GPU Accelerated RANSAC documentation

First, we parse the input file given from the command line parameter. The provided test files did not include a header line for number of contour vertices, as such we hard coded the length in a macro since all given test files had the same length.

```
char const* const inputFile = argv[1];
FILE* file = fopen(inputFile, "r");
int vert_x[num_vert];
int vert_y[num_vert];

char line[256];
int i = 0;

while (fgets(line, sizeof(line), file) != NULL) {
    sscanf(line, "%d %d", &vert_x[i], &vert_y[i]);
    ++i;
}

fclose(file);
```

We initialize our host-side fields that are used for either inputting values into our GPU threads or to extract the results from them.

```
int ts = threads * sizeof(int);
int fs = threads * sizeof(float);
int ns = num_vert * sizeof(int);

int* h_centerx = (int*)malloc(ts);
int* h_centery = (int*)malloc(ts);
float* h_centerrad = (float*)malloc(fs);
int* h_eval = (int*)malloc(ts);
int* h_randinds = (int*)malloc(ts * 3);
```

The initial three values, **ts, fs, ns** are just size variables that are referenced throughout the field allocation and copy functions.

h_centerx and **h_centery** correspond to the x and y coordinates of every resulting circle center computed from every single GPU thread. Of all of these, we will only use filter out one, that we consider the end result of the execution.

h_centerrad corresponds to the circle radius results computed from every single GPU thread, similarly to the coordinates.

h_eval corresponds to the evaluation computed from every single GPU thread. To be more exact, when a GPU thread finds its corresponding circle center and radius, it then parses through the contour vertices to check which fit the line and which not, the resulting number of vertices being the integer evaluation score. We use this to determine which of the many circles is the actual best fitting circle.

h_randinds is a necessary vector to generate 3 random contour indices per executed thread, since we cannot access the rand() function from within a device code segment. These are possible contour vertex

indices, not thread or block indices, and correspond to the 3 random vertices from which we generate our given circle in the given thread.

We also allocate and initialize the variable used on the device-end.

```
int* d_centerx;
int* d_centery;
float* d_centerrad;
int* d_eval;
int* d_vert_x;
int* d_vert_y;
int* d_randinds;

cudaMalloc((void**)& d_centerx, ts);
cudaMalloc((void**)& d_centery, ts);
cudaMalloc((void**)& d_centerrad, fs);
cudaMalloc((void**)& d_eval, ts);
cudaMalloc((void**)& d_vert_x, ns);
cudaMalloc((void**)& d_vert_y, ns);
cudaMalloc((void**)& d_randinds, ts * 3);
```

The first four and the last simply correspond to the device-end copies of the aforementioned variables.

d_vert_x and **d_vert_y** are device-copies of the input contour vertex arrays, one for each coordinate.

Then we randomize the indices for each thread, making sure that the three indices themselves are not the same, otherwise we couldn't fit a circle on that to begin with.

```
for (int i = 0; i < threads; ++i) {
    int i1 = rand() % num_vert;
    int i2 = rand() % num_vert;
    int i3 = rand() % num_vert;
    while (i2 == i1) {
        i2 = rand() % num_vert;
    }
    while ((i3 == i1) || (i3 == i2)) {
        i3 = rand() % num_vert;
    }
    h_randinds[i * 3] = i1;
    h_randinds[i * 3 + 1] = i2;
    h_randinds[i * 3 + 2] = i3;
}
```

We pass on the input values to the device.

```
cudaMemcpy(d_vert_x, vert_x, ns, cudaMemcpyHostToDevice);
cudaMemcpy(d_vert_y, vert_y, ns, cudaMemcpyHostToDevice);
cudaMemcpy(d_randinds, h_randinds, (ts * 3), cudaMemcpyHostToDevice);
```

We execute RANSAC on the given amount of threads and blocks. My GPU supports 1024 threads per block so I used that setting.

```
ransac_circle << <threads / threads_per_block, threads_per_block >> > (d_centerx, d_centery, d_centerrad, d_eval, d_vert_x, d_vert_y, d_randinds, num_vert);
```

Each thread will do the following:

```
int index = threadIdx.x + blockIdx.x * blockDim.x;
int i1 = d_randinds[index];
int i2 = d_randinds[index + 1];
int i3 = d_randinds[index + 2];
int x1 = d_vert_x[i1];
int x2 = d_vert_x[i2];
int x3 = d_vert_x[i3];
int y1 = d_vert_y[i1];
int y2 = d_vert_y[i2];
int y3 = d_vert_y[i3];
```

First we fetch the three pairs of coordinates.

Then we calculate the determinant values with which we can get the circle properties. Think of the typical circle equation as a system of 3 equations with 3 values to find. Thus can be solved and the following is a simplification of the determinant system one has to solve for. For more information visit

http://mathforum.org/library/drmath/view/54323.html

```
float temp = x2 * x2 + y2 * y2;
float bc = (x1 * x1 + y1 * y1 - temp) / 2;
float cd = (temp - x3 * x3 - y3 * y3) / 2;
float det = (x1 - x2) * (y2 - y3) - (x2 - x3) * (y1 - y2);

float cx = (bc * (y2 - y3) - cd * (y1 - y2)) / det;
float cy = ((x1 - x2) * cd - (x2 - x3) * bc) / det;
float radius = sqrt((cx - x1) * (cx - x1) + (cy - y1) * (cy - y1));
```

Then we save these values and evaluate the circle fit.

```
d_centerx[index] = (int)cx;
d_centery[index] = (int)cy;
d_centerrad[index] = radius;

int count = 0;
for (int i = 0; i < d_n; i++) {
    int x_i = d_vert_x[i];
    int y_i = d_vert_y[i];
    if (((x_i - cx) * (x_i - cx) + (y_i - cy) * (y_i - cy)) - radius * radius == 0) count++;
}
d_eval[index] = count;
```

After that, in our host function, we first collect the values generated:

```
cudaMemcpy(h_centerx, d_centerx, ts, cudaMemcpyDeviceToHost);
cudaMemcpy(h_centery, d_centery, ts, cudaMemcpyDeviceToHost);
cudaMemcpy(h_centerrad, d_centerrad, fs, cudaMemcpyDeviceToHost);
cudaMemcpy(h_eval, d_eval, ts, cudaMemcpyDeviceToHost);
```

Then do one search for the best fitting circle:

```
int best_eval = h_eval[0];
int best_eval_index = 0;
for (int i = 1; i < threads; i++) {
    if (h_eval[i] > best_eval) {
        best_eval = h_eval[i];
        best_eval_index = i;
    }
}
```

And we release the GPU memory:

```
cudaFree(d_vert_x);
cudaFree(d_vert_y);
cudaFree(d_randinds);
cudaFree(d_centerx);
cudaFree(d_centery);
cudaFree(d_centerrad);
cudaFree(d_eval);
```

Test results:

Note that these results are uniform regardless of the randomized indices / threads. Image results soon.

For *points1.txt* the best fitting circle via RANSAC has the center **(491,1190)** with radius **400.134** which goes through 27 points.

For *points2.txt* the best fitting circle via RANSAC has the center **(1622,1294)** with radius **362.935** which goes through 23 points.

For *points3.txt* the best fitting circle via RANSAC has the center **(1161,930)** with radius **371.419** which goes through 22 points.

For *points4.txt* the best fitting circle via RANSAC has the center **(1723,936)** with radius **435.974** which goes through 16 points.