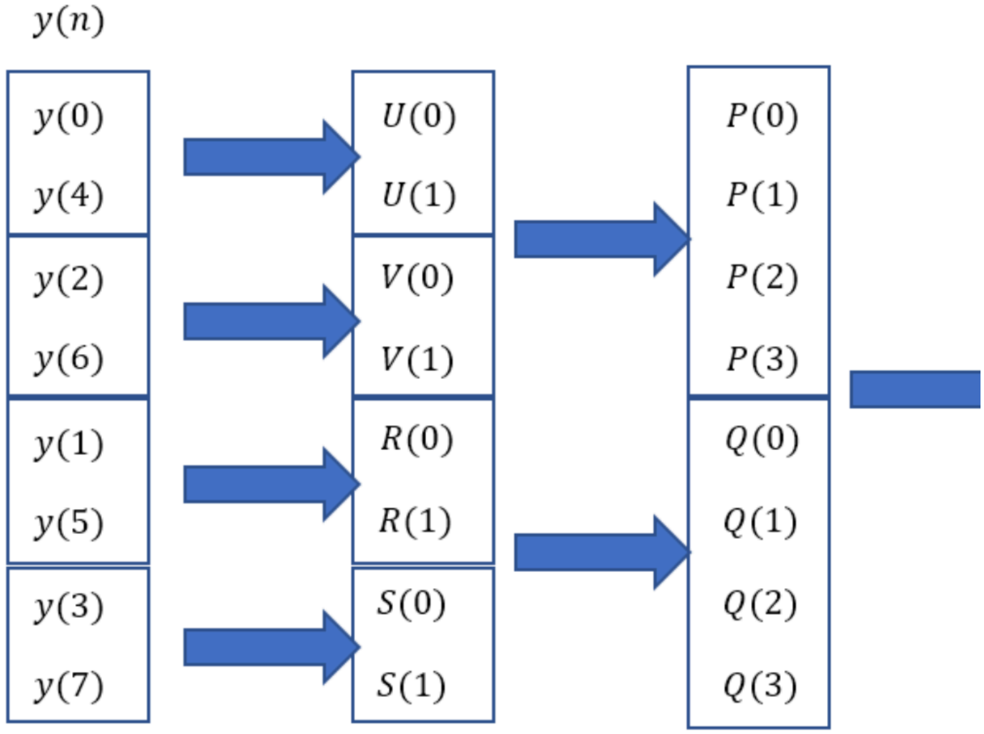
- FFT_main
  - SRC dim -> TEX dim
  - Real2Complex func으로 복소수 변환
  - pot를 check하는 이유??
    - FFT로 원활하게 진행할 수 있기 때문이다
  - 이후 DFT or FFT 진행
- userdata-fft
  - DFT
    - bind TEX, 1, 0 -> first pass를 TEX의 layer 1에 저장
    - row 기준으로 fourier 할 때는 src
    - bind TEX, 0, 0 -> second pass를 TEX의 layer 0에 저장(bypass)
    - col 에서는 row의 결과 값을 위해서 TEX->view(0,1,1,1)
      - level 0부터 1 level, layer 1부터 1의 layer에 대한 view(level -> mipmap)
      - 중간 저장용인 1 index layer 사용하겠다는 의미
      - 0 1 1 0 하면 안쓰니까 row fourier만 생기고 오류(하나 이상 layer 쓰라는 msg)
    - return view(0, 1, 0, 1), 최종결과로 0 index layer 사용하겠다
  - FFT
    - FFT preprocessing(butterfly algorithm)
      - w, h에 맞게 비트 뒤집어서 짝을 맞춰야 하기 때문에 bit-reversal 과정이 필요
      - vector<vec2> 꼴에 각 x, y의 width, height에 맞게 bit-reverse
        - ex) 0,1,2,3,4,5,6,7 -> Lv1 : [0,4],[2,6],[3,5],[4,7] -> Lv 2 : [0,2,4,6],[3,4,5,7]
          -> Lv3 : [0,1,2,3,4,5,6,7] (max level = lg N)
      - 마찬가지로 각 level 에 맞는 weight 미리 계산
        - 각 level n마다 2^(n-1)개의 weight 필요
        - Forward 기준으로 $W_k = W_{k-1} * e^{(-2\pi i/d)}, W_0 = 1$ 의 점화식이다.
          - 이를 sinusoidal 하게 나타내기 위해서 오일러 식을 써서 나타내면
            $e^{(-2\pi i/d)} = cos(-2\pi/d) + i * sin(-2\pi/d)$
            만약 $W_{k-1} = a + b \cdot i$에서 $W_k = (a + b \cdot i)(cos(-t) + i * sin(-t))$
            이므로 실수부, 허수부로 나눈다면
            $W_k.\mathrm{x} = a\cos(-t) - b\sin(-t), W_k.\mathrm{y} = a\sin(-t) + b\cos(-t)$
            코드와 동일하다는 것을 알 수 있었다.
        - 예를 들어서 N = 8이면 총 level 3 단계로
          - level 1 : $W_0 = 1$

