Canny Edge Detection
Canny Edge Detection

GreyScale → Blur → Differentiate

Mag / Dir → Maxima → Link Edges
Canny Edge Detection

1. GreyScale → Blur → Differentiate
2. Data Parallel
3. Data Parallel
4. Data Parallel
5. Mag / Dir → Maxima → Link Edges
6. Data Parallel
7. Data Parallel
8. Recursive O(edge pixels)
Canny Edge Detection

**GreyScale** → **Blur** → **Differentiate**

**Pixel/Pixel**

- **Binomial\(_{7X}\)**
  \[
  \begin{bmatrix}
  1 & 6 & 15 & 20 & 15 & 6 & 1
  \end{bmatrix}
  \]

- **Binomial\(_{7Y}\)**
  \[
  \begin{bmatrix}
  1 & 6 & 15 & 20 & 15 & 6 & 1
  \end{bmatrix}^T
  \]

**Stencil Convolution**

- **Sobel\(_X\)**
  \[
  \begin{bmatrix}
  -1 & 0 & +1
  -2 & 0 & +2
  -1 & 0 & +1
  \end{bmatrix}
  \]

- **Sobel\(_Y\)**
  \[
  \begin{bmatrix}
  -1 & -2 & -1
  0 & 0 & 0
  +1 & +2 & +1
  \end{bmatrix}
  \]

**Mag / Dir** → **Maxima** → **Link Edges**

**Pixel/Pixel**

**Comparison of adjacent pixels**

- **Wildfire Algorithm**
A single point result from a 3x3 stencil.

\[(A \ast K)(x, y) = \sum_i \sum_j A(x + i, y + j) \cdot K(i, j)\]

\[r = a[i-1][j-1] \ast k[-1][-1] + a[i-1][j] \ast k[-1][0] + a[i-1][j+1] \ast k[-1][+1] + a[i][j-1] \ast k[0][-1] + a[i][j] \ast k[0][0] + a[i][j+1] \ast k[0][+1] + a[i+1][j-1] \ast k[+1][-1] + a[i+1][j] \ast k[+1][0] + a[i+1][j+1] \ast k[+1][+1]\]
Several options all of which are used in practice, regarding the border. The border itself is marked in grey. There are regions of the array where we can apply the stencil without worry. In the figure, the white squares indicate the regions.

Figure 7 shows the application of a 6x6 stencil close to the top left case where the stencil “falls off” the edge of the array. For example, in the case of stencil functions, an immediate concern is what to do about the region where the stencil overlaps the border.

### TODO:

4.1 Border Handling

Handling which we will discuss further in the next section.

#### 4.1.1 Common Cases

- **Case 1:** Stencils smaller than a certain fixed size would allow us to support most of the common cases.
- **Case 2:** Stencils larger than a certain fixed size would allow us to support most of the common cases.

#### 4.1.2 Optimization Techniques

- **Optimization 1:** Look for patterns and optimization opportunities in stencil functions.
- **Optimization 2:** Implement optimizations for specific stencil sizes.

#### 4.1.3 Limitations

- **Limitation 1:** Optimization techniques may not be applicable for all stencil sizes.
- **Limitation 2:** There may be cases where optimization is not possible.

#### 4.1.4 Future Work

- **Future Work 1:** Investigate the effectiveness of optimization techniques for different stencil sizes.
- **Future Work 2:** Explore the possibility of integrating optimization techniques with other computational methods.

#### 4.1.5 Conclusion

- **Conclusion 1:** Optimization techniques can significantly improve the performance of stencil computations.
- **Conclusion 2:** Further research is needed to fully explore optimization opportunities.

---

#### References

1. **Sobel**
   - Roberts
   - X

---

** IMPLEMENTATION:**

- **Implementation 1:** A general purpose implementation for stencil functions.
- **Implementation 2:** A specialized implementation for specific stencil sizes.

---

**Applying Stencils:**

- **Applying 1:** Applying it to the image.
- **Applying 2:** Applying the stencil along the X axis.
- **Applying 3:** Applying the stencil along the Y axis.

---

**Related Works:**

- **Related Works 1:** Related kernels are obtained by rotating the kernel.
- **Related Works 2:** Rotating the kernel by 3 degrees yields a related kernel.

---

**Efficiency:**

- **Efficiency 1:** Most kernels have odd dimensions.
- **Efficiency 2:** Most kernels are square.
- **Efficiency 3:** Most kernels fit in a 8x8 matrix.
- **Efficiency 4:** All coefficients are statically known.
- **Efficiency 5:** Many coefficients are zero.
- **Efficiency 6:** All kernels are symmetric.
- **Efficiency 7:** Many kernels are based on the values of the kernel coefficients.
- **Efficiency 8:** Some kernels are based on the values of the kernel coefficients.

