Academy

slimme softwareontwikkeling

Rendering distance fields

using

a systems programming language
expressive, concurrent, garbage-collected

GO

- Go is a compiled, garbage-collected, concurrent programming language developed by Google.
- Go was officially announced in November 2009 ...
- ... with implementations released for the Linux and Mac OS X platforms.

GO is ...

… simple

```
package main

import "fmt"

func main() {
  fmt.Printf("Hello, 世界\n")
}
```

… fast

Go compilers produce fast code fast. Typical builds take a fraction of a second yet the resulting programs run nearly as quickly as comparable C or C++ code.

… safe

Go is type safe and memory safe. Go has pointers but no pointer arithmetic. For random access, use slices, which know their limits.

… concurrent

Go promotes writing systems and servers as sets of lightweight communicating processes, called goroutines, with strong support from the language. Run thousands of goroutines if you want—and say good-bye to stack overflows.

… fun

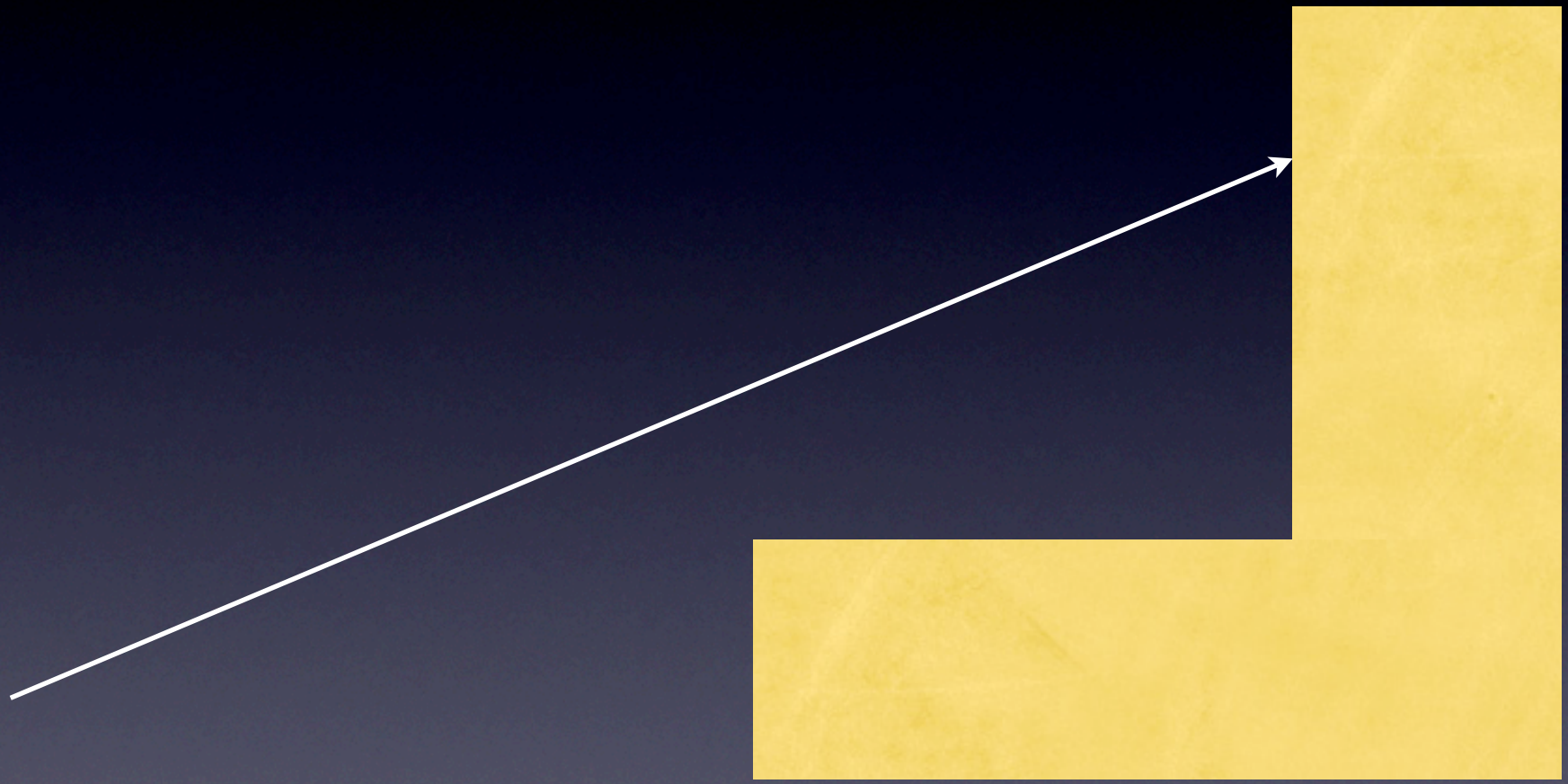
Go has fast builds, clean syntax, garbage collection, methods for any type, and run-time reflection. It feels like a dynamic language but has the speed and safety of a static language. It's a joy to use.

