/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2016 - Lecture 13: “Practical”

Welcome!

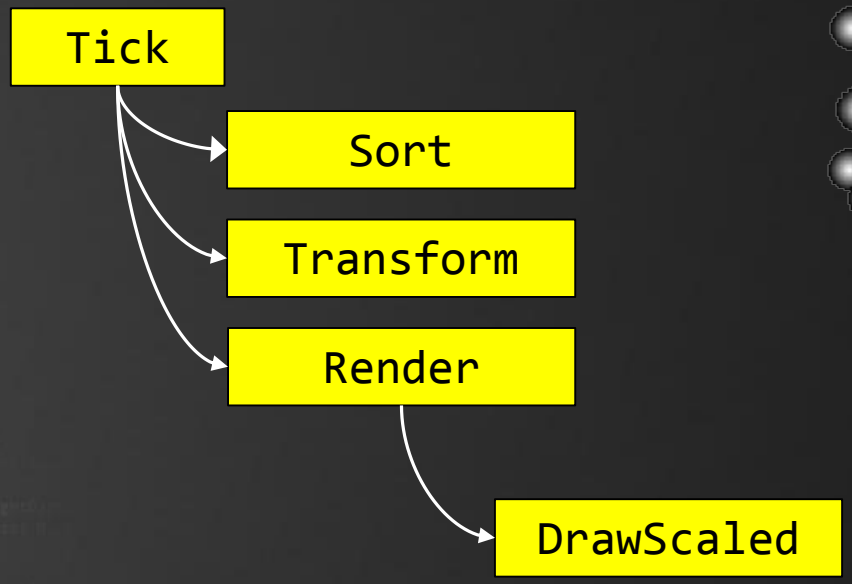
Today’s Agenda:

- DotCloud: profiling & high-level (1)
- DotCloud: low-level and blind stupidity
- DotCloud: high-level (2)
- Wu’s Algorithm for Anti-aliased Lines
- Digest

Overview

DotCloud

Application breakdown:

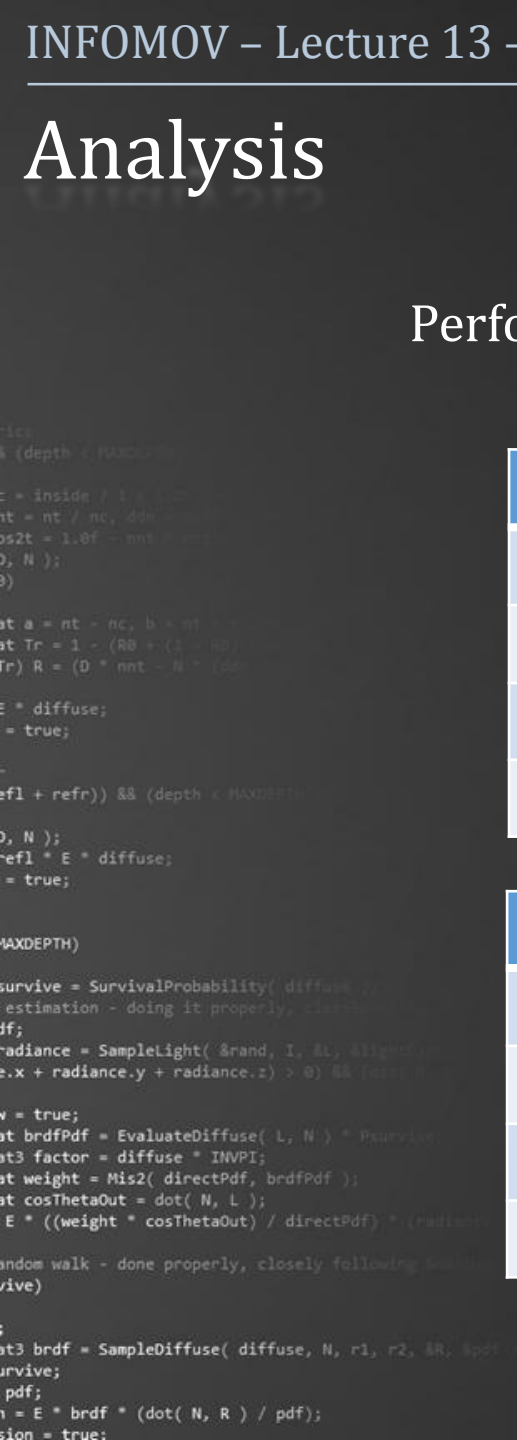


Analysis

Performance Analysis & Scalability

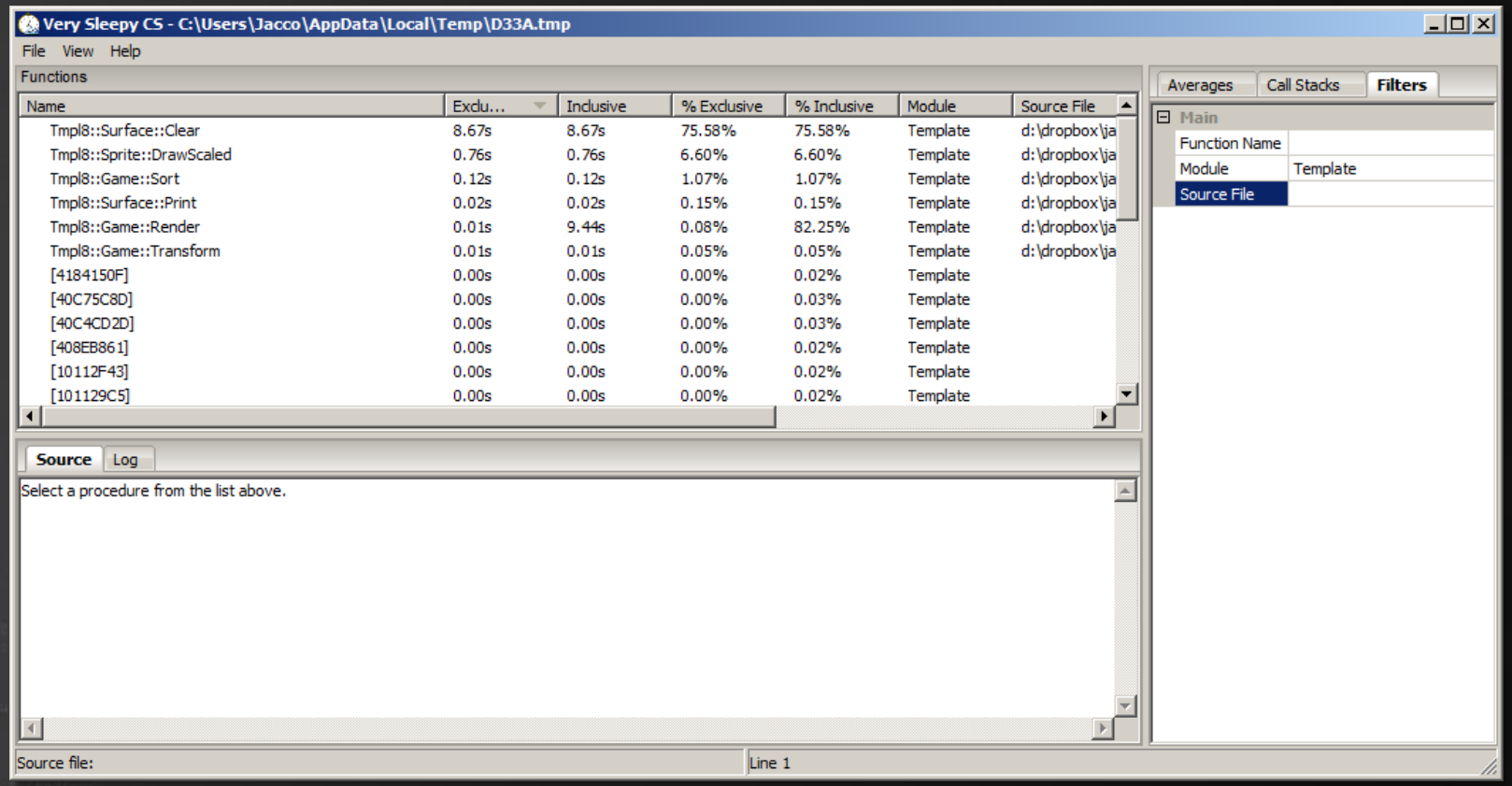
ms per frame	256	1024	4096	16384
Transform	0.01	0.01	0.07	0.23
Sort	0.45	7.26	116.80	1846.00
Render	2.26	6.43	22.65	98.41

ms per dot	256	1024	4096	16384
Transform	0.0000	0.0000	0.0000	0.0000
Sort	0.0018	0.0071	0.0285	0.1127
Render	0.0088	0.0063	0.0055	0.0060



Analysis

Performance Analysis & Scalability

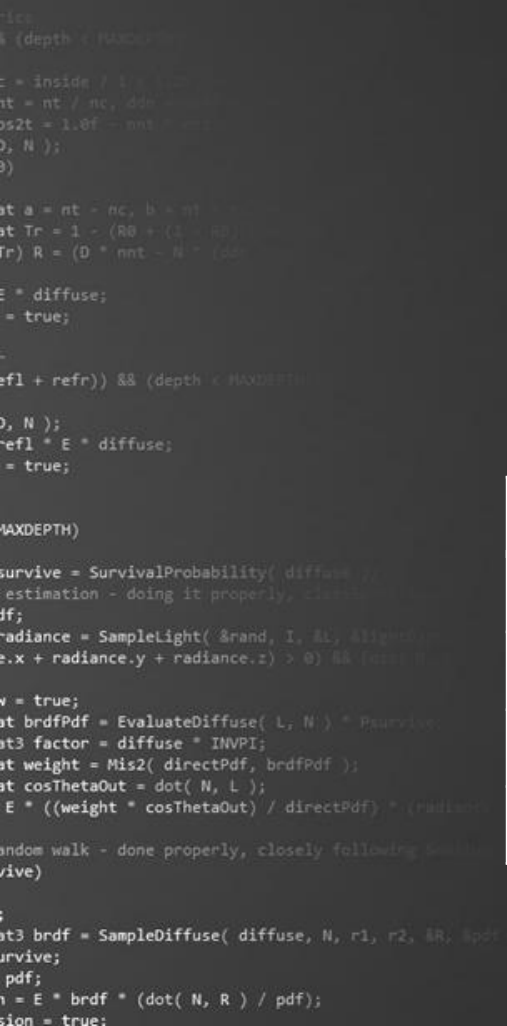


Analysis

Performance Analysis & Scalability

ms per frame	256	1024	4096	16384
Transform	0.01	0.01	0.07	0.23
Sort	0.45	7.26	116.80	1846.00
Render	2.26	6.43	22.65	98.41
Clear	0.91	0.91	0.91	0.91

ms per dot	256	1024	4096	16384
Transform	0.0000	0.0000	0.0000	0.0000
Sort	0.0018	0.0071	0.0285	0.1127
Render	0.0088	0.0063	0.0055	0.0060
Clear	0.0036	0.0009	0.0002	0.0001



