

P3 phase 3 Write-up

The example I used to test render go is the one to produce cube-rgb.nrrd. After running the command:

```
./render go -petc $CUBE -fov 14 -us 0.03 -s 0.03 -k ctmr -p rgbalit -b over -lut lut.nrrd -lit $SCIVIS/lit/rgb.txt -o cube-rgb.nrrd
```

using Time Profiler, I noticed in the call tree as shown below that the bottleneck was from the function **rndConvoEval**, which took 5.20 seconds out of the total render go running time 6.06 seconds:

Weight	Self Weight	Symbol Name
6.06 s	100.0%	0 s
6.06 s	100.0%	0 s
6.06 s	100.0%	0 s
6.06 s	100.0%	0 s
6.06 s	100.0%	0 s
2.00 ms	0.0%	0 s
1.00 ms	0.0%	0 s
6.06 s	99.9%	14.00 ms
1.00 ms	0.0%	0 s
3.00 ms	0.0%	0 s
69.00 ms	1.1%	69.00 ms
27.00 ms	0.4%	3.00 ms
5.94 s	98.0%	53.00 ms
38.00 ms	0.6%	38.00 ms
1.00 ms	0.0%	0 s
3.00 ms	0.0%	0 s
5.85 s	96.5%	429.00 ms
63.00 ms	1.0%	63.00 ms
33.00 ms	0.5%	0 s
1.00 ms	0.0%	1.00 ms
1.00 ms	0.0%	1.00 ms
12.00 ms	0.1%	7.00 ms
4.00 ms	0.0%	4.00 ms
106.00 ms	1.7%	104.00 ms
5.20 s	85.7%	4.23 s
493.00 ms	8.1%	466.00 ms
27.00 ms	0.4%	27.00 ms
11.00 ms	0.1%	11.00 ms
462.00 ms	7.6%	454.00 ms

the call tree before code change

Then I checked the **rndConvoEval** function in convo.c, and found possible code fragments for improvement:

1) three for loops for checking the `cnv->inside` is true or false which should be combined into one:

```
for(int i = ctx->lower; i <= ctx->upper; i++) {
    if(i+n3 < 0 || i+n3 > (signed)(ctx->vol->size[2])-1
        || i+n2 < 0 || i+n2 > (signed)(ctx->vol->size[1])-1
        || i+n1 < 0 || i+n1 > (signed)(ctx->vol->size[0])-1) {
        cnv->inside = 0;
    }
}
```

2) two 3-layer embedded loops dealing with convolution and gradient respectively which should be combined into one 3-layer embedded loops:

```
for(int i3 = ctx->lower, j3=0; i3<=ctx->upper; i3++, j3++)
for(int i2 = ctx->lower, j2=0; i2<=ctx->upper; i2++, j2++)
for(int i1 = ctx->lower, j1=0; i1<=ctx->upper; i1++, j1++){

    int index_1D = (n3+i3)*(ctx->vol->size[1]*ctx->vol->size[0])+
        (n2+i2)*(ctx->vol->size[0])+n1+i1;
    if(ctx->vol->dtype == rndTypeFloat)
        cov_sum += ctx->vol->data.fl[index_1D]*cnv->kvalues1[j1]*cnv->kvalues2[j2]*cnv->kvalues3
    else if(ctx->vol->dtype == rndTypeShort)
        cov_sum += ctx->vol->data.ss[index_1D]*cnv->kvalues1[j1]*cnv->kvalues2[j2]*cnv->kvalues3
    if (cnv->grad_need){
        if(ctx->vol->dtype == rndTypeFloat){
            cov_sum_deriv1 += ctx->vol->data.fl[index_1D]*cnv->the_kvalues1[j1]*cnv->kvalues2[j2]*cn
            cov_sum_deriv2 += ctx->vol->data.fl[index_1D]*cnv->kvalues1[j1]*cnv->the_kvalues2[j2]*cn
            cov_sum_deriv3 += ctx->vol->data.fl[index_1D]*cnv->kvalues1[j1]*cnv->kvalues2[j2]*cnv->t
        } else if(ctx->vol->dtype == rndTypeShort){
            cov_sum_deriv1 += ctx->vol->data.ss[index_1D]*cnv->the_kvalues1[j1]*cnv->kvalues2[j2]*cn
            cov_sum_deriv2 += ctx->vol->data.ss[index_1D]*cnv->kvalues1[j1]*cnv->the_kvalues2[j2]*cn
            cov_sum_deriv3 += ctx->vol->data.ss[index_1D]*cnv->kvalues1[j1]*cnv->kvalues2[j2]*cnv->t
        }
    }
}
```

then running the same command use Time Profiler, and the function **rndConvoEval** took 4.58 seconds out of the total rendr go running time 5.43 seconds:

Time Profiler | Profile | Root | rendr (90633) | Main Thread 0x100f72c

☒ Ignore Case ☐ Auto Expand

Weight Self Weight Symbol Name

5.43 s	100.0%	0 s	▼ Main Thread 0x100f72c
5.43 s	100.0%	0 s	▼ start libdyld.dylib
5.43 s	100.0%	0 s	▼ main rendr
5.43 s	100.0%	0 s	▼ unrduCmdMain rendr
5.43 s	100.0%	0 s	▼ rnd_goMain rendr
7.00 ms	0.1%	0 s	► <Unknown Address>
2.00 ms	0.0%	0 s	► rndImageSave rendr
1.00 ms	0.0%	1.00 ms	rndRayStep This address is not in a known library range and cannot be symbolicated.
5.42 s	99.8%	17.00 ms	▼ rndRender rendr
2.00 ms	0.0%	0 s	► <Unknown Address>
4.00 ms	0.0%	0 s	► fflush libsystem_c.dylib
76.00 ms	1.4%	76.00 ms	rndRayBlend rendr
19.00 ms	0.3%	3.00 ms	► rndRayStart rendr
5.30 s	97.6%	36.00 ms	▼ rndRayStep rendr
44.00 ms	0.8%	44.00 ms	0x7fff65be5d20 libsystem_m.dylib
3.00 ms	0.0%	0 s	► <Unknown Address>
4.00 ms	0.0%	0 s	► rndRayFinish rendr
5.21 s	96.0%	417.00 ms	▼ rndRaySample rendr
65.00 ms	1.1%	65.00 ms	0x7fff65be5d20 libsystem_m.dylib
39.00 ms	0.7%	0 s	► <Unknown Address>
8.00 ms	0.1%	4.00 ms	► __tg_pow(float, float) rendr
12.00 ms	0.2%	12.00 ms	► __tg_sqrt(float) rendr
88.00 ms	1.6%	83.00 ms	► rndBlinnPhong rendr
4.58 s	84.4%	3.66 s	▼ rndConvoEval rendr
535.00 ms	9.8%	506.00 ms	▼ CtmrEval rendr
29.00 ms	0.5%	29.00 ms	► __tg_fabs(float) rendr
12.00 ms	0.2%	12.00 ms	► __tg_floor(float) rendr
379.00 ms	6.9%	374.00 ms	► dCtmrEval rendr

the call tree after code change

Comparing two total running time before and after code change, the improvement $6.06 - 5.43 = 0.63(s)$, which approximately equals to the running time improvement from the function **rndConvoEval** $5.20 - 4.58 = 0.62(s)$, which indicates that the code change on the function **rndConvoEval** effectively eliminate the bottleneck.