---

**Efficiency Analysis:**

- **Analysis 1:** Efficiency analysis shows that some kernels are more efficient than others.
- **Analysis 2:** Efficiency analysis can help in selecting the most efficient kernel for a given task.

---

**Optimization Opportunities:**

- **Opportunities 1:** There are opportunities for optimization in stencil computations.
- **Opportunities 2:** Optimization opportunities can lead to significant performance improvements.

---

**Conclusion:**

- **Conclusion 1:** Stencil computations offer significant performance opportunities.
- **Conclusion 2:** Further research is needed to fully explore these opportunities.

---

**License:**

- **License 1:** This work is licensed under the [Apache License 2.0](https://www.apache.org/licenses/LICENSE-2.0).
- **License 2:** All rights reserved.

---

**Contact:**

- **Contact 1:** Contact the authors for more information.
- **Contact 2:** Feedback and contributions are welcome.

---

**Acknowledgments:**

- **Acknowledgments 1:** Thanks to the team for their support.
- **Acknowledgments 2:** Special thanks to the reviewers for their valuable feedback.

---

**References:**

- **Reference 1:** [Sobel](https://www.sobel.org/).
- **Reference 2:** [Roberts](https://www.roberts.org/).
- **Reference 3:** [X](https://www.x.org/).
- **Reference 4:** [Y](https://www.y.org/).

---

**Data:**

- **Data 1:** The data for this project is available in [this directory](https://www.data.org/).
- **Data 2:** Data generated using the [code](https://www.code.org/).

---

**Authors:**

- **Authors 1:** [Author 1](https://www.author1.org/).
- **Authors 2:** [Author 2](https://www.author2.org/).

---

**Copyright:**

- **Copyright 1:** Copyright © 2023 [Company](https://www.company.org/).
- **Copyright 2:** All rights reserved.

---

**License:**

- **License 1:** The license for this work is [Apache License 2.0](https://www.apache.org/licenses/LICENSE-2.0).
- **License 2:** The license for this work is [GPLv3](https://www.gnu.org/licenses/gpl-3.0).
Testing the border at every pixel is slow....

{-# INLINE relaxLaplace #-}
relaxLaplace :: Image -> Image
relaxLaplace arr
  = traverse arr id elemFn
where _ ::. height ::. width = extent arr

{-# INLINE elemFn #-}
elemFn get d@(Z ::. i ::. j)
  = if isBorder i j
    then get d
    else (get (Z ::. (i-1) ::. j)
          + get (Z ::. i ::. (j-1))
          + get (Z ::. (i+1) ::. j)
          + get (Z ::. i ::. (j+1))) / 4

{-# INLINE isBorder #-}
isBorder i j
  = (i == 0) || (i >= width - 1)
  || (j == 0) || (j >= height - 1)
Testing the border at every pixel is slow....

```haskell
{--# INLINE relaxLaplace #-}  
relaxLaplace :: Image -> Image  
relaxLaplace arr  
    = traverse arr id elemFn  
where _ :: height :: width = extent arr

{--# INLINE elemFn #-}  
elemFn get d@(_ :: i :: j)  
    = if isBorder i j  
    then get d  
    else (get (_ :: (i-1) :: j)  
           + get (_ :: i     :: (j-1))  
           + get (_ :: (i+1) :: j)  
           + get (_ :: i     :: (j+1))) / 4

{--# INLINE isBorder #-}  
isBorder i j  
    = (i == 0) || (i >= width - 1)  
    || (j == 0) || (j >= height - 1)
```

DIE!
Sharing in computations of adjacent pixels.

\[
\begin{align*}
3 \times 3 \times 4 &= 36 \\
3 \times 6 &= 18 \\
36 / 18 &= 2
\end{align*}
\]
Application of a single Laplace stencil.

```plaintext
case quotInt# ixLinear width of { ix ->
case remInt# ixLinear width of { iy ->
  writeFloatArray# world arrDest ixLinear
    (+## (indexFloatArray# arrBV
      (+# arrBV_start (+# (*# arrBV_width iy) ix)))
    (*## (indexFloatArray# arrBM
      (+# arrBM_start (+# (*# arrBM_width iy) ix)))
    (/## (+## (+## (+## (indexFloatArray# arrSrc
      (+# arrSrc_start (+# (*# (-# width 1) iy) ix))))
     (indexFloatArray# arrSrc
      (+# arrSrc_start (+# (*# width iy) (-# ix 1)))))
    (indexFloatArray# arrSrc
      (+# arrSrc_start (+# (*# (+# width 1) iy) ix)))
    (indexFloatArray# arrSrc
      (+# arrSrc_start (+# (*# width iy) (+# ix 1)))))
  4.0)))
}
```
Application of a single Laplace stencil.

\[
x + y \times \text{width}
\]
Two new features:

**Partitioned arrays**
Represent the partitioning into border and internal regions directly, to avoid the test in the inner loop.