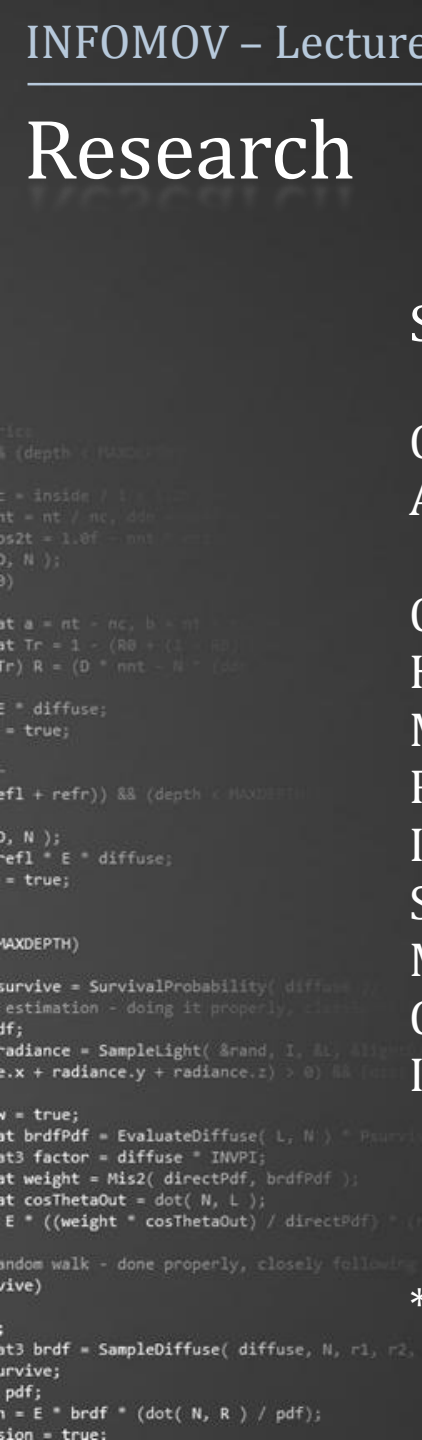
Research

Solving the Sort Problem

Current Sort: bubblesort ($O(N^2)$).
Alternatives*:

Quicksort	Shell sort	Pigeonhole sort
Heapsort	Binary tree sort	Bucket sort
Mergesort	Library sort	Spread sort
Radixsort	Smoothsort	Burstsort
Insertionsort	Strand sort	Flashsort
Selectionsort	Cocktail sort	Postman sort
Monkeysort	Comb sort	Bread sort
Countingsort	Block sort	Bitonic sort
Introsort	Odd-even sort	Stooge sort

* See e.g.: http://www.sorting-algorithms.com



Research

Solving the Sort Problem

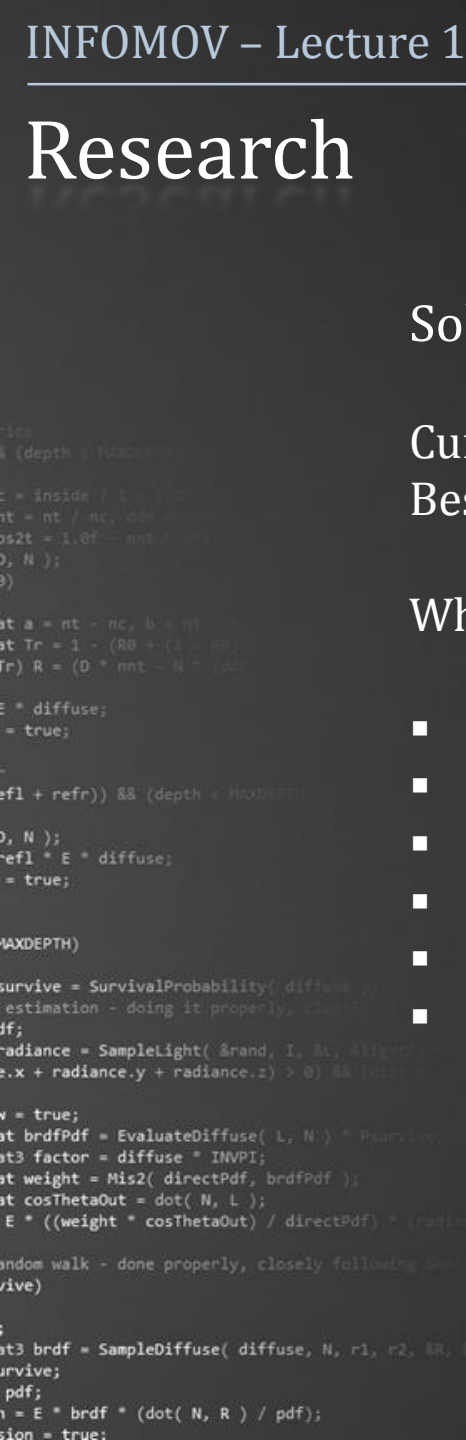
Current Sort: bubblesort ($O(N^2)$).
Best case: O(N).

Which case do we have here? Factors:

- Size of set
- Already sorted / almost sorted?
- Distributed (even / uneven)
- Type of data (just key / full records)
- Key type (float / int / string)
- …

How much effort should we spend on this?

- For small sets, sorting takes far less time than rendering
- Anything that is not $O(N^2)$ will probably be fine.
- Would be nice if we can find something that fits well in the current code (safe time for other optimizations).

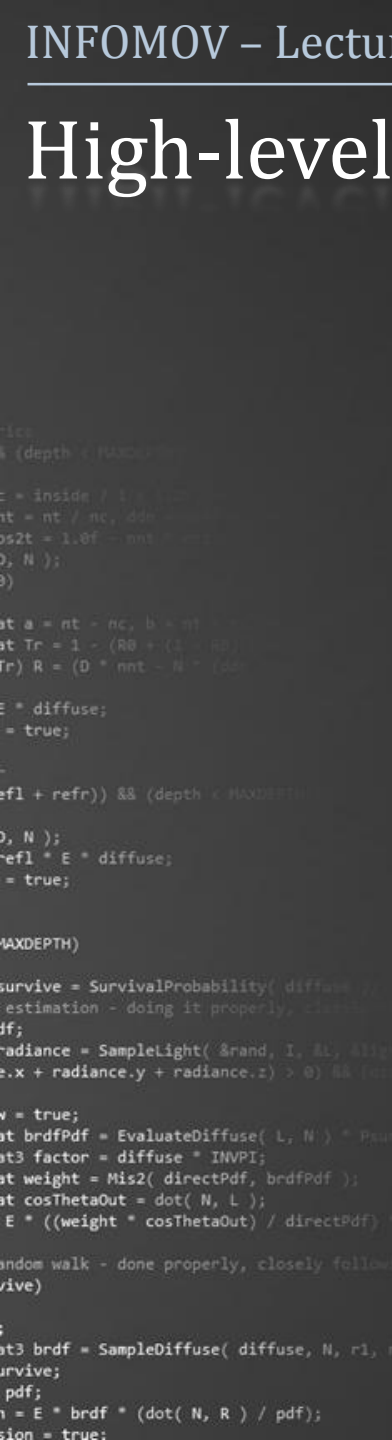


High-level

Solving the Sort Problem

Current Sort: bubblesort ($O(N^2)$). Alternative: QuickSort ($O(N \log N)$).

```
void Swap( float3& a, float3& b ) { float3 t = a; a = b; b = t; }
int Pivot( float3 a[], int first, int last )
{
   int p = first;
   float3 e = a[first];
   for( int i = first + 1; i <= last; i++ ) if (a[i].z <= e.z) Swap( a[i], a[++p] );
   Swap( a[p], a[first] );
   return p;
}
void QuickSort( float3 a[], int first, int last)
{
   int pivotElement;
   if (first >= last) return;
   pivotElement = Pivot( a, first, last );
   QuickSort( a, first, pivotElement - 1 );
   QuickSort( a, pivotElement + 1, last );
}
```



Profile

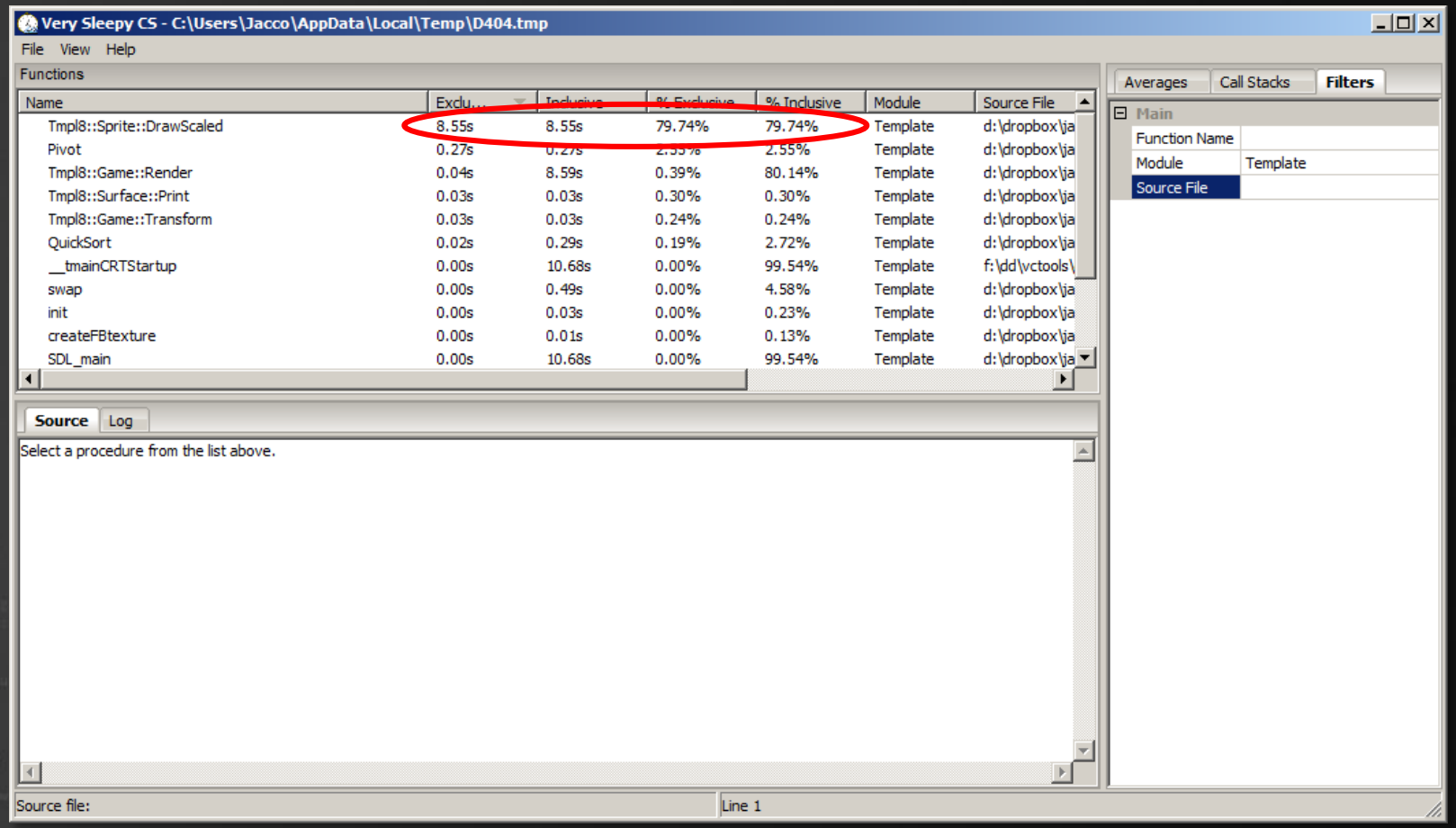
Repeated Profiling

bubblesort	256	1024	4096	16384
Transform	0.01	0.01	0.07	0.23
Sort (bubble)	0.45	7.26	116.80	1846.00
Sort (quick)	**0.03**	**0.13**	**0.63**	**2.90**
Render	2.26	6.43	22.65	98.41
Clear	0.91	0.91	0.91	0.91

Note: Clear is implemented using a loop; a memset is faster for clearing to zero (~0.43).

