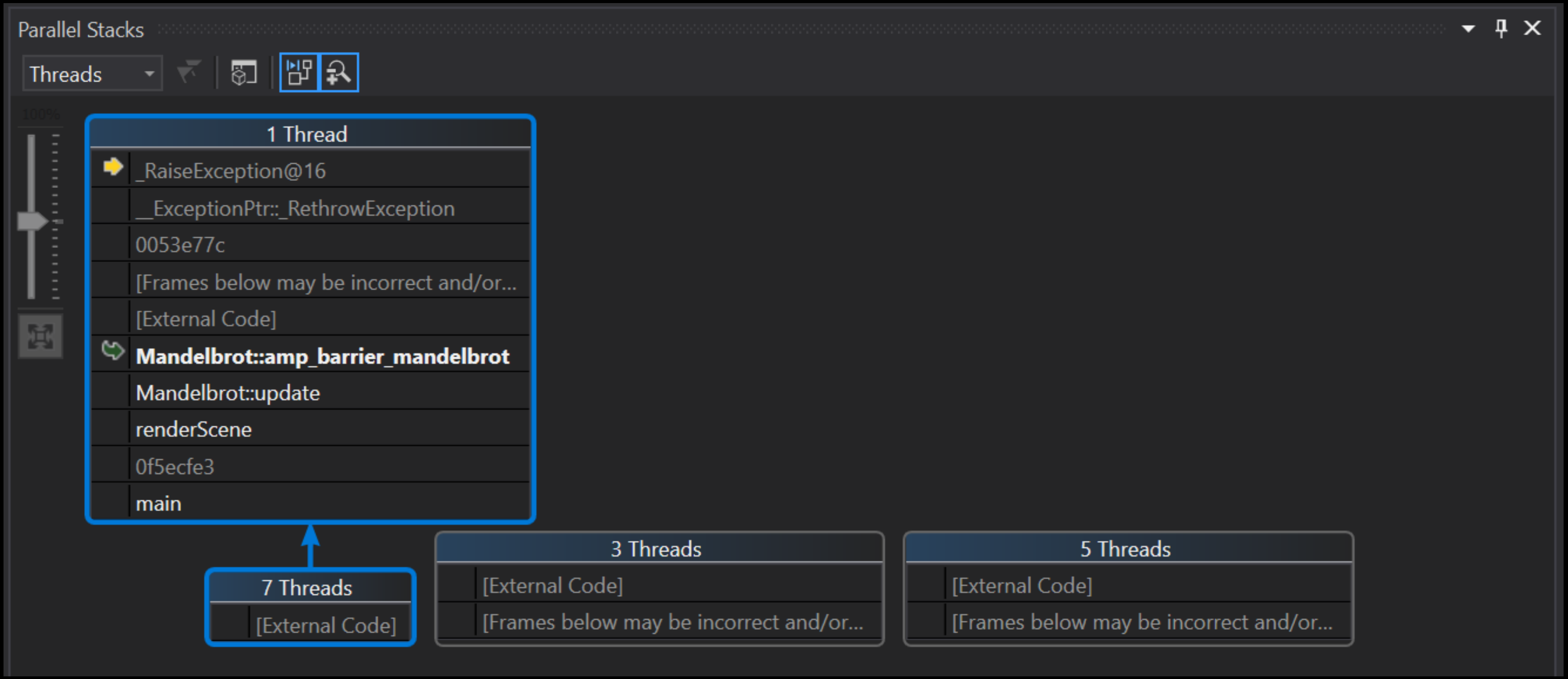
- The Mandelbrot set calculation methods put into a `mandelbrot` class
- Calculation methods are AMP restricted
- 3 different methods of creating pixel array to display the Mandelbrot set
- Passing maximum number of iterations and colour values to the parallel for each lambda function
- Possible to change those values during the runtime and to switch between 3 different Mandelbrot sets
- Update and render functions for the `mandelbrot` class
- Displaying the Mandelbrot set using glut API for OpenGL

Initilaze (`Mandelbrot` class constructor)



```
oid Mandelbrot::init(Input * in)

    input = in;
    camera = &freeCamera;
    scale_.set(1.0f, 1.0f, 0.0f);
    translate_.set(0.0f, 0.0f, 0.0f);
    zoom_.set(1.0f, 1.0f, 0.0f);
    zoom_scale_ = 1.0f;
    // default mandelbrot calculation function
    calc_mandelbrot_ = AMP_MANDELBROT;
    // maximum number of iterations for all the Mandelbrot functions
    max_iterations_ = 0;
    // condition flag to calculate Mandlebrot only when the funciton is called
    calculate_ = false;
    // vector to hold all accelerators
    accls_ = accelerator::get_all();
    // varaibles to hold number of the current accelerator
    current_accelerator_ = 0;
    //
    pixel_amp_mandelbrot_.reserve(DATA_SIZE * 3);
    pixel_amp_barrier_mandelbrot_ = std::vector<int>(DATA_SIZE * 3);
    // colours
    r_ = 250;
    g_ = 68;
    b_ = 32;
    // timing number of times variables
    i_ = 0;
    max_timings_ = 100;
    timing_ = false;
    // files to write timings to
    file_amp_mandelbrot_nvidia_.open("amp_mandelbrot_NVIDIA_.csv");
    file_amp_mandelbrot_msc_basic_render_driver_.open("amp_mandelbrot_MICROSOFT_basic_render_driver_.csv");
    file_amp_mandelbrot_software_adapter_.open("amp_mandelbrot_software_adapter_.csv");
    file_amp_mandelbrot_cpu_accelerator_.open("amp_mandelbrot_cpu_accelerator_.csv");
    file_amp_pixel_mandelbrot_nvidia_.open("amp_pixel_mandelbrot_NVIDIA_.csv");
    file_amp_pixel_mandelbrot_msc_basic_render_driver_.open("amp_pixel_mandelbrot_MICROSOFT_basic_render_driver_.csv");
    file_amp_pixel_mandelbrot_software_adapter_.open("amp_pixel_mandelbrot_software_adapter_.csv");
    file_amp_pixel_mandelbrot_cpu_accelerator_.open("amp_pixel_mandelbrot_cpu_accelerator_.csv");
    file_amp_barrier_mandelbrot_nvidia_.open("amp_barrier_mandelbrot_NVIDIA_.csv");
    file_amp_barrier_mandelbrot_msc_basic_render_driver_.open("amp_barrier_mandelbrot_MICROSOFT_basic_render_driver_.csv");
    file_amp_barrier_mandelbrot_software_adapter_.open("amp_barrier_mandelbrot_software_adapter_.csv");
    file_amp_barrier_mandelbrot_cpu_accelerator_.open("amp_barrier_mandelbrot_cpu_accelerator_.csv");
```

Parallel stack



1 Thread

- _RaiseException@16
- __ExceptionPtr::_RethrowException
- 0053e77c
- [Frames below may be incorrect and/or...
- [External Code]
- **Mandelbrot::amp_barrier_mandelbrot**
- Mandelbrot::update
- renderScene
- 0f5ecfe3
- main

7 Threads

- [External Code]

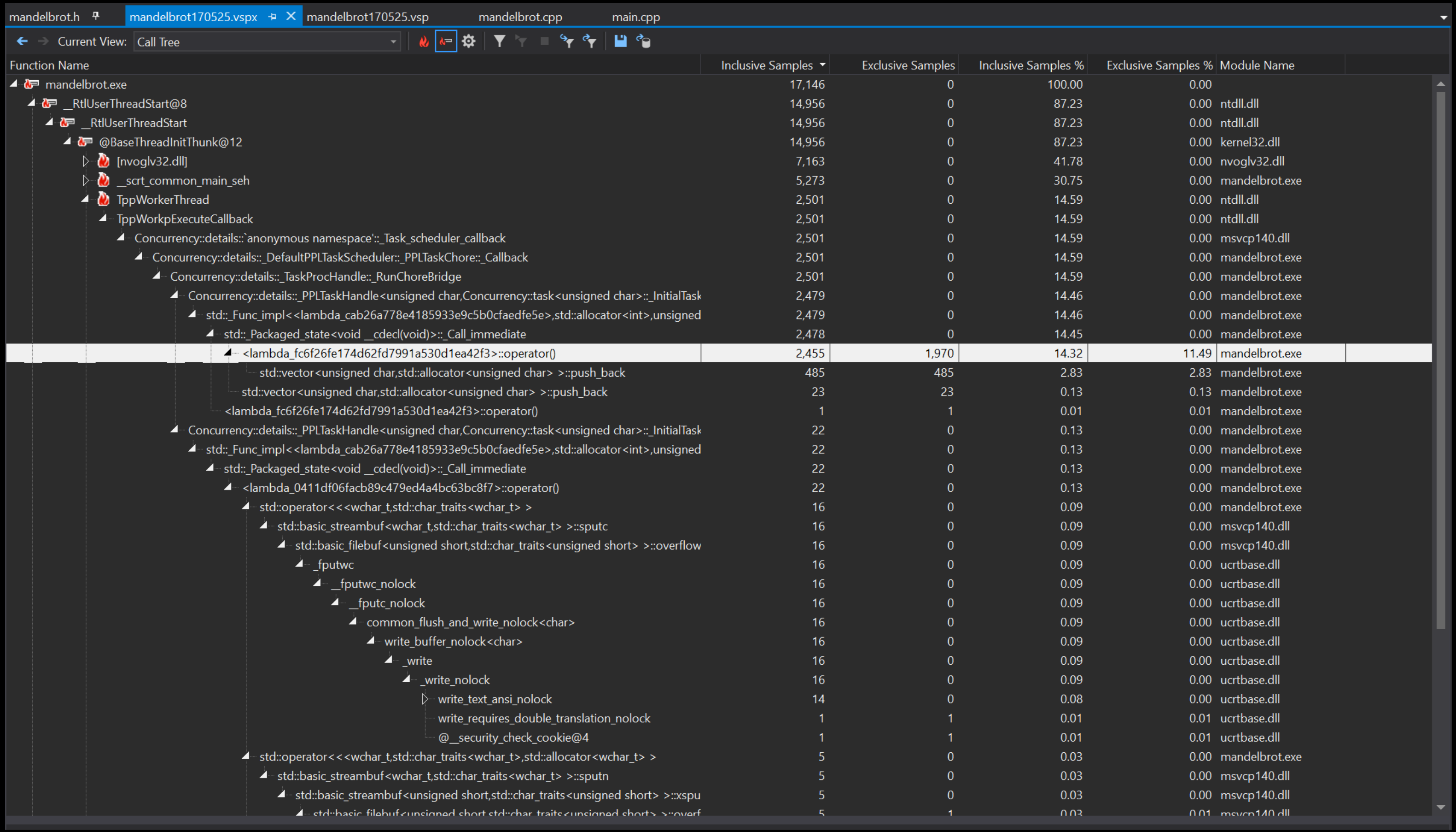
3 Threads

- [External Code]
- [Frames below may be incorrect and/or...