… open source

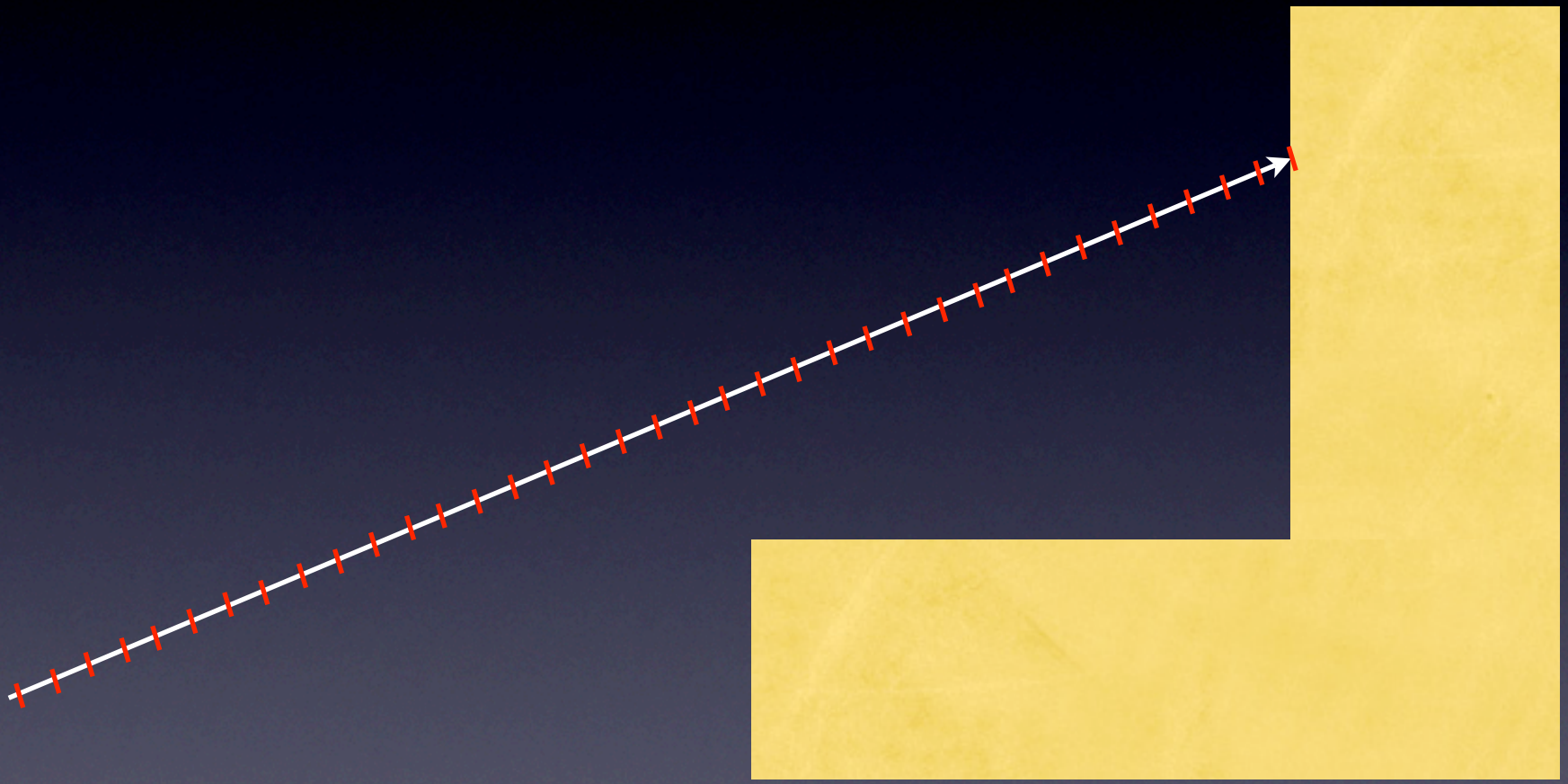
Distance fields

- Check: http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf !!