**Cursored arrays**
Expose intermediate linear indices when calculating array offsets, to avoid repeated use of \(x + y \times \text{width}\).
New Repa Array Types:

```haskell
data Array sh a = Array { arrayExtent :: sh, arrayRegions :: [Region sh a] }
data Region sh a = Region { regionRange :: Range sh, regionGen :: Generator sh a }
data Range sh = RangeAll | RangeRects { rangeMatch :: sh -> Bool, rangeRects :: [Rect sh] }
data Rect sh = Rect sh sh
```
New Repa Array Types:

data Generator sh a
  = GenManifest { genVector :: Vector a }

| forall cursor.
GenCursored { genMake :: sh -> cursor
  , genShift :: sh -> cursor -> cursor
  , genLoad  :: cursor -> a }
Defining the stencil

```haskell
data Stencil sh a
    = Stencil { stencilSize :: sh,
                stencilZero :: b,
                stencilAcc :: sh -> a -> a -> a }

makeStencil :: sh -> (sh -> Maybe a) -> Stencil sh a
makeStencil ex getCoeff
    = Stencil ex 0
        \ix val acc
             -> case getCoeff ix of
                    Nothing  -> acc
                    Just coeff  -> acc + val * coeff

laplace :: Stencil sh a
laplace = makeStencil (Z :. 3 :. 3)
        \ix -> case ix of
                Z :.  0 :.  1 -> Just 1
                Z :.  0 :. -1 -> Just 1
                Z :.  1 :.  0 -> Just 1
                Z :. -1 :.  0 -> Just 1
                _             -> Nothing
```
Defining the stencil

```haskell
data Stencil sh a
  = Stencil { stencilSize :: sh,
              stencilZero :: b,
              stencilAcc :: sh -> a -> a -> a }

makeStencil :: sh -> (sh -> Maybe a) -> Stencil sh a
makeStencil ex getCoeff
  = Stencil ex 0
    $ 
    \ix val acc
    -> case getCoeff ix of
        Nothing -> acc
        Just coeff -> acc + val * coeff

laplace :: Stencil sh a
laplace = [ 1 0 1
            0 1 0
            ]
```

Not a Number

{-# RULES
  "add-id" forall (x :: Float). x + 0 = x
  "mul-id" forall (x :: Float). x * 0 = 0
#-}
Not a Number

{-# RULES
  "add-id" forall (x :: Float). x + 0 = x
  "mul-id" forall (x :: Float). x * 0 = 0
#-}

With IEEE 754 Floats

\[ \infty \times 0 = \text{NaN} \]
{--# RULES
    “add-id”  \textit{forall} (x :: \text{Float}). x + 0 = x
    “mul-id”  \textit{forall} (x :: \text{Float}). x * 0 = 0
#-}
Applying a Stencil

```haskell
-- | Compute gradient in the X direction.
gradientX :: Array DIM2 Float -> Array DIM2 Float
gradientX img = force2 $ forStencil2 (BoundConst 0) img
               [stencil2 | -1 0 1
                          -2 0 2
                          -1 0 1 ]
```
Detection of Local Maxima

-- | Suppress pixels which are not local maxima.

```
maxima :: Float -> Float -> Image (Float, Float) -> Image Word8
maxima threshLow threshHigh dMagOrient
  = force2 $ makeBordered2 (extent dMagOrient) 1 (GenCursor id addDim (const 0))
    (GenCursor id addDim compare)

  where compare ix@(sh :. i :. j)
    | o == undef    = edge None
    | o == horiz    = isMax (getMag (sh :. i :: j-1)) (getMag (sh :. i :: j+1))
    | o == vert     = isMax (getMag (sh :. i-1 :: j))  (getMag (sh :. i+1 :: j))
    | o == negDiag  = isMax (getMag (sh :. i-1 :: j-1)) (getMag (sh :. i+1 :: j+1))
    | o == posDiag  = isMax (getMag (sh :. i-1 :: j+1)) (getMag (sh :. i+1 :: j-1))
    | otherwise     = edge None

  where
    o    = getOrient ix
    m    = getMag   ix

  getMag    = fst . (dMagOrient !)
  getOrient = snd . (dMagOrient !)

  isMax mag1 mag2
    | m < threshLow = edge None
    | m < mag1      = edge None
    | m < mag2      = edge None
    | m < threshHigh = edge Weak
    | otherwise     = edge Strong
```
mapStencil2 :: Boundary a -> Stencil DIM2 a -> Array DIM2 a -> Array DIM2 a

mapStencil2 boundary (Stencil sExtent _ _) arr
= let (Z ::. aHeight ::. aWidth) = extent arr
     (Z ::. sHeight ::. sWidth) = sExtent

     rectsInternal   = ...
     rectsBorder     = ...
     inInternal ix   = ...
     inBorder ix     = ...

