MANDELBROT CODE FOR STUDY

mandelbrot_ispc/main.cpp

```
#include
<stdio.h>
#include <algorithm>
#include <getopt.h>

#include "CycleTimer.h"
#include "mandelbrot_ispc.h"

extern void mandelbrotSerial(
    float x0, float y0, float x1, float y1,
    int width, int height,
    int startRow, int numRows,
    int maxIterations,
    int output[]);

extern void mandelbrotThread(
    int numThreads,
    float x0, float y0, float x1, float y1,
    int width, int height,
    int maxIterations,
    int output[]);

extern void writePPMImage(
    int* data,
    int width, int height,
    const char *filename,
    int maxIterations);

bool verifyResult (int *gold, int *result, int width, int height) {
    int i, j;

    for (i = 0; i < height; i++) {
        for (j = 0; j < width; j++) {
            if (gold[i * width + j] != result[i * width + j]) {
                printf ("Mismatch : [%d][%d], Expected : %d, Actual : %d\n",
                                i, j, gold[i * width + j], result[i * width + j]);
```

```
                    return 0;
                }
            }
        }

        return 1;
    }

    void
    scaleAndShift(float& x0, float& x1, float& y0, float& y1,
                  float scale,
                  float shiftX, float shiftY)
    {

        x0 *= scale;
        x1 *= scale;
        y0 *= scale;
        y1 *= scale;
        x0 += shiftX;
        x1 += shiftX;
        y0 += shiftY;
        y1 += shiftY;

    }

    using namespace ispc;

    void usage(const char* progname) {
        printf("Usage: %s [options]\n", progname);
        printf("Program Options:\n");
        printf("  -t  --tasks        Run ISPC code implementation with tasks\n");
        printf("  -v  --view <INT>   Use specified view settings\n");
        printf("  -?  --help         This message\n");
    }


    int main(int argc, char** argv) {

        const unsigned int width = 1200;
```

```
    const unsigned int height = 800;
    const int maxIterations = 256;

    float x0 = -2;
    float x1 = 1;
    float y0 = -1;
    float y1 = 1;

    bool useTasks = false;

    // parse commandline options ////////////////////////////////////////
    int opt;
    static struct option long_options[] = {
        {"tasks", 0, 0, 't'},
        {"view",  1, 0, 'v'},
        {"help",  0, 0, '?'},
        {0 ,0, 0, 0}
    };

    while ((opt = getopt_long(argc, argv, "tv:?", long_options, NULL)) != EOF) {

        switch (opt) {
        case 't':
            useTasks = true;
            break;
        case 'v':
        {
            int viewIndex = atoi(optarg);
            // change view settings
            if (viewIndex == 2) {
                float scaleValue = .015f;
                float shiftX = -.986f;
                float shiftY = .30f;
                scaleAndShift(x0, x1, y0, y1, scaleValue, shiftX, shiftY);
            } else if (viewIndex > 1) {
                fprintf(stderr, "Invalid view index\n");
                return 1;
            }
            break;
```

```
            }
            case '?':
            default:
                usage(argv[0]);
                return 1;
            }
        }
        // end parsing of commandline options

        int *output_serial = new int[width*height];
        int *output_ispc = new int[width*height];
        int *output_ispc_tasks = new int[width*height];

        for (unsigned int i = 0; i < width * height; ++i)
            output_serial[i] = 0;

        //
        // Run the serial implementation. Report the minimum time of three
        // runs for robust timing.
        //
        double minSerial = 1e30;
        for (int i = 0; i < 3; ++i) {
            double startTime = CycleTimer::currentSeconds();
            mandelbrotSerial(x0, y0, x1, y1, width, height, 0, height, maxIterations, ou
            double endTime = CycleTimer::currentSeconds();
            minSerial = std::min(minSerial, endTime - startTime);
        }

        printf("[mandelbrot serial]:\t\t[%.3f] ms\n", minSerial * 1000);
        writePPMImage(output_serial, width, height, "mandelbrot-serial.ppm", maxIteratio

        // Clear out the buffer
        for (unsigned int i = 0; i < width * height; ++i)
            output_ispc[i] = 0;

        //
        // Compute the image using the ispc implementation
        //
        double minISPC = 1e30;
```

```
    for (int i = 0; i < 3; ++i) {
        double startTime = CycleTimer::currentSeconds();
        mandelbrot_ispc(x0, y0, x1, y1, width, height, maxIterations, output_ispc);
        double endTime = CycleTimer::currentSeconds();
        minISPC = std::min(minISPC, endTime - startTime);
    }

    printf("[mandelbrot ispc]:\t\t[%.3f] ms\n", minISPC * 1000);
    writePPMImage(output_ispc, width, height, "mandelbrot-ispc.ppm", maxIterations);


    if (! verifyResult (output_serial, output_ispc, width, height)) {
        printf ("Error : ISPC output differs from sequential output\n");

        delete[] output_serial;
        delete[] output_ispc;
        delete[] output_ispc_tasks;

        return 1;
    }

    // Clear out the buffer
    for (unsigned int i = 0; i < width * height; ++i) {
        output_ispc_tasks[i] = 0;
    }

    double minTaskISPC = 1e30;
    if (useTasks) {
        //
        // Tasking version of the ISPC code
        //
        for (int i = 0; i < 3; ++i) {
            double startTime = CycleTimer::currentSeconds();
            mandelbrot_ispc_withtasks(x0, y0, x1, y1, width, height, maxIterations,
            double endTime = CycleTimer::currentSeconds();
            minTaskISPC = std::min(minTaskISPC, endTime - startTime);
        }

        printf("[mandelbrot multicore ispc]:\t[%.3f] ms\n", minTaskISPC * 1000);
```

```
        writePPMImage(output_ispc_tasks, width, height, "mandelbrot-task-ispc.ppm",

        if (! verifyResult (output_serial, output_ispc_tasks, width, height)) {
            printf ("Error : ISPC output differs from sequential output\n");
            return 1;
        }
    }

    printf("\t\t\t\t(%.2fx speedup from ISPC)\n", minSerial/minISPC);
    if (useTasks) {
        printf("\t\t\t\t(%.2fx speedup from task ISPC)\n", minSerial/minTaskISPC);
    }

    delete[] output_serial;
    delete[] output_ispc;
    delete[] output_ispc_tasks;


    return 0;
}
```

