기존 연구와 문제점

- 기존 방법에는 real-time과 higher quality 사이에 trade-off 발생(Ray tracing, Hullin et al, flare mapping based matrices)
- lens flare와 veiling glare를 분리하여 처리하면 성능 개선 가능

Contribution

- fourier series 기반 parametric 표현
  - glare model을 큰 주기를 갖는 주기 함수로 간주
  - screen size에 대응하는 high period > periodicity가 screen에서 보이지 않음
  - coefficient 계산을 효율적으로 계산하는 parametric 접근
- parametric multiple glares expression
  - screen의 모든 glare를 하나의 fourier series 틀에서 표현이 가능
  - moving light source and camera에 대한 coefficient를 look-up table > 효율적 계산
  - compute cost가 ghost의 개수 k에 independent, pixel 연산 단계에서 constant 유지
- Enhanced real-time glare rendering performance
  - glare rendering의 기존 접근보다 간소화된 연산으로 real-time rendering performance
  - Fourier series 기반 근사를 통해 glare quality 및 accuracy 개선
  - Physically-plausible veiling glare 모델을 효율적으로 표현

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

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

$$\begin{aligned} &\text{where} \\ &\quad \kappa = 1 \text{ if n= 0 and m = 0} \\ &\quad \kappa = 2 \text{ if n = 0 or m = 0} \\ &\quad \kappa = 4 \text{ if n > 0 and m > 0} \end{aligned}$$

- K개의 ghost 상에서 각 center를 x_k, y_k라고 했을 때

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

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

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

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

Implementation

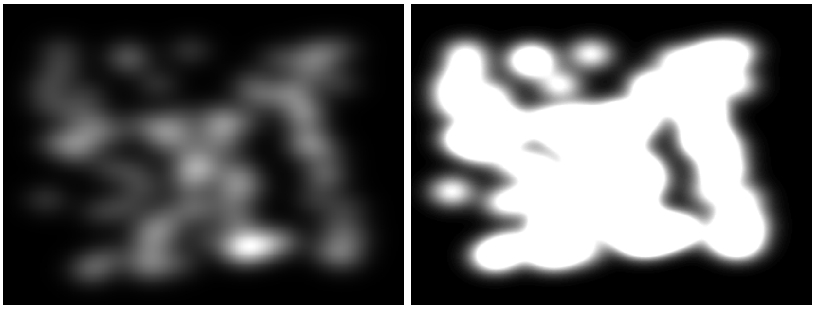
- userdata에 coefficients 저장하는 table과 SSBO setting

```
gl::Buffer* COEF = nullptr;
vector<float> A = vector<float>(N * M, 0.0);
```

- initial_update 파트에서 각 ghost에 대한 x_k, y_k를 포함한 fourier coefficient를 pre-compute. 각 계수를 A, B, C, D table에 저장

```
// initial_update()
for m, n in [0, M-1], [0, N-1]
    a_n, b_m = gaussian_coeff(n, m, sigma)
    kappa = (n==0 && m==0) ? 1.0 : ((n==0 || m==0) ? 2.0 : 4.0);
    coef = kappa * a_n * b_m / (T * T)
    for k in [0, K-1]
        x_k, y_k = ghost_means[k]
        A[m, n] += coef * cos(n * omega * x_k) * cos(m * omega * y_k)
        B[m, n] += coef * cos(n * omega * x_k) * sin(m * omega * y_k)
        C[m, n] += coef * sin(n * omega * x_k) * cos(m * omega * y_k)
        D[m, n] += coef * sin(n * omega * x_k) * sin(m * omega * y_k)
```

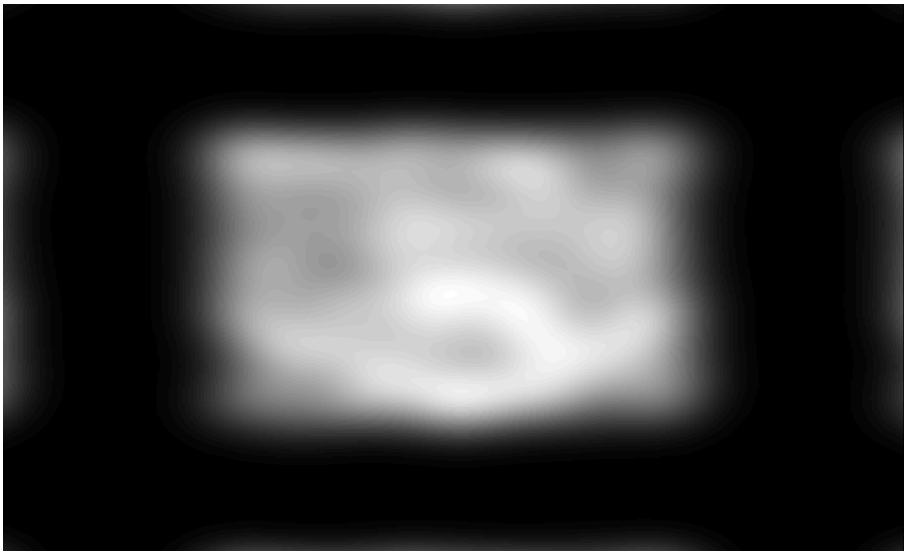
- A, B, C, D와 ghost의 mean(x_k, y_k)값을 SSBO에 저장한다. b_parametric_compute에 따라 shader에서 parmetric compute와 non-parametric compute 수행
- mipmap texture 생성 후, last_mip의 min, max를 통해 normalize
  - 결과(K = 100, N, M = 15, sigma = 0.7, T = 20.0)
  - 좌측 사진 normalize, 우측 사진 clamp



Benchmark

1. # of ghost K = 5, fourier degree N = 10, M = 10
    1. parametric case :
       FPS 4000~5000
       VGlare_FSD2.draw_glare = 0.080 ms, mean GL time = 0.207ms
       

    2. non-parametric case :
       FPS 5000~6000
       VGlare_FSD2.draw_glare = 0.014ms, mean GL time = 0.161ms
       

2. # of ghost K = 1000, fourier degree N = 10, M = 10, normalize
    1. parametric case :
       FPS 4000~5000
       VGlare_FSD2.draw_glare = 0.082 ms, mean GL time = 0.207ms
       
    2. non-parametric case :
       FPS ±2000
       VGlare_FSD2.draw_glare = 0.311ms, mean GL time = 0.442ms