Introduction: Raymarching

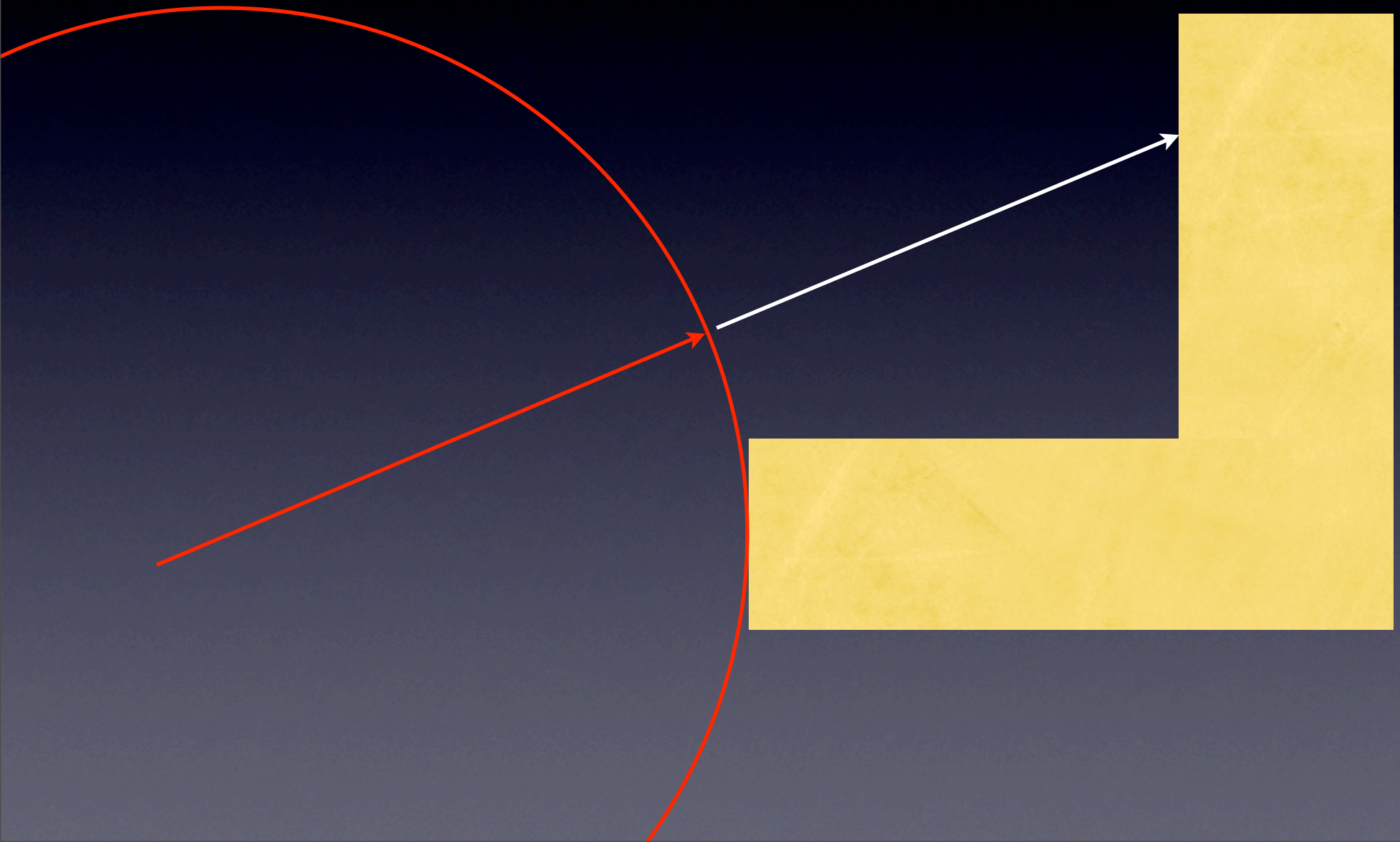


Naive implementation

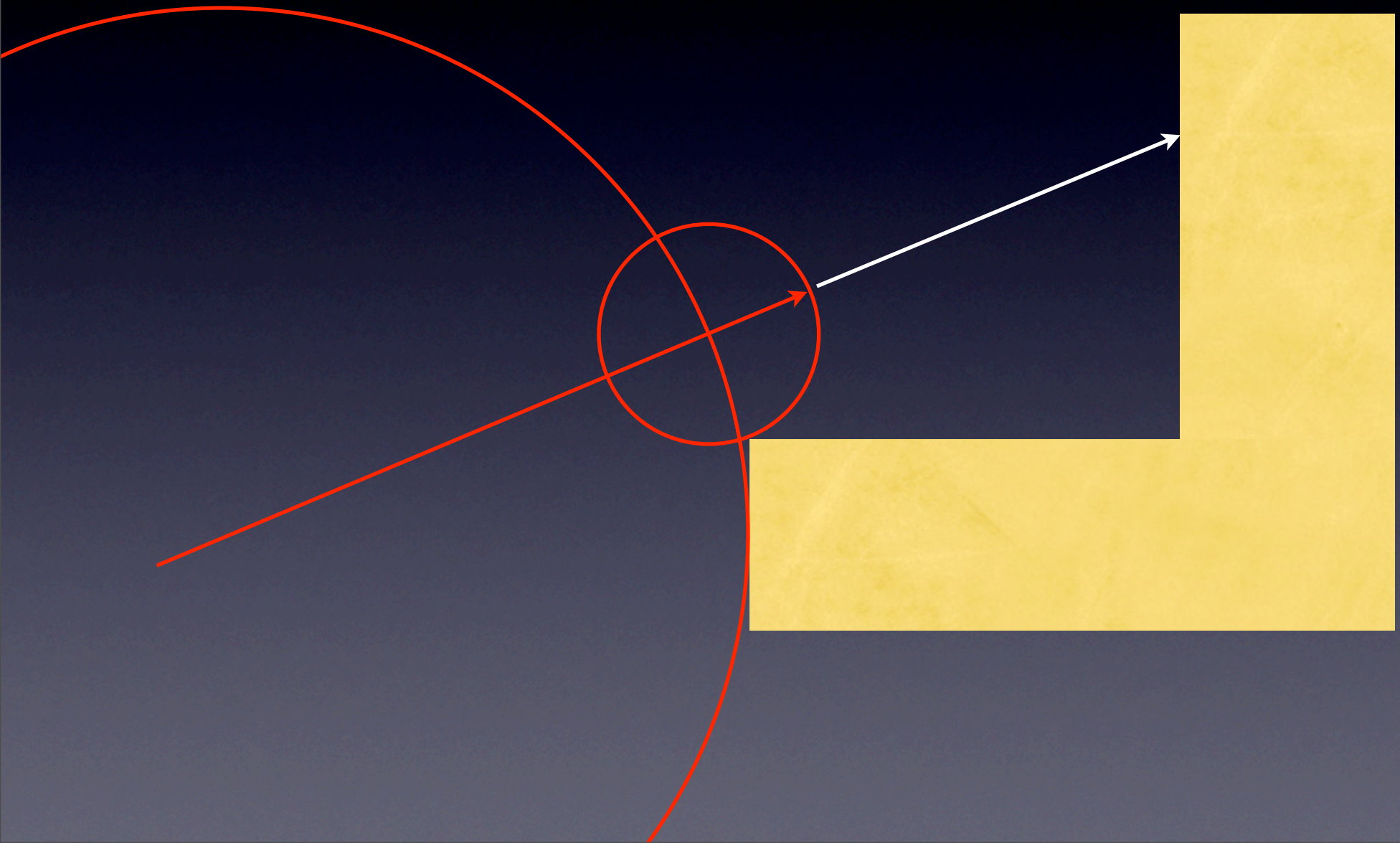


(objects without an analytic intersection function)

