2025-03-07-vglare

지난 주 진행 사항

- mexican hat 모양을 가진 polynomial sample을 쓰기 위해 $1-(x^2+y^2)$꼴을 선택. 2d fourier series로 나타낼 때 다음과 같이 표현

$$a_n = \int_{-1}^{1}\int_{0}^{\sqrt{1-x^2}} (1-(x^2+y^2))\cos(n\omega x)\cos(n\omega y)dydx$$

- 이는 반구 상에서의 근사 뿐 아니라 극좌표계로 쓰면 좀 더 간단하게 전개할 수 있을 것이라 생각함. r,θ를 이용한 극좌표계로 나타내었을 때, $cos(n\omega r\cos\theta)$ 꼴을 r을 이용한 bessel function J_0로 나타낼 수 있기 때문에 bessel function J_0를 basis로 갖는 fourier-bessel series로 표현
- fourier-bessel series는 polynomial formula에 대해서 원형으로 구간을 잡아 glare의 바닥이 원형으로 퍼지는 구간을 표현하기에 적합했지만 $x+dx$ 로 중심이 이동하는 과정에서 $\cos(x-dx)$ 처럼 $J(x)J(dx)$ 꼴의 분해가 되지 때문에 논문의 취지에 맞출 수 없었다. 따라서 우선 다른 방법을 찾기로 결정했다.
- (-T,T) x (-T',T') 구간의 사각형 cartesian coordinate에서 자연스럽게 mexican hat 모양을 구현하기 위해 gaussian의 analytic integral에서 문제가 되었던 erf함수를 근사하는 모델을 찾아 2d gaussian을 적용함

진행사항

- 2d gaussian의 erf approximation을 통한 analytic integral form을 이용해서 2d fourier series 구현
- bell curve 형태인 gaussian function은 근사 없이 정적분 형태가 잘 나오지 않아 bell curve형태와 비슷한 $cos^2(x)cos^2(y)$ 를 선정하여 fourier series 2D를 rex로 구현
- bessel function으로 translate 했을 때 분해 가능성 확인

2d gaussian의 analytic integral

$$z = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x^2+y^2)/2\sigma^2}$$

- 지수 분리를 통해 $e^{-(x^2+y^2)}$항을 $e^{-x^2}\cdot e^{-y^2}$꼴로 분해할 수 있고 1d fourier series의 곱으로 나타낼 수 있다. 우선 1d gaussian의 fourier series는 아래와 같이 전개할 수 있다.

$$f(x) \approx \frac{a_0}{2} + \sum_{n=1}^{N} a_n\cos(n\omega x), \omega = \frac{2\pi}{T}$$

- coefficient는 다음과 같이 나타낸다.

$$\begin{aligned} a_n &= \int_0^{T/2} e^{-x^2/2\sigma^2} \cos(n\omega x) \\ &= \frac{\sqrt{2\pi}\sigma}{2} e^{-(n\omega\sigma)^2/2} R\left(erf\left(\frac{T}{2\sqrt{2}\sigma}\right)\right) \end{aligned}$$

- erf는 $\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$ 꼴로 closed-form solution이 없고 polynomial series로 근사할 수 있다. 상수 값 $a_1, \dots, a_5, p$에 대해서 다음과 같이 근사할 수 있다.

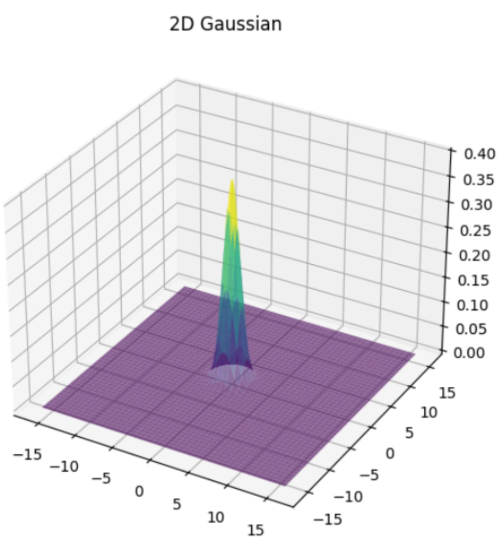
$$erf(x) \approx 1 - \left(a_1 \frac{1}{1+px} + a_2 \frac{1}{(1+px)^2} \cdots + a_5 \frac{1}{(1+px)^5}\right) e^{-x^2} + \epsilon(x), |\epsilon(x)| \leq 5 \times 10^{-7}$$

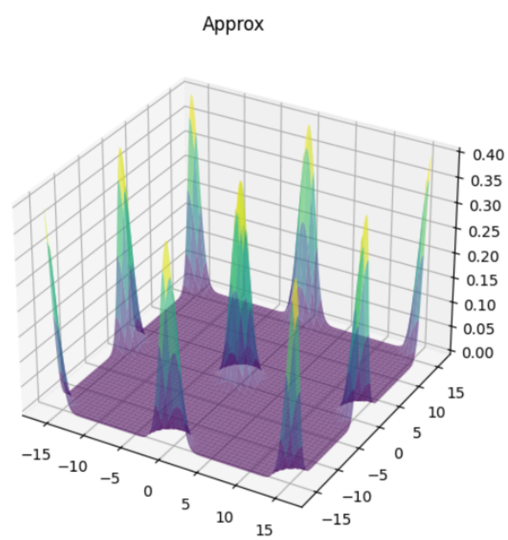
  erf 근사 참고자료 : p299(p88), 7.1.26

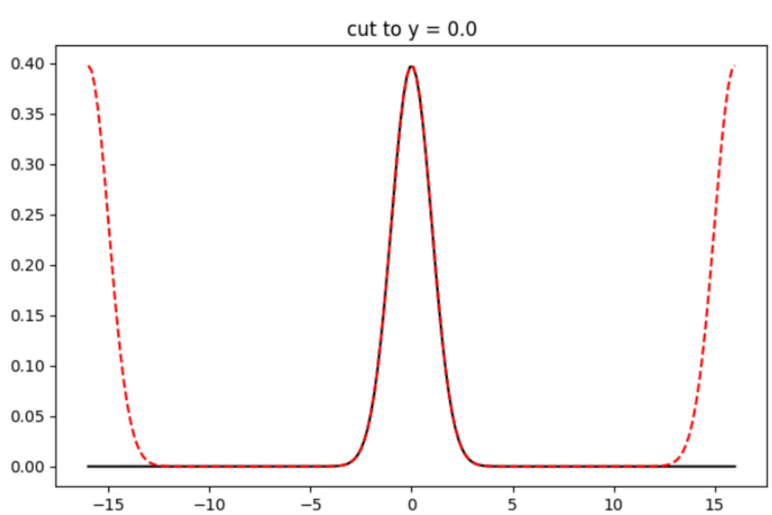
- 2d gaussian은 두 1d fourier series의 곱으로 표현할 수 있다.