     make (Z::y::x) = Cursor (x + y*aWidth)
     shift (Z::y::x) (Cursor offset)
         = Cursor (offset + x + y*aWidth)

     loadBorder ix   = case boundary of ... 
     loadInner cursor = unsafeAppStencil2 stencil arr shift cursor

     in Array (extent arr)
         [ Region (RangeRects inBorder rectsBorder)
           (GenCursored id addIndex loadBorder)
         , Region (RangeRects inInternal rectsInternal)
           (GenCursored make shift loadInner) ]
unsafeAppStencil2
:: Stencil DIM2 a -> Array DIM2 a
-> (DIM2 -> Cursor -> Cursor) -- shift cursor
-> Cursor -> a

unsafeAppStencil2
stencil@(Stencil sExtent sZero sAcc)
arr@(Array aExtent [Region RangeAll (GenManifest vec)])
shift cursor

| _ :: sHeight :: sWidth <- sExtent
, sHeight <= 3, sWidth <= 3
= template3x3 loadFromOffset sZero

| otherwise = error "stencil too big for this method"

where getData (Cursor index)
  = vec `unsafeIndex` index

loadFromOffset oy ox
  = let offset = Z :: oy :: ox
      cur'   = shift offset cursor
      in    sAcc offset (getData cur')
template3x3 :: (Int -> Int -> a -> a) -> a -> a

template3x3 f sZero
  = f (-1) (-1) $ f (-1) 0 $ f (-1) 1
  $ f 0 (-1) $ f 0 0 0 $ f 0 1
  $ f 1 (-1) $ f 1 0 0 $ f 1 1 1
  $ sZero

... dreaming of supercompilation
fillCursoredBlock2
:: Elt a => IOVector a          -- vec
-> (DIM2   -> cursor)           -- makeCursor
-> (DIM2   -> cursor -> cursor) -- shiftCursor
-> (cursor -> a) -> Int         -- loadElem, width
-> Int -> Int -> Int -> Int     -- x0 y0 x1 y1
-> IO ()

fillCursoredBlock2 !vec !make !shift !load !width !x0 !y0 !x1 !y1
= fillBlock y0
where
  fillBlock !y
  | y > y1            = return ()
  | otherwise
  = do fillLine4 x0
       fillBlock (y + 1)
where
  fillLine4 !x
  | x + 4 > x1       = fillLine1 x
  | otherwise
  = do BODY
       fillLine4 (x + 4)