5 Threads

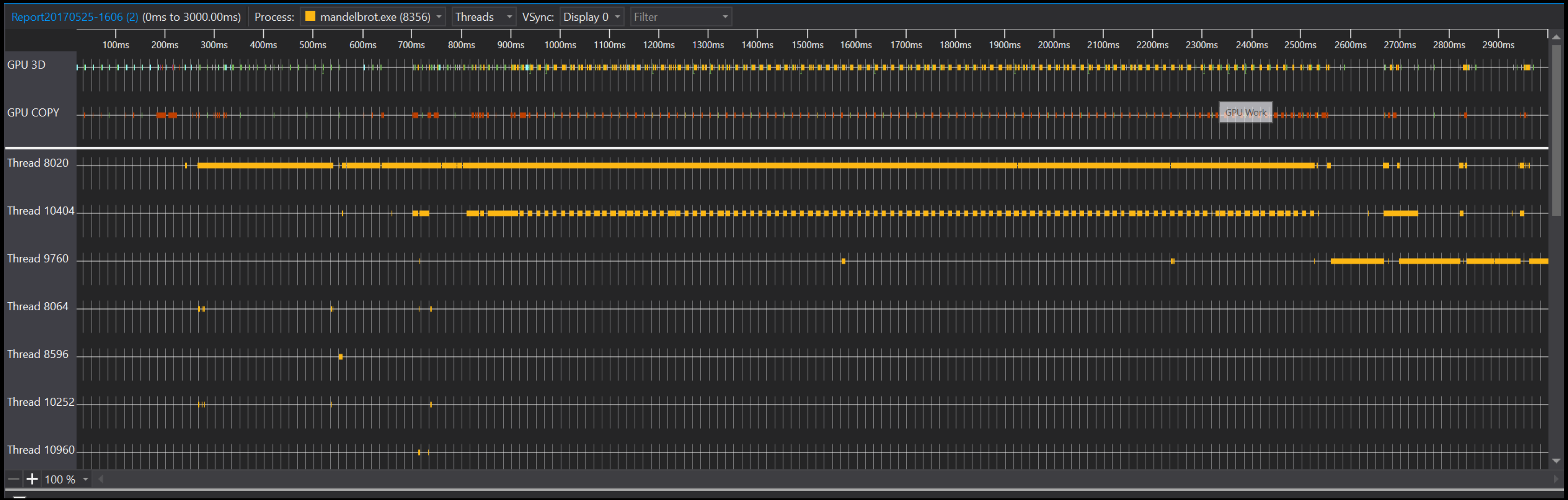
- [External Code]
- [Frames below may be incorrect and/or...

Call tree



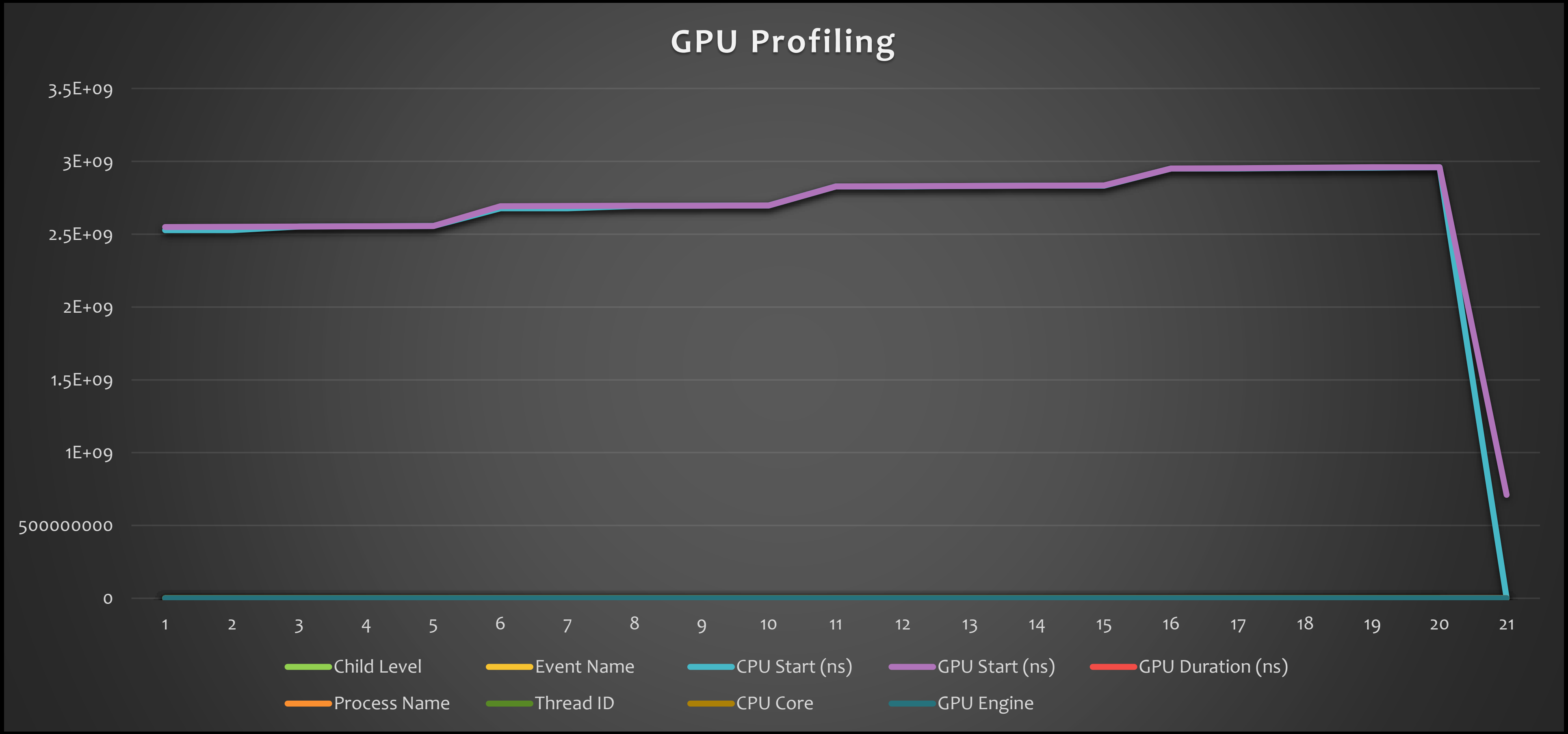
Function Name	Inclusive Samples	Exclusive Samples	Inclusive Samples %	Exclusive Samples %	Module Name
mandelbrot.exe	17,146	0	100.00	0.00	
_RtlUserThreadStart@8	14,956	0	87.23	0.00	ntdll.dll
__RtlUserThreadStart	14,956	0	87.23	0.00	ntdll.dll
@BaseThreadInitThunk@12	14,956	0	87.23	0.00	kernel32.dll
[nvoglv32.dll]	7,163	0	41.78	0.00	nvoglv32.dll
__scrt_common_main_seh	5,273	0	30.75	0.00	mandelbrot.exe
TppWorkerThread	2,501	0	14.59	0.00	ntdll.dll
TppWorkpExecuteCallback	2,501	0	14.59	0.00	ntdll.dll
Concurrency::details::`anonymous namespace'::_Task_scheduler_callback	2,501	0	14.59	0.00	msvcp140.dll
Concurrency::details::_DefaultPPLTaskScheduler::_PPLTaskChore::_Callback	2,501	0	14.59	0.00	mandelbrot.exe
Concurrency::details::_TaskProcHandle::_RunChoreBridge	2,501	0	14.59	0.00	mandelbrot.exe
Concurrency::details::_PPLTaskHandle<unsigned char,Concurrency::task<unsigned char>::_InitialTask	2,479	0	14.46	0.00	mandelbrot.exe
std::_Func_impl<<lambda_cab26a778e4185933e9c5b0cfaedfe5e>,std::allocator<int>,unsigned	2,479	0	14.46	0.00	mandelbrot.exe
std::_Packaged_state<void __cdecl(void)>::_Call_immediate	2,478	0	14.45	0.00	mandelbrot.exe
<lambda_fc6f26fe174d62fd7991a530d1ea42f3>::operator()	2,455	1,970	14.32	11.49	mandelbrot.exe
std::vector<unsigned char,std::allocator<unsigned char> >::push_back	485	485	2.83	2.83	mandelbrot.exe
std::vector<unsigned char,std::allocator<unsigned char> >::push_back	23	23	0.13	0.13	mandelbrot.exe
<lambda_fc6f26fe174d62fd7991a530d1ea42f3>::operator()	1	1	0.01	0.01	mandelbrot.exe
Concurrency::details::_PPLTaskHandle<unsigned char,Concurrency::task<unsigned char>::_InitialTask	22	0	0.13	0.00	mandelbrot.exe
std::_Func_impl<<lambda_cab26a778e4185933e9c5b0cfaedfe5e>,std::allocator<int>,unsigned	22	0	0.13	0.00	mandelbrot.exe
std::_Packaged_state<void __cdecl(void)>::_Call_immediate	22	0	0.13	0.00	mandelbrot.exe
<lambda_0411df06facb89c479ed4a4bc63bc8f7>::operator()	22	0	0.13	0.00	mandelbrot.exe
std::operator<<<wchar_t,std::char_traits<wchar_t> >	16	0	0.09	0.00	mandelbrot.exe
std::basic_streambuf<wchar_t,std::char_traits<wchar_t> >::sputc	16	0	0.09	0.00	msvcp140.dll
std::basic_filebuf<unsigned short,std::char_traits<unsigned short> >::overflow	16	0	0.09	0.00	msvcp140.dll
_fputwc	16	0	0.09	0.00	ucrtbase.dll
__fputwc_nolock	16	0	0.09	0.00	ucrtbase.dll
__fputc_nolock	16	0	0.09	0.00	ucrtbase.dll
common_flush_and_write_nolock<char>	16	0	0.09	0.00	ucrtbase.dll
write_buffer_nolock<char>	16	0	0.09	0.00	ucrtbase.dll
_write	16	0	0.09	0.00	ucrtbase.dll
_write_nolock	16	0	0.09	0.00	ucrtbase.dll
write_text_ansi_nolock	14	0	0.08	0.00	ucrtbase.dll
write_requires_double_translation_nolock	1	1	0.01	0.01	ucrtbase.dll
@__security_check_cookie@4	1	1	0.01	0.01	ucrtbase.dll
std::operator<<<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t> >	5	0	0.03	0.00	mandelbrot.exe
std::basic_streambuf<wchar_t,std::char_traits<wchar_t> >::sputn	5	0	0.03	0.00	msvcp140.dll
std::basic_streambuf<unsigned short,std::char_traits<unsigned short> >::xspu	5	0	0.03	0.00	msvcp140.dll
std::basic_filebuf<unsigned short,std::char_traits<unsigned short> >::overf	5	1	0.03	0.01	msvcp140.dll