$$f(x, y, \sigma) \approx \left[\frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(n\omega x)\right] \left[\frac{b_0}{2} + \sum_{m=1}^{M} b_n \cos(m\omega y)\right]$$

- 결과 사진(T = 16으로 ± 8이후로 반복된다. center = (0.0, 0.0), N = 20, sigma = 1.0)







2d gaussian general form

- 주기가 T_1, T_2를 갖고 중심이 (dx, dy) 만큼 이동한 함수 $f(x, y)$는 다음과 같이 표현할 수 있다.

$$f(x,y) = \frac{a_0}{4} + \sum_{n,m} a_{n,m} \cos(n\omega_1(x-dx)) \cos(m\omega_2(y-dy))$$

- 이 역시 두 1d fourier series의 곱으로 나타낼 수 있다.

$$\begin{aligned} f(x,y,\sigma) &\approx \left[\frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(n\omega(x-dx))\right]\left[\frac{b_0}{2} + \sum_{m=1}^{M} b_n \cos(m\omega(y-dy))\right] \\ &= \left[\frac{a_0}{2} + \sum_{n=1}^{N} a_n (c_n(x)c_n(dx) + s_n(x)s_n(dx))\right]\left[\frac{b_0}{2} + \sum_{m=1}^{M} b_n (c_m(y)c_m(dy) + s_m(y)s_m(dy))\right] \end{aligned}$$

- 전개해서 정리하면

$$\begin{aligned} f(x,y,\sigma) \approx \frac{\kappa}{T^2} \sum_{n=0}^{N} \sum_{m=0}^{M} a_n b_m [&c_n(x)c_n(dx)c_m(y)c_m(dy) \\ &+ c_n(x)c_n(dx)s_m(y)s_m(dy) \\ &+ s_n(x)s_n(dx)c_m(y)c_m(dy) \\ &+ s_n(x)s_n(dx)s_m(y)s_m(dy)] \\ \text{where} \\ &\kappa = 1 \text{ if n} = 0 \text{ and m} = 0 \\ &\kappa = 2 \text{ if n} = 0 \text{ or m} = 0 \\ &\kappa = 4 \text{ if n} > 0 \text{ and m} > 0 \end{aligned}$$

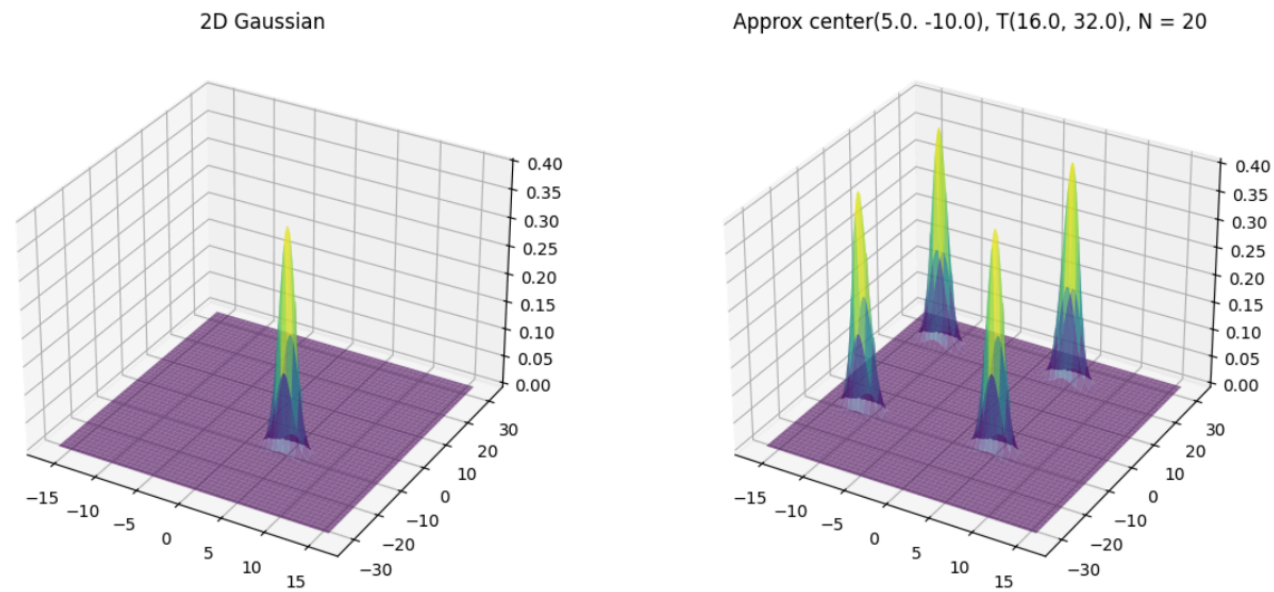
- K개의 ghost 상에서 각 center를 x_k, y_k라고 했을 때