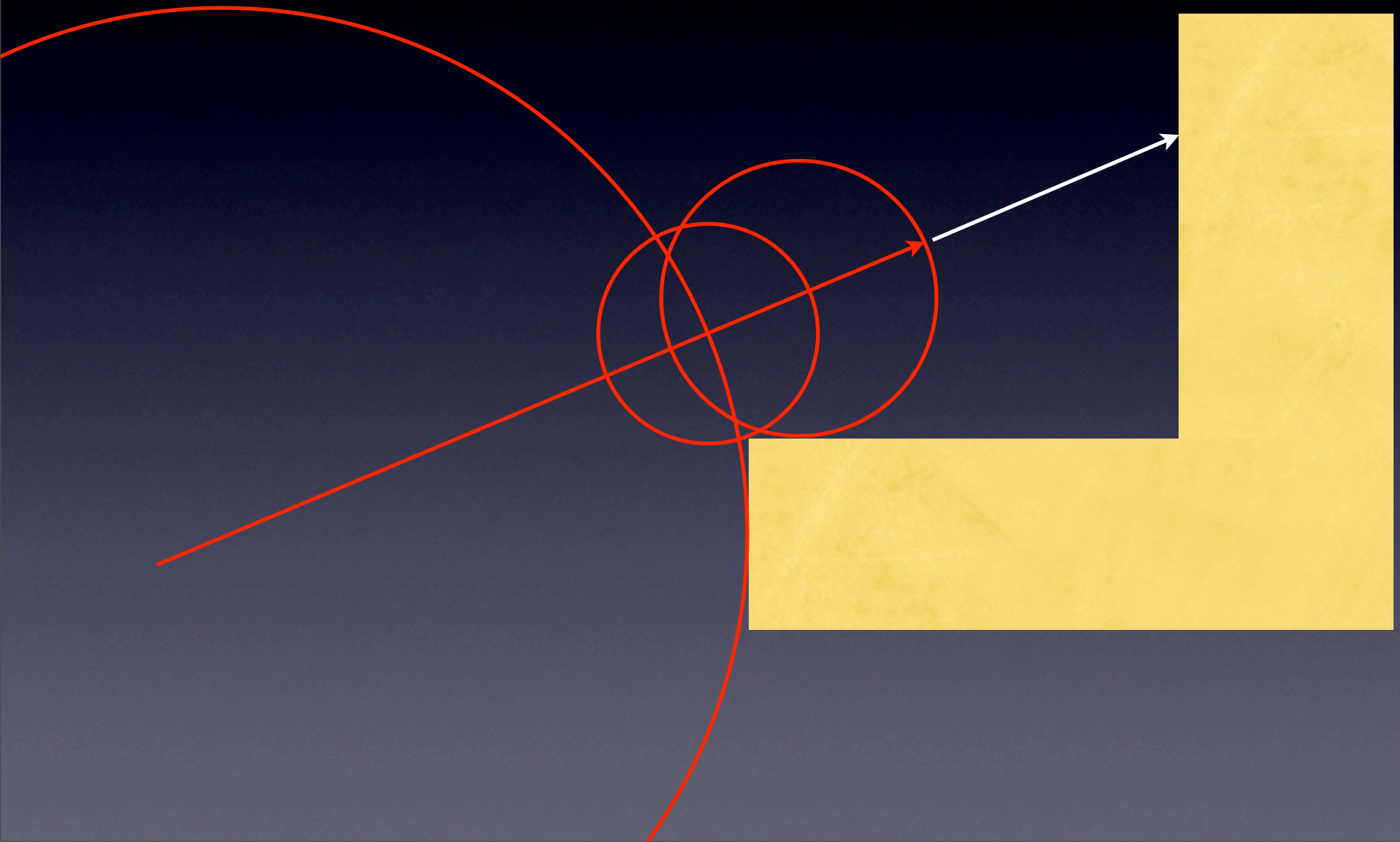
Distance field aided



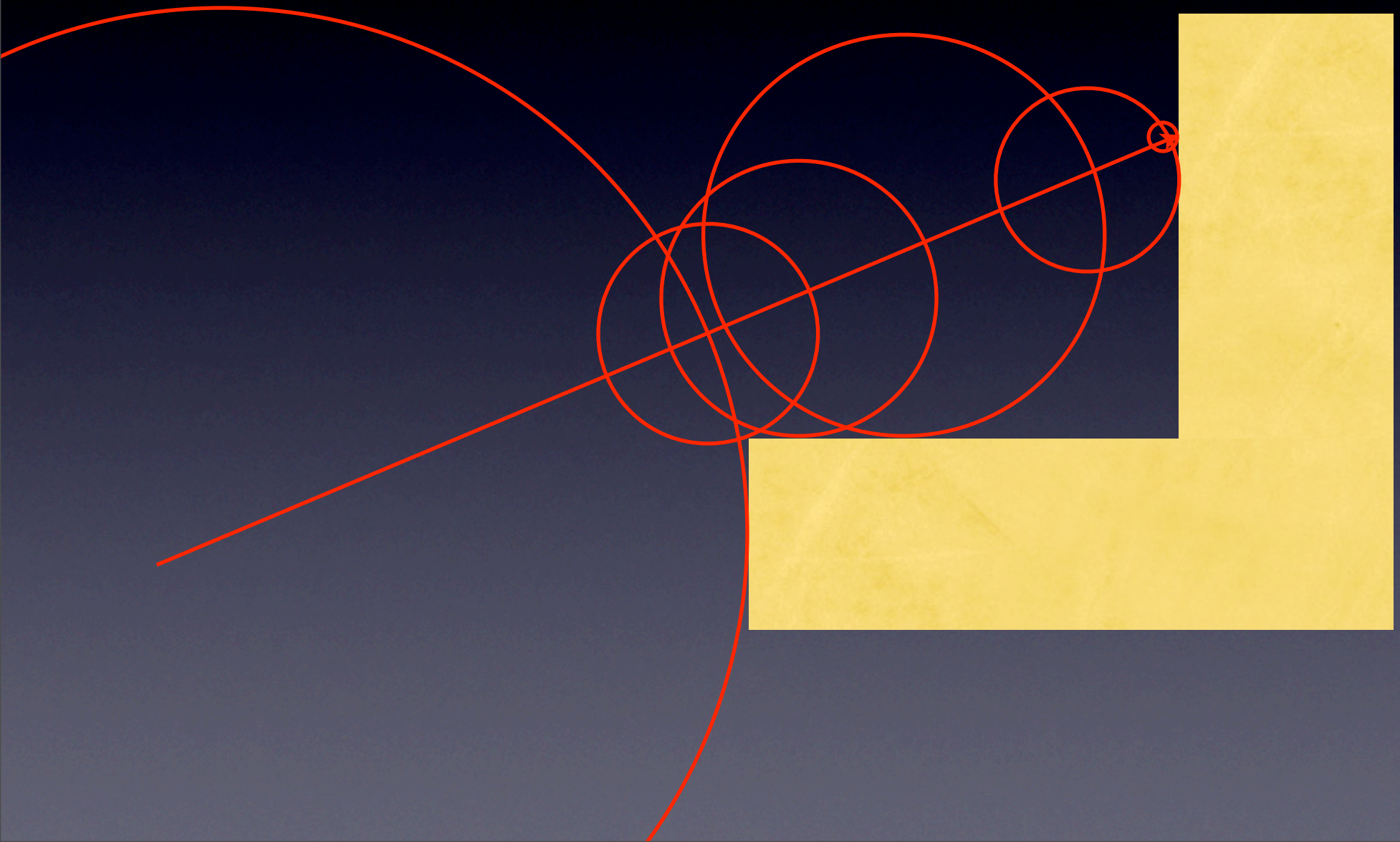
Distance field aided



Distance field aided



Distance field aided



Problem

- Constructing a distance field can be hard and needs a lot of pre-calculation.
- Can we construct and render distance fields directly?