Profile

Repeated Profiling



Low-level

Low Level Optimization of DrawScaled

```
void Sprite::DrawScaled( float a_X, float a_Y, float a_Width, float a_Height, Surface* a_Target )
{
   Pixel* dest = a_Target->GetBuffer() + (int)a_X + (int)a_Y * a_Target->GetPitch();
   Pixel* src = GetBuffer() + m_CurrentFrame * m_Width;
   for ( int y = 0; y < (int)a_Height; y++ ) for ( int x = 0; x < (int)a_Width; x++ )
   {
      int v = (int)((y * m_Height) / a_Height);
      int u = (int)((x * m_Pitch) / a_Width);
      if (src[u + v * m_Pitch] & 0xffffff)
         *(dest + x + y * a_Target->GetWidth()) = src[u + v * m_Pitch];
   }
}
```

Functionality:

- for every pixel of the rectangular target image,
- find the corresponding source pixel,
- using interpolation,
- plot if it’s not black.

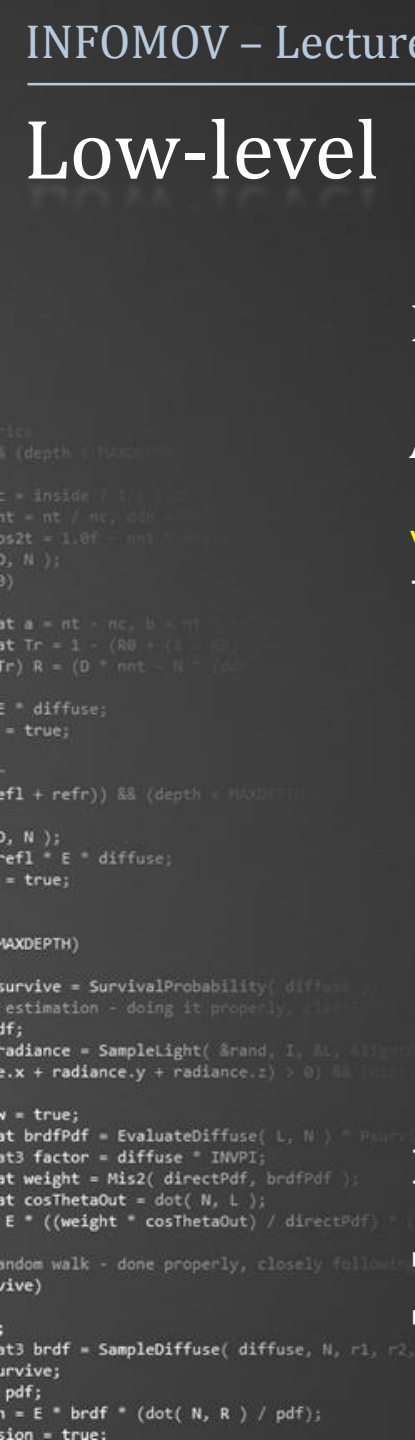
Low-level

Low Level Optimization of DrawScaled

A few basic optimizations:

```
void Sprite::DrawScaled( float a_X, float a_Y, float a_Width, float a_Height, Surface* a_Target )
{
   Pixel* dest = a_Target->GetBuffer() + (int)a_X + (int)a_Y * a_Target->GetPitch();
   Pixel* src = GetBuffer() + m_CurrentFrame * m_Width;
   for ( int y = 0; y < (int)a_Height; y++ )
   {
      int v = (int)((y * m_Height) / a_Height);
      for ( int x = 0; x < (int)a_Width; x++ )
      {
         int u = (int)((x * m_Pitch) / a_Width);
         Pixel color = src[u + v * m_Pitch] & 0xffffff;
         if (color) *(dest + x + y * a_Target->GetWidth()) = color;
      }
   }
}
```

- Loop hoisting (variable v is constant inside x loop)
- Reading source pixel only once



Low-level

Low Level Optimization of DrawScaled

More basic optimizations:

```
void Sprite::DrawScaled( float a_X, float a_Y, float a_Width, float a_Height, Surface* a_Target )
{
   Pixel* dest = a_Target->GetBuffer() + (int)a_X + (int)a_Y * a_Target->GetPitch();
   Pixel* src = GetBuffer() + m_CurrentFrame * m_Width;
   float rw = (float)m_Width / a_Width;
   float rh = (float)m_Height / a_Height;
   int iw = (int)a_Width, ih = (int)a_Height;
   for ( int y = 0; y < ih; y++ )
   {
      int v = (int)(y * rh);
      Pixel* line = dest + y * a_Target->GetWidth();
      for ( int x = 0; x < iw; x++ )
      {
         int u = (int)(x * rw);
         Pixel color = src[u + v * m_Width] & 0xffffff;
         if (color) line[x] = color;
      }
   }
}
```

- Precalculate m_Height / a_Height, m_Width / a_Width
- Calculate target address once per line; index using x

Low-level

Low Level Optimization of DrawScaled

Fixed point optimization:

```
void Sprite::DrawScaled( int a_X, int a_Y, int a_Width, int a_Height, Surface* a_Target )
{
   const int rh = (m_Height << 10) / a_Height, rw = (m_Width << 10) / a_Width;
   Pixel* line = a_Target->GetBuffer() + a_X + a_Y * a_Target->GetPitch();
   for ( int y = 0; y < a_Height; y++, line += a_Target->GetPitch() )
   {
      const int v = (y * rh) >> 10;
      for ( int x = 0; x < a_Width; x++ )
      {
         const int u = (x * rw) >> 10;
         const Pixel color = GetBuffer()[u + v * m_Pitch];
         if (color & 0xffffff) line[x] = color;
      }
   }
}
```

- Fixed point works really well here… but doesn’t improve performance.
- Seems we reached the end here…

Blind Stupidity

Low Level Optimization of DrawScaled

Now what?

- Plot multiple pixels at a time?
- …

How many different ball sizes do we encounter?

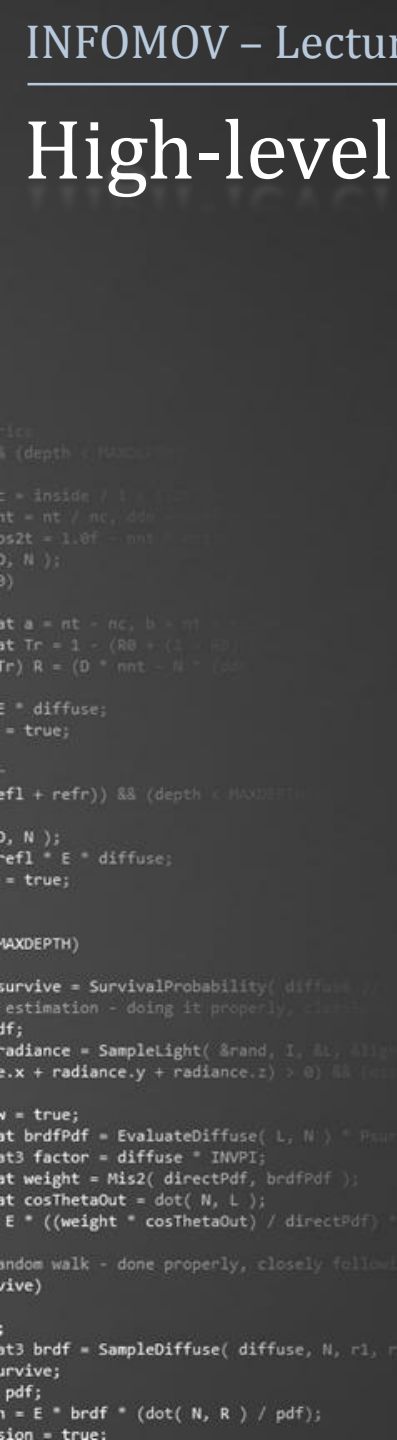
…Why don’t we simply precalculate those frames?

“More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity.” (W.A. Wulff)

High-level

High Level Optimization of DrawScaled

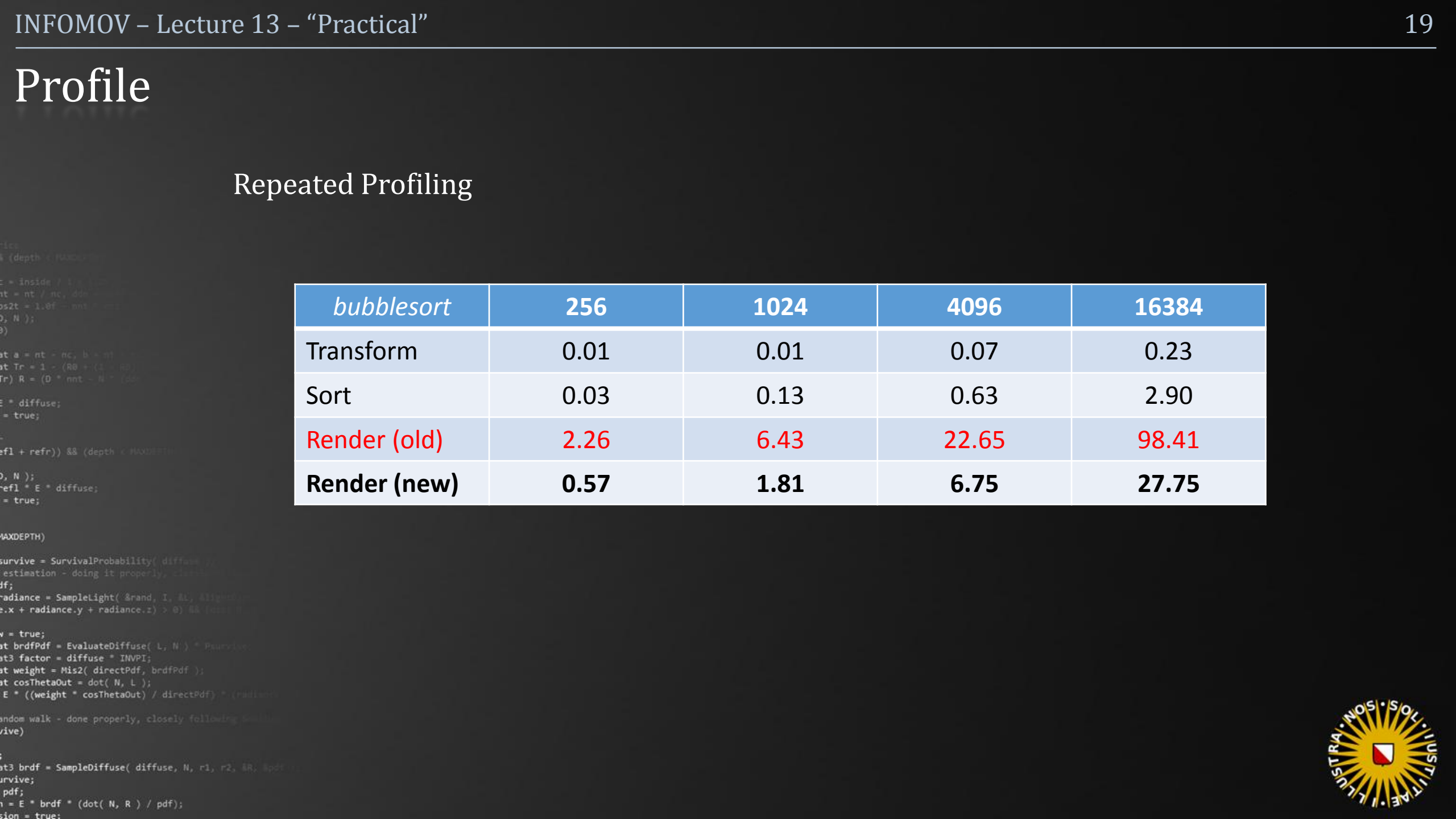
```
Sprite* scaled[64];
void Game::Init()
{
   ...
   for( int i = 0; i < 64; i++ )
   {
      int size = i + 1;
      scaled[i] = new Sprite( new Surface( size, size ), 1 );
      scaled[i]->GetSurface()->Clear( 0 );
      m_Dot->DrawScaled( 0, 0, size, size, scaled[i]->GetSurface() );
   }
}

scaled[size]->Draw( (sx - size / 2), (sy - size / 2), m_Surface );
```



Profile

Repeated Profiling

bubblesort	256	1024	4096	16384
Transform	0.01	0.01	0.07	0.23
Sort	0.03	0.13	0.63	2.90
Render (old)	2.26	6.43	22.65	98.41
Render (new)	**0.57**	**1.81**	**6.75**	**27.75**



Profile

What about the compiler?

bubblesort	256	1024	4096	16384
Transform	0.01	0.01	0.07	0.23
Sort	0.03	0.13	0.63	2.90
Render (vs’13)	0.57	1.81	6.75	27.75
Render (vs’15)	**0.56**	**1.82**	**?**	**26.30**

Profile

What about the compiler?

bubblesort	256	1024	4096	16384
Transform	0.01	0.01	0.07	0.23
Sort	0.03	0.13	0.63	2.90
Render (vs’13)	0.57	1.81	6.75	27.75
Render (’15,32)	0.56	1.82	?	26.30
Render (‘15,64)	**0.59**	**1.92**	**7.05**	**27.50**

Dotting i’s

Optimization of Dense Clouds

Observation: beyond a certain dot count, a large number of particles is occluded.