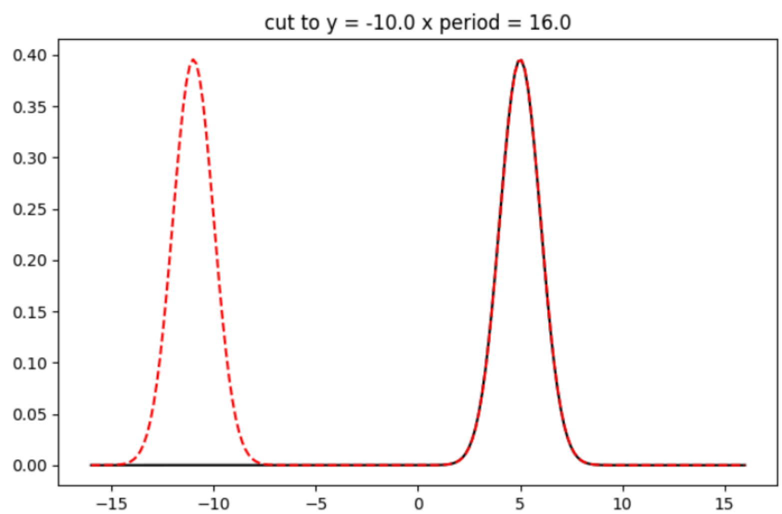
$$\begin{aligned} G_{n,m}(x,y) &= A_{n,m} c_n(x) c_m(y) + B_{n,m} c_n(x) s_m(y) \\ &+ C_{n,m} s_n(x) c_m(y) + D_{n,m} s_n(x) s_m(y) \\ \text{where} \\ A_{n,m} &= \sum^{K} a_{n,k} b_{m,k} (c_n(x_k) c_m(y_k)) \\ B_{n,m} &= \sum^{K} a_{n,k} b_{m,k} (c_n(x_k) s_m(y_k)) \\ C_{n,m} &= \sum^{K} a_{n,k} b_{m,k} (s_n(x_k) c_m(y_k)) \\ D_{n,m} &= \sum^{K} a_{n,k} b_{m,k} (s_n(x_k) s_m(y_k)) \end{aligned}$$

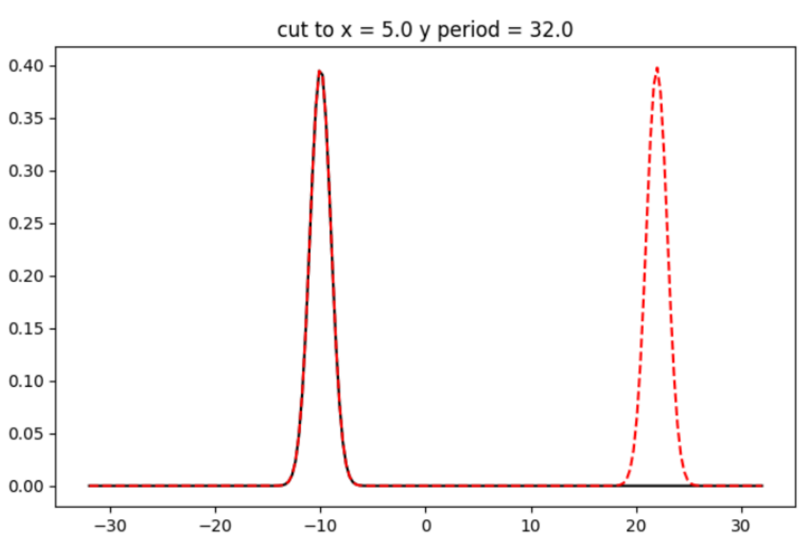
- $a_{n,k}, b_{m,k}$는 1d fourier series에 대한 계수로 간주할 수 있기 때문에 각각 다음과 같이 표현된다.

$$a_{n,k} = \frac{2}{T_1}\int_0^{T_1/2} g_k(x, y_k)\cos(n\omega x)dx$$
$$b_{n,k} = \frac{2}{T_2}\int_0^{T_2/2} g_k(x_k, y)\cos(n\omega y)dx$$

- 결과 (T1 = 16.0, T2 = 32.0, sigma = 1.0, N = 20, center = (5, -10))







$cos^2(x)cos^2(y)$

함수 정의

$$f(x,y) = \begin{cases} \cos^2(x)cos^2(y), & -\frac{\pi}{2} \le x \le \frac{\pi}{2}, -\frac{\pi}{2} \le y \le \frac{\pi}{2} \\ 0, & \text{otherwise} \end{cases}$$

Fourier Series 전개

$$\begin{aligned} G_{n,m}(x,y) = A_{n,m}c_n(x)c_m(y) + B_{n,m}c_n(x)s_m(y) \\ +C_{n,m}s_n(x)c_m(y) + D_{n,m}s_n(x)s_m(y) \end{aligned}$$

$$\begin{aligned} A_{n,m} &= \sum_{k=0}^{K} a_{k,n,m}c_n(x_k)c_m(y_k) + b_{k,n,m}s_n(x_k)s_m(y_k), \\ B_{n,m} &= \sum_{k=0}^{K} a_{k,n,m}c_n(x_k)s_m(y_k) - b_{k,n,m}s_n(x_k)c_m(y_k), \\ C_{n,m} &= \sum_{k=0}^{K} a_{k,n,m}s_n(x_k)c_m(y_k) - b_{k,n,m}c_n(x_k)s_m(y_k), \\ D_{n,m} &= \sum_{k=0}^{K} a_{k,n,m}s_n(x_k)s_m(y_k) + b_{k,n,m}c_n(x_k)c_m(y_k), \end{aligned}$$

$$\begin{aligned} a_{k,n,m} &= \frac{\kappa}{\lambda^2}\int_{-\lambda/2}^{\lambda/2}\int_{-\lambda/2}^{\lambda/2} f(x,y)c_n(x)c_m(y)dxdy \\ &= \frac{\kappa}{\lambda^2}\int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} cos^2(x)cos^2(y)c_n(x)c_m(y)dxdy \\ &= \frac{\kappa}{\lambda^2}\int_{-\pi/2}^{\pi/2} cos^2(x)c_n(x)dx \int_{-\pi/2}^{\pi/2} cos^2(y)c_m(y)dy \end{aligned}$$

$$\int_{-\pi/2}^{\pi/2} cos^2(x)c_n(x)dx = \begin{cases} \frac{\pi}{2} & n=0, \\ \frac{4sin(\frac{\pi n\omega}{2})}{4n\omega - n^3\omega^3} & n>0 \end{cases}$$

$$\int_{-\pi/2}^{\pi/2} cos^2(y)c_m(y)dy = \begin{cases} \frac{\pi}{2} & m=0, \\ \frac{4sin(\frac{\pi m\omega}{2})}{4m\omega - m^3\omega^3} & m>0 \end{cases}$$

$$\begin{aligned} b_{k,n,m} &= \frac{\kappa}{\lambda^2}\int_{-\lambda/2}^{\lambda/2}\int_{-\lambda/2}^{\lambda/2} f(x,y)s_n(x)s_m(y)dxdy \\ &= \frac{\kappa}{\lambda^2}\int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} cos^2(x)cos^2(y)s_n(x)s_m(y)dxdy \\ &= \frac{\kappa}{\lambda^2}\int_{-\pi/2}^{\pi/2} cos^2(x)s_n(x)dx \int_{-\pi/2}^{\pi/2} cos^2(y)s_m(y)dy \\ &= 0 \end{aligned}$$

