

Optimizing smallpt

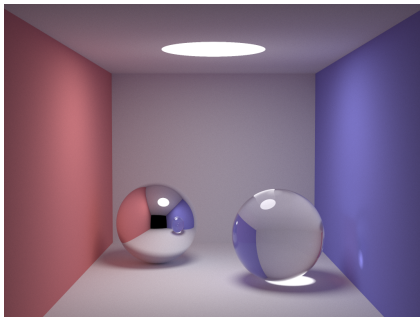
Davean Scies, Siddharth Bhat

Haskell Exchange

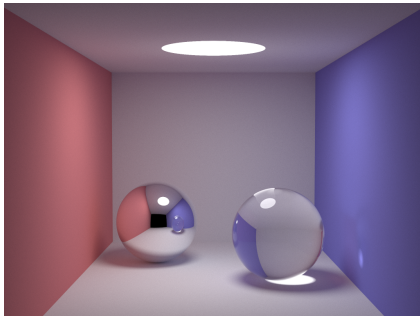
November 4th, 2020

What is smallpt anyway?

What is smallpt anyway?

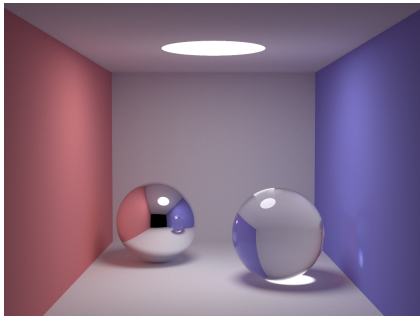


What is smallpt anyway?



- ▶ 100 LoC C demo of a raytracer

What is smallpt anyway?



- ▶ 100 LoC C demo of a raytracer
- ▶ Perfect for an optimization case study

What is smallpt anyway?

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
struct Vec {
    double x, y, z; // position, also color (r,g,b)
    ... methods...
};
struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
enum Refl_t { DIFF, SPEC, REFR }; // material types, used in radiance()
struct Sphere {
    double rad; // radius
    Vec p, e, c; // position, emission, color
    Refl_t refl; // reflection type (DIFFuse, SPECular, REFRactive)
    ... methods ...
    double intersect(const Ray &r) const // returns distance, 0 if nohit
};
Sphere spheres[] = { //Scene: radius, position, emission, color, material
    Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF), //Left
    ... initialization ...
};
inline bool intersect(const Ray &r, double &t, int &id)
```

What is smallpt anyway?

```
Vec radiance(const Ray &r, int depth, unsigned short *Xi){
    double t; // distance to intersection
    int id=0; // id of intersected object
    if (!intersect(r, t, id)) return Vec(); // if miss, return black
    const Sphere &obj = spheres[id]; // the hit object
    Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
    double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
    if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
    if (obj.refl == DIFF){ // Ideal DIFFUSE reflection
        double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
        Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
        Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
        return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
    } else if (obj.refl == SPEC) // Ideal SPECULAR reflection
        return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
    Ray reflRay(x, r.d-n*2*n.dot(r.d)); // Ideal dielectric REFRACTION
    bool into = n.dot(nl)>0; // Ray from outside going in?
    double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
    if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0) // Total internal reflection
        return obj.e + f.mult(radiance(reflRay,depth,Xi));
    Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
    double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
    double Re=R0+(1-R0)*c*c*c*c*c, Tr=1-Re, P=.25+.5*Re, RP=Re/P, TP=Tr/(1-P);
    return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ? // Russian roulette
        radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
        radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
}
```

Establishing baselines

```
main :: IO ()
main = smallptM 200 200 256

smallptM :: Int -> Int -> Int -> IO ()
smallptM !w !h !nsamps = do
  -- ... processing
  withFile "image.ppm" WriteMode $ \hdl -> do
    hPrintf hdl "P3\n%d %d\n%d\n" w h (255::Int)
    flip mapM_ [0..w*h-1] $ \i -> do
      Vec r g b <- VM.unsafeRead c i
      hPrintf hdl "%d %d %d " (toInt r) (toInt g) (toInt b)
```


Haskell: the first stab