fillLine1 !x
| x > x1       = return ()
| otherwise
= do unsafeWrite vec (x + y * imageWidth)
    (getElem $ makeCursor (Z:.y:.x))
    fillLine1 (x + 1)
```plaintext
fillLine4 !x
| x + 4 > x1  = fillLine1 x
| otherwise    = do let srcCur0 = make (Z:.y:.x)
                let srcCur1 = shift (Z:.0:.1) srcCur0
                let srcCur2 = shift (Z:.0:.1) srcCur1
                let srcCur3 = shift (Z:.0:.1) srcCur2

                let val0 = load srcCur0
                let val1 = load srcCur1
                let val2 = load srcCur2
                let val3 = load srcCur3

                let !dstCur0 = x + y * width
                unsafeWrite vec (dstCur0) val0
                unsafeWrite vec (dstCur0 + 1) val1
                unsafeWrite vec (dstCur0 + 2) val2
                unsafeWrite vec (dstCur0 + 3) val3
                fillLine4 (x + 4)
```
case ># (+# w4_s3lq 4) ipv8_i30r of _ {
  False ->
    let { a22_s4SQ = +# w4_s3lq (*# w3_s3ly ipv1_X2LM) } in
    let { Vector rb_i2YQ _ rb2_i2YS _ <- ds6_d2b5 `cast` ... } in
    let { a23_i30Y = +# w4_s3lq (*# w3_s3ly ipv1_X2LM) } in
    let { __DEFAULT ~ s#_X39w <- writeFloatArray#
        arr#_i2Pd
        a23_i30Y
        (plusFloat#
          (plusFloat#
            (plusFloat#
              (plusFloat#
                (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ ipv1_X2LM) 1))))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ ipv1_X2LM) (1)))) __float -1.0))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a22_s4SQ 1)) __float 2.0)))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ (-1)))) __float -2.0))
                (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ (*# (-1) ipv1_X2LM) 1)))))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a22_s4SQ (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
          (w2_s3ls `cast` ...) }) in
    let { __DEFAULT ~ s#1_X39F <- writeFloatArray#
        arr#_i2Pd
        (+# a23_i30Y 1)
        (plusFloat#
          (plusFloat#
            (plusFloat#
              (plusFloat#
                (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) 1))))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) (-1)))) __float -1.0))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG 1)) __float 2.0)))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (-1)))) __float -2.0))
                (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (*# (-1) ipv1_X2LM) 1))))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
          (w2_s3ls `cast` ...) }) in
    let { a24_s4TG = +# a22_s4SQ 1 } in
    let { __DEFAULT ~ s#_X39w <- writeFloatArray#
        arr#_i2Pd
        (+# a23_i30Y 1)
        (plusFloat#
          (plusFloat#
            (plusFloat#
              (plusFloat#
                (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) 1))))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG ipv1_X2LM) (-1)))) __float -1.0))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# (+# a24_s4TG 1)) __float 2.0)))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (-1)))) __float -2.0))
                (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (*# (-1) ipv1_X2LM) 1))))
                (timesFloat# (indexFloatArray# rb2_i2YS (+# rb_i2YQ (+# a24_s4TG (*# (-1) ipv1_X2LM)) (-1)))) __float -1.0))
          (w2_s3ls `cast` ...) }) in
  _
movl 0x03(%edi),%ecx
movl 0x07(%edi),%edx
movl 0x08(%ebp),%esi
movl 0x10(%ebp),%ebx
movl %ebx,0x04(%esp)
leal 0x02(%esi,%edx),%eax
movl %eax,(%esp)
movl 0x14(%ebp),%eax
leal 0x02(%esi,%eax),%edi
addl %edx,%ebx
addl %edx,%edi
movss 0x08(%ecx,%edi,4),%xmm1
subss 0x08(%ecx,%ebx,4),%xmm1
movl (%esp),%edi
movss 0x08(%ecx,%edi,4),%xmm2
addss %xmm2,%xmm2
addss %xmm1,%xmm2
leal 0x02(%esi,%eax),%edi
movss 0x08(%ecx,%edi,4),%xmm1
mulss %xmm0,%xmm1
addss %xmm2,%xmm1
leal 0x02(%esi),%edi
movl %edi,(%esp)
movl %edi,%ebx
subl %eax,%ebx
addl %edx,%ebx
addss 0x08(%ecx,%ebx,4),%xmm1
movl $0x3ffffff,%ebx
subl %eax,%ebx
leal 0x01(%ebx,%esi),%eax
addl %edx,%eax
subss 0x08(%ecx,%eax,4),%xmm1
movl 0x04(%esp),%eax
movl 0x14(%esp),%ecx
movss 0x08(%ecx,%edi,4),%xmm1
subss 0x08(%ecx,%ebx,4),%xmm1
movl 0x10(%ebp),%eax
movl %eax,0x04(%esp)
movl 0x14(%ebp),%edx
leal 0x01(%esi,%edx),%ebx
movl 0x10(%esp),%edi
movl 0x03(%edi),%eax
addl %ecx,%ebx
movl 0x07(%edi),%ecx
movl 0x03(%esi,%edx),%edi
addl %ecx,%edi
subl %edx,%edi
addl %ecx,%edi
addss 0x08(%ecx,%edi,4),%xmm1
leal 0x01(%esi,%ecx),%edi
movss 0x08(%eax,%edi,4),%xmm2
addss %xmm2,%xmm2
addss %xmm1,%xmm2
leal 0x01(%esi,%ecx),%edi
movl 0x04(%esp),%eax
movl 0x14(%esp),%ecx
movl 0x03(%esi,%edx),%edi
addl %ecx,%edi
subl %edx,%edi
addl %ecx,%edi
movl 0x04(%esp),%eax
movl 0x14(%esp),%ecx
movss %xmm1,0x0c(%eax,%ecx,4)
movl 0x10(%ebp),%eax
movl %eax,0x04(%esp)
movl 0x14(%ebp),%edx
leal 0x01(%esi,%edx),%ebx
movl 0x10(%esp),%edi
movl %eax,0x04(%esp)
movl 0x03(%edi),%eax
addl %ecx,%ebx
movl 0x07(%edi),%ecx
movl 0x03(%esi,%edx),%edi
addl %ecx,%edi
subl %eax,%ebx
addl %edx,%ebx
movl 0x14(%esp),%ecx
movss %xmm1,0x10(%eax,%ecx,4)
\$(w_{4_{s_s3lq}} :: \text{Int#}) \ (w_{2_{s_s3ls}} :: \text{State# RealWorld}) \rightarrow \$

\text{case }\>\> (\text{(# w}_{4_{s_s3lq}} 4) \ \text{ipv8}_{i_30r} \text{ of } \ _{\text{ (}}} \ $

\text{False \}{ \text{ let } \{ \ a_{22_{s_s4SQ}} = \# \ w_{4_{s_s3lq}} (\#\ w_{3_{s_s3ly}} \ \text{ipv1}_{X2LM}) \ } \text{ in} \ $