$cos(x)^2 cos(y)^2$ 에 대한 $a_{k,m,n}$ 는 위의 식을 그대로 계산하여 적분식을 얻었으며, rex에서의 c++ 함수로 다음과 같이 표현한다.

$$a_{k,m,n} = 16.0 \frac{\kappa}{\lambda^2} \frac{sin(\pi n\omega/2.0)}{4.0n\omega - n^3\omega^3} \frac{sin(\pi m\omega/2.0)}{4.0m\omega - m^3\omega^3} (n! = 0, m! = 0)$$

```
    std::function<double(float, int, int)> temp_func_a = [](float w, int n, int m) {
        double coef_n = PI<double> / 2.0;
        if(n != 0)
        {
            coef_n = 4.0 * sin(PI<double>*n*w / 2.0) / (4.0*n*w - n*n*n*w*w*w);
        }
        double coef_m = PI<double> / 2.0;
        if(m != 0)
        {
            coef_m = 4.0 * sin(PI<double>*m*w / 2.0) / (4.0*m*w - m*m*m*w*w*w);
        }
        return coef_n * coef_m;
    };
```

각각의 ghost k에 대한 $a_{k,m,n}$ 를 기반으로 initial*update()에서 coefficients $A*{m,n},B*{m,n}*,C{m,n}, D_{m,n}$
를 계산하여 vector 배열로 저장한다.
각각의 ghost의 mean은 pair의 vector 구조로 저장한다.

```
    vector<double> A = vector<double>(N * M, 0);
    vector<double> B = vector<double>(N * M, 0);
    vector<double> C = vector<double>(N * M, 0);
    vector<double> D = vector<double>(N * M, 0);

    vector<std::pair<float, float>> means = vector<std::pair<float, float>>(K);
```

```
    float n_ka = 1.0f;
    float m_ka = 1.0f;
    for(int m = 0; m < M; m++)
    {
```

```
        if(n_ka != 0) n_ka = 2.0f;
        for(int n = 0; n < N; n++)
        {
            if(m_ka != 0) m_ka = 2.0f;
            for(int k = 0; k < K; k++)
            {
                float ka = n_ka * m_ka;
                double coef_a = temp_func_a(omega, n, m) * ka / (T * T);
                double coef_b = 0.0;

                A[m * N + n] += coef_a * cos(n * omega * means[k].first) *
cos(m * omega * means[k].second) + coef_b * sin(n * omega * means[k].first) *
sin(m * omega * means[k].second);
                B[m * N + n] += coef_a * cos(n * omega * means[k].first) *
sin(m * omega * means[k].second) - coef_b * sin(n * omega * means[k].first) *
cos(m * omega * means[k].second);
                C[m * N + n] += coef_a * sin(n * omega * means[k].first) *
cos(m * omega * means[k].second) - coef_b * cos(n * omega * means[k].first) *
sin(m * omega * means[k].second);
                D[m * N + n] += coef_a * sin(n * omega * means[k].first) *
sin(m * omega * means[k].second) + coef_b * cos(n * omega * means[k].first) *
cos(m * omega * means[k].second);
            }
        }
    }
```

모든 k를 루프를 돌며, 각각의 m,n index에 대하여 coefficients를 계산하여 저장했다.

initial_update()에서 한번 계산한 coefficients들은 render frame마다 각 픽셀 별로 m x n번 계산하여 화면에 출력했다. x,y [-3.0,3.0] 범위를 스크린으로 렌더링했으며, 함수 결과값을 rgb값의 세기로 표현했다.

```
vector<vertex> UserFunc::export_result()
{
    vector<vertex> vertices;

    float omega = 2.0f * PI<float> / T;
    for(float x = -(T / 2.0f); x < (T / 2.0f); x += 0.01f)
    {
        for(float y = -(T / 2.0f); y < (T / 2.0f); y += 0.01f)
```

```
        {
            double sum = 0.0;
            for(int m = 0; m < M; m++)
            {
                for(int n = 0; n < N; n++)
                {
                    int idx = m * N + n;
                    sum += A[idx] * cos(n * omega * x) * cos(m * omega * y);
                    sum += B[idx] * cos(n * omega * x) * sin(m * omega * y);
                    sum += C[idx] * sin(n * omega * x) * cos(m * omega * y);
                    sum += D[idx] * sin(n * omega * x) * sin(m * omega * y);
                }
            }
            vertices.push_back({vec3(x / (T / 2.0f), y / (T / 2.0f), 0.0f),
vec3(sum, 0.0f, 0.0f), vec2(0)});
        }
    }

    return vertices;
}
```