Sphere

Distance field for sphere is easy:

```
func distanceToSphere( p Vector ) float64 {
    return p.Len() - 1.0;
}
```

(positioned in origin with radius 1.0)

Go language: backward declaration:

```
var a, b *int;
```

c/c++: int* a, b

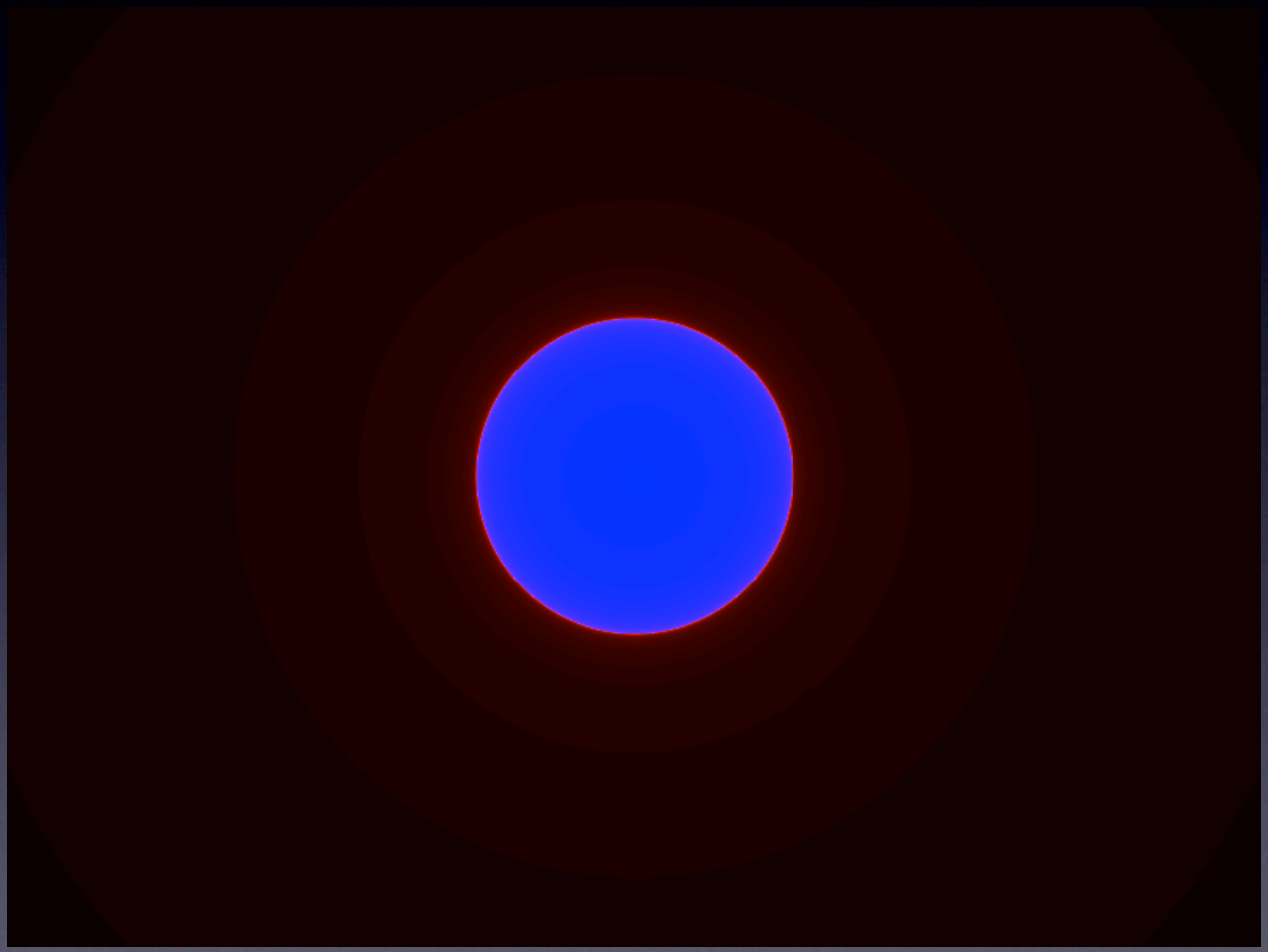
Short declaration:

```
a := 1;
p := new(foo.Foo);
```

```
var a int = 1;
var p *foo.Foo = new(foo.Foo);
```

Sphere :)

#steps in red, sphere intersection in blue



GO

Flags:

```
import (
    "flag";
)

var w = flag.Int("w", 800, "width");
var h = flag.Int("h", 600, "height");

func main() {
    flag.Parse();

    width := *w;
    height := *h;
}
```

```
reinder-nijhoffs-computer:go reinder$ ./8.out -width 600
flag provided but not defined: -width
Usage of ./8.out:
  -h=600: height
  -w=800: width
```

GO

Methods:

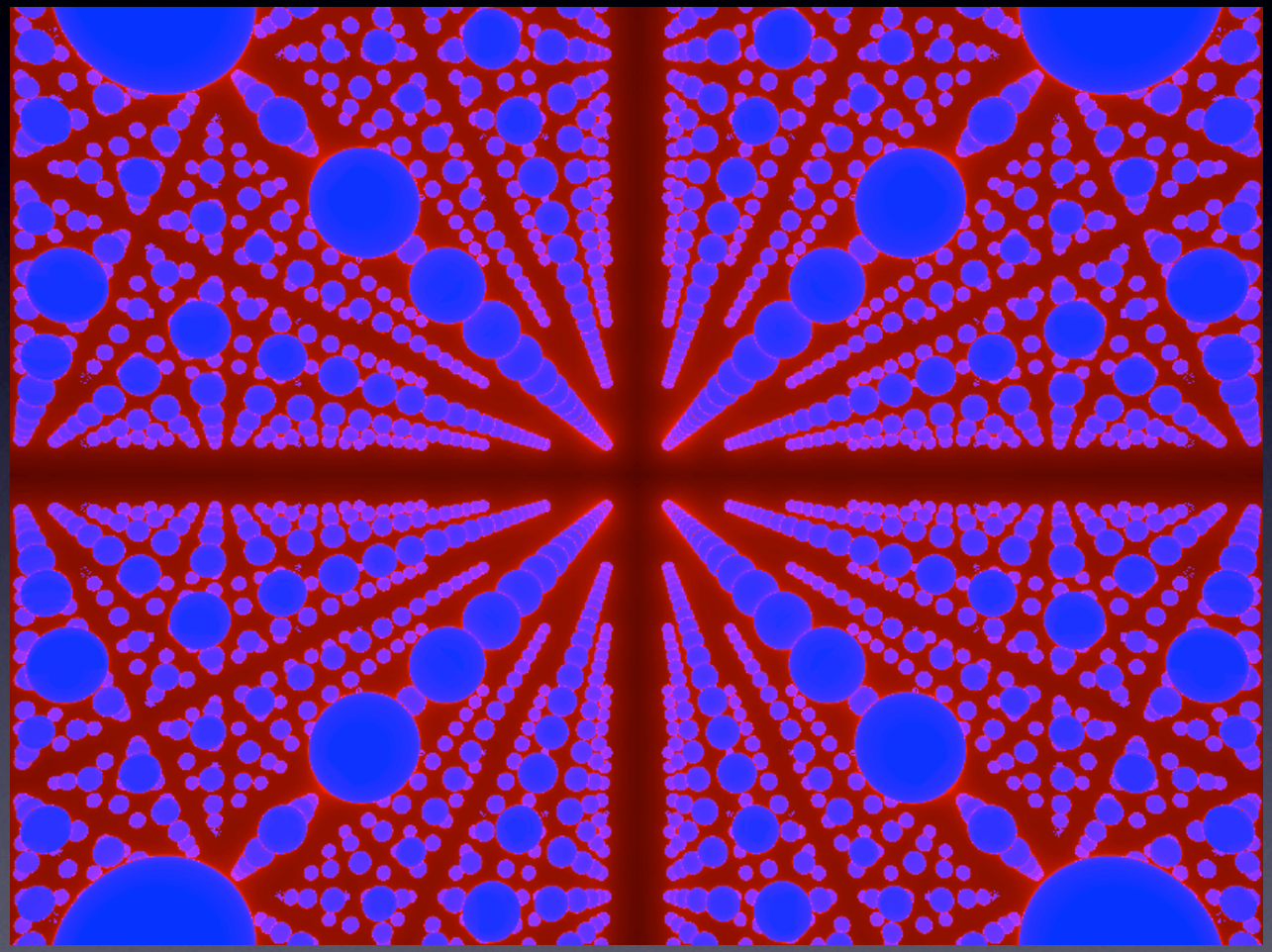
```
type Vector struct {
    X, Y, Z float64;
}

func ( p *Vector ) Len() float64 {
    return math.Sqrt( p.X*p.X + p.Y*p.Y + p.Z*p.Z );
}

func ( p *Vector ) Normalize() {
    len := p.Len();
    p.X /= len;
    p.Y /= len;
    p.Z /= len;
}
```