Specifically, we won’t be able to see the back half.

```
if (m_Rotated[i].z > -0.2f)
          scaled[size]->Draw( (sx - size / 2), (sy - size / 2), screen );
```

(perhaps we could also limit rendering to the outer shell of the cloud?)

Rendering is now significantly faster, and sorting is significant again:

At 65536 dots, we get 11ms for sorting, 3ms for rendering.

High-level

Sorting in O(1)

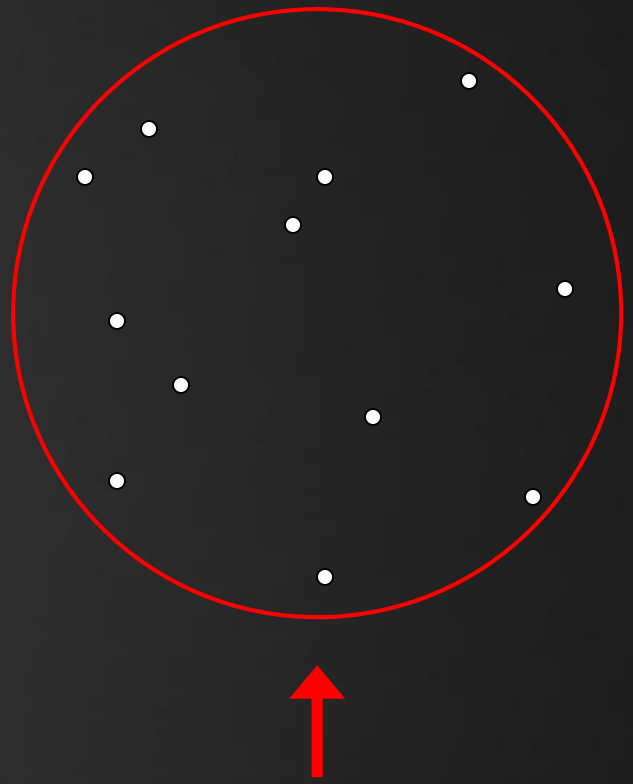
For this specific situation, we can sort in O(1), e.g., independent of particle count.

Observation: dots do not move independently.

Intuition: why rotate 64k dots if you can rotate a single camera?

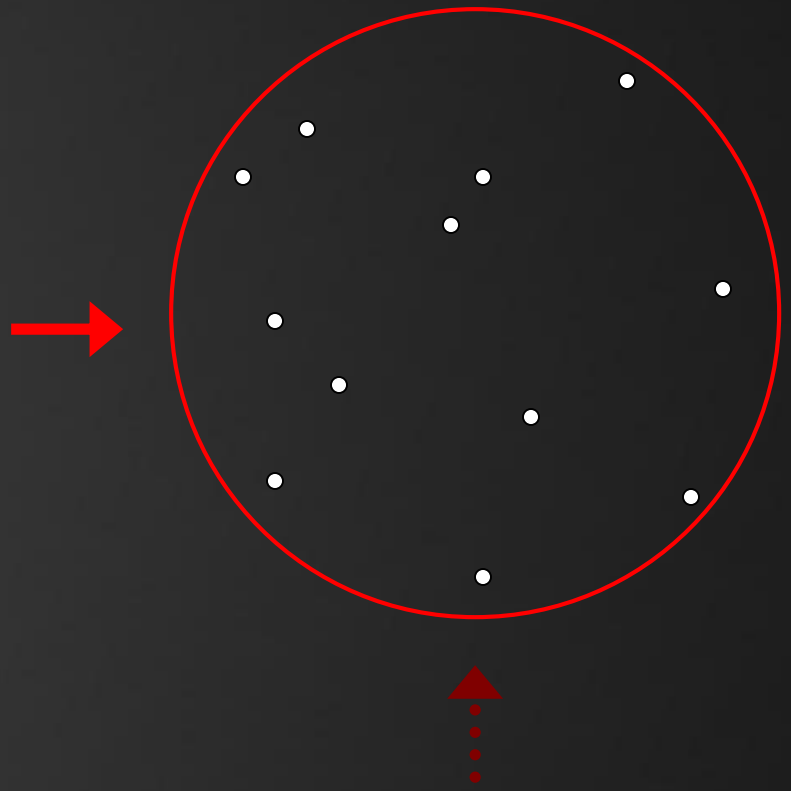
High-level

Sorting in O(1)



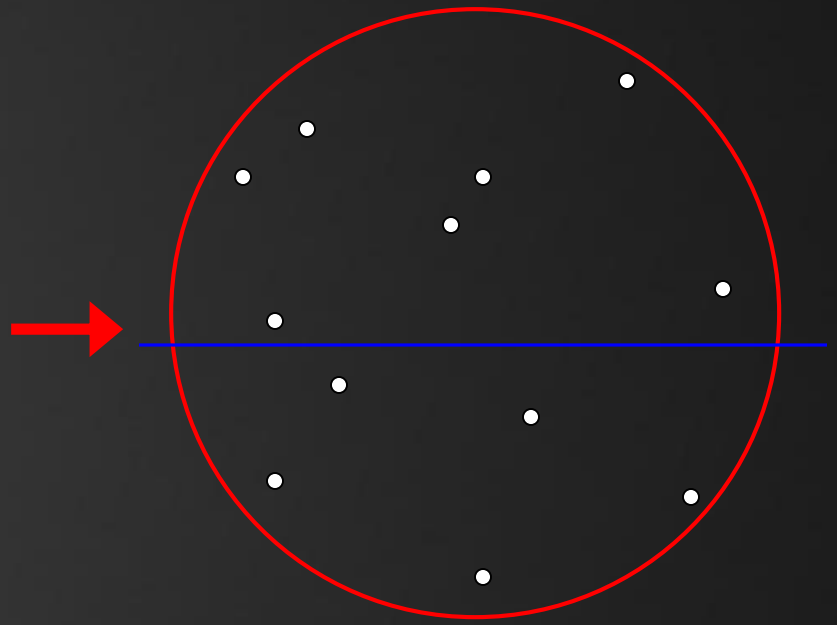
High-level

Sorting in O(1)



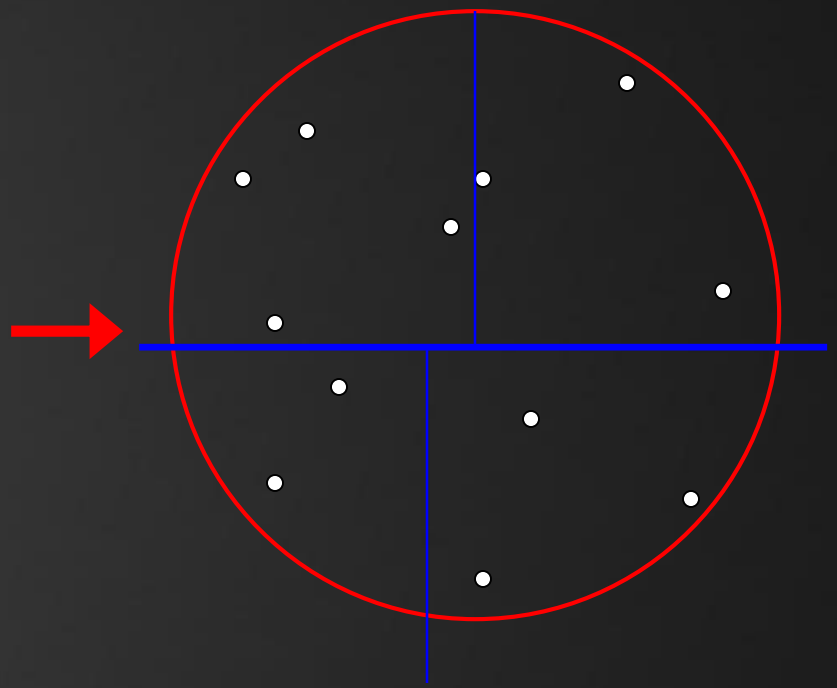
High-level

Sorting in O(1)



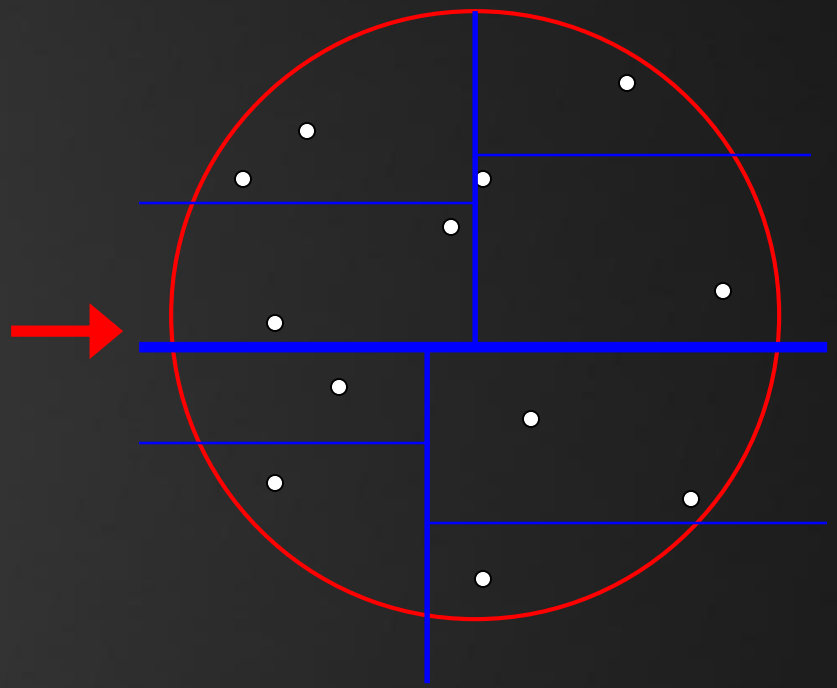
High-level

Sorting in O(1)