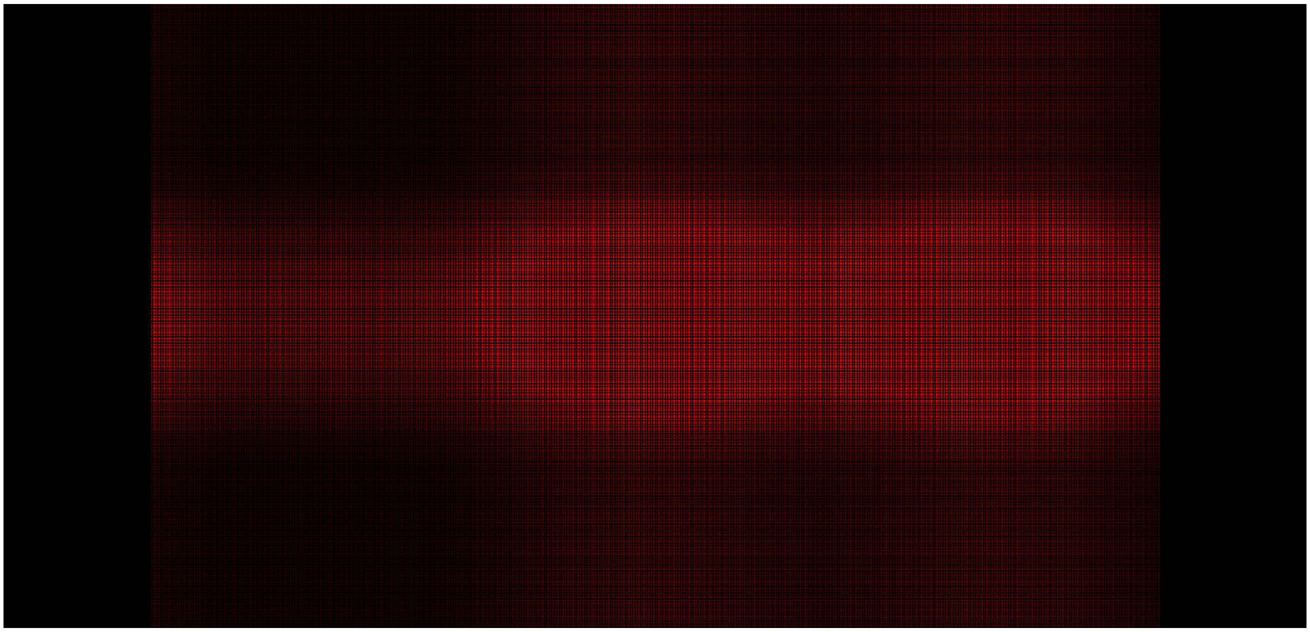
쉐이더 코드는 다음과 같다.

```
interface PSIN { vec2 tex; vec3 norm; };
uniform sampler2D SRC;

//*********************************************************************
shader vsQuad( in vec3 position:0, in vec3 normal:1, in vec2 texcoord:2, out
PSIN vout )
{
    gl_Position = vec4( position, 1 );
    vout.tex = texcoord;
    vout.norm = normal;
}

shader psDrawPoints(in PSIN pin, out vec4 pout)
{
    pout = vec4(2.0 * pin.norm.xyz, pin.norm.x);
}
```

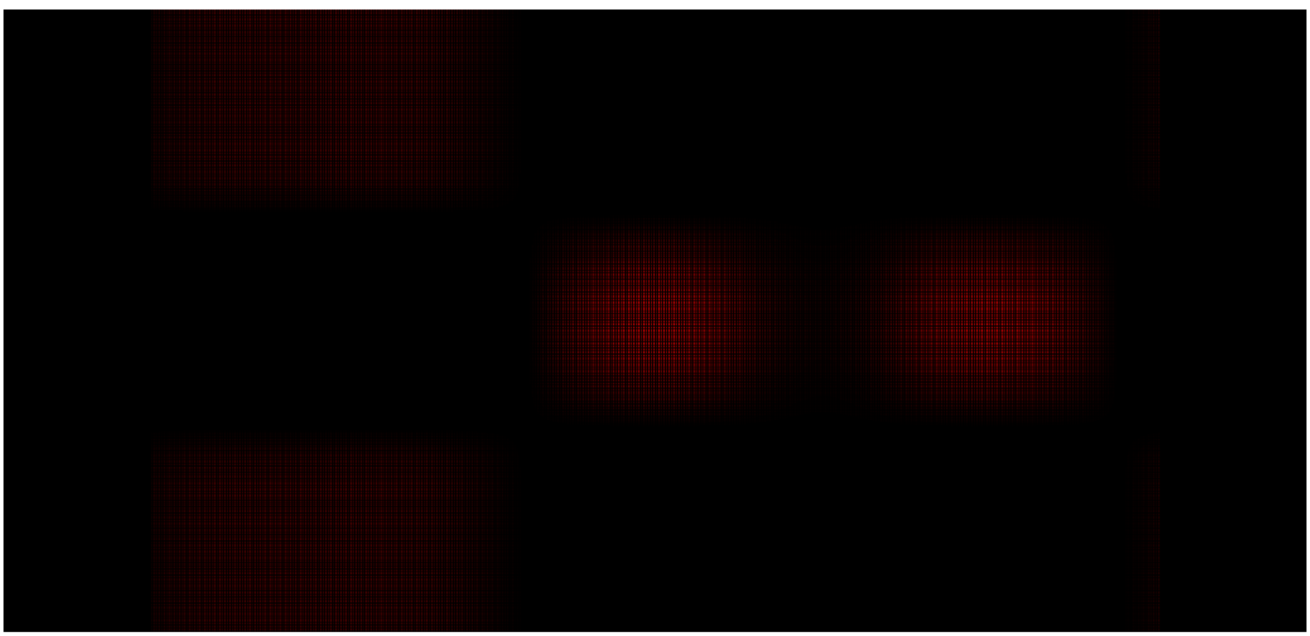
```
program draw_fs { vs(440)=vsQuad(); fs(440)=psDrawPoints(); };
```

두 개의 ghost를 rendering했다. 0차항(dc성분)을 포함했을 때 값이 퍼지는 현상이 심하여 0차항을 제거하고도 측정했다.

