HalfSize Code Test

The submitted is an OpenCL tool which loads a .tga image file and resizes it to half of its original width and height, and then saves it back to disk, as a new .tga file. A zipped Visual Studio project is delivered.

Solution Overview:

The supports 8 bit Monochrome, 24-bit uncompressed RGB and 32-bit uncompressed RGBA. The RLE support can be plugged in, where additional implementation involves reading the decoded input image data into the host buffer. Hence an extension can be provided with the current code.

The kernel extension to support the 16 bit RGB and MAP has be to be implemented as one of the if/else cases in the kernel code. The code has been designed to be modular so easy extension and plugin is possible.

The code is distributed in 3 sections with 5 files:

1. The Host execution- halfsize_tga.cpp, halfsize_tga.h
2. The TGA helper code- tga_helper.cpp, tga_helper.h
3. The kernel execution - halfsize_tga.cl

The TGA image is read, parsed and stored in the structure TGA. By default, all the elements are public such that the member elements can be accessed directly. This structure can be replaced by a class, so as to make all member variables private and provide a Get() and Set() function for each member variable respectively, to provide data hiding and protection.

```
// TGA Image as structure
struct TGA {

        // By default all kept public
        // Member Variables
        tgaHeader                       mHeader;        // Header of the image decoded
        uint32_t                mPixelDataLen;  // Pixel data length
        std::vector<uint8_t>    mID;                    // ID
        std::vector<uint8_t>    mColorMapSpec           // ColorMapSpec
        std::vector<uint8_t>    mPixelData;             // Pixel Data

        // Default Constructor
        TGA() {
                mHeader = {0};
                mPixelDataLen = 0;
        }

        // Parameterised Constructor
        TGA(const char *imgName) {
        read(imgName);
        }

        // Member functions
        void read(const char *imgName);
        bool write(const char *imgName);
};
```

Kernel Implementation:

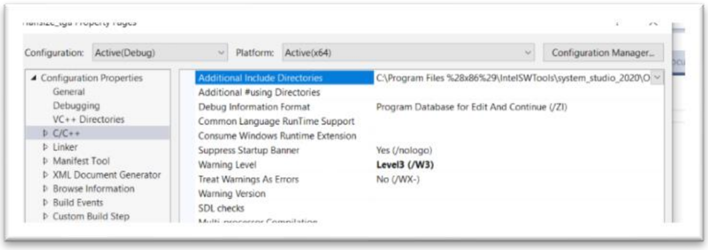
- The half sizing is done by clustering 4 pixels (as a box) and averaging the pixel values.
- The buffers and image data is always represented using 1 byte array.
- The number of workitems are always equal to the size of the output image, i.e (InImageWidth/2 * InImageHeight/2). For the 24-bit RGB and 32-bit RGBA a serial computation of all R, G, B and A values is done in one execution thread and core.

Delivery:

- A zipped Visual Studio solution directory is delivered, which can be unzipped, to review the code, and built in Debug and Release mode. The desired output can be seen on the console.

Settings related to the Code Run and Execution:

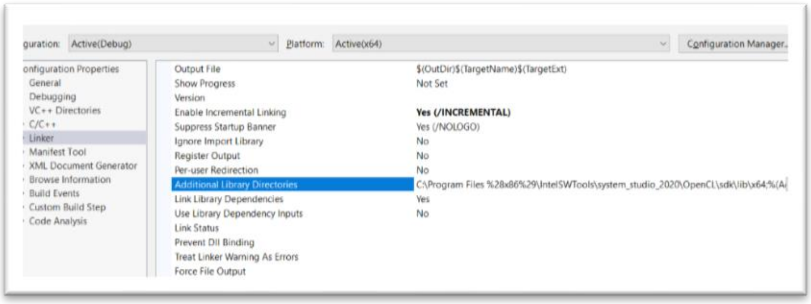
- Currently the Intel OpenCL sdk includes folder path is added to the 'Additional Include Directories' to provide for OpenCL related references and usage in the code.