\text{let } \{ \ \text{Vector rb}_{i_2YQ} _{\_} \ rb_{2_{i_2YS}} \_ \ <- \ \text{ds6}_{d_2b5} `\text{cast}` \ldots \ \text{ in} \ $

\text{let } \{ \ a_{23_{_i30Y}} = \# \ w_{4_{s_s3lq}} (\#\ w_{3_{s_s3ly}} \ \text{ipv1}_{X2LM}) \ \text{ in} \ $

\text{let } \{ \ _\text{DEFAULT} \rightarrow \ s_{\#_{X39w}} \ \text{ in} \ $

\text{<- writeFloatArray#} \ $

arr\_i2Pd \ $

a_{23_{i30Y}} \ $

(\text{plusFloat#} \ $

(\text{plusFloat#} \ $

(\text{plusFloat#} \ $

(\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{22_{s_s4SQ}} \ \text{ipv1}_{X2LM}) 1))} \ $

(\text{timesFloat#} (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{22_{s_s4SQ}} \ \text{ipv1}_{X2LM}) (-1))} \text{ float 1.0)))} \ $

(\text{timesFloat#} (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{22_{s_s4SQ}} 1)))} \text{ float 2.0)))} \ $

(\text{timesFloat#} (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{22_{s_s4SQ}} (-1)))} \text{ float 2.0)))} \ $

(\text{timesFloat#} (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{22_{s_s4SQ}} (\# (-1) \ \text{ipv1}_{X2LM})) (-1)))} \text{ float 1.0)))} \ $

\text{ (w}_{2_{s_s3ls}} `\text{cast`} \ldots \}$ \ $

\text{in } \text{......} \}$

\text{<- writeFloatArray#} \ $

arr\_i2Pd \ $

(\text{plusFloat#} \ $

(\text{plusFloat#} \ $

(\text{plusFloat#} \ $

\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{24_{s_s4TG}} \ \text{ipv1}_{X2LM}) 1))} \ $

\text{timesFloat# (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{24_{s_s4TG}} \ \text{ipv1}_{X2LM}) (-1)))} \text{ float 1.0)))} \ $

\text{timesFloat# (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{24_{s_s4TG}} 1)))} \text{ float 2.0)))} \ $

\text{timesFloat# (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{24_{s_s4TG}} (-1)))} \text{ float 2.0)))} \ $

\text{timesFloat# (\text{indexFloatArray#} \ rb_{2_{i2YS}} (\# \ rb_{i2YQ} (\text{(# (\# a_{24_{s_s4TG}} (\# (-1) \ \text{ipv1}_{X2LM})) (-1)))} \text{ float 1.0)))} \ $

s_{\#_{X39w}} \}$ \ $

\text{in } \text{......} \}$
fillLine4 !x
| \( x + 4 > x_1 \) = fillLine1 x
| otherwise
= do let srcCur0 = make (Z:.y:.x)
    let srcCur1 = shift (Z:.0:.1) srcCur0
    let srcCur2 = shift (Z:.0:.1) srcCur1
    let srcCur3 = shift (Z:.0:.1) srcCur2

    let val0 = load srcCur0
    let val1 = load srcCur1
    let val2 = load srcCur2
    let val3 = load srcCur3

    let !dstCur0 = x + y * width
    unsafeWrite vec (dstCur0) val0
    unsafeWrite vec (dstCur0 + 1) val1
    unsafeWrite vec (dstCur0 + 2) val2
    unsafeWrite vec (dstCur0 + 3) val3
    fillLine4 (x + 4)
The poison

touch# :: forall o

  . o -> State# RealWorld
  -> State# RealWorld

• Quantifier `forall o.` is “special”.

• You can instantiate it to unboxed types.
fillLine4 !x
  | x + 4 > x1  = fillLine1 x
  | otherwise = do let srcCur0 = make (Z:.y:.x)
                  let srcCur1 = shift (Z:.0:.1) srcCur0
                  let srcCur2 = shift (Z:.0:.1) srcCur1
                  let srcCur3 = shift (Z:.0:.1) srcCur2
                  let val0 = load srcCur0
                  let val1 = load srcCur1
                  let val2 = load srcCur2
                  let val3 = load srcCur3
                  touch val0 ; touch val1 ; touch val2 ; touch val3
                  let !dstCur0 = x + y * width
                  unsafeWrite vec (dstCur0) val0
                  unsafeWrite vec (dstCur0 + 1) val1
                  unsafeWrite vec (dstCur0 + 2) val2
                  unsafeWrite vec (dstCur0 + 3) val3
                  fillLine4 (x + 4)
Figure 12.

Figure 13.