1. 0차항 포함



2. 0차항 제외



결과적으로 여러 개의 ghost를 k에 independent하게 fourier series를 이용해 n x m으로 계산할 수 있다.

fourier-bessel series separability

문제점

- fourier-bessel series 식은 x, y가 아닌 r을 사용하기 때문에 만약 중심이 이동한다면 r에서 $r - dr$을 대입해야한다. fourier series의 핵심은 삼각함수 덧셈정리를 통해 dx, dy 항을 분리하여 모든 glare에 대해 공통항을 모을 수 있다는 것이었기 때문에 bessel function을 적용하려면 이 부분을 해결해야 했음.

transaltion bessel function

- Graf's addition formula

$$J_v(\sqrt{x^2+y^2-2xy\cos\psi})\left(\frac{x-ye^{-i\psi}}{x-ye^{i\psi}}\right)^{v/2} = \sum_{m=-\infty}^{\infty} J_{v+m}(x)J_m(y)e^{im\psi}$$

$J_v(|\vec{r}-\vec{dr}|)$에서 v = 0이고 실수 범위에서 우함수이기 때문에 다음과 같이 간소화할 수 있다.

$$J_0(||\vec{r}-\vec{dr}||) = J_0(|\vec{r}|)J_0(|\vec{dr}|) + 2\sum_{m=1}^{M} J_m(|\vec{r}|)J_m(|\vec{dr}|)\cos(m\theta)$$

따라서 $A - B(\vec{r}-\vec{dr})$꼴의 glare model은 다음과 같이 fourier-bessel series로 나타낼 수 있다. fourier-bessel coefficient 구할 때 r은 glare 중심으로 부터 거리 r을 의미한다.

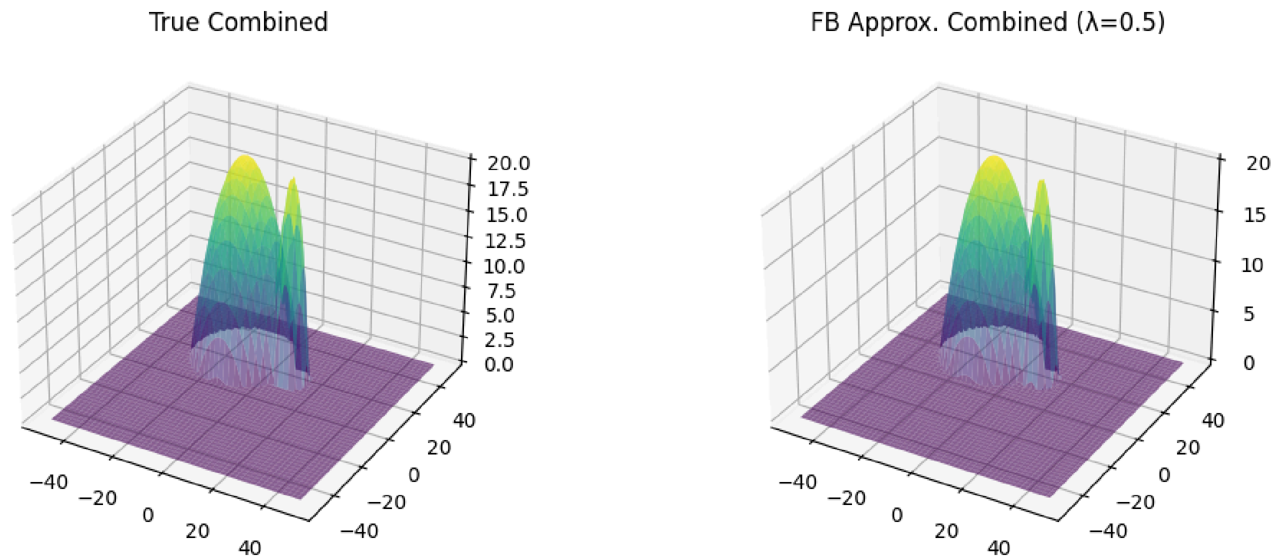
$$\begin{aligned} a_k &= \frac{2}{L^2\cdot(J_1(\alpha_k))^2}\int_0^L rf(r)J_0\left(\frac{\alpha_k r}{L}\right)dr \\ &= \frac{2}{L^2\cdot(J_1(\alpha_k))^2}\int_0^{\sqrt{A/B}} rf(r)J_0\left(\frac{\alpha_k r}{L}\right)dr \\ &= \frac{2}{L^2\cdot(J_1(\alpha_k))^2}\frac{2AL^2}{\alpha_k^2}J_2\left(\frac{\alpha_k}{L}\sqrt{\frac{A}{B}}\right) \\ &= \frac{4A}{\alpha_k^2\cdot(J_1(\alpha_k))^2}J_2\left(\frac{\alpha_k}{L}\sqrt{\frac{A}{B}}\right) \end{aligned}$$

$$\begin{aligned} f(r) &\approx \sum_{k=1}^{K} a_k J_0\left(\frac{\alpha_k(||\vec{r}-\vec{dr}||)}{L}\right) \\ &= \sum_{k=1}^{K}\sum_{m=0}^{M} a_k\kappa J_m\left(\frac{\alpha_k}{L}|\vec{r}|\right)J_m\left(\frac{\alpha_k}{L}|\vec{dr}|\right)\cos(m\theta) \end{aligned}$$
$$\text{where}$$
$$\kappa = 1, \text{if } m = 1$$
$$\kappa = 2, \text{ if } m > 1$$