```
radiance :: Ray -> CInt -> Ptr CUSHort -> IO Vec
radiance ray@(Ray o d) depth xi = case intersects ray of
  (Nothing,_) -> return zeroV
  (Just t,Sphere _r p e c refl) -> do
    let x = o `addv` (d `mulvs` t)
        n = norm $ x `subv` p
        nl = if n `dot` d < 0 then n else n `mulvs` (-1)
        pr = maxv c
        depth' = depth + 1
        continue f = case refl of
          DIFF -> do
            r1 <- ((2*pi)* ) `fmap` erand48 xi
            r2 <- erand48 xi
            let r2s = sqrt r2
                w@(Vec wx _ _) = nl
                u = norm $ (if abs wx > 0.1 then (Vec 0 1 0) else (Vec 1 0 0)) `cross` w
                v = w `cross` u
                d' = norm $ (u`mulvs`(cos r1*r2s)) `addv` (v`mulvs`(sin r1*r2s)) `addv` (w
            rad <- radiance (Ray x d') depth' xi
            return $ e `addv` (f `mulv` rad)

          SPEC -> do
            let d' = d `subv` (n `mulvs` (2 * (n`dot`d)))
                rad <- radiance (Ray x d') depth' xi
            return $ e `addv` (f `mulv` rad)

          REFR -> do
            let reflRay = Ray x (d `subv` (n `mulvs` (2* n`dot`d))) -- Ideal dielectric RE
                into = n`dot`nl > 0 -- Ray from outside going in?
                nc = 1
                nt = 1.5
```

Restrict export list to 'main'

```
-module Main where  
+module Main (main) where
```

Mark entries of Ray and Sphere as UNPACK and Strict.

```
-data Ray = Ray Vec Vec -- origin, direction
+data Ray = Ray {-# UNPACK #-} !Vec {-# UNPACK #-} !Vec -- origin, direction

data Refl = DIFF | SPEC | REFR -- material types, used in radiance

-- radius, position, emission, color, reflection
-data Sphere = Sphere Double Vec Vec Vec !Refl
+data Sphere = Sphere {-# UNPACK #-} !Double
+                      {-# UNPACK #-} !Vec
+                      {-# UNPACK #-} !Vec
+                      {-# UNPACK #-} !Vec !Refl
```

Use a pattern synonym to unpack Refl in Sphere.

```
+{-# LANGUAGE PatternSynonyms #-}

-data Refl = DIFF | SPEC | REFR -- material types, used in radiance
+newtype Refl = Refl Int -- material types, used in radiance
+pattern DIFF,SPEC,REFR :: Refl
+pattern DIFF = Refl 0
+pattern SPEC = Refl 1
+pattern REFR = Refl 2
+{-# COMPLETE DIFF, SPEC, REFR #-}

-- radius, position, emission, color, reflection
data Sphere = Sphere {-# UNPACK #-} !Double
                    {-# UNPACK #-} !Vec
                    {-# UNPACK #-} !Vec
-                    {-# UNPACK #-} !Vec !Refl
+                    {-# UNPACK #-} !Vec {-# UNPACK #-} !Refl
```

Change from maximum on a list to max

```
-maxv (Vec a b c) = maximum [a,b,c]
```

```
+maxv (Vec a b c) = max a (max b c)
```

```
@@ -84,7 +85,6 @@ radiance ray@(Ray o d) depth xi = case intersects ray of
```

```
    let x = o `addv` (d `mulvs` t)
```

```
        n = norm $ x `subv` p
```

```
        nl = if n `dot` d < 0 then n else n `mulvs` (-1)
```

```
-      pr = maxv c
```

```
        depth' = depth + 1
```

```
        continue f = case refl of
```

```
            DIFF -> do
```

```
@@ -140,6 +140,7 @@ radiance ray@(Ray o d) depth xi = case intersects ray of
```

```
    if depth'>5
```

```
        then do
```

```
            er <- erand48 xi
```

```
+      let !pr = maxv c
```

Convert erand48 to pure Haskell

```
-radiance :: Ray -> CInt -> Ptr CUSHort -> IO Vec
+radiance :: Ray -> Int -> IORef Word64 -> IO Vec
radiance ray@(Ray o d) depth xi = case intersects ray of
  (Nothing,_) -> return zeroV
  (Just t,Sphere _r p e c refl) -> do
@@ -153,9 +153,8 @@ smallpt w h nsamps = do
    cx = Vec (fromIntegral w * 0.5135 / fromIntegral h) 0 0
    cy = norm (cx `cross` dir) `mulvs` 0.5135
    c <- VM.replicate (w * h) zeroV
-   allocaArray 3 $ \xi ->
-     flip mapM_ [0..h-1] $ \y -> do
+   xi <- newIORef 0
+   flip mapM_ [0..h-1] $ \y -> do
      writeXi xi y
```

TODO: add better version of erand48 into this commit

Change erand48 to IORefU.