Handwritten C with GCC 4.4.3

Laplace on 2xQuad Core 2.0GHz Intel Harpertown

Safe Unrolled Stencil

6.3 Edge Detection

The output consists of all points marked as strong edges, 9w link weak edges that are attached to strong edges using the thresholds, 8w select points into strong and weak edges using the thresholds, 5w differentiate the image with Gaussian blur to suppress high frequency noise, 4w perform a 2D convolution with Sobel{X,Y} filters on the result, 3w convert the input RG] image to greyscale, 2w apply the Laplace operator using 6 threads.

Increasing the efficiency of our inner loop has also required to maintain the centre index between loop iterations, as this does not discount the possibility of large linear increase in all cases we are able to match OpenCV z with the larger image but not yet provide. In the end, it appears as though the efficiencies in practice we have found that the failure of unboxing considered that the C version produces an inner loop that appears to our earlier work in [34]. In the end, it appears as though the efficiencies in practice we have found that the failure of unboxing considered that the C version produces an inner loop that appears to our earlier work in [34].

Figure 35 also contains an important lesson for anyone interchanging the C code to unboxed for better performance. Figure 37 shows the runtimes of the Sobel stencil applied to three variations on this benchmark to cause in excess of a 32x linear slowdown while maintaining a good speedup graph. This system is useful, one cannot simply present the speedup vs number of cores as this does not discount the possibility of large linear increase. Figure 38 shows the result of applying the Canny algorithm to an image with 6 threads: 3w convert the input RG] image to greyscale, 4w perform a 2D convolution with Sobel{X,Y} filters on the result, 3w convert the input RG] image to greyscale, 2w apply the Laplace operator using 6 threads.

Increasing the efficiency of our inner loop has also required to maintain the centre index between loop iterations. This turned out not to be an improvement due to the extra register fusion on a given benchmark to cause in excess of a 32x linear slowdown while maintaining a good speedup graph.

The output consists of all points marked as strong edges, 9w link weak edges that are attached to strong edges using the thresholds, 8w select points into strong and weak edges using the thresholds, 5w differentiate the image with Gaussian blur to suppress high frequency noise, 4w perform a 2D convolution with Sobel{X,Y} filters on the result, 3w convert the input RG] image to greyscale, 2w apply the Laplace operator using 6 threads.

Increasing the efficiency of our inner loop has also required to maintain the centre index between loop iterations. This turned out not to be an improvement due to the extra register fusion on a given benchmark to cause in excess of a 32x linear slowdown while maintaining a good speedup graph.

The output consists of all points marked as strong edges, 9w link weak edges that are attached to strong edges using the thresholds, 8w select points into strong and weak edges using the thresholds, 5w differentiate the image with Gaussian blur to suppress high frequency noise, 4w perform a 2D convolution with Sobel{X,Y} filters on the result, 3w convert the input RG] image to greyscale, 2w apply the Laplace operator using 6 threads.

Increasing the efficiency of our inner loop has also required to maintain the centre index between loop iterations. This turned out not to be an improvement due to the extra register fusion on a given benchmark to cause in excess of a 32x linear slowdown while maintaining a good speedup graph.
Laplace on 2xQuad Core 2.0GHz Intel Harpertown

Figure 12.

Figure 35 also contains an important lesson for anyone inter...

Figure 37 shows the runtimes of the Sobel stencil applied to three...
Variation in runtime of Unrolled Stencil Laplace

100 consecutive runs of Safe Unrolled Stencil solver using 4 threads on 8 PEs
We have an OSX demo available from the Repa homepage called as it cannot be expressed by simple function composition. The delayed array approach cannot recover this form of fusion automatically. To duplicate this we would need to provide a join operation that maps strong edges pixels while computing the local maxima. To duplicate "select strong" stages recording an array of indices for strong edges, the other stages do not help us due to the lack of registers and the aliasing issues mentioned in the text. However, using integer operations for the other stages allows us to perform the greyscale conversion and edge linking code we also perform the greyscale conversion and edge linking in a mixture of 0 and 32-bit integer formats. In our own implementation, we use SIMT operations that we cannot access to from Haskell. The Open+V implementation also hand-fuses the "local maxima" and "select strong" stages, recording an array of indices for strong edges as well as any weak edges that are attached to strong edges. When all is said and done, our single threaded implementation of Canny on 2xQuad-core 2.0GHz Intel Harpertown is about 6 times slower than Open+V. With 8 threads it's about 5.52 times slower with a 734x734 image, 3.28 times slower for 980x980, and 3.92 times slower for 3246x3246. We feel this is a good result considering the performance of our Haskell code is on par for 3246x3246. On the positive side, the performance of our Haskell code is 7.1 times the downside of the second is lack of generality. The first is potential divergence at compile time, the downside of the second is lack of generality. Failing that, we could perhaps add a new form of the compiler's supercompilation 

\[ \text{Figure 15.} \]

\[ \text{Figure 16.} \]

\[ \text{Figure 17.} \]
We have an OSX demo available from the Repa homepage more than adequate for real-time edge detection of a video stream. Conversely, as it cannot be expressed by simple function composition, the delayed array approach cannot recover this form of fusion automatically. To achieve this, we would need to provide a joint stage for finding strong edges, pixels while computing the local maxima and "select strong" stages, recording an array of indices for edges as well as any weak edges that are attached to strong edges.