GPU profiling



Report20170525-1606 (2) (0ms to 3000.00ms) Process: mandelbrot.exe (8356) Threads VSync: Display 0 Filter

GPU profiling



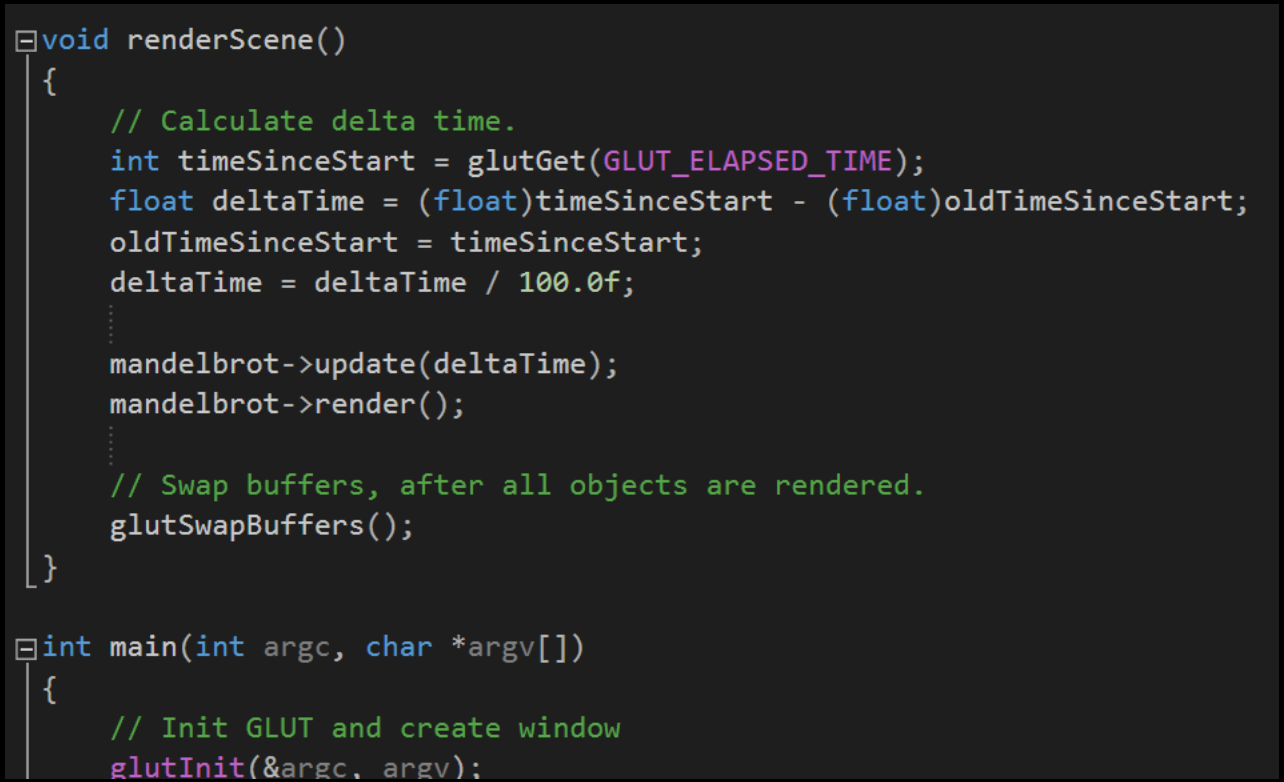
update

- Handle input
- Update variables
- Set accelerators
- Set the Mandelbrot set to display
- Calculate the Mandelbrot set with the chosen method
  - Calculate only when the function is being called
  - Time how long it took to calculate
  - Put results into files depending on the accelerator and the method
- Update Camera

render

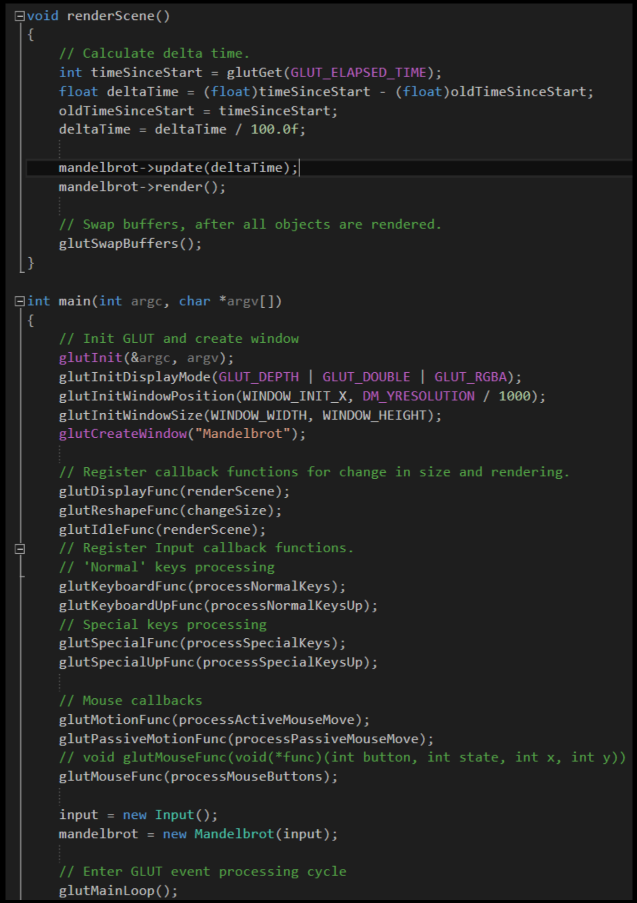
- Set the camera
- Create texture
- Apply texture to the quad (made out of two triangles)
- Switch between 3 Mandelbrot sets

main.cpp



```
void renderScene()
{
    // Calculate delta time.
    int timeSinceStart = glutGet(GLUT_ELAPSED_TIME);
    float deltaTime = (float)timeSinceStart - (float)oldTimeSinceStart;
    oldTimeSinceStart = timeSinceStart;
    deltaTime = deltaTime / 100.0f;

    mandelbrot->update(deltaTime);
    mandelbrot->render();

    // Swap buffers, after all objects are rendered.
    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    // Init GLUT and create window
    glutInit(&argc, argv);
```

main.cpp

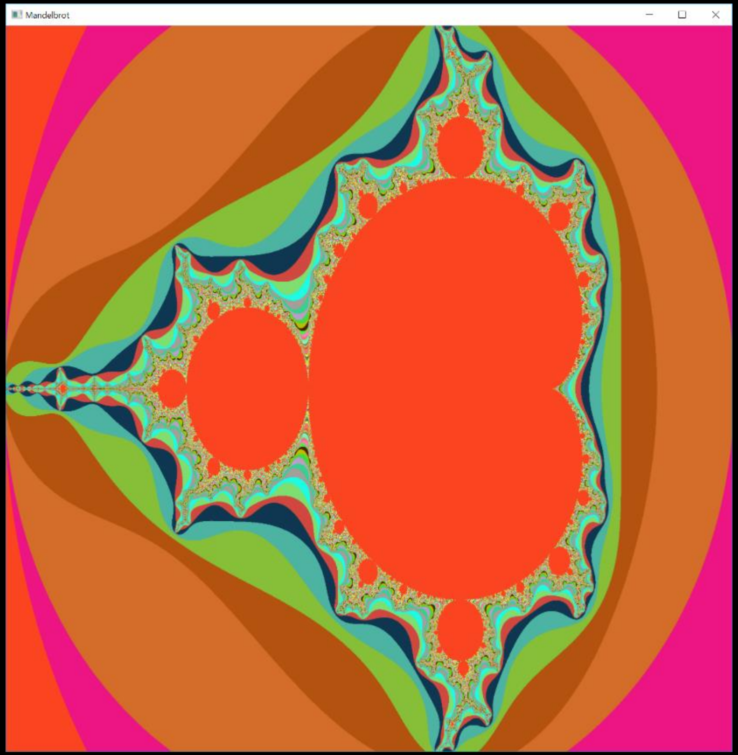


```cpp
void renderScene()
{
    // Calculate delta time.
    int timeSinceStart = glutGet(GLUT_ELAPSED_TIME);
    float deltaTime = (float)timeSinceStart - (float)oldTimeSinceStart;
    oldTimeSinceStart = timeSinceStart;
    deltaTime = deltaTime / 100.0f;

    mandelbrot->update(deltaTime);
    mandelbrot->render();

    // Swap buffers, after all objects are rendered.
    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    // Init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(WINDOW_INIT_X, DM_YRESOLUTION / 1000);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("Mandelbrot");

    // Register callback functions for change in size and rendering.
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(renderScene);
    // Register Input callback functions.
    // 'Normal' keys processing
    glutKeyboardFunc(processNormalKeys);
    glutKeyboardUpFunc(processNormalKeysUp);
    // Special keys processing
    glutSpecialFunc(processSpecialKeys);
    glutSpecialUpFunc(processSpecialKeysUp);

    // Mouse callbacks
    glutMotionFunc(processActiveMouseMove);
    glutPassiveMotionFunc(processPassiveMouseMove);
    // void glutMouseFunc(void(*func)(int button, int state, int x, int y))
    glutMouseFunc(processMouseButtons);

    input = new Input();
    mandelbrot = new Mandelbrot(input);

    // Enter GLUT event processing cycle
    glutMainLoop();
```