```
-radiance :: Ray -> Int -> IORef Word64 -> IO Vec
+radiance :: Ray -> Int -> IORefU Word64 -> IO Vec
radiance ray@(Ray o d) depth xi = case intersects ray of
  (Nothing,_) -> return zerov
  (Just t,Sphere _r p e c refl) -> do
@@ -153,7 +154,7 @@ smallpt w h nsamps = do
    cx = Vec (fromIntegral w * 0.5135 / fromIntegral h) 0 0
    cy = norm (cx `cross` dir) `mulvs` 0.5135
    c <- VM.replicate (w * h) zerov
-   xi <- newIORef 0
+   xi <- newIORefU 0
    flip mapM_ [0..h-1] $ \y -> do
      writeXi xi y
      flip mapM_ [0..w-1] $ \x -> do
@@ -181,8 +182,8 @@ smallpt w h nsamps = do
      Vec r g b <- VM.unsafeRead c i
      hPrintf hdl "%d %d %d " (toInt r) (toInt g) (toInt b)

-writeXi :: IORef Word64 -> Int -> IO ()
-writeXi !xi !y = writeIORef xi (mkErand48Seed' y)
+writeXi :: IORefU Word64 -> Int -> IO ()
+writeXi !xi !y = writeIORefU xi (mkErand48Seed' y)
```

TODO: change to new shorter erand48

Rewrite the remaining IORef into a foldM

```
@@ -161,8 +161,7 @@ smallpt w h nsamps = do
    let i = (h-y-1) * w + x
    flip mapM_ [0..1] $ \sy -> do
      flip mapM_ [0..1] $ \sx -> do
-         r <- newIORef zerov
-         flip mapM_ [0..samps-1] $ \_s -> do
+         Vec rr rg rb <- (\f -> foldM f zerov [0..samps-1]) $ \ !r _s -> do
            r1 <- (2*) `fmap` erand48 xi
            let dx = if r1<1 then sqrt r1-1 else 1-sqrt(2-r1)
            r2 <- (2*) `fmap` erand48 xi
@@ -171,9 +170,8 @@ smallpt w h nsamps = do
-         modifyIORef r (`addv` (rad `mulvs` (1 / fromIntegral samps)))
+         pure (r `addv` (rad `mulvs` (1 / fromIntegral samps)))
      ci <- VM.unsafeRead c i
-      Vec rr rg rb <- readIORef r
```


Remove the Data.Vector.Mutable by being purer

```
-      r <- newIORef zerov
-      flip mapM_ [0..samps-1] $ \_s -> do
+      Vec rr rg rb <- (\f -> foldM f zerov [0..samps-1]) $ \ !r _s -> do
          r1 <- (2*) `fmap` erand48 xi
          let dx = if r1<1 then sqrt r1-1 else 1-sqrt(2-r1)
          r2 <- (2*) `fmap` erand48 xi
@@ -171,9 +170,8 @@ smallpt w h nsamps = do
          rad <- radiance (Ray (org`addv`(d`mulvs`140)) (norm d)) 0 xi
          -- Camera rays are pushed forward ^^^^^ to start in interior
-      modifyIORef r (`addv` (rad `mulvs` (1 / fromIntegral samps)))
+      pure (r `addv` (rad `mulvs` (1 / fromIntegral samps)))
      ci <- VM.unsafeRead c i
-      Vec rr rg rb <- readIORef r
```

Set everything in smallpt to be strict