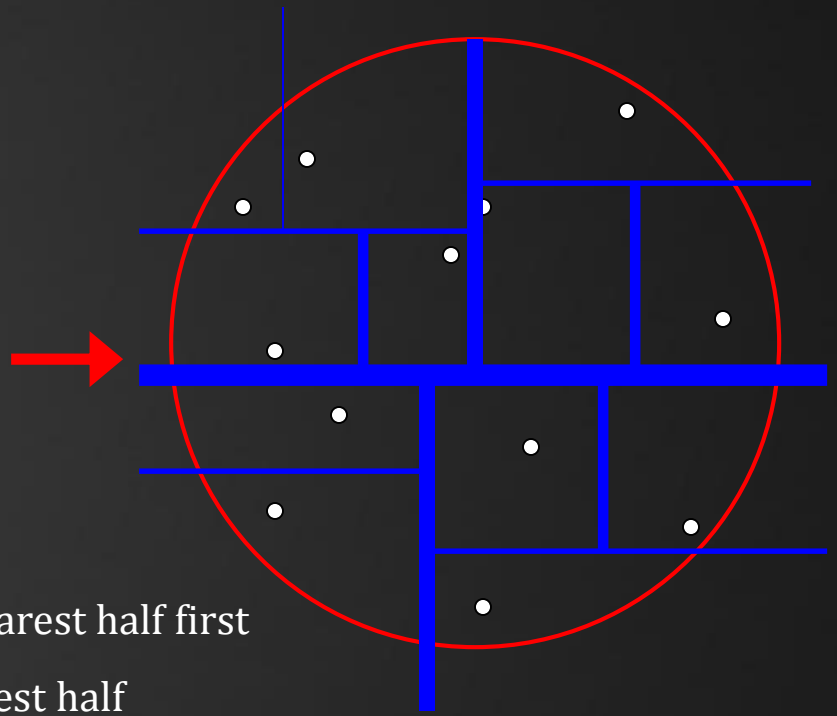
High-level

Sorting in O(1)



High-level

Sorting in O(1)



For each split:

- Process nearest half first
- Then farthest half
- Recurse

Where ‘nearest’ is the side that the ‘camera’ is on.

Quo Vadis

It’s Fast, Now What?

We need additional work:

- Smaller particles
- Sub-pixel movement, alpha blending for sprite edges
- Higher resolution

We could multi-thread:

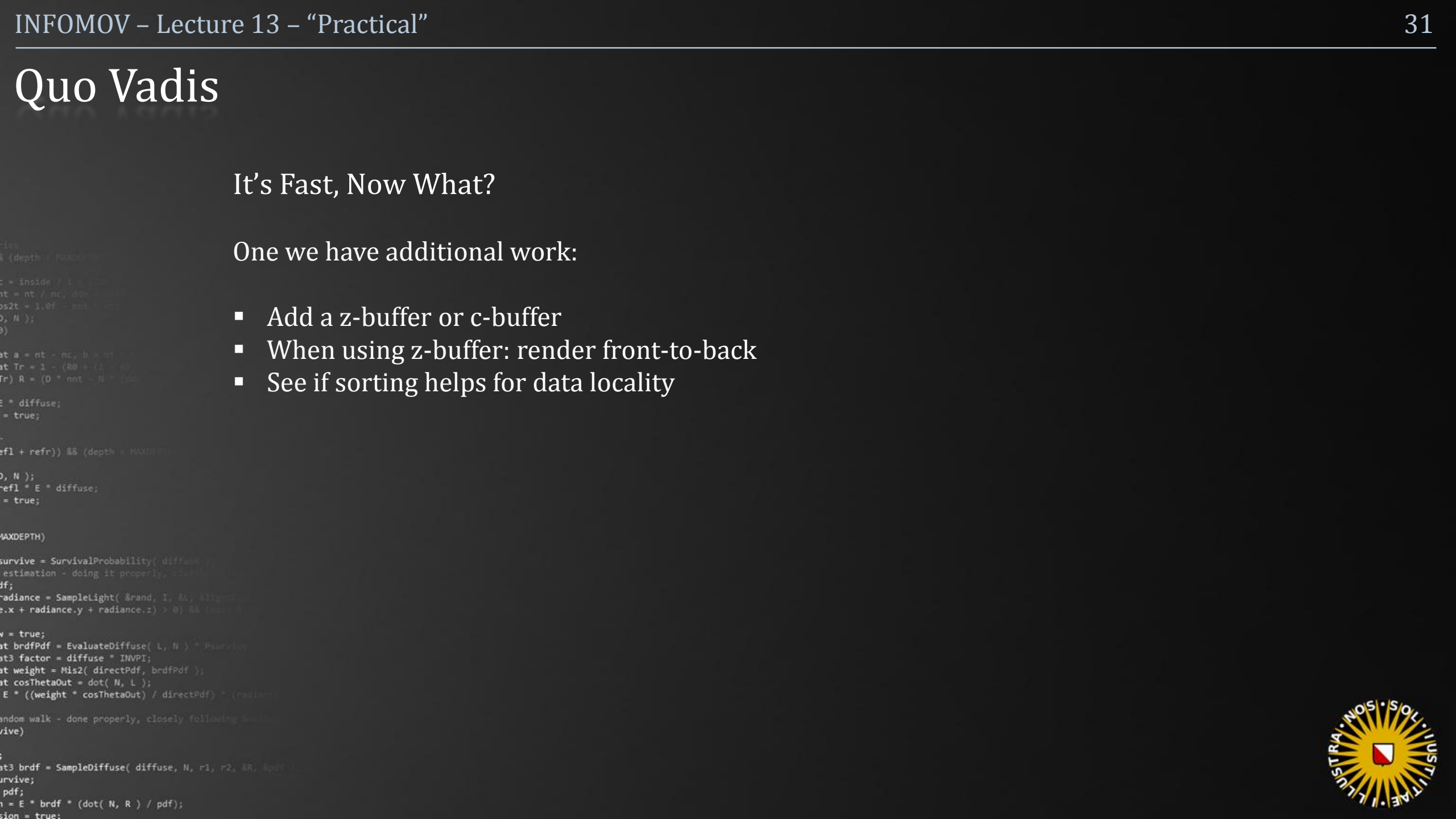
- Two screen halves / four quarters: clipping
- A grid, for GPGPU

Quo Vadis

It’s Fast, Now What?

One we have additional work:

- Add a z-buffer or c-buffer
- When using z-buffer: render front-to-back
- See if sorting helps for data locality



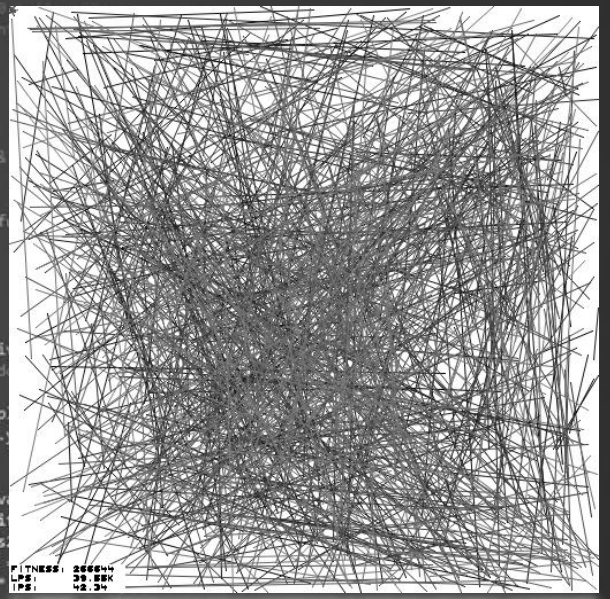
Today’s Agenda:

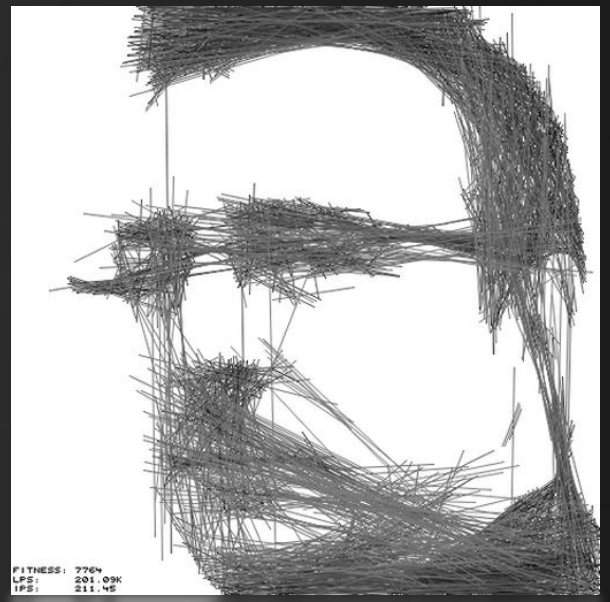
- DotCloud: profiling & high-level (1)
- DotCloud: low-level and blind stupidity
- DotCloud: high-level (2)
- Wu’s Algorithm for Anti-aliased Lines
- Digest

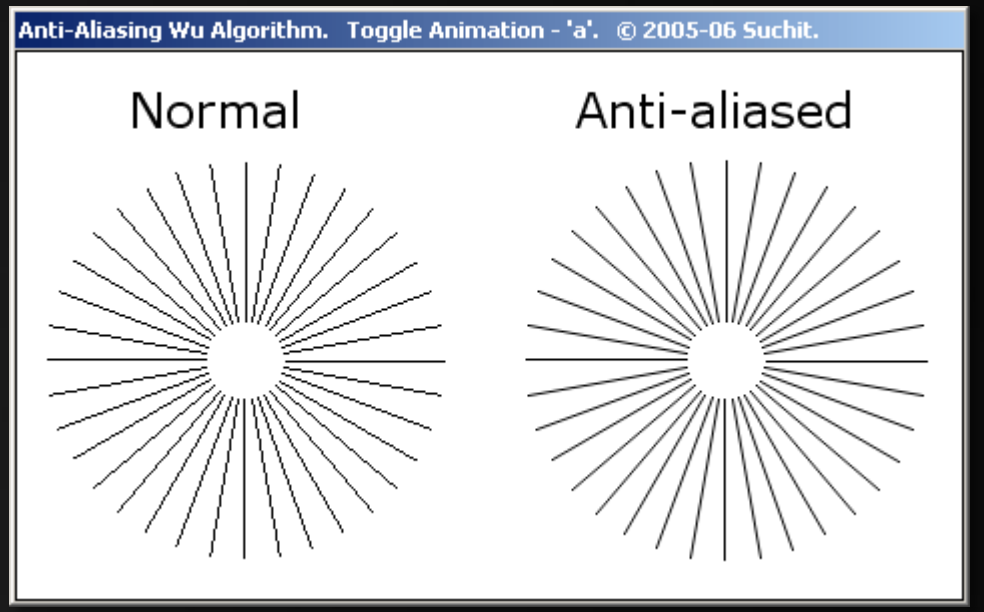
Wu

Wu’s Algorithm

Bresenham-style line rendering, with anti-aliasing.







http://www.codeproject.com/Articles/13360/Antialiasing-Wu-Algorithm

Idea: draw 1024 random lines, then change the image one line at the time and check if the result is now closer to the target image. If so, keep the mutation, otherwise, revert it.

Bottleneck: rendering lines.