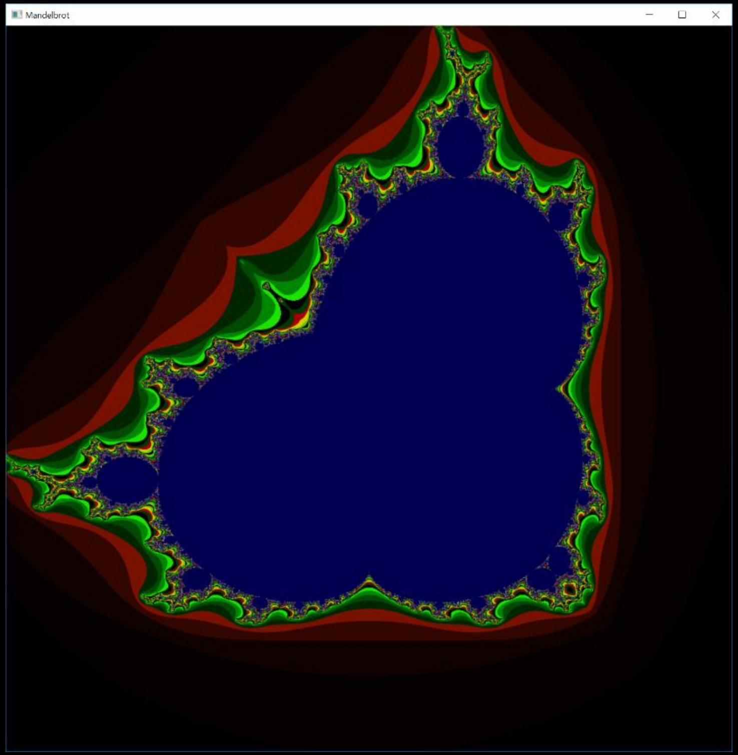
Application build properties

- Solution configuration – Release
- Solution Platform – Win32
- SubSystem – Console (for couting variables)

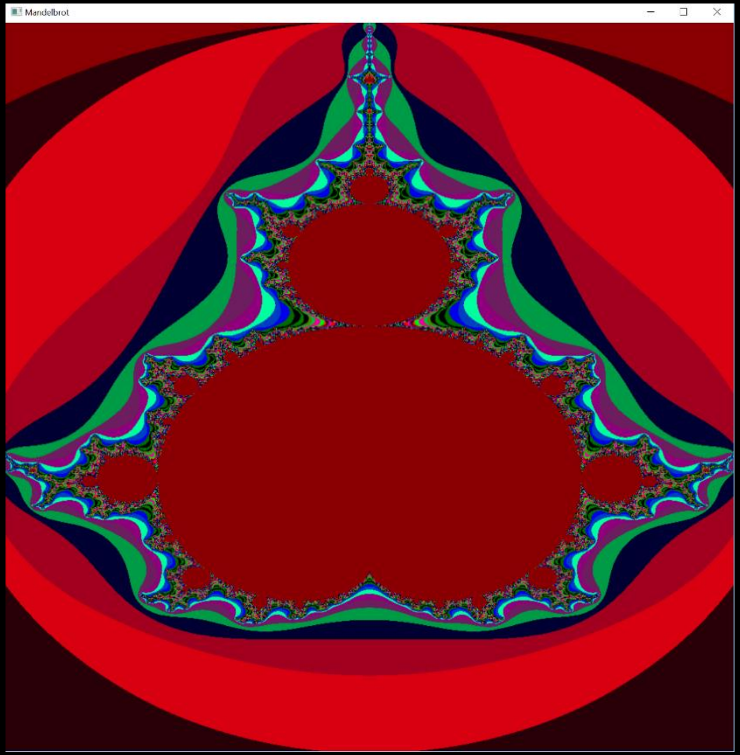
Explanation of 3 C++ AMP methods to calculate the Mandelbrot set



amp_mandelbrot



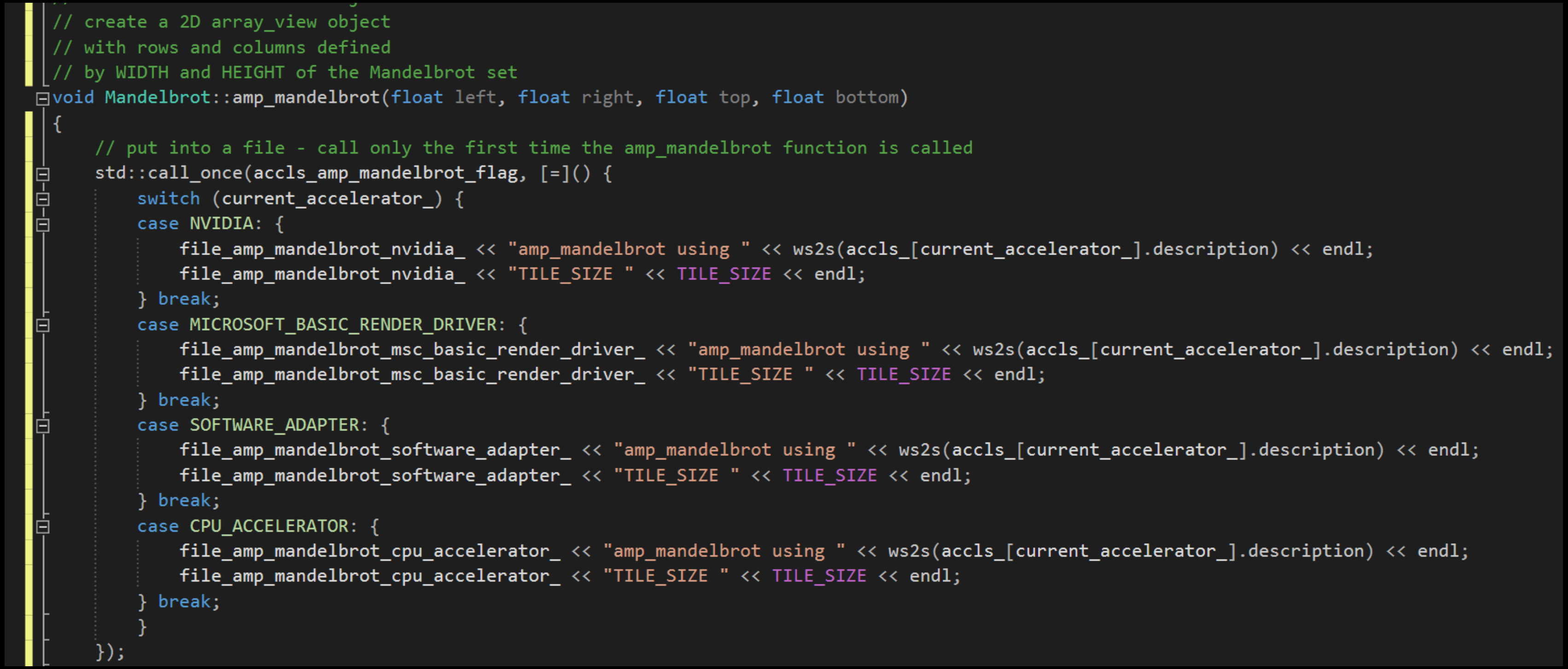
amp_pixel_mandelbrot



amp_barrier_mandelbrot

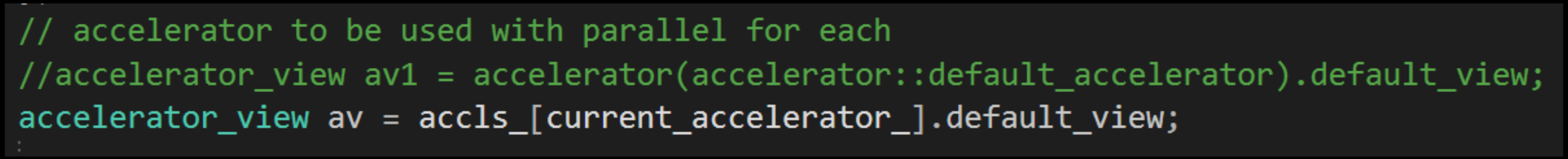
amp_mandelbrot function structure

- call_once function puts information about what function, accelerator and number of tiles are currently being used