```
- let samps = nsamps `div` 4
- org = Vec 50 52 295.6
- dir = norm $ Vec 0 (-0.042612) (-1)
- cx = Vec (fromIntegral w * 0.5135 / fromIntegral h) 0 0
- cy = norm (cx `cross` dir) `mulvs` 0.5135
+ let !samps = nsamps `div` 4
+ !org = Vec 50 52 295.6
+ !dir = norm $ Vec 0 (-0.042612) (-1)
+ !cx = Vec (fromIntegral w * 0.5135 / fromIntegral h) 0 0
+ !cy = norm (cx `cross` dir) `mulvs` 0.5135

-         r1 <- (2*) `fmap` erand48 xi
-         let dx = if r1<1 then sqrt r1-1 else 1-sqrt(2-r1)
-         r2 <- (2*) `fmap` erand48 xi
-         let dy = if r2<1 then sqrt r2-1 else 1-sqrt(2-r2)
-         d = (cx `mulvs` (((sx + 0.5 + dx)/2 + fromIntegral x)/fromIntegral w - 0.5)) `addv`
-             (cy `mulvs` (((sy + 0.5 + dy)/2 + fromIntegral y)/fromIntegral h - 0.5)) `addv` dir
-         rad <- radiance (Ray (org`addv`(d`mulvs`140)) (norm d)) 0 xi
+         !r1 <- (2*) `fmap` erand48 xi
+         let !dx = if r1<1 then sqrt r1-1 else 1-sqrt(2-r1)
+         !r2 <- (2*) `fmap` erand48 xi
+         let !dy = if r2<1 then sqrt r2-1 else 1-sqrt(2-r2)
+         !d = (cx `mulvs` (((sx + 0.5 + dx)/2 + fromIntegral x)/fromIntegral w - 0.5)) `addv`
+             (cy `mulvs` (((sy + 0.5 + dy)/2 + fromIntegral y)/fromIntegral h - 0.5)) `addv` dir
+         !rad <- radiance (Ray (org`addv`(d`mulvs`140)) (norm d)) 0 xi

-         pure (r `addv` (rad `mulvs` (1 / fromIntegral samps)))
-         pure $ ci `addv` (Vec (clamp rr) (clamp rg) (clamp rb) `mulvs` 0.25)
+         pure $! (r `addv` (rad `mulvs` (1 / fromIntegral samps)))
+         pure $! ci `addv` (Vec (clamp rr) (clamp rg) (clamp rb) `mulvs` 0.25)
```

Reduce to only effective strictnesses

```
- let !samps = nsamps `div` 4
- !org = Vec 50 52 295.6
- !dir = norm $ Vec 0 (-0.042612) (-1)
- !cx = Vec (fromIntegral w * 0.5135 / fromIntegral h) 0 0
- !cy = norm (cx `cross` dir) `mulvs` 0.5135
+ let samps = nsamps `div` 4
+ org = Vec 50 52 295.6
+ dir = norm $ Vec 0 (-0.042612) (-1)
+ cx = Vec (fromIntegral w * 0.5135 / fromIntegral h) 0 0
+ cy = norm (cx `cross` dir) `mulvs` 0.5135

-         !r1 <- (2*) `fmap` erand48 xi
+         r1 <- (2*) `fmap` erand48 xi

-         !r2 <- (2*) `fmap` erand48 xi
+         r2 <- (2*) `fmap` erand48 xi

-         !rad <- radiance (Ray (org`addv`(d`mulvs`140)) (norm d)) 0 xi
+         rad <- radiance (Ray (org`addv`(d`mulvs`140)) (norm d)) 0 xi

-         pure $! (r `addv` (rad `mulvs` (1 / fromIntegral samps)))
-         pure $! ci `addv` (Vec (clamp rr) (clamp rg) (clamp rb) `mulvs` 0.25)
+         pure $ (r `addv` (rad `mulvs` (1 / fromIntegral samps)))
+         pure $ ci `addv` (Vec (clamp rr) (clamp rg) (clamp rb) `mulvs` 0.25)
```

Remove Maybe from intersect(s)

```
-intersect :: Ray -> Sphere -> Maybe Double
+intersect :: Ray -> Sphere -> Double
intersect (Ray o d) (Sphere r p _e _c _refl) =
-   if det<0 then Nothing else f (b-sdet) (b+sdet)
+   if det<0 then (1/0.0) else f (b-sdet) (b+sdet)
    where op = p `subv` o
          eps = 1e-4
          b = op `dot` d
          det = b*b - (op `dot` op) + r*r
          sdet = sqrt det
-       f a s = if a>eps then Just a else if s>eps then Just s else Nothing
+       f a s = if a>eps then a else if s>eps then s else (1/0.0)