Preview

Transform (input) domain

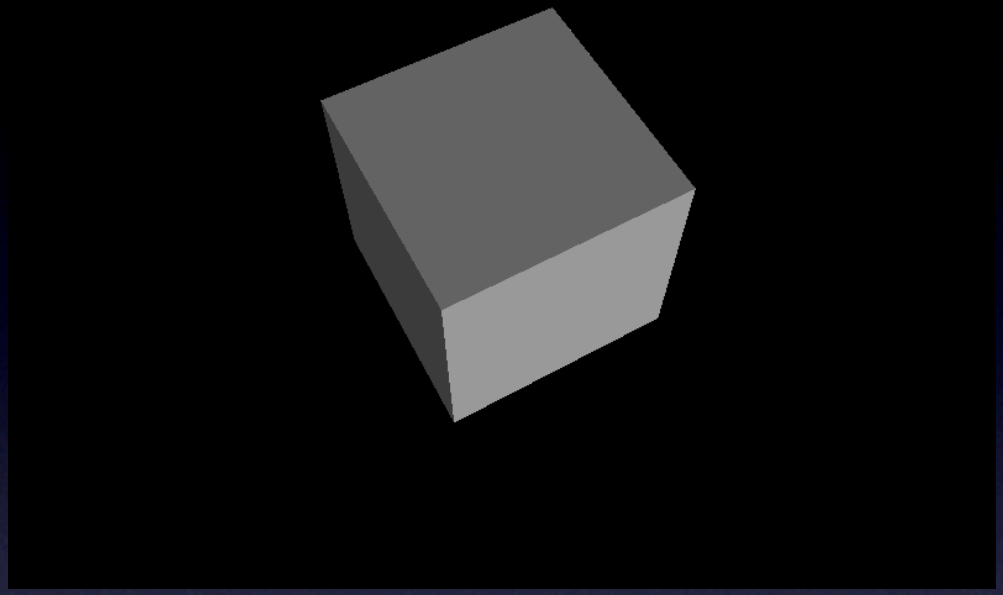


(modulo p.X, p.Y and p.Z before calling distanceToSphere(p))

Cube:

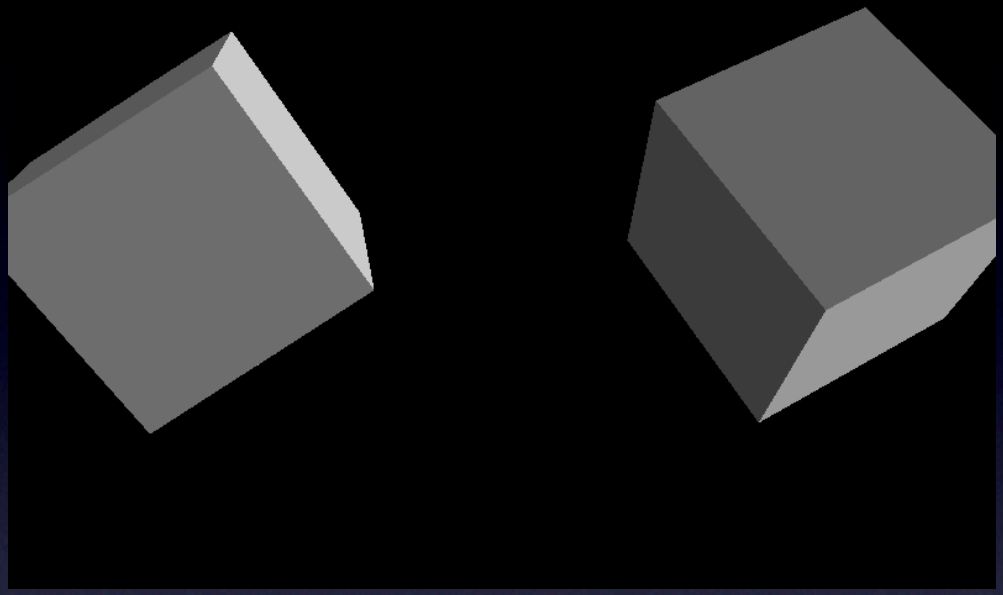
```
func distanceToCube( p Vector ) float64 {
    return max( math.Fabs( p.X ),
      max( math.Fabs( p.Y ), math.Fabs( p.Z ) ) ) -1.0;
}

func distanceField( p Vector ) float64 {
    return distanceToCube( p );
}
```