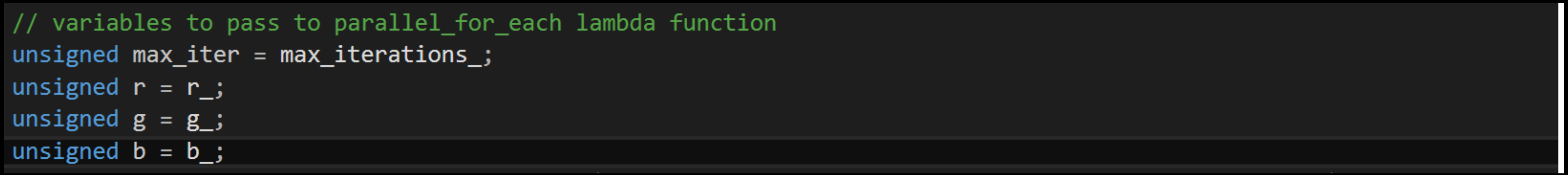
```cpp
// create a 2D array_view object
// with rows and columns defined
// by WIDTH and HEIGHT of the Mandelbrot set
void Mandelbrot::amp_mandelbrot(float left, float right, float top, float bottom)
{
    // put into a file - call only the first time the amp_mandelbrot function is called
    std::call_once(accls_amp_mandelbrot_flag, [=]() {
        switch (current_accelerator_) {
        case NVIDIA: {
            file_amp_mandelbrot_nvidia_ << "amp_mandelbrot using " << ws2s(accls_[current_accelerator_].description) << endl;
            file_amp_mandelbrot_nvidia_ << "TILE_SIZE " << TILE_SIZE << endl;
        } break;
        case MICROSOFT_BASIC_RENDER_DRIVER: {
            file_amp_mandelbrot_msc_basic_render_driver_ << "amp_mandelbrot using " << ws2s(accls_[current_accelerator_].description) << endl;
            file_amp_mandelbrot_msc_basic_render_driver_ << "TILE_SIZE " << TILE_SIZE << endl;
        } break;
        case SOFTWARE_ADAPTER: {
            file_amp_mandelbrot_software_adapter_ << "amp_mandelbrot using " << ws2s(accls_[current_accelerator_].description) << endl;
            file_amp_mandelbrot_software_adapter_ << "TILE_SIZE " << TILE_SIZE << endl;
        } break;
        case CPU_ACCELERATOR: {
            file_amp_mandelbrot_cpu_accelerator_ << "amp_mandelbrot using " << ws2s(accls_[current_accelerator_].description) << endl;
            file_amp_mandelbrot_cpu_accelerator_ << "TILE_SIZE " << TILE_SIZE << endl;
        } break;
        }
    });
```

Set a default accelerator



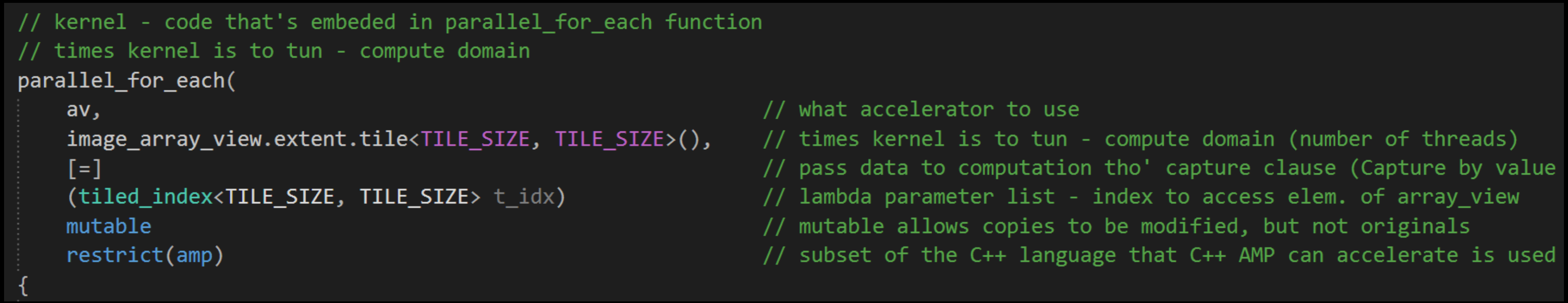
```
// accelerator to be used with parallel for each
//accelerator_view av1 = accelerator(accelerator::default_accelerator).default_view;
accelerator_view av = accls_[current_accelerator_].default_view;
```

Set variables to pass to the kernel



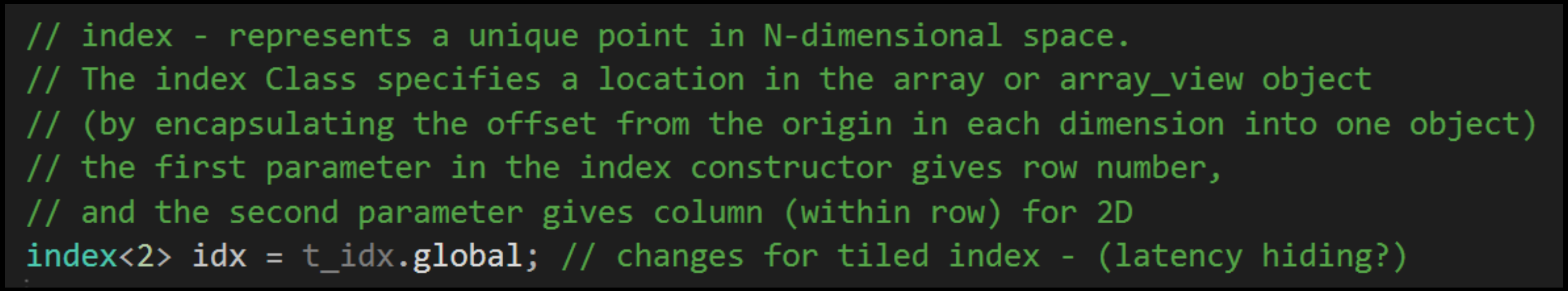
```
// variables to pass to parallel_for_each lambda function
unsigned max_iter = max_iterations_;
unsigned r = r_;
unsigned g = g_;
unsigned b = b_;
```

amp_mandelbrot - kernel



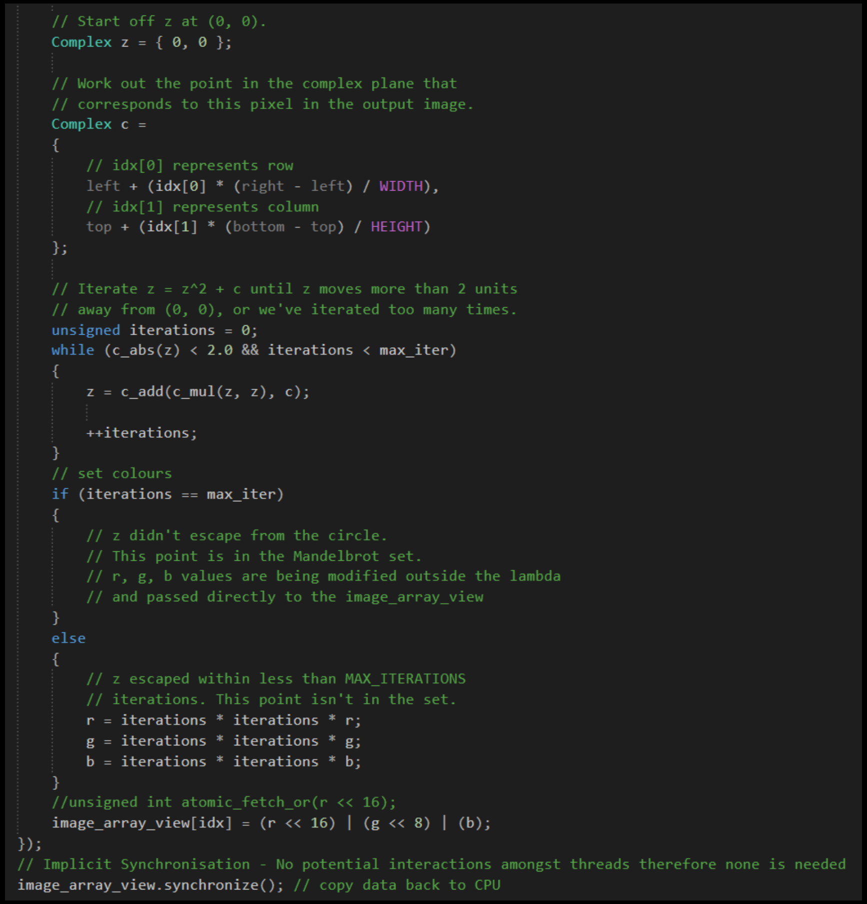
```
// kernel - code that's embeded in parallel_for_each function
// times kernel is to tun - compute domain
parallel_for_each(
    av,                                                      // what accelerator to use
    image_array_view.extent.tile<TILE_SIZE, TILE_SIZE>(),    // times kernel is to tun - compute domain (number of threads)
    [=]                                                      // pass data to computation tho' capture clause (Capture by value
    (tiled_index<TILE_SIZE, TILE_SIZE> t_idx)                // lambda parameter list - index to access elem. of array_view
    mutable                                                  // mutable allows copies to be modified, but not originals
    restrict(amp)                                            // subset of the C++ language that C++ AMP can accelerate is used
{
```