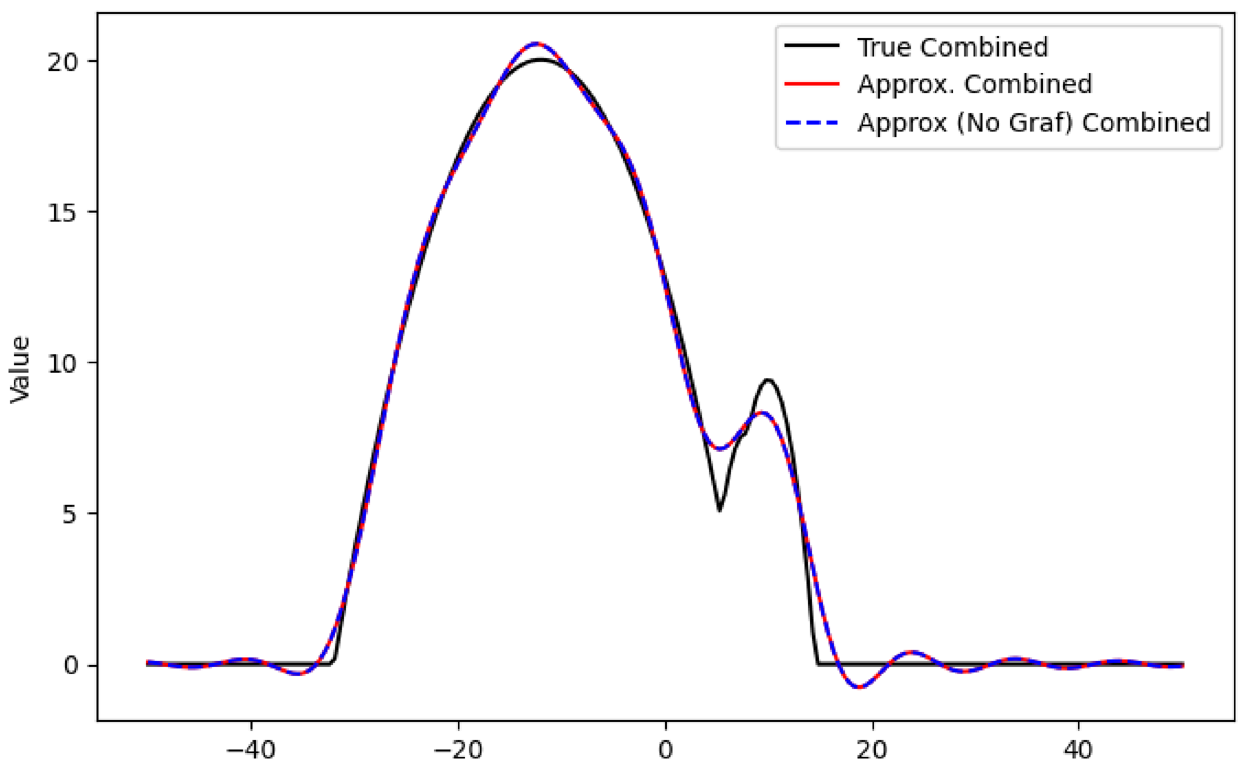
glare마다 $J_m\left(\frac{\alpha_k}{L}|\vec{r}|\right)$의 공통항이 생기게 된다. 하지만 main logic의 loop 내부에 M-summation이 생겼기 때문에 시간이 크게 늘어났다. 예를 들어 colab 환경에서 N = 100, M = 100으로 실행했을

때, 28분이 걸렸다. 코드 최적화 과정과 gpu programming을 해서 줄이는 과정이 필요.

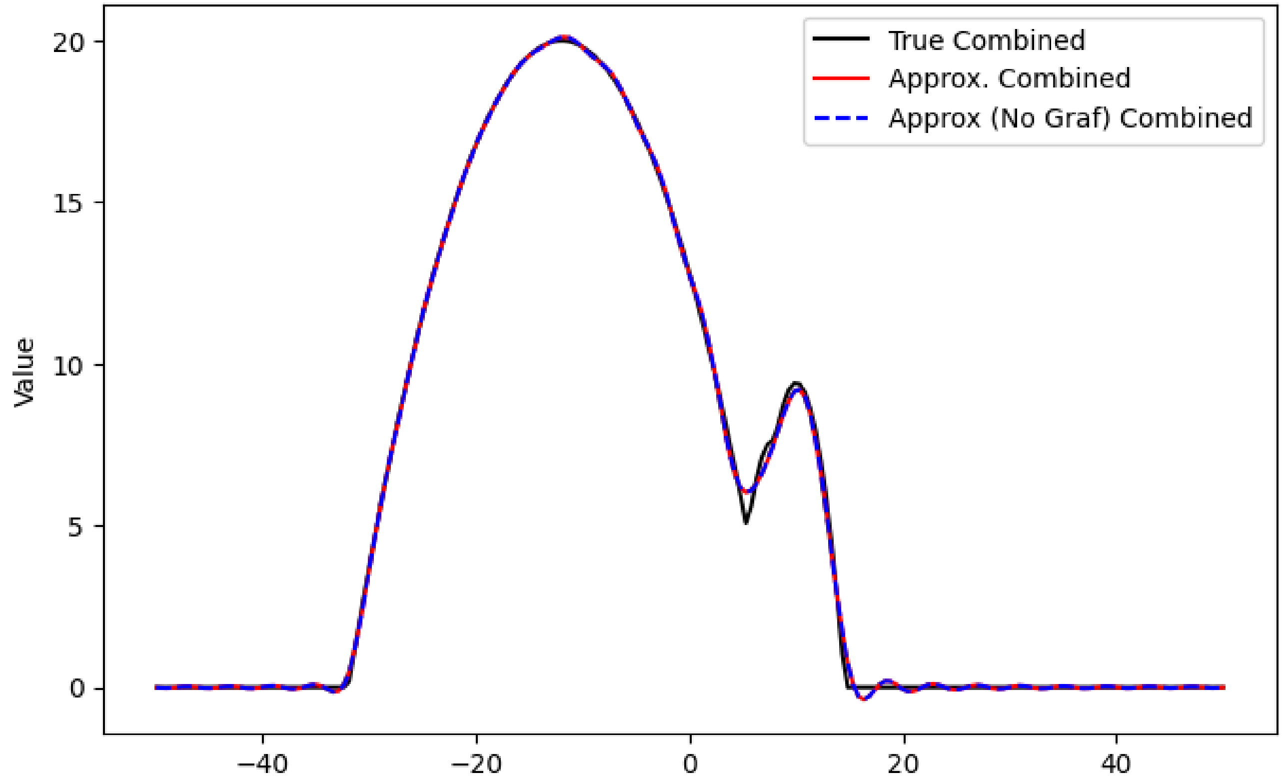
- 실행 결과
- 3D(2개의 glare를 각각 0.5씩 weight 가중, 그래프는 y=dy 기준으로 자른 단면)