Low-level

Wu’s Algorithm

```
void Surface::WULine( int X0, int Y0, int X1, int Y1, Pixel clrLine )
{
   // make sure the line runs top to bottom
   if (Y0 > Y1) { /* flip */ }

   // special-case horizontal, vertical, and diagonal lines
   int DeltaY = Y1 - Y0;
   if (DeltaY == 0) { /* horizontal line */ ... return; }
   if (DeltaX == 0) { /* vertical line */ ... return; }
   if (DeltaX == DeltaY) { /* diagonal line */ ... return; }

   // is this an X-major or Y-major line?
   if (DeltaY > DeltaX)
   {
      // it’s a y-major line
      ...
      return;
   }
   // it's an x-major line
   ...
}
```

Low-level

Wu’s Algorithm

Step 1:

Shorts are evil (16-bit)

➔ Use unsigned ints instead
➔ Mimic behavior of short: & 0xFFFF

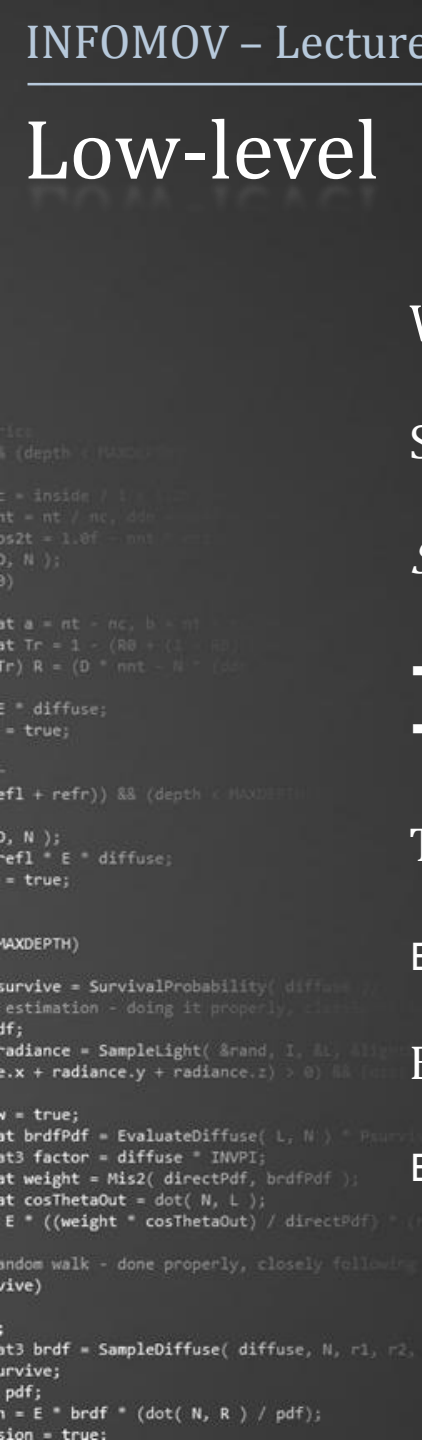
Turns out this is only actually needed for this line:

```
ErrorAcc += ErrorAdj;
```

Becomes:

```
ErrorAcc = (ErrorAcc + ErrorAdj) & 0xffff;
```

```
unsigned short ErrorAdj, ErrorAccTemp, Weighting;

// line is not horizontal, diagonal, or vertical
unsigned short ErrorAcc = 0;
```



Low-level

```
BYTE r1 = GetRValue( clrLine );
BYTE g1 = GetGValue( clrLine );
BYTE b1 = GetBValue( clrLine );
```

Wu’s Algorithm

Step 2:

Bytes are also evil (8-bit)

➔ Use unsigned ints instead
➔ This has no consequences

Low-level

```
BYTE rl = GetRValue( clrLine );
BYTE gl = GetGValue( clrLine );
BYTE bl = GetBValue( clrLine );

Plot( X0, Y0, RGB( rr, gr, br ) );
```

Wu’s Algorithm

Step 3:

Don’t trust Windows functions.
In fact, don’t trust my functions.

➔ Plot becomes Plot_ (no if statements)
➔ GetRValue etc. get replaced by some basic masking
➔ RGB(r,g,b) gets replaced by some shifting magic

Low-level

```
BYTE rr = ( rb > rl ? ( ( BYTE )( ( ( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0 * ( rb - rl ) + rl ) ) : ( ( BYTE )( ( ( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0 * ( rl - rb ) + rb ) ) );
BYTE gr = ( gb > gl ? ( ( BYTE )( ( ( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0 * ( gb - gl ) + gl ) ) : ( ( BYTE )( ( ( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0 * ( gl - gb ) + gb ) ) );
BYTE br = ( bb > bl ? ( ( BYTE )( ( ( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0 * ( bb - bl ) + bl ) ) : ( ( BYTE )( ( ( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0 * ( bl - bb ) + bb ) ) );
```

Wu’s Algorithm

Step 4:

That’s an awful lot of repetitive comparisons.

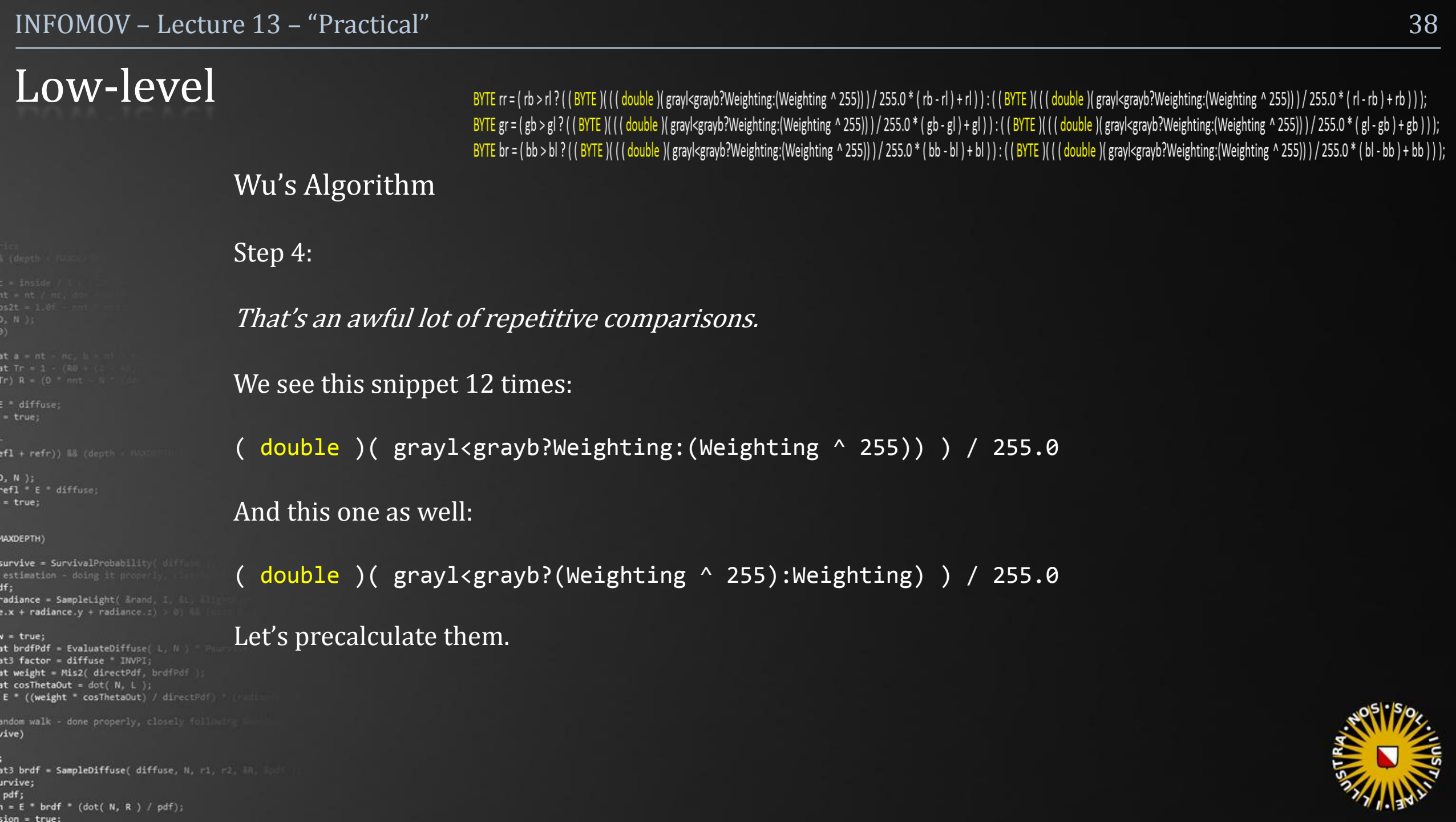
We see this snippet 12 times:

```
( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0
```

And this one as well:

```
( double )( grayl<grayb?(Weighting ^ 255):Weighting) ) / 255.0
```

Let’s precalculate them.



Low-level

Wu’s Algorithm

Step 5:

Comparing doubles seems overdone.

The comparisons compare these values:

```
double grayl = rl * 0.299 + gl * 0.587 + bl * 0.114;
double grayb = rb * 0.299 + gb * 0.587 + bb * 0.114;
```

That’s silly; let’s compare unsigned integers instead.
We use 8-bit fixed point, that should be enough accuracy:

```
uint grayl = rl * 77 + gl * 150 + bl * 29;
uint grayb = rb * 77 + gb * 150 + bb * 29;
```

0.299 * 256 = 77
0.587 * 256 = 150
0.114 * 256 = 29

77 + 150 + 29 = 256

Low-level

Wu’s Algorithm

Step 6:

In fact, we don’t need doubles at all.

We still have these:

```
( double )( grayl<grayb?Weighting:(Weighting ^ 255)) ) / 255.0
( double )( grayl<grayb?(Weighting ^ 255):Weighting) ) / 255.0
```

Everything is ‘int’ until the cast; so let’s not divide by 255 and keep the ints as fixed point numbers.

That does mean that when we multiply by these values, we have to divide by 256 to undo the fixed point multiply scale.

Low-level

Wu’s Algorithm

Final performance: roughly doubled.

Takeaway: casts are evil. And: don‘t do doubles if you don’t need to.

Today’s Agenda:

- DotCloud: profiling & high-level (1)
- DotCloud: low-level and blind stupidity
- DotCloud: high-level (2)
- Wu’s Algorithm for Anti-aliased Lines
- Digest