        - level 2 : $W_0 = 1, W_1 = e^{(-2\pi i/4)}$
        - level 3 : $W_0 = 1, W_1 = e^{(-2\pi i/8)}, W_2 = e^{(-4\pi i/8)}, W_3 = e^{(-6\pi i/8)}$
      - weight texture와 bit-reversal-map은 해당 SRC의 모든 FFT에 사용될 수 있기 때문에 미리 table로 만들어 놓는다.
    - 0 layer bind, FFTShiftScale, src shift(+ scaling)
    - 1 layer bind, PermuteBits, tex layer 0을 읽음
      - 요런 식으로 tex 0, 1 layer를 번갈아가면서 사용하면서 bind
    - FFT
      - 미리 계산되어 rpm에 맞게 순서가 바뀐 tex layer와 weight texture 적용
      - 1 stage부터 log N stage까지 진행한다.
      - TEX는 0과 1을 번갈아가면서 bind하여 사용
- FFT.fx
  - Real2Complex
    - texelFetch로 TEX의 현재 xy값에 대응되는 rgb 값을 가져온느데 그 중에서 x 값을 가져오고 y 값은 0으로 vec2(texelFetch().x, 0) 꼴을 반환한다
    - 텍스쳐에서 하나의 실수 값을 가져와서 $R + 0 \cdot i$꼴의 복소수 형태로 만듦
  - DFT
    - 이때 받은 SRC level 1은 위의 Real2Complex를 거친 상태
    - tex : current pixel coordinate, ivec2
    - UV : 좌표계 중심을 이미지 중앙으로 이동
      - 왜 와이?? frequency domain에서 0 center로 하고 좌우로 보내기 위해서
      - 0,1,2,3,4,5,6,7 pixel 좌표를 -4 -3 -2 -1 0 1 2 3 꼴
    - u = UV[mode]/m*(-1.0 or 1.0) 이게 무엇인가??
      - m은 mode에 다른 width or height 값
      - 크기를 -0.5~0.5로 정규화
      - DFT에서 forward 값에 따라 exponential의 지수가 -1 붙는 것을 보면 됨
    - $f(u) = \sum F(n)e^{-i2\pi un/N}$ 을 계산하기 위해서 DFT 수행(mode : row 가정)
      - tex[mode] = n 으로 row 값 고정 현재 처리할 부분들 fix
      - vec2 f = tFetch().xy로 현재 위치의 현재 F(n) 값이라고 볼 수 있다
      - $2\pi u(n - \frac{M}{2})$
      - $I = vec2(0,0) = 0 + 0 \cdot i$에서 0부터 M에 대해 각각 구한 complex의 exp꼴을 sin함수 꼴로 변환 후 누적 sqrt(m)은 너무 작게 나오는 걸 방지
  - FFT
    - FFTShiftScale
      - brute하게 하면 주로 보는 low frequency영역이 가장자리에 위치하기 때문에 이를 가운데로 옮기기 위함, 시각적으로 잘 보이게 하기 위해 scale
      - 기존 x, y와 전체 width, height를 dim = vec2(M, N)으로 저장해서 shift

    - [0, ... ,n-1]에서 [n/2, ... n-1, 0, 1, ... ,n/2 - 1] shift, 다시 말해 n/2 씩 더하고 mod n
    - scale은 1.0으로 원본 상태를 유지
- PermuteBits
    - row 또는 col 방향으로 선택해서 진행
    - rpm은 미리 계산된 bit-reversal map이기 때문에 해당 mode에 따른 x, y 좌표만 바꿔서 계산한다.
    - dir = ivec2(1, 0)이면 행 연산, tc[row][y]의 row 값을 위의 rpm으로 재배열한다
- FFT
    - r = tc[dir.y==1?1:0] % (1<<stage) 이게 무엇인가??
        - 위 bit-inversal map 참고해서 현재 위치를 보고 stage에 따른 상대적 위치
            - - ex) 0,1,2,3,4,5,6,7 -reversal-> 0,4,2,6,3,5,4,7
                -> stage 1 : [0,4],[2,6],[3,5],[4,7]
                -> stage 2 : [0,2,4,6],[3,4,5,7]
                -> stage 3 : [0,1,2,3,4,5,6,7]
        - 예를 들어 row 연산에서 tc.x: 0 1 2 3 4 5 6 7 일 때 이미 rpm 적용되었기에 옆으로 적당히 그룹을 지어주기만 하면 된다.
          tc.x: 0 1 2 3 4 5 6 7
          r(stage 1) : 0 1 0 1 0 1 0 1 (hd = 1)
          r(stage 2) : 0 1 2 3 0 1 2 3 (hd = 2)
          r(stage 3) : 0 1 2 3 4 5 6 7 (hd = 4) 각 stage마다 더 큰 그룹으로 합쳐짐
        - hd = ivec2(1<<(stage-1))
            - 상대적인 간격? cell 크기? 를 절반으로 나눔(weight 적용할 부분)
            - 하단, 상단으로 구분했을 때 다음과 같은 차이가 있다.
            - ex) N = 8 중 k = 2, 하단의 경우
              $Y(2) = S(2) + W_2 T(2), Y(k) = S(k) + W_k T(k)$
              k = 5, 상단의 경우
              $Y(5) = S(1) - W_1 T(1), Y(k) = S(k-hd) - W_{k-hd} T(k-hd)$
              S(0)~S(3)과 T(0)~T(3) 역시
              $S(0) = A(0) + W_0 B(0) \ldots S(3) = A(1) - W_1 B(1)$
              분해 가능 (T도 동일 한 방식으로 C, D)
              계속 내려가서 kd = 1인 stage 1에서는 src 값 y()만으로 구한다
              $X(0) = y(0) + y(1), X(1) = y(0) - y(1)$
        - 유도를 할 때와는 반대로 실제로 현재 vertex k에 대한 FFT 값 Y(k)는 stage 1의 y(k)를 가지고 stage 2의 A, B, C, D를 구하고 stage 3로 가는 bottom up 방식이다.

- 식의 일관성을 위해서
  하단부는
  $fragColor(k) = P(k) + W_k Q(k + hd)$
  상단부는
  $fragColor(k) = P(k - hd) + W_{k-hd} Q(k)$
  

  출처
- 위 사진처럼 row or col을 고정시켜놓고 보았을 때 1차원으로 했을 때 각 r 값으로 구역을 나누고 한 셀 크기를 hd라고 생각하면 된다.