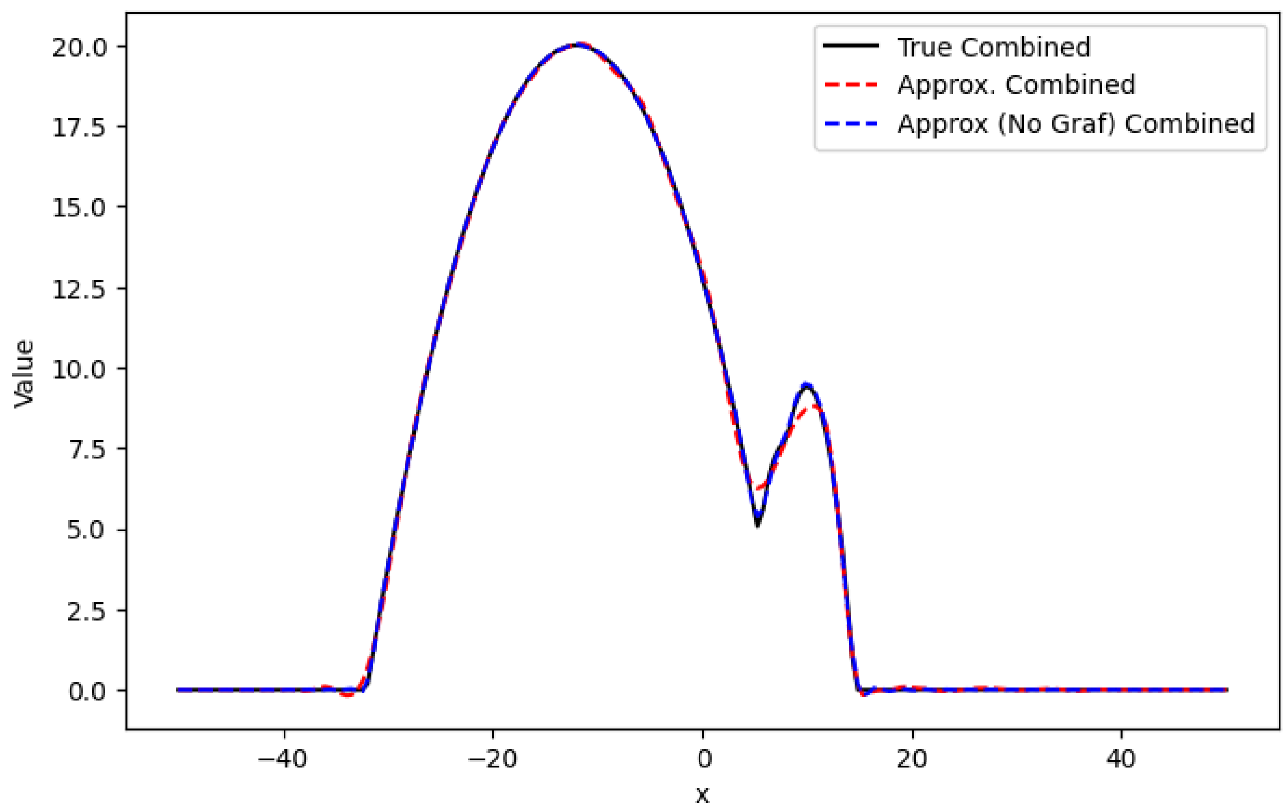
- N = 20, M = 20
  (검은 실선은 실제로 합쳤을 때, 붉은 점선은 graf를 이용한 근사, 파란 점선은 $||\vec{r} - \vec{dr}||$항을 분리하지 않은 근사)



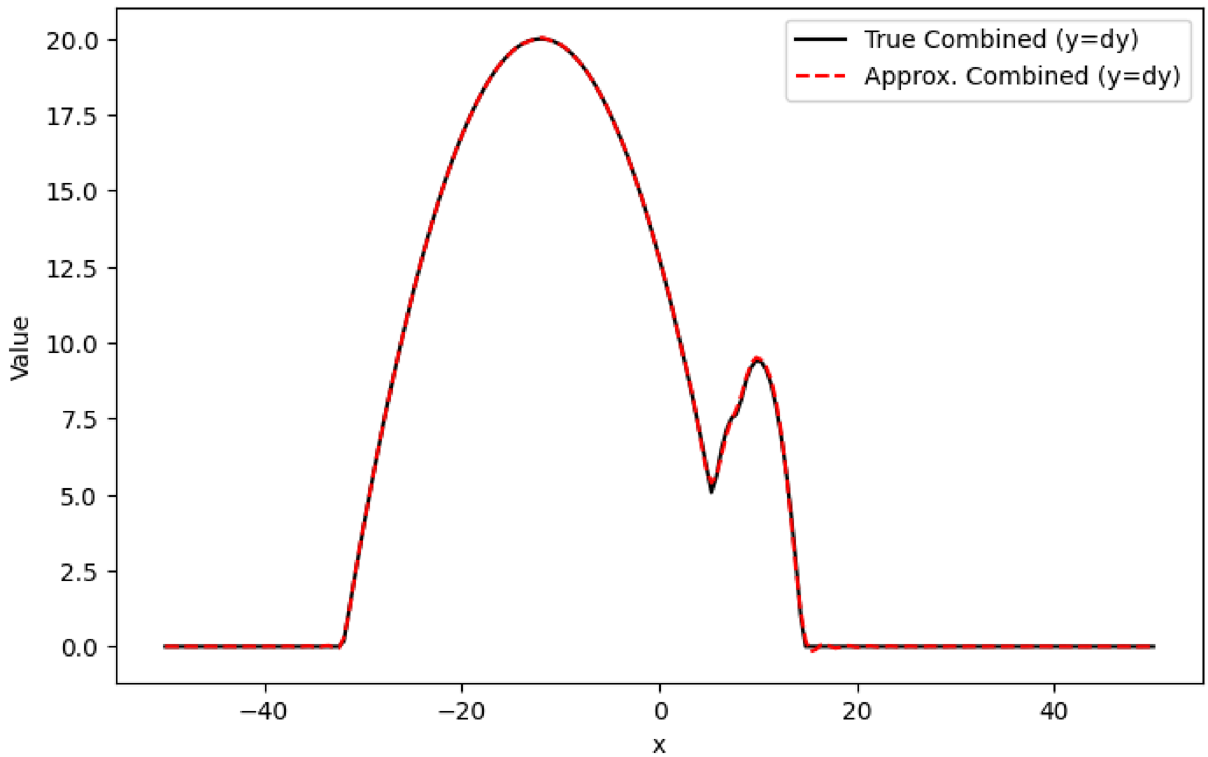
- N = 50, M = 50



- N = 100, M = 20



- N = 100, M = 100


True Combined (y=dy)
Approx. Combined (y=dy)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Value
−40
−20
0
20
40
x