Configuration: Active(Debug) Platform: Active(x64)

Additional Include Directories	C:\Program Files %28x86%29\IntelSWTools\system_studio_2020\O
Additional #using Directories	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
Warning Version	
SDL checks	

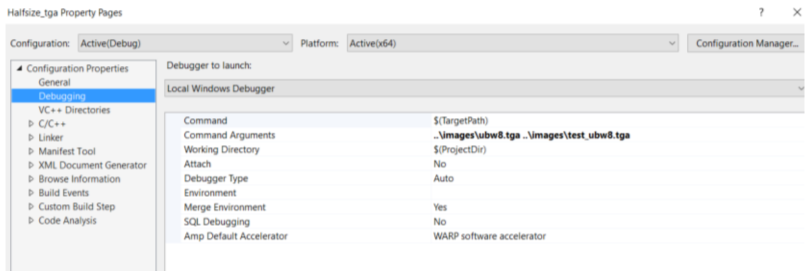
- The sdk library path is added to the Linker -> Additional Include Directories' to provide for OpenCL related linkages during compilation.



guration: Active(Debug) Platform: Active(x64)

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	C:\Program Files %28x86%29\IntelSWTools\system_studio_2020\OpenCL\sdk\lib\x64;%(A
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	

- OpenCL 1.2 is used for compiling the kernel code.
- Th execution can be performed in both Debug and Release Mode, by adding the inputs to the command line execution using the 'command arguments', as in the below picture:

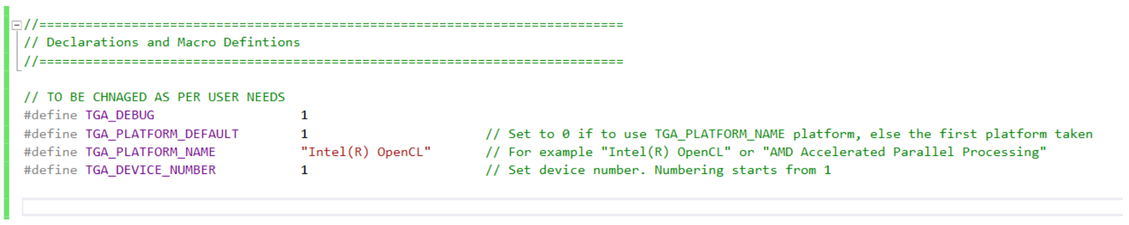


Configuration: Active(Debug) Platform: Active(x64)

Debugger to launch: Local Windows Debugger

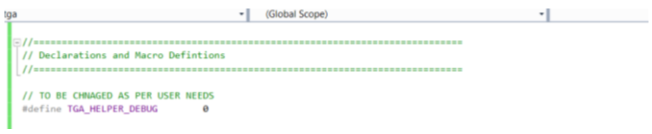
Command	$(TargetPath)
Command Arguments	**..\images\ubw8.tga ..\images\test_ubw8.tga**
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

- By default, the necessary/ desired output is available on the console window. The MACRO TGA_DEBUG can be enabled in the halfsize_tga.h, so more detailed debug info may be printed onto the console. (Refer pic below).
  TGA_PLATFORM_DEFAULT Should to set to 0, if to use TGA_PLATFORM_NAME platform, else the first platform with the device will be used for execution.
  Similarly, TGA_PLATFORM_NAME and TGA_DEVICE_NUMBER need to be modified as per user needs.
  In line 83 of halfsize_tga.cpp, the device is set to CL_DEVICE_TYPE_GPU ,which can be changed as per user needs.



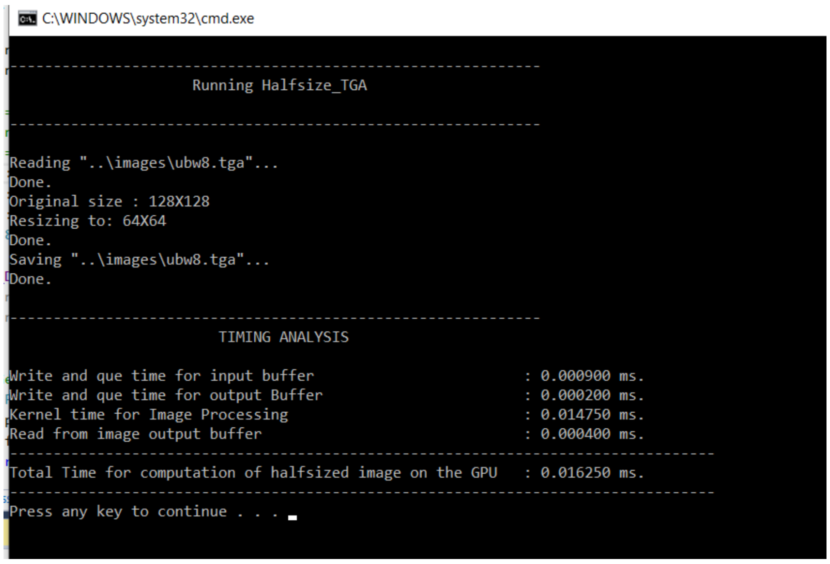
```
//===========================================================================
// Declarations and Macro Defintions
//===========================================================================

// TO BE CHNAGED AS PER USER NEEDS
#define TGA_DEBUG                    1
#define TGA_PLATFORM_DEFAULT         1                    // Set to 0 if to use TGA_PLATFORM_NAME platform, else the first platform taken
#define TGA_PLATFORM_NAME            "Intel(R) OpenCL"    // For example "Intel(R) OpenCL" or "AMD Accelerated Parallel Processing"
#define TGA_DEVICE_NUMBER            1                    // Set device number. Numbering starts from 1
```