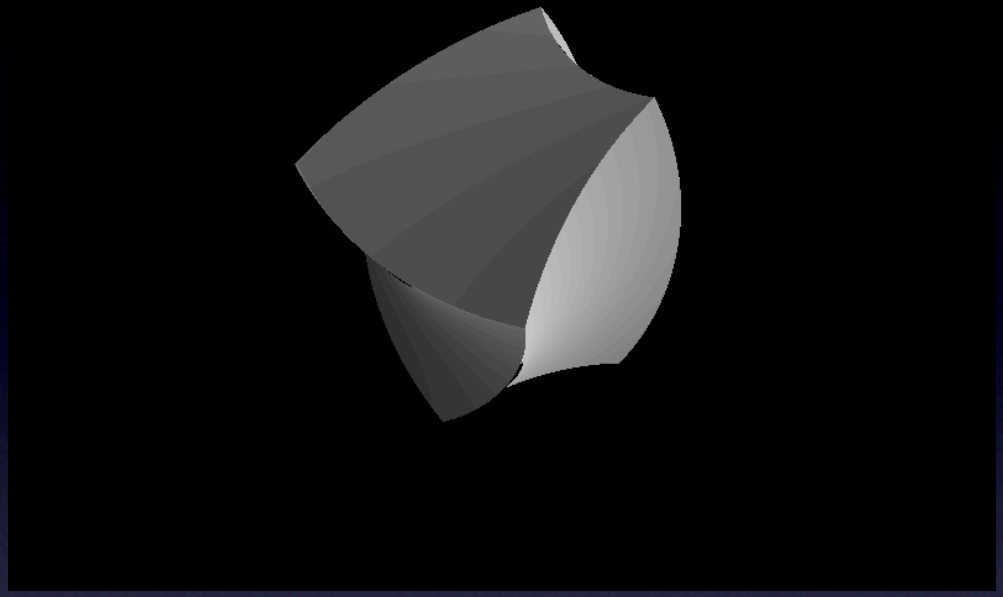
Transform the domain:

```
func distanceField( p Vector ) float64 {
    return distanceToCube( p.RotateX( 1.0 ).RotateY( 0.5 ) );
}
```



Combine two distance fields:

```
func distanceField( p Vector ) float64 {
   return min(
   distanceToCube( p.Translate( -2.5, 0.0, 0.0 ).RotateX( 1.0 ).RotateY( 0.5 ) ),
   distanceToCube( p.Translate( 2.5, 0.0, 0.0 ).RotateZ( 1.0 ).RotateY( 3.5 ) ) );
}
```



Or some other deformations:

```
func distanceField( p Vector ) float64 {
    t := p.RotateZ( p.Z / 2.0 );

    return distanceToCube(t.RotateX( 1.0 ).RotateY( 0.5 ) );
}
```

GO

Trying to write some oo-code using interfaces:

```
type Sphere struct {
    Position Vector;
    Radius float64;
}

type Cube struct {
    Position, Rotation Vector;
    Size float64;
}

type DistanceFieldObject interface {
    Distance( p Vector ) float64;
}
```

GO

Distance functions:

```
func ( s Sphere ) Distance( p Vector ) float64 {
    p = p.TranslateBy( s.Position );

    return p.Len() - s.Radius;
}

func  ( c Cube ) Distance ( p Vector ) float64 {
    p = p.TranslateBy( c.Position );
    p = p.RotateX( c.Rotation.X ).RotateY
( c.Rotation.Y ).RotateZ( c.Rotation.Z );

    return max( math.Fabs( p.X ),
     max( math.Fabs( p.Y ), math.Fabs( p.Z ) ) ) -c.Size;
}
```

GO

Init distance field:

```
field := make([]DistanceFieldObject, 60);

for i:=0; i< 30; i++ {
    field[ i ] = Cube{ ...
}

for i:=30; i< 60; i++ {
    field[ i ] = Sphere{ ..
}
```

GO

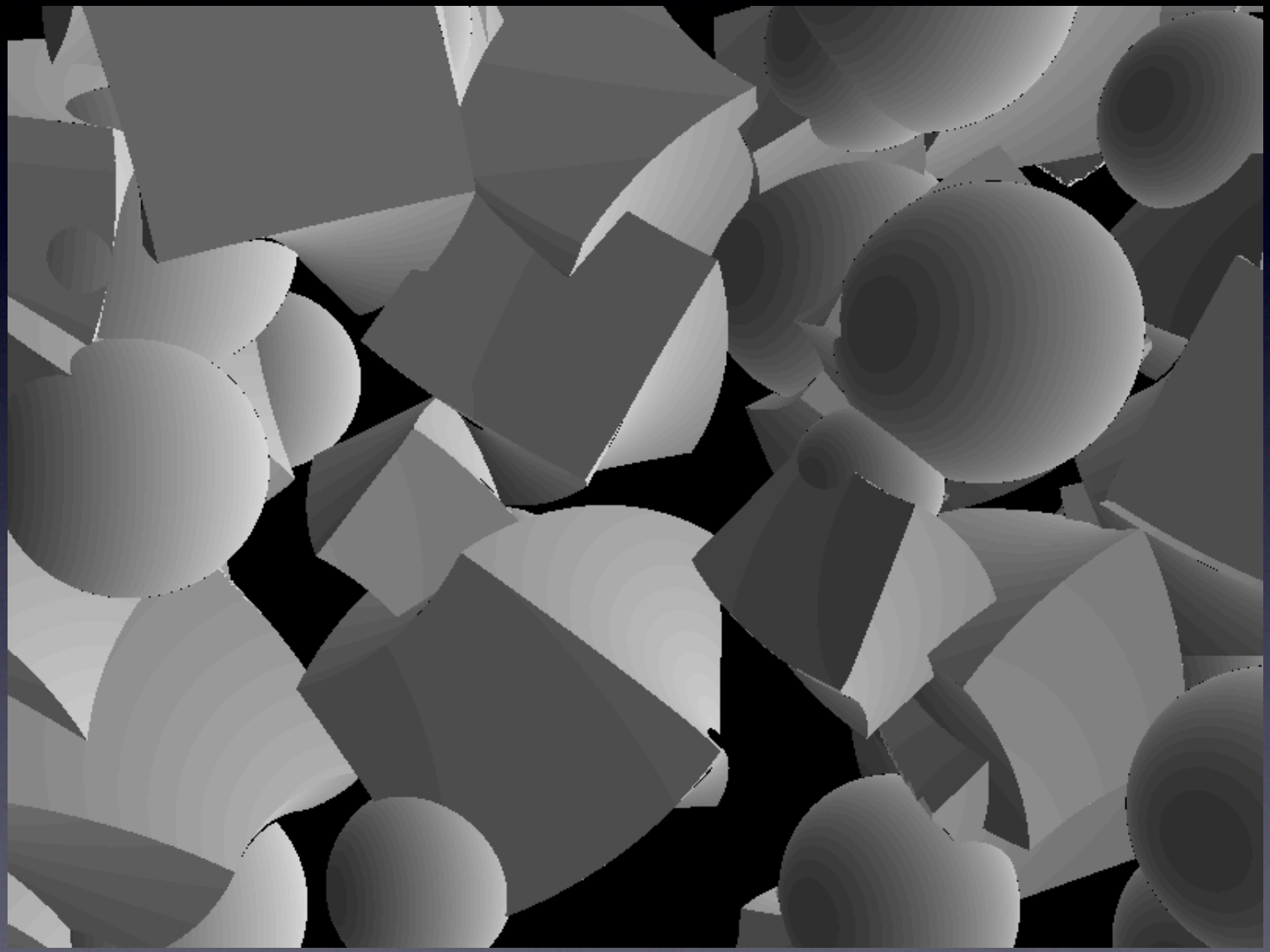
Render distance field:

```
func distanceField( p Vector ) float64 {
    d := float64( 9999 );

    for i:=0; i<60; i++ {
        d = min( d, field[i].Distance(p) );
    }

    return d;
}
```