kernel - index



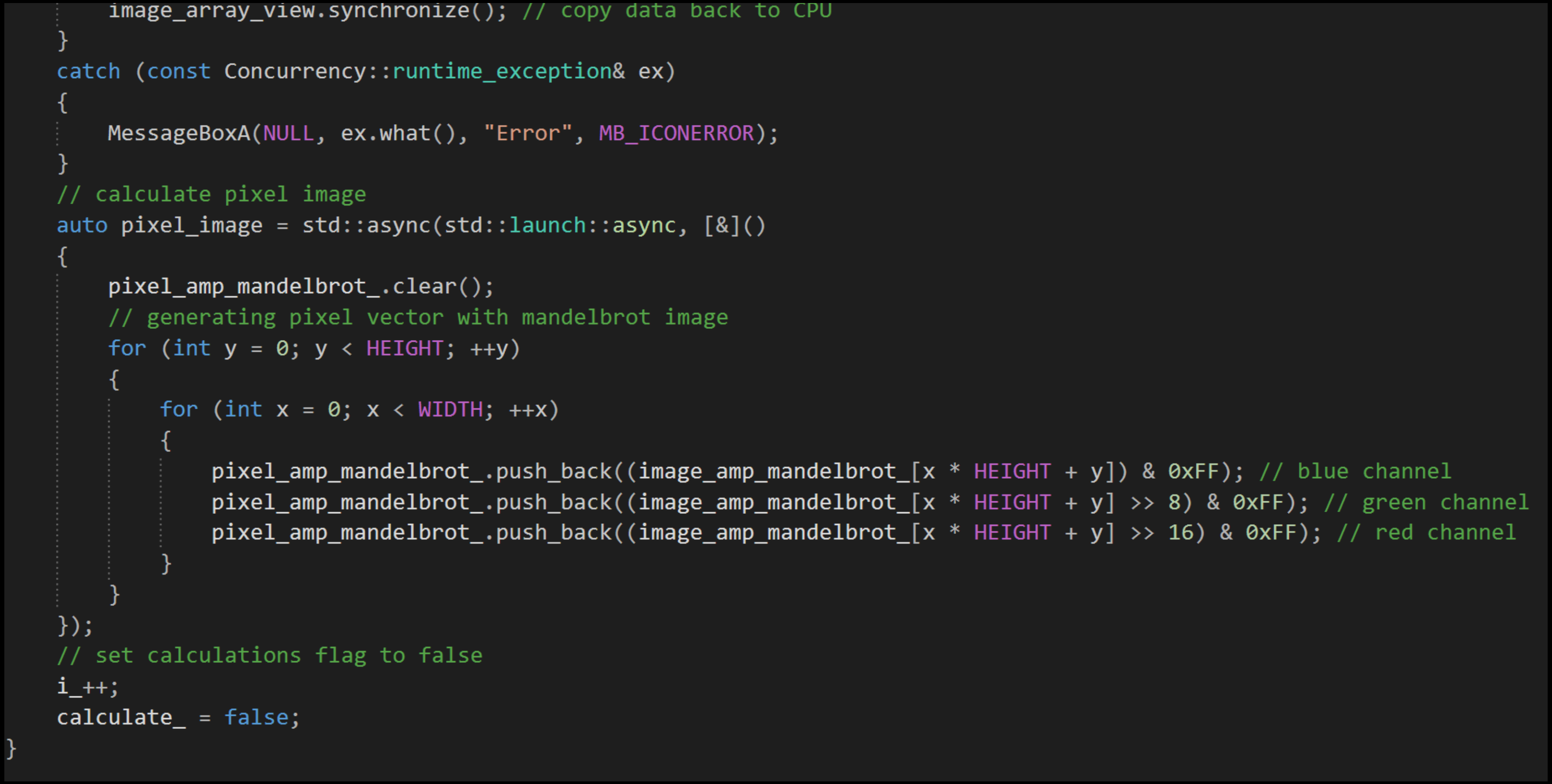
```
// index - represents a unique point in N-dimensional space.
// The index Class specifies a location in the array or array_view object
// (by encapsulating the offset from the origin in each dimension into one object)
// the first parameter in the index constructor gives row number,
// and the second parameter gives column (within row) for 2D
index<2> idx = t_idx.global; // changes for tiled index - (latency hiding?)
```

kernel – calculate the Mandelbrot set



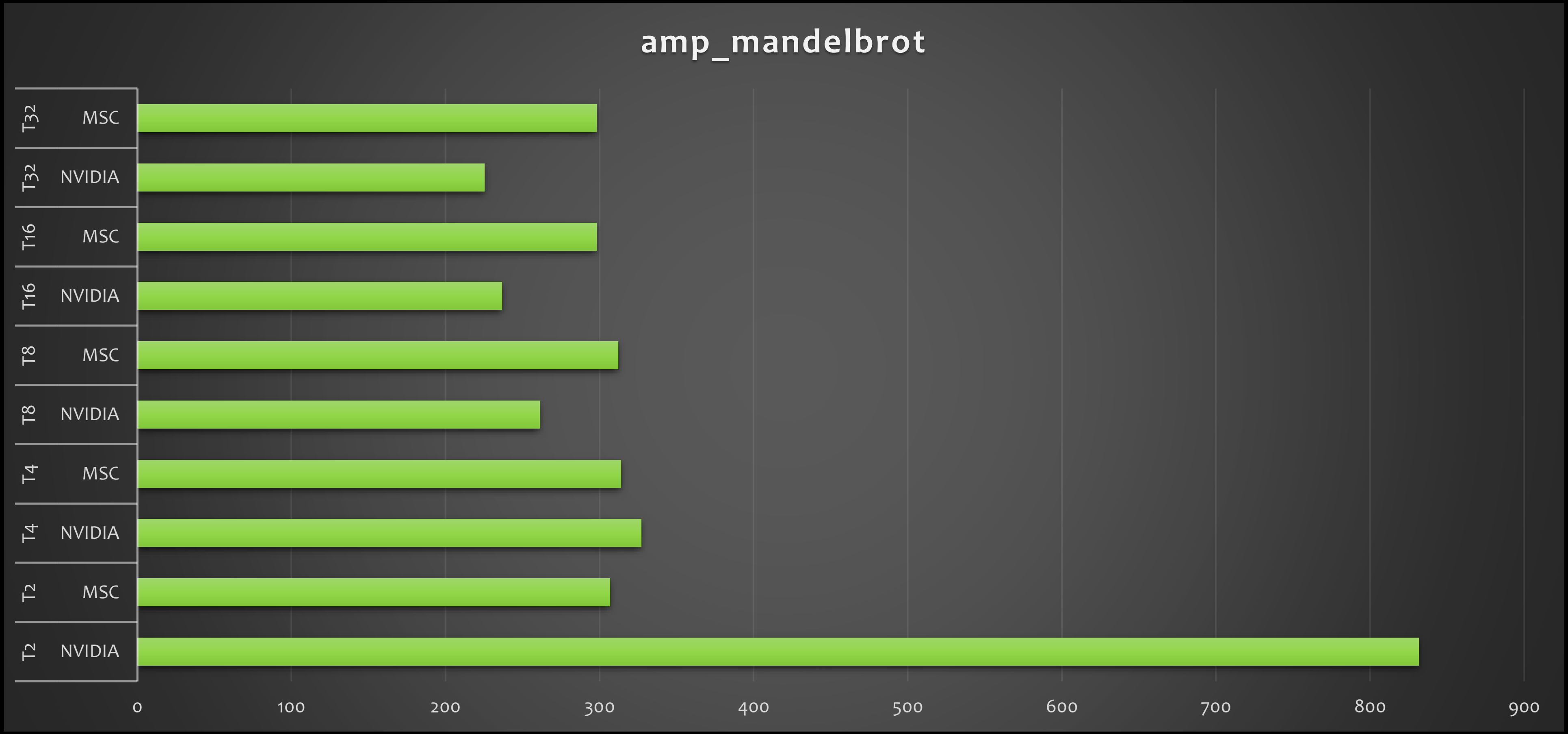
```
    // Start off z at (0, 0).
    Complex z = { 0, 0 };

    // Work out the point in the complex plane that
    // corresponds to this pixel in the output image.
    Complex c =
    {
        // idx[0] represents row
        left + (idx[0] * (right - left) / WIDTH),
        // idx[1] represents column
        top + (idx[1] * (bottom - top) / HEIGHT)
    };

    // Iterate z = z^2 + c until z moves more than 2 units
    // away from (0, 0), or we've iterated too many times.
    unsigned iterations = 0;
    while (c_abs(z) < 2.0 && iterations < max_iter)
    {
        z = c_add(c_mul(z, z), c);

        ++iterations;
    }
    // set colours
    if (iterations == max_iter)
    {
        // z didn't escape from the circle.
        // This point is in the Mandelbrot set.
        // r, g, b values are being modified outside the lambda
        // and passed directly to the image_array_view
    }
    else
    {
        // z escaped within less than MAX_ITERATIONS
        // iterations. This point isn't in the set.
        r = iterations * iterations * r;
        g = iterations * iterations * g;
        b = iterations * iterations * b;
    }
    //unsigned int atomic_fetch_or(r << 16);
    image_array_view[idx] = (r << 16) | (g << 8) | (b);
});
// Implicit Synchronisation - No potential interactions amongst threads therefore none is needed
image_array_view.synchronize(); // copy data back to CPU
```

async pixel array creation



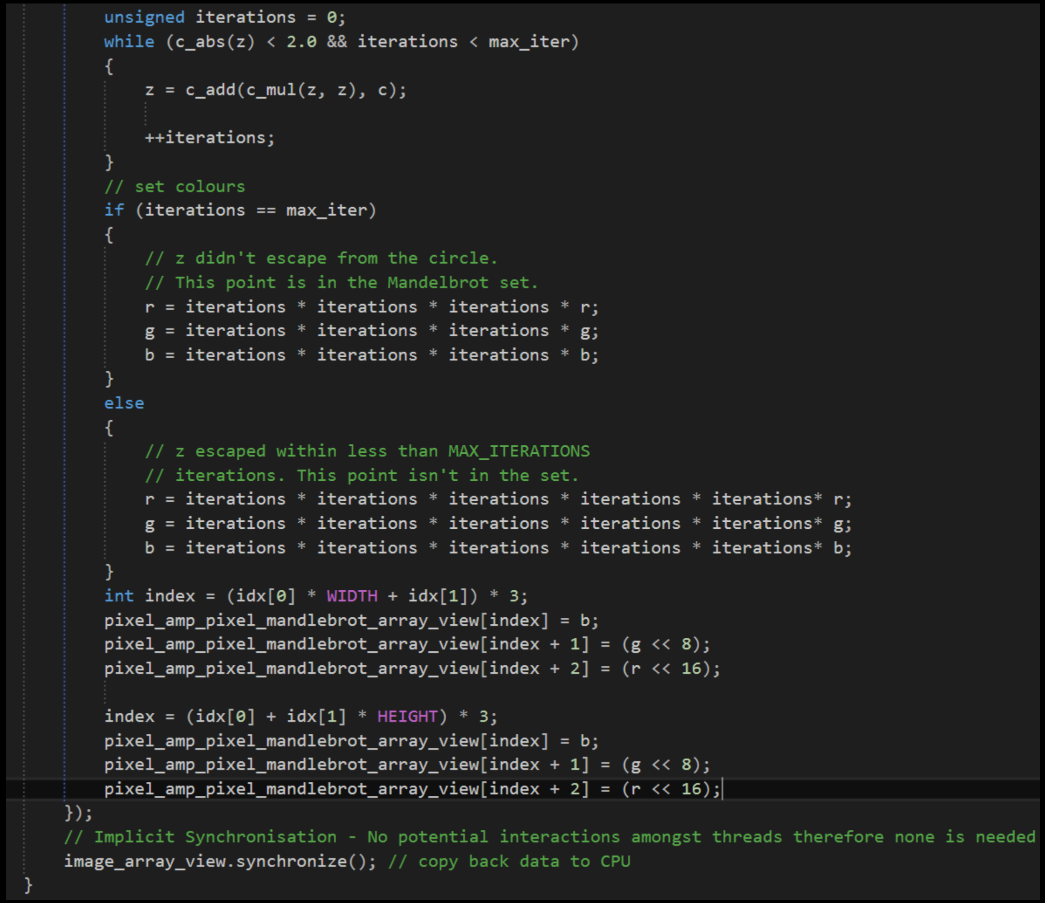
```
        image_array_view.synchronize(); // copy data back to CPU
    }
    catch (const Concurrency::runtime_exception& ex)
    {
        MessageBoxA(NULL, ex.what(), "Error", MB_ICONERROR);
    }
    // calculate pixel image
    auto pixel_image = std::async(std::launch::async, [&]()
    {
        pixel_amp_mandelbrot_.clear();
        // generating pixel vector with mandelbrot image
        for (int y = 0; y < HEIGHT; ++y)
        {
            for (int x = 0; x < WIDTH; ++x)
            {
                pixel_amp_mandelbrot_.push_back((image_amp_mandelbrot_[x * HEIGHT + y]) & 0xFF); // blue channel
                pixel_amp_mandelbrot_.push_back((image_amp_mandelbrot_[x * HEIGHT + y] >> 8) & 0xFF); // green channel
                pixel_amp_mandelbrot_.push_back((image_amp_mandelbrot_[x * HEIGHT + y] >> 16) & 0xFF); // red channel
            }
        }
    });
    // set calculations flag to false
    i_++;
    calculate_ = false;
}
```