- Similarly, the MACRO TGA_HELPER_DEBUG can be enabled in the tga_helper.h, if more detailed debug information is required to be printed onto the console, related to the packing and unpacking of the frames



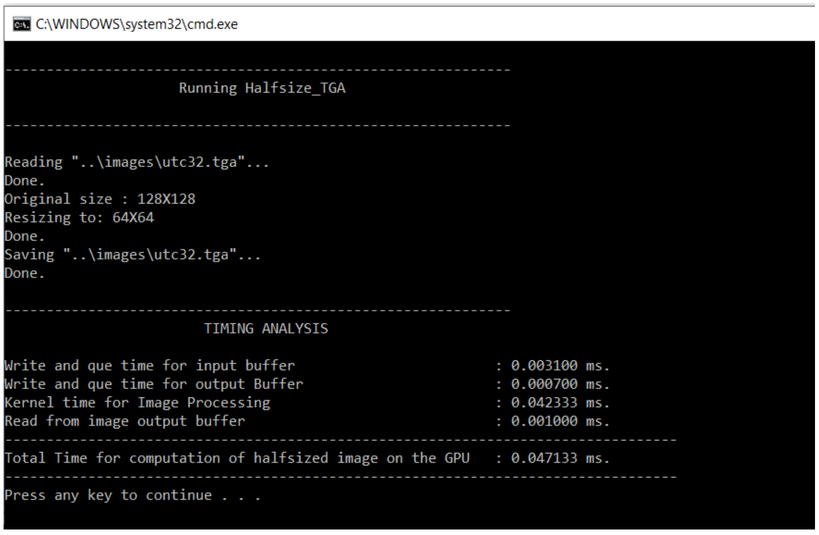
```
//===========================================================================
// Declarations and Macro Defintions
//===========================================================================

// TO BE CHNAGED AS PER USER NEEDS
#define TGA_HELPER_DEBUG        0
```

- The OUTPUT: The final output has the following details as below. The timing analysis has been performed to give the time executed in memory operations, GPU executions and total time



```
-------------------------------------------------------------
                     Running Halfsize_TGA

-------------------------------------------------------------

Reading "..\images\ubw8.tga"...
Done.
Original size : 128X128
Resizing to: 64X64
Done.
Saving "..\images\ubw8.tga"...
Done.

-------------------------------------------------------------
                        TIMING ANALYSIS

Write and que time for input buffer                     : 0.000900 ms.
Write and que time for output Buffer                    : 0.000200 ms.
Kernel time for Image Processing                        : 0.014750 ms.
Read from image output buffer                           : 0.000400 ms.
---------------------------------------------------------------------------------
Total Time for computation of halfsized image on the GPU   : 0.016250 ms.
---------------------------------------------------------------------------------
Press any key to continue . . .
```

For 8 bit Monochrome



```
-------------------------------------------------------------
                     Running Halfsize_TGA

-------------------------------------------------------------

Reading "..\images\utc32.tga"...
Done.
Original size : 128X128
Resizing to: 64X64
Done.
Saving "..\images\utc32.tga"...
Done.

-------------------------------------------------------------
                        TIMING ANALYSIS

Write and que time for input buffer                     : 0.003100 ms.
Write and que time for output Buffer                    : 0.000700 ms.
Kernel time for Image Processing                        : 0.042333 ms.
Read from image output buffer                           : 0.001000 ms.
---------------------------------------------------------------------------------
Total Time for computation of halfsized image on the GPU   : 0.047133 ms.
---------------------------------------------------------------------------------
Press any key to continue . . .
```

For 32 bit RGBA

- The tested images are not shared along with the project, due to copyright concerns
- The files are annotated with the necessary copyright information, such that it may be used and distributed freely as per the guidelines.