mandelbrot_ispc/mandelbrot.ispc

```
static inline int
mandel(float c_re,
float c_im, int
count) {
```

```
    float z_re = c_re, z_im = c_im;
    int i;
    for (i = 0; i < count; ++i) {

        if (z_re * z_re + z_im * z_im > 4.f)
           break;

        float new_re = z_re*z_re - z_im*z_im;
        float new_im = 2.f * z_re * z_im;
        z_re = c_re + new_re;
        z_im = c_im + new_im;
    }

    return i;
}

export void mandelbrot_ispc(uniform float x0, uniform float y0,
                            uniform float x1, uniform float y1,
                            uniform int width, uniform int height,
                            uniform int maxIterations,
                            uniform int output[])
{
    float dx = (x1 - x0) / width;
    float dy = (y1 - y0) / height;

    foreach (j = 0 ... height, i = 0 ... width) {
            float x = x0 + i * dx;
            float y = y0 + j * dy;

            int index = j * width + i;
            output[index] = mandel(x, y, maxIterations);
    }
}

// slightly different kernel to support tasking
task void mandelbrot_ispc_task(uniform float x0, uniform float y0,
                               uniform float x1, uniform float y1,
                               uniform int width, uniform int height,
                               uniform int rowsPerTask,
```

```
                                    uniform int maxIterations,
                                    uniform int output[])
{

    // taskIndex is an ISPC built-in

    uniform int ystart = taskIndex * rowsPerTask;
    uniform int yend = ystart + rowsPerTask;

    uniform float dx = (x1 - x0) / width;
    uniform float dy = (y1 - y0) / height;

    foreach (j = ystart ... yend, i = 0 ... width) {
            float x = x0 + i * dx;
            float y = y0 + j * dy;

            int index = j * width + i;
            output[index] = mandel(x, y, maxIterations);
    }
}

export void mandelbrot_ispc_withtasks(uniform float x0, uniform float y0,
                                      uniform float x1, uniform float y1,
                                      uniform int width, uniform int heigh
                                      uniform int maxIterations,
                                      uniform int output[])
{

    uniform int rowsPerTask = height / 2;

    // create 2 tasks
    launch[2] mandelbrot_ispc_task(x0, y0, x1, y1,
                                   width, height,
                                   rowsPerTask,
                                   maxIterations,
                                   output);
}
```

mandelbrot_ispc/mandelbrotSerial.cpp

```
This code was
modified from
example code
originally provided by Intel.  To comply with Intel's open source
licensing agreement, their copyright is retained below.

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

Copyright (c) 2010-2011, Intel Corporation
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

  * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

  * Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

  * Neither the name of Intel Corporation nor the names of its
    contributors may be used to endorse or promote products derived from
    this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
 OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
```

```
    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


static inline int mandel(float c_re, float c_im, int count)
{
    float z_re = c_re, z_im = c_im;
    int i;
    for (i = 0; i < count; ++i) {

        if (z_re * z_re + z_im * z_im > 4.f)
            break;

        float new_re = z_re*z_re - z_im*z_im;
        float new_im = 2.f * z_re * z_im;
        z_re = c_re + new_re;
        z_im = c_im + new_im;
    }

    return i;
}

//
// MandelbrotSerial --
//
// Compute an image visualizing the mandelbrot set.  The resulting
// array contains the number of iterations required before the complex
// number corresponding to a pixel could be rejected from the set.
//
// * x0, y0, x1, y1 describe the complex coordinates mapping
//   into the image viewport.
// * width, height describe the size of the output image
// * startRow, totalRows describe how much of the image to compute
void mandelbrotSerial(
    float x0, float y0, float x1, float y1,
    int width, int height,
    int startRow, int totalRows,
```

```
    int maxIterations,
    int output[])
{
    float dx = (x1 - x0) / width;
    float dy = (y1 - y0) / height;

    int endRow = startRow + totalRows;

    for (int j = startRow; j < endRow; j++) {
        for (int i = 0; i < width; ++i) {
            float x = x0 + i * dx;
            float y = y0 + j * dy;

            int index = (j * width + i);
            output[index] = mandel(x, y, maxIterations);
        }
    }
}
```