amp_mandelbrot



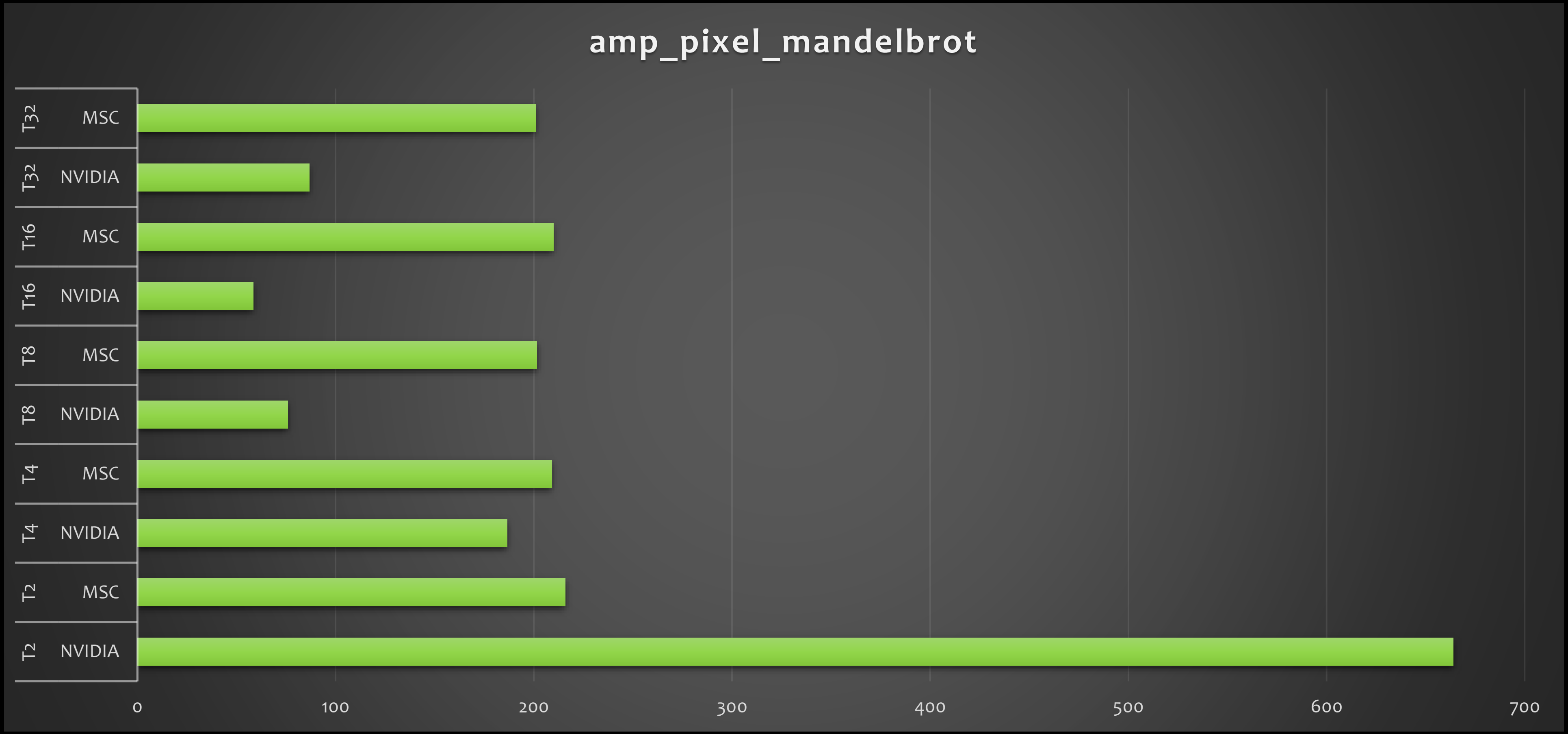
amp_Mandelbrot video

amp_pixel_mandelbrot



```
        unsigned iterations = 0;
        while (c_abs(z) < 2.0 && iterations < max_iter)
        {
            z = c_add(c_mul(z, z), c);

            ++iterations;
        }
        // set colours
        if (iterations == max_iter)
        {
            // z didn't escape from the circle.
            // This point is in the Mandelbrot set.
            r = iterations * iterations * iterations * r;
            g = iterations * iterations * iterations * g;
            b = iterations * iterations * iterations * b;
        }
        else
        {
            // z escaped within less than MAX_ITERATIONS
            // iterations. This point isn't in the set.
            r = iterations * iterations * iterations * iterations * iterations* r;
            g = iterations * iterations * iterations * iterations * iterations* g;
            b = iterations * iterations * iterations * iterations * iterations* b;
        }
        int index = (idx[0] * WIDTH + idx[1]) * 3;
        pixel_amp_pixel_mandlebrot_array_view[index] = b;
        pixel_amp_pixel_mandlebrot_array_view[index + 1] = (g << 8);
        pixel_amp_pixel_mandlebrot_array_view[index + 2] = (r << 16);

        index = (idx[0] + idx[1] * HEIGHT) * 3;
        pixel_amp_pixel_mandlebrot_array_view[index] = b;
        pixel_amp_pixel_mandlebrot_array_view[index + 1] = (g << 8);
        pixel_amp_pixel_mandlebrot_array_view[index + 2] = (r << 16);
    });
    // Implicit Synchronisation - No potential interactions amongst threads therefore none is needed
    image_array_view.synchronize(); // copy back data to CPU
}
```