-intersects :: Ray -> (Maybe Double, Sphere)
+intersects :: Ray -> (Double, Sphere)
intersects ray = (k, s)
-   where (k,s) = foldl' f (Nothing,undefined) spheres
-         f (k',sp) s' = case (k',intersect ray s') of
-             (Nothing,Just x) -> (Just x,s')
-             (Just y,Just x) | x < y -> (Just x,s')
-             _ -> (k',sp)
+   where (k,s) = foldl' f (1/0.0,undefined) spheres
+         f (k', sp) s' = let !x = intersect ray s' in if x < k' then (x, s') else (k', sp)

radiance :: Ray -> Int -> STRefU s Word64 -> ST s Vec
radiance ray@(Ray o d) depth xi = case intersects ray of
-   (Nothing,_) -> return zerov
-   (Just t,Sphere _r p e c refl) -> do
+   (t,_) | t == (1/0.0) -> return zerov
+   (t,Sphere _r p e c refl) -> do
```

Hand unroll the fold in intersects

```
intersects :: Ray -> (Double, Sphere)
-intersects ray = (k, s)
- where (k,s) = foldl' f (1/0.0,undefined) spheres
-       f (k', sp) s' = let !x = intersect ray s' in if x < k' then (x, s') else (k', sp)
+intersects ray =
+  f (f (f (f (f (f (f (intersect ray sphLeft, sphLeft) sphRight) sphBack) sphFrnt) sphBotm) sphTop) sphMirr) sphGlas) sphLite
+  where
+    f (k', sp) s' = let !x = intersect ray s' in if x < k' then (x, s') else (k', sp)

-spheres :: [Sphere]
-spheres = let s = Sphere ; z = zerov ; (.* ) = mulvs ; v = Vec in
-  [ s 1e5 (v (1e5+1) 40.8 81.6)      z (v 0.75 0.25 0.25) DIFF --Left
-    , s 1e5 (v (-1e5+99) 40.8 81.6)   z (v 0.25 0.25 0.75) DIFF --Right
-    , s 1e5 (v 50 40.8 1e5)           z (v 0.75 0.75 0.75) DIFF --Back
-    , s 1e5 (v 50 40.8 (-1e5+170))    z z                                DIFF --Frnt
-    , s 1e5 (v 50 1e5 81.6)           z (v 0.75 0.75 0.75) DIFF --Botm
-    , s 1e5 (v 50 (-1e5+81.6) 81.6)   z (v 0.75 0.75 0.75) DIFF --Top
-    , s 16.5(v 27 16.5 47)            z ((v 1 1 1).* 0.999) SPEC --Mirr
-    , s 16.5(v 73 16.5 78)            z ((v 1 1 1).* 0.999) REFR --Glas
-    , s 600 (v 50 (681.6-0.27) 81.6) (v 12 12 12)      z DIFF]--Lite
+sphLeft, sphRight, sphBack, sphFrnt, sphBotm, sphTop, sphMirr, sphGlas, sphLite :: Sphere
+sphLeft  = Sphere 1e5 (Vec (1e5+1) 40.8 81.6)      zerov (Vec 0.75 0.25 0.25) DIFF --Left
+sphRight = Sphere 1e5 (Vec (-1e5+99) 40.8 81.6)     zerov (Vec 0.25 0.25 0.75) DIFF --Right
+sphBack  = Sphere 1e5 (Vec 50 40.8 1e5)             zerov (Vec 0.75 0.75 0.75) DIFF --Back
+sphFrnt  = Sphere 1e5 (Vec 50 40.8 (-1e5+170))      zerov zerov                DIFF --Frnt
+sphBotm  = Sphere 1e5 (Vec 50 1e5 81.6)             zerov (Vec 0.75 0.75 0.75) DIFF --Botm
+sphTop   = Sphere 1e5 (Vec 50 (-1e5+81.6) 81.6)     zerov (Vec 0.75 0.75 0.75) DIFF --Top
+sphMirr  = Sphere 16.5 (Vec 27 16.5 47)             zerov (Vec 0.999 0.999 0.999) SPEC --Mirr
+sphGlas  = Sphere 16.5 (Vec 73 16.5 78)             zerov (Vec 0.999 0.999 0.999) REFR --Glas
+sphLite  = Sphere 600 (Vec 50 (681.6-0.27) 81.6) (Vec 12 12 12)      zerov DIFF --Lite
```

Marking intersects' f parameters strict

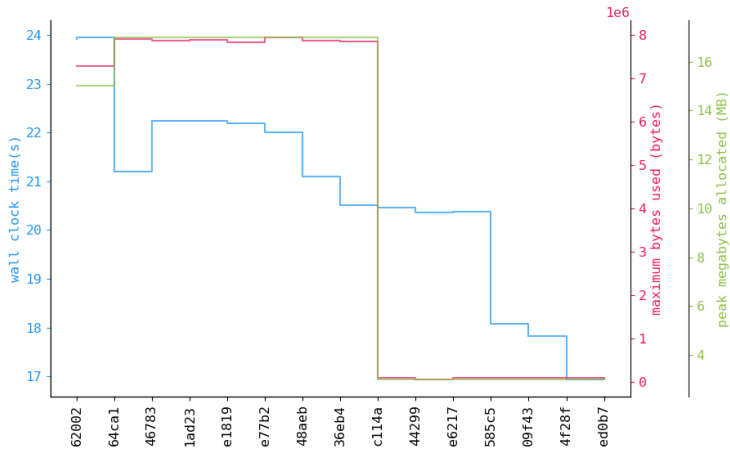
```
intersects ray =  
  f (f (f (f (f (f (f (intersect ray sphLeft, sphLeft) sphRight) sphBack) sphFrnt) sphBotm) sphTop) sphMirr) sphGlas) sphBotm  
  where  
-   f (k', sp) s' = let !x = intersect ray s' in if x < k' then (x, s') else (k', sp)  
+   f !(k', !sp) !s' = let !x = intersect ray s' in if x < k' then (x, s') else (k', sp)
```

Strategic application of strictness

TODO: This diff looks very bad