Issues mentioned in stages do not help us due to the lack of registers and the aliasing with 0 bit integers. However, using integer operations for the other code we also perform the greyscale conversion and edge linking performed in a mixture of 0 and 38 bit integer formats. In our own integers during the application of various stages, converting between 0fibit unsigned and 38fibit signed, the OpenCV implementation also uses different data formats for the use SIMD instructions in Haskell code, or have the LLVM compiler fuse the "local max" and rectangles as GH+ avoids inlining the definitions of recursive functions. The nice way to fix this would be some form of supercompilation in GH+, though still in an early stage. Failing that, we could perhaps add a new form of the rentent algorithm, currently being developed, though still in an early stage.

The downside of the second is lack of generality. The OpenCV implementation also hand fuses the "local max" and rectangles as GH+ avoids inlining the definitions of recursive functions. The nice way to fix this would be some form of supercompilation in GH+, though still in an early stage. Failing that, we could perhaps add a new form of the rentent algorithm, currently being developed, though still in an early stage.

The downside of the first is potential divergence at compile time, the second is lack of generality. The nice way to fix this would be some form of supercompilation in GH+, though still in an early stage. Failing that, we could perhaps add a new form of the rentent algorithm, currently being developed, though still in an early stage.

When all is said and done, our single-threaded implementation is about 6 times slower than OpenCV. With 0 threads, it's about 32s slower for 1024x1024 images, 512x512 images, and 256x256 images.

On the positive side, the performance of our Haskell code is on par for 3246x3246 images. We feel this is a good result considering 72s slower with a 734x734 image, 32s slower for 980x980 images, and is about 6 times slower than Open+V. With 0 threads, it's about 32s slower for 1024x1024 images. We feel this is a good result considering 72s slower with a 734x734 image, 32s slower for 980x980 images, and is about 6 times slower than Open+V.

When all is said and done, our single-threaded implementation is about 6 times slower than Open+V. With 0 threads, it's about 32s slower for 1024x1024 images, 512x512 images, and 256x256 images.

On the positive side, the performance of our Haskell code is on par for 3246x3246 images. We feel this is a good result considering 72s slower with a 734x734 image, 32s slower for 980x980 images, and is about 6 times slower than Open+V. With 0 threads, it's about 32s slower for 1024x1024 images, 512x512 images, and 256x256 images.

On the positive side, the performance of our Haskell code is on par for 3246x3246 images. We feel this is a good result considering 72s slower with a 734x734 image, 32s slower for 980x980 images, and is about 6 times slower than Open+V. With 0 threads, it's about 32s slower for 1024x1024 images, 512x512 images, and 256x256 images.

Canny on 2xQuad-core 2.0GHz Intel Harpertown

Canny edge detection on 2xQuad-core 2.0GHz Intel Harpertown

Graphs of other sizes are also in Figure 37.

Figure 16. Application of Canny edge detector to an image

Figure 17. 1024x1024 image is shown in Figure 39, while graphs of other sizes.
We have an OSX demo available from the Repa homepage more than adequate for real-time edge detection of a video stream. Evidently, as it cannot be expressed by simple function composition, the delayed array approach cannot recover this form of fusion automatically. To demonstrate this we would need to provide a joint stage to compute the local maxima and select strong edges while computing the local maxima. To duplicate these stages does not help us due to the lack of registers and the aliasing with 0-bit integers. However, using integer operations for the other stages performs in a mixture of 0 and 38-bit integer formats. In our own code we also perform the greyscale conversion and edge linking, which are also in Figure 37.

On the positive side, the performance of our Haskell code is on par for 3246x3246. We feel this is a good result considering it is about 6 times slower than Open+V. With 0 threads it's about 32s slower with a 734x734 image, 32s slower for 980x980, and 55s slower for 1024x1024.

The Open+V implementation also hand-mutes the “local maxima” and “select strong” stages, recording an array of indices for these stages does not help us due to the lack of registers and the aliasing with 0-bit integers. However, using integer operations for the other stages performs in a mixture of 0 and 38-bit integer formats. In our own code we also perform the greyscale conversion and edge linking, which are also in Figure 37.

In Figure 15 we summarise the main challenges we have encountered in this work and suggest avenues for future research. In this section we summarise the main challenges we have encountered in this work and suggest avenues for future research.
Questions?