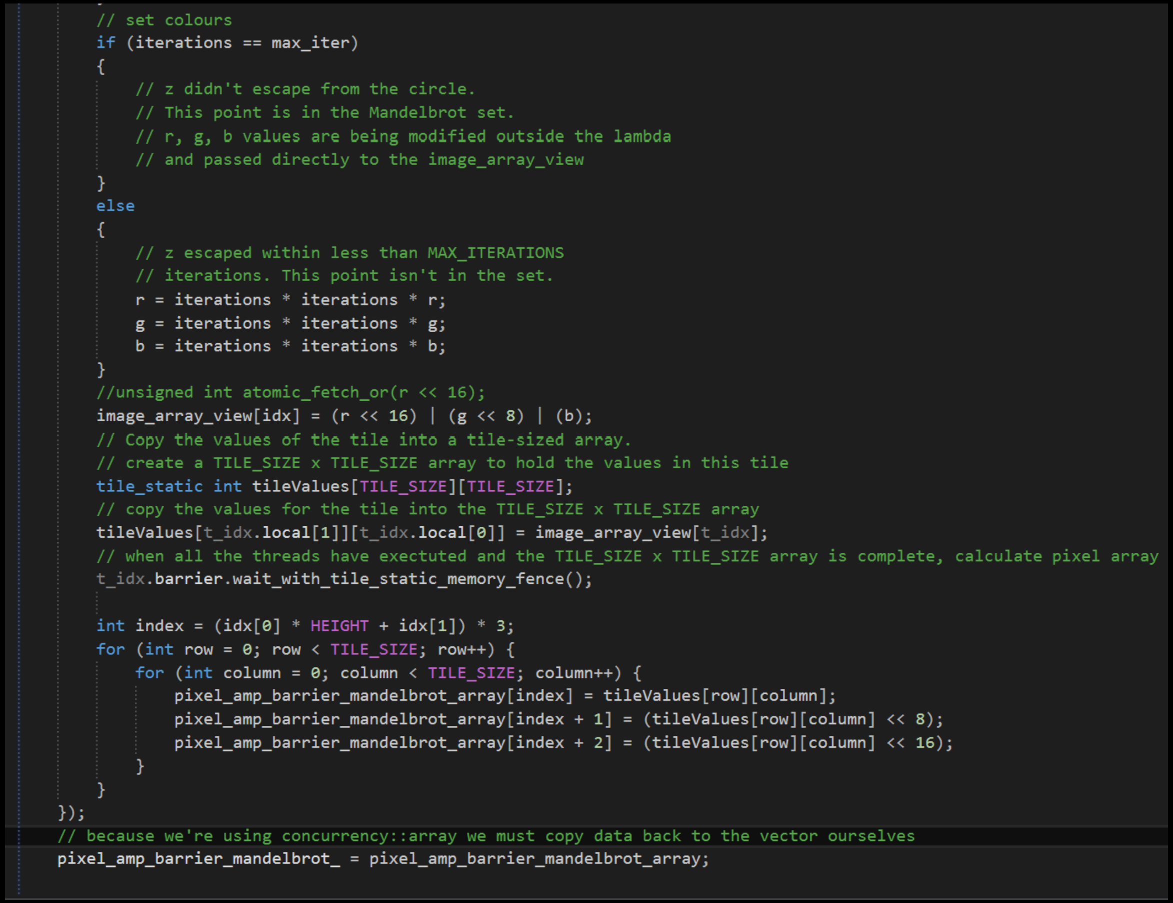
amp_pixel_mandelbrot



pixel_Mandelbrot video

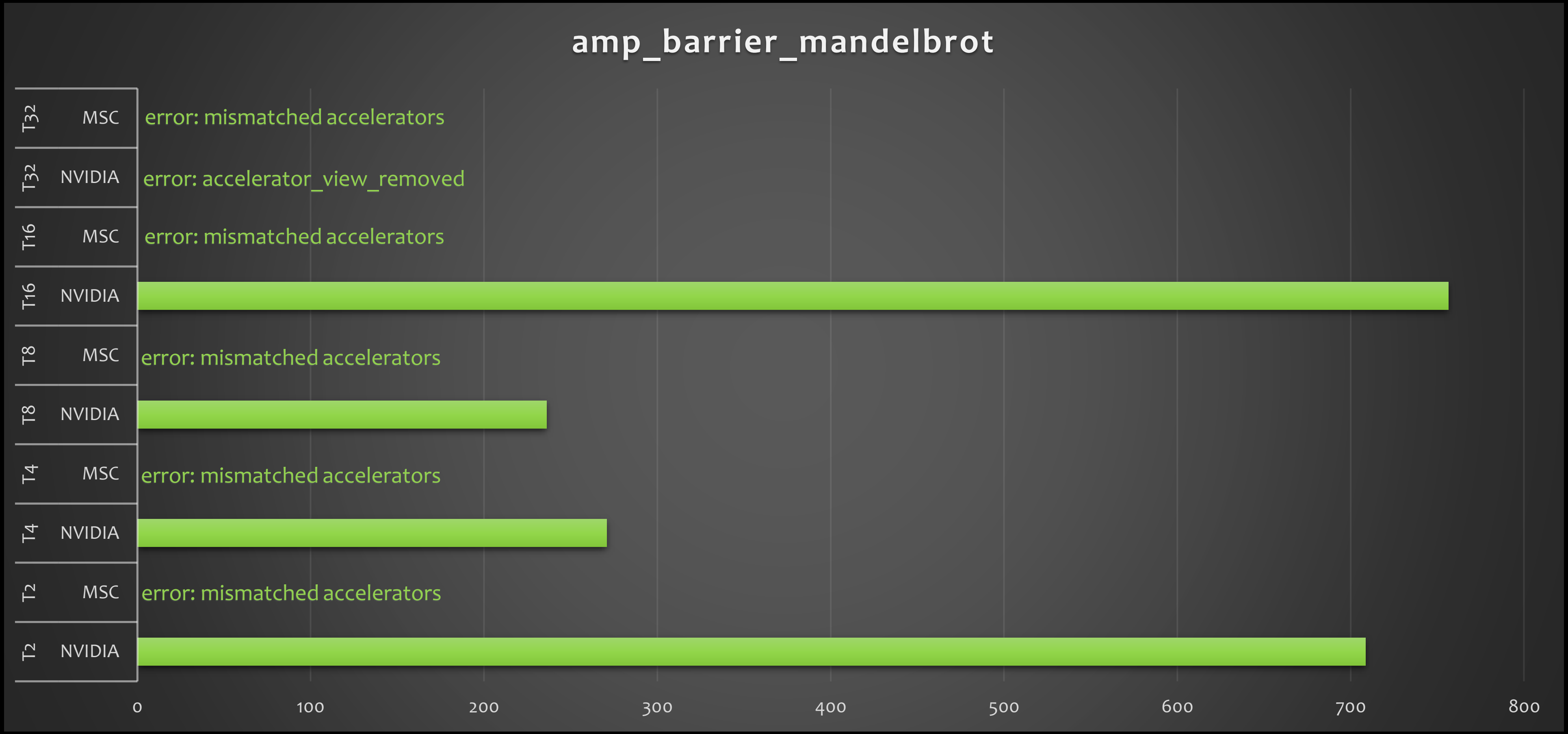
pixel_Mandelbrot video

amp_barrier_mandelbrot

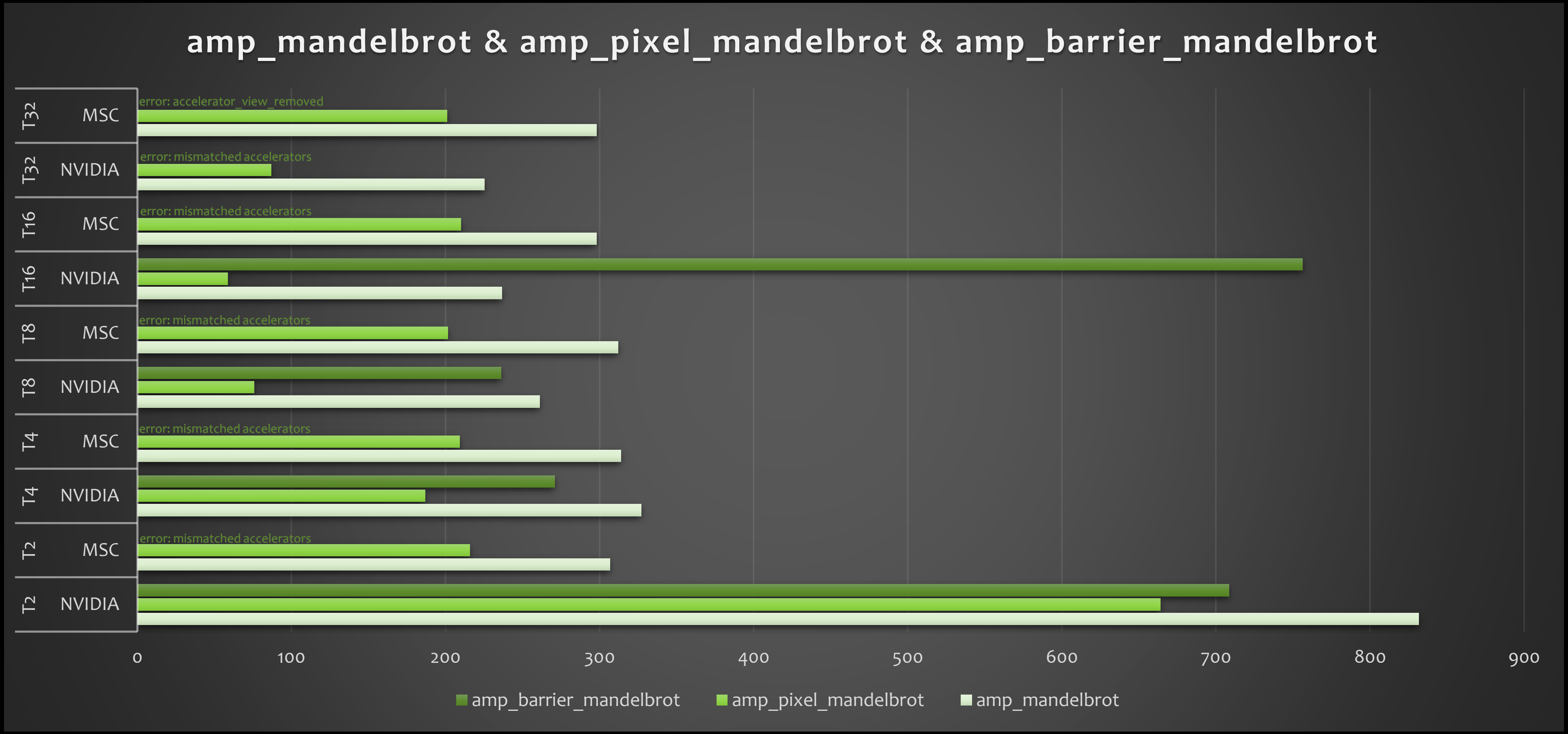


```
    // set colours
    if (iterations == max_iter)
    {
        // z didn't escape from the circle.
        // This point is in the Mandelbrot set.
        // r, g, b values are being modified outside the lambda
        // and passed directly to the image_array_view
    }
    else
    {
        // z escaped within less than MAX_ITERATIONS
        // iterations. This point isn't in the set.
        r = iterations * iterations * r;
        g = iterations * iterations * g;
        b = iterations * iterations * b;
    }
    //unsigned int atomic_fetch_or(r << 16);
    image_array_view[idx] = (r << 16) | (g << 8) | (b);
    // Copy the values of the tile into a tile-sized array.
    // create a TILE_SIZE x TILE_SIZE array to hold the values in this tile
    tile_static int tileValues[TILE_SIZE][TILE_SIZE];
    // copy the values for the tile into the TILE_SIZE x TILE_SIZE array
    tileValues[t_idx.local[1]][t_idx.local[0]] = image_array_view[t_idx];
    // when all the threads have exectuted and the TILE_SIZE x TILE_SIZE array is complete, calculate pixel array
    t_idx.barrier.wait_with_tile_static_memory_fence();

    int index = (idx[0] * HEIGHT + idx[1]) * 3;
    for (int row = 0; row < TILE_SIZE; row++) {
        for (int column = 0; column < TILE_SIZE; column++) {
            pixel_amp_barrier_mandelbrot_array[index] = tileValues[row][column];
            pixel_amp_barrier_mandelbrot_array[index + 1] = (tileValues[row][column] << 8);
            pixel_amp_barrier_mandelbrot_array[index + 2] = (tileValues[row][column] << 16);
        }
    }
});
// because we're using concurrency::array we must copy data back to the vector ourselves
pixel_amp_barrier_mandelbrot_ = pixel_amp_barrier_mandelbrot_array;
```

amp_barrier_mandelbrot



3 methods comparison



3 mandelbrots video

Possible improvements

- WarmUp timing
- Use high-resolution timer for C++
  https://blogs.msdn.microsoft.com/nativeconcurrency/2011/12/27/high-resolution-timer-for-c/
- Runtime error eliminations

Thanks

Matthew Wallace - 1502616@abertay.ac.uk
Data and Structures 2 – CMP202