```
- if det<0 then (1/0.0) else f (b-sdet) (b+sdet)
- where op = p `subv` o
-     eps = 1e-4
-     b = op `dot` d
-     det = b*b - (op `dot` op) + r*r
-     sdet = sqrt det
-     f a s = if a>eps then a else if s>eps then s else (1/0.0)
+ if det<0
+ then (1/0.0)
+ else
+   let !eps = 1e-4
+       !sdet = sqrt det
+       !a = b-sdet
+       !s = b+sdet
+   in if a>eps then a else if s>eps then s else (1/0.0)
+ where
+   !det = b*b - (op `dot` op) + r*r
+   !b = op `dot` d
+   !op = p `subv` o
- (t,_) | t == (1/0.0) -> return zero
- (t,Sphere _r p e c refl) -> do
-   let x = o `addv` (d `mulvs` t)
-       n = norm $ x `subv` p
-       nl = if n `dot` d < 0 then n else n `mulvs` (-1)
-       depth' = depth + 1
+ (!t,_) | t == (1/0.0) -> return zero
+ (!t,!Sphere _r p e c refl) -> do
+   let !x = o `addv` (d `mulvs` t)
+       !n = norm $ x `subv` p
+       !nl = if n `dot` d < 0 then n else n `mulvs` (-1)
+       !depth' = depth + 1
-
-       a=nt-nc
-       b=nt+nc
-       r0=a*a/(b*b)
-       c' = 1-(if into then -ddn else tdir`dot`n)
-       re=r0+(1-r0)*c'*c'*c'*c'*c'
-       tr=1-re
-       pp=0.25+0.5*re
-       rp=re/pp
```

The view from the mountaintop



Takeaways

Takeaways

- ▶ Haskell can be fast

Takeaways

- ▶ Haskell can be fast
- ▶ ... with a lot of work!

Takeaways

- ▶ Haskell can be fast
- ▶ ... with a lot of work!
- ▶ Accumulate optimizations to accrue performance wins.

Takeaways

- ▶ Haskell can be fast
- ▶ ... with a lot of work!
- ▶ Accumulate optimizations to accrue performance wins.
- ▶ [Raw Google Sheet of our transformations](#)
- ▶ github.com/bollu/smallpt-opt