Wow!



(this took a long time to render :( )

Concurrency

- Goroutines:
  - Run functions in background.
  - Communicate using channels.

Concurrency

Example:

```
c := make(chan int)  // Allocate a channel.

// Start the sort in a goroutine;
// when it completes, signal on the channel.

go func() {
    list.Sort()
    c <- 1  // Send a signal; value does not matter.
}()

doSomethingForAWhile()

<-c   // Wait for sort to finish; discard sent value.
```

Concurrency

Parallelization, in main:

```
const NCPU = 4  // number of CPU cores

c := make(chan int, NCPU);
for i := 0; i < NCPU; i++ {
  go renderDistanceField( .., c );
}
// Drain the channel.
for i := 0; i < NCPU; i++ {
  <-c    // wait for one task to complete
}
```

At end of renderDistanceField function:

```
  ...
  c <- 1    // signal that this piece is done
}
```

Results

```
real    0m9.163s
user    0m9.029s
sys     0m0.083s
reinder-nijhoffs-computer:go reinder$ time go channel > test2.tga

real    0m9.122s
user    0m8.986s
sys     0m0.084s
```

4 goroutines

vs.

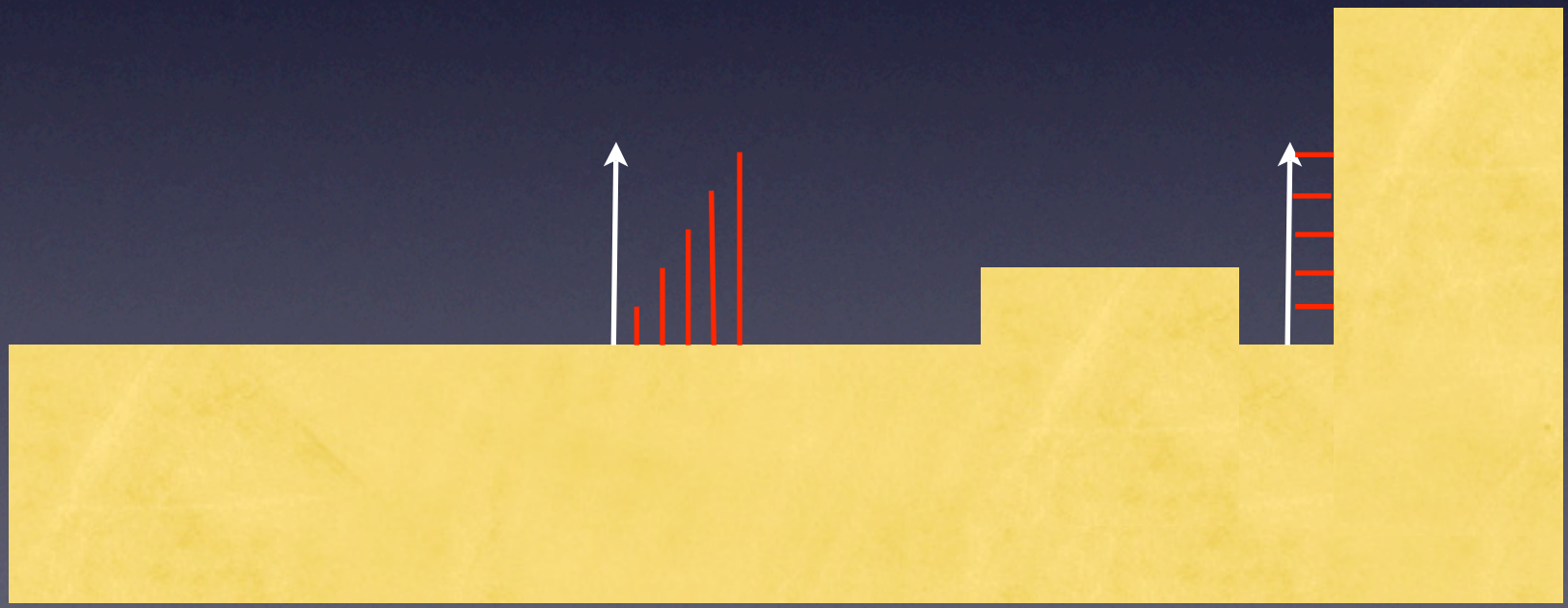
1 goroutine

(N=1 - still slow)

(something went wrong...)

Ambient Occlusion

- Fake ambient occlusion using distance field:
- Sample distance field along normal of intersectionpoint:



Ambient Occlusion

```
func ambientOcclusion( start, normal Vector ) float64 {
    a := float64(0.0);
    k := float64(1.0);

    for i:=1; i<6; i++ {
        start.X += normal.X * DELTA;
        start.Y += normal.Y * DELTA;
        start.Z += normal.Z * DELTA;
        a += ( 1.0 / k ) * ( float64(i) * DELTA
                - distanceField( start )  );
        k = k * 2.0;
    }
    return 1.0 - a;
}
```