1 P_{nest}

```
type Image = [:[:Int:]:]
type Hist a = [:a:]

hbalance :: Image -> Image
hbalance img =
   let h = hist img
       a = accu h
       a0 = headP a
       agmax = lastP a
       n = normalize a0 agmax a
       s = scale gmax n
       img’ = apply s img
   in  img’

hist :: Image -> Hist Int
hist =
   sparseToDenseP (gmax+1) 0
   . mapP (\g -> (headP g,lengthP g))
   . groupP
   . sortP
   . concatP

accu :: Hist Int -> Hist Int
accu = scanlP (+) 0

normalize :: Int -> Int -> Hist Int -> Hist Double
normalize a0’ agmax’ as =
   let a0 = fromIntegral a0’
       agmax = fromIntegral agmax’
       divisor = agmax - a0
   in  [: (fromIntegral freq’ - a0) / divisor | freq’ <- as :]

scale :: Int -> Hist Double -> Hist Int
scale gmax as = [: floor (a * fromIntegral gmax) | a <- as :]

apply :: Hist Int -> Image -> Image
apply as img = mapP (mapP (as !:)) img
```

2 Work and Depth Table

- n sei die Anzahl der Bildpixel
- w sei die Bildbreite
- h sei die Bildhöhe
- p sei die Anzahl der PUs (gang members).

Table 1: Work and Depth complexities

function or variable	O(W)	O(D)
hbalance	$\max(n \log n, gmax)$	$\log \max(n, gmax)$
hist	$\max(n \log n, gmax)$	log n
sparseToDenseP	gmax	1
groupP	n	$\log n$
sortP	n $\log n$	$\log n$
concatP	1	1
accu	gmax	$\log gmax$
scanlP	gmax	$\log gmax$
normalize	gmax	1
scale	gmax	1
apply	$n = w \cdot h \cdot O(1)$	1
mapP f xs	$W(f, x) \cdot size(xs)$	1
headP/lastP	1	1
indexP, !:	1	1

3 Other aspects **e.g. sync-points, programmer workload, simplicity**

- optimisations: ?
- progammer-workload: ?
- simplicity: ?