Digest

High Level Optimization

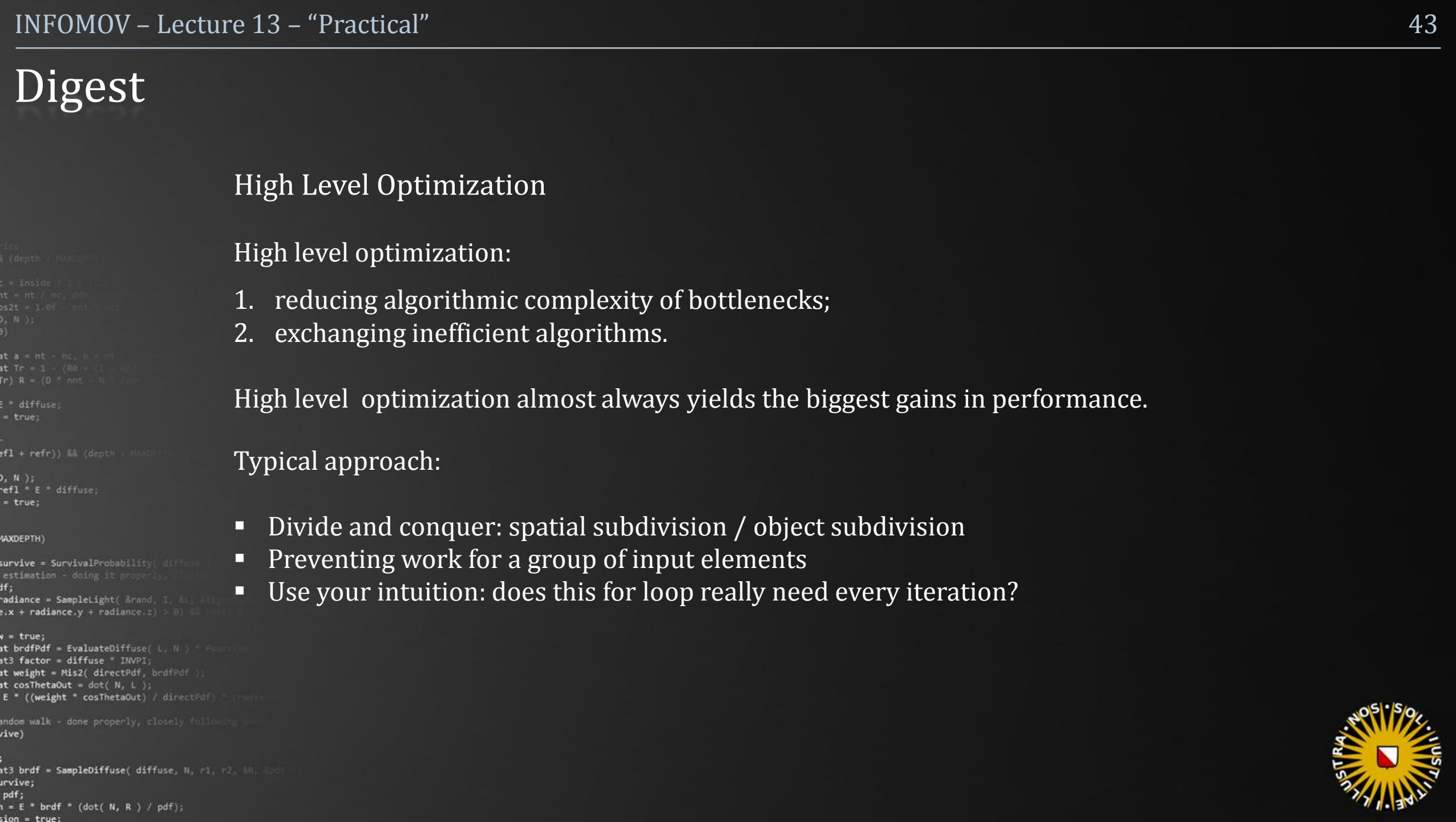
High level optimization:

1. reducing algorithmic complexity of bottlenecks;
2. exchanging inefficient algorithms.

High level optimization almost always yields the biggest gains in performance.

Typical approach:

- Divide and conquer: spatial subdivision / object subdivision
- Preventing work for a group of input elements
- Use your intuition: does this for loop really need every iteration?



Digest

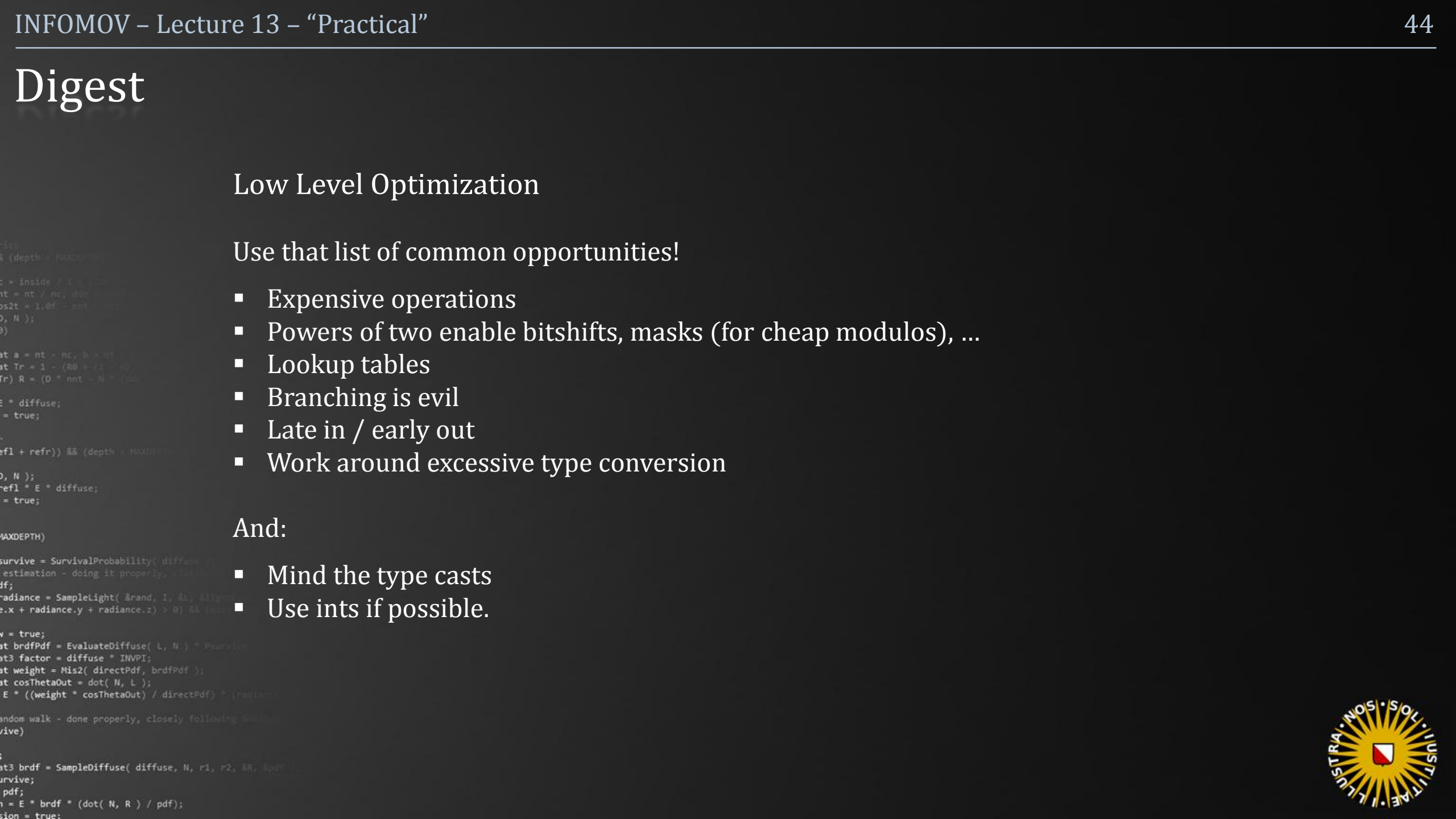
Low Level Optimization

Use that list of common opportunities!

- Expensive operations
- Powers of two enable bitshifts, masks (for cheap modulos), …
- Lookup tables
- Branching is evil
- Late in / early out
- Work around excessive type conversion

And:

- Mind the type casts
- Use ints if possible.



Today’s Agenda:

- DotCloud: profiling & high-level (1)
- DotCloud: low-level and blind stupidity
- DotCloud: high-level (2)
- Wu’s Algorithm for Anti-aliased Lines
- Digest

...

Low Level Optimization of Sprite::Draw

Pre-scaled sprites are faster than on-the-fly scaling.

But, we still have loops, and if-statements, and look-ups. I wonder…

...

Low Level Optimization of Sprite::Draw

Extreme Optimization:

- We simply generate a function that plots every pixel, without the need for a loop.

Side effect:

L1 data cache is now used for the screen buffer;
L1 instruction cache is used for the sprite data. ☺

```
void Sprite::DrawBall( int x, int y, int size,
                       Surface* target )
{
   uint* a = target->GetBuffer() + x + y * SCRWIDTH;
   switch( size )
   {
   case 1:
      break;
   case 2:
      a[1]=1052688;
      a[800]=1052688;
      a[801]=15724527;
      break;
   case 3:
      a[801]=9737364;
      a[802]=8684676;
      a[1601]=8684676;
      a[1602]=8092539;
      break;
   case 4:
      a[2]=1052688;
      a[801]=6513507;
      a[802]=9737364;
      a[803]=4868682;
      a[1600]=1052688;
      a[1601]=9737364;
      a[1602]=15724527;
      a[1603]=7566195;
      a[2401]=4868682;
      a[2402]=7566195;
      a[2403]=3223857;
      break;
```

...

Low Level Optimization of Sprite::Draw

Extreme Optimization:

- We simply generate a function that plots every pixel, without the need for a loop.

```
FILE* f = fopen( "drawfunc.h", "w" );
fprintf( f, "void Sprite::DrawBall( int x, int y, int size, Surface* target )\n" );
fprintf( f, "{\nuint* a = target->GetBuffer() + x + y * SCRWIDTH;\nswitch( size )\n{\n" );
for( int i = 0; i < 64; i++ )
{
   ...
   fprintf( f, "case %i:\n", size );
   for( int y = 0; y < size; y++) for( int x = 0; x < size; x++ )
   {
      int a = y * SCRWIDTH + x;
      if (scaled[i]->GetBuffer()[x + y * size] & 0xffffff)
         fprintf( f, "a[%i]=%i;\n", a, scaled[i]->GetBuffer()[x + y * size] & 0xffffff );
   }
   fprintf( f, "break;\n" );
}
fprintf( f, "}\n}\n" );
```

```
switch( size )
{
case 1:
break;
case 2:
a[1]=1052688;
a[800]=1052688;
a[801]=15724527;
break;
case 3:
a[801]=9737364;
a[802]=8684676;
a[1601]=8684676;
a[1602]=8092539;
break;
case 4:
a[2]=1052688;
a[801]=6513507;
a[802]=9737364;
a[803]=4868682;
a[1600]=1052688;
a[1601]=9737364;
a[1602]=15724527;
a[1603]=7566195;
a[2401]=4868682;
a[2402]=7566195;
a[2403]=3223857;
break;
case 5:
a[2]=526344;
a[3]=526344;
a[801]=3223857;
a[802]=7039851;
a[803]=6513507;
a[804]=2697513;
a[1600]=526344;
a[1601]=7039851;
a[1602]=12434877;
a[1603]=11908533;
a[1604]=5921370;
a[2400]=526344;
a[2401]=6513507;
a[2402]=11908533;
a[2403]=11382189;
a[2404]=5395026;
a[3201]=2697513;
a[3202]=5921370;
a[3203]=5395026;
a[3204]=2171169;
break;
case 6:
a[3]=1052688;
a[801]=2171169;
a[802]=5395026;
a[803]=6513507;
a[804]=4868682;
a[805]=1052688;
a[1601]=5395026;
a[1602]=9737364;
a[1603]=11908533;
a[1604]=8684676;
a[1605]=4342338;
a[2400]=1052688;
a[2401]=6513507;
a[2402]=11908533;
a[2403]=15724527;
a[2404]=10855845;
a[2405]=5395026;
a[3201]=4868682;
a[3202]=8684676;
a[3203]=10855845;
a[3204]=8092539;
a[3205]=3750201;
a[4001]=1052688;
a[4002]=4342338;
a[4003]=5395026;
a[4004]=3750201;
a[4005]=526344;
break;
case 7:
a[3]=1052688;
a[4]=526344;
a[801]=526344;
a[802]=3750201;
a[803]=5395026;
a[804]=4868682;
a[805]=3223857;
a[1601]=3750201;
a[1602]=8092539;
a[1603]=10263708;
a[1604]=10263708;
a[1605]=7039851;
a[1606]=2697513;
a[2400]=1052688;
a[2401]=5395026;
a[2402]=10263708;
a[2403]=14079702;
a[2404]=13553358;
a[2405]=9211020;
a[2406]=4342338;
a[3200]=526344;
a[3201]=4868682;
a[3202]=10263708;
a[3203]=13553358;
a[3204]=12434877;
a[3205]=9211020;
a[3206]=3750201;
a[4001]=3223857;
a[4002]=7039851;
a[4003]=9211020;
a[4004]=9211020;
a[4005]=6513507;
a[4006]=2171169;
a[4802]=2697513;
a[4803]=4342338;
a[4804]=3750201;
a[4805]=2171169;
break;
case 8:
a[3]=526344;
a[4]=1052688;
a[801]=526344;
a[802]=3223857;
a[803]=4868682;
a[804]=5395026;
a[805]=4342338;
a[806]=1579032;
a[1601]=3223857;
a[1602]=6513507;
a[1603]=9211020;
a[1604]=9737364;
a[1605]=8092539;
a[1606]=4868682;
a[1607]=1052688;
a[2400]=526344;
a[2401]=4868682;
a[2402]=9211020;
a[2403]=12434877;
a[2404]=14079702;
a[2405]=10855845;
a[2406]=7039851;
a[2407]=2697513;
a[3200]=1052688;
a[3201]=5395026;
a[3202]=9737364;
a[3203]=14079702;
a[3204]=15724527;
a[3205]=11908533;
a[3206]=7566195;
a[3207]=3223857;
a[4001]=4342338;
a[4002]=8092539;
a[4003]=10855845;
a[4004]=11908533;
a[4005]=9737364;
a[4006]=5921370;
a[4007]=2171169;
a[4801]=1579032;
a[4802]=4868682;
a[4803]=7039851;
a[4804]=7566195;
a[4805]=5921370;
a[4806]=3223857;
a[5602]=1052688;
a[5603]=2697513;
a[5604]=3223857;
a[5605]=2171169;
break;
case 9:
a[4]=1052688;
a[5]=1052688;
a[802]=1579032;
a[803]=3223857;
a[804]=4342338;
a[805]=4342338;
a[806]=2697513;
a[807]=1052688;
a[1601]=1579032;
a[1602]=4868682;
a[1603]=7039851;
a[1604]=8684676;
a[1605]=8092539;
a[1606]=6513507;
a[1607]=4342338;
a[1608]=526344;
a[2401]=3223857;
a[2402]=7039851;
a[2403]=9737364;
a[2404]=11908533;
a[2405]=11382189;
a[2406]=8684676;
a[2407]=5921370;
a[2408]=2171169;
a[3200]=1052688;
a[3201]=4342338;
a[3202]=8684676;
a[3203]=11908533;
a[3204]=15724527;
a[3205]=14606046;
a[3206]=10855845;
a[3207]=7566195;
a[3208]=3223857;
a[4000]=1052688;
a[4001]=4342338;
a[4002]=8092539;
a[4003]=11382189;
a[4004]=14606046;
a[4005]=14079702;
a[4006]=10263708;
a[4007]=7566195;
a[4008]=3223857;
a[4801]=2697513;
a[4802]=6513507;
a[4803]=8684676;
a[4804]=10855845;
a[4805]=10263708;
a[4806]=8092539;
a[4807]=5395026;
a[4808]=1579032;
a[5601]=1052688;
a[5602]=4342338;
a[5603]=5921370;
a[5604]=7566195;
a[5605]=7566195;
a[5606]=5395026;
a[5607]=3223857;
a[6402]=526344;
a[6403]=2171169;
a[6404]=3223857;
a[6405]=3223857;
a[6406]=1579032;
break;
case 10:
a[4]=526344;
a[5]=1052688;
a[6]=526344;
a[802]=1052688;
a[803]=2697513;
a[804]=3750201;
a[805]=4342338;
a[806]=3750201;
a[807]=2171169;
a[808]=526344;
a[1601]=1052688;
a[1602]=3223857;
a[1603]=5395026;
a[1604]=7039851;
a[1605]=7566195;
a[1606]=6513507;
a[1607]=4868682;
a[1608]=2697513;
a[2401]=2697513;
a[2402]=5395026;
a[2403]=8092539;
a[2404]=10263708;
a[2405]=10855845;
a[2406]=9211020;
a[2407]=7039851;
a[2408]=4868682;
a[2409]=1579032;
a[3200]=526344;
a[3201]=3750201;
a[3202]=7039851;
a[3203]=10263708;
a[3204]=12434877;
a[3205]=14079702;
a[3206]=11908533;
a[3207]=9211020;
a[3208]=5921370;
a[3209]=2697513;
a[4000]=1052688;
a[4001]=4342338;
a[4002]=7566195;
a[4003]=10855845;
a[4004]=14079702;
a[4005]=15724527;
a[4006]=13027014;
a[4007]=9737364;
a[4008]=6513507;
a[4009]=3223857;
a[4800]=526344;
a[4801]=3750201;
a[4802]=6513507;
a[4803]=9211020;
a[4804]=11908533;
a[4805]=13027014;
a[4806]=11382189;
a[4807]=8684676;
a[4808]=5395026;
a[4809]=2697513;
a[5601]=2171169;
a[5602]=4868682;
a[5603]=7039851;
a[5604]=9211020;
a[5605]=9737364;
a[5606]=8684676;
a[5607]=6513507;
a[5608]=3750201;
a[5609]=1052688;
a[6401]=526344;
a[6402]=2697513;
a[6403]=4868682;
a[6404]=5921370;
a[6405]=6513507;
a[6406]=5395026;
a[6407]=3750201;
a[6408]=2171169;
a[7203]=1579032;
a[7204]=2697513;
a[7205]=3223857;
a[7206]=2697513;
a[7207]=1052688;
break;
case 11:
a[4]=526344;
a[5]=1052688;
a[6]=1052688;
a[803]=1052688;
a[804]=2697513;
a[805]=3223857;
a[806]=3223857;
a[807]=2171169;
a[808]=526344;
a[1602]=2171169;
a[1603]=3750201;
a[1604]=5395026;
a[1605]=6513507;
a[1606]=6513507;
a[1607]=5395026;
a[1608]=3223857;
a[1609]=1052688;
a[2401]=1052688;
a[2402]=3750201;
a[2403]=6513507;
a[2404]=8684676;
a[2405]=9737364;
a[2406]=9211020;
a[2407]=8092539;
a[2408]=5921370;
a[2409]=3223857;
a[2410]=526344;
a[3200]=526344;
a[3201]=2697513;
a[3202]=5395026;
a[3203]=8684676;
a[3204]=11382189;
a[3205]=13027014;
a[3206]=12434877;
a[3207]=10263708;
a[3208]=7566195;
a[3209]=4868682;
a[3210]=1579032;
a[4000]=1052688;
a[4001]=3223857;
a[4002]=6513507;
a[4003]=9737364;
a[4004]=13027014;
a[4005]=15724527;
a[4006]=14606046;
a[4007]=11908533;
a[4008]=8684676;
a[4009]=5395026;
a[4010]=2171169;
a[4800]=1052688;
a[4801]=3223857;
a[4802]=6513507;
a[4803]=9211020;
a[4804]=12434877;
a[4805]=14606046;
a[4806]=14079702;
a[4807]=11382189;
a[4808]=8092539;
a[4809]=5395026;
a[4810]=2171169;
a[5601]=2171169;
a[5602]=5395026;
a[5603]=8092539;
a[5604]=10263708;
a[5605]=11908533;
a[5606]=11382189;
a[5607]=9737364;
a[5608]=7039851;
a[5609]=4342338;
a[5610]=1052688;
a[6401]=526344;
a[6402]=3223857;
a[6403]=5921370;
a[6404]=7566195;
a[6405]=8684676;
a[6406]=8092539;
a[6407]=7039851;
a[6408]=4868682;
a[6409]=2697513;
a[7202]=1052688;
a[7203]=3223857;
a[7204]=4868682;
a[7205]=5395026;
a[7206]=5395026;
a[7207]=4342338;
a[7208]=2697513;
a[7209]=526344;
a[8003]=526344;
a[8004]=1579032;
a[8005]=2171169;
a[8006]=2171169;
a[8007]=1052688;
break;
case 12:
a[5]=1052688;
a[6]=1052688;
a[7]=526344;
a[803]=1052688;
a[804]=2171169;
a[805]=3223857;
a[806]=3223857;
a[807]=2697513;
a[808]=1579032;
a[1602]=2171169;
a[1603]=3750201;
a[1604]=5395026;
a[1605]=6513507;
a[1606]=6513507;
a[1607]=5921370;
a[1608]=4868682;
a[1609]=2697513;
a[1610]=1052688;
a[2401]=1052688;
a[2402]=3750201;
a[2403]=6513507;
a[2404]=8092539;
a[2405]=9211020;
a[2406]=9737364;
a[2407]=9211020;
a[2408]=7039851;
a[2409]=4868682;
a[2410]=3223857;
a[2411]=526344;
a[3201]=2171169;
a[3202]=5395026;
a[3203]=8092539;
a[3204]=9737364;
a[3205]=11382189;
a[3206]=11908533;
a[3207]=10855845;
a[3208]=8684676;
a[3209]=5921370;
a[3210]=4342338;
a[3211]=1052688;
a[4000]=1052688;
a[4001]=3223857;
a[4002]=6513507;
a[4003]=9211020;
a[4004]=11382189;
a[4005]=14079702;
a[4006]=14606046;
a[4007]=13553358;
a[4008]=10263708;
a[4009]=7566195;
a[4010]=5395026;
a[4011]=2171169;
a[4800]=1052688;
a[4801]=3223857;
a[4802]=6513507;
a[4803]=9737364;
a[4804]=11908533;
a[4805]=14606046;
a[4806]=15724527;
a[4807]=14079702;
a[4808]=10855845;
a[4809]=7566195;
a[4810]=5395026;
a[4811]=2171169;
a[5600]=526344;
a[5601]=2697513;
a[5602]=5921370;
a[5603]=9211020;
a[5604]=10855845;
a[5605]=13553358;
a[5606]=14079702;
a[5607]=12434877;
a[5608]=10263708;
a[5609]=7039851;
a[5610]=4868682;
a[5611]=1579032;
a[6401]=1579032;
a[6402]=4868682;
a[6403]=7039851;
a[6404]=8684676;
a[6405]=10263708;
a[6406]=10855845;
a[6407]=10263708;
a[6408]=8092539;
a[6409]=5395026;
a[6410]=3750201;
a[6411]=526344;
a[7202]=2697513;
a[7203]=4868682;
a[7204]=5921370;
a[7205]=7566195;
a[7206]=7566195;
a[7207]=7039851;
a[7208]=5395026;
a[7209]=3223857;
a[7210]=1579032;
a[8002]=1052688;
a[8003]=3223857;
a[8004]=4342338;
a[8005]=5395026;
a[8006]=5395026;
a[8007]=4868682;
a[8008]=3750201;
a[8009]=1579032;
a[8010]=526344;
a[8803]=526344;
a[8804]=1052688;
a[8805]=2171169;
a[8806]=2171169;
a[8807]=1579032;
a[8808]=526344;
break;
case 13:
a[5]=526344;
a[6]=1052688;
a[7]=1052688;
a[8]=526344;
a[803]=526344;
a[804]=1579032;
a[805]=2697513;
a[806]=3223857;
a[807]=3223857;
a[808]=2697513;
a[809]=1052688;
a[1602]=526344;
a[1603]=2697513;
a[1604]=3750201;
a[1605]=4868682;
a[1606]=5395026;
a[1607]=5395026;
a[1608]=4868682;
a[1609]=3223857;
a[1610]=1579032;
a[2401]=526344;
a[2402]=2697513;
a[2403]=4868682;
a[2404]=6513507;
a[2405]=8092539;
a[2406]=8684676;
a[2407]=8092539;
a[2408]=7566195;
a[2409]=5921370;
a[2410]=4342338;
a[2411]=1579032;
a[3201]=1579032;
a[3202]=3750201;
a[3203]=6513507;
a[3204]=8092539;
a[3205]=10263708;
a[3206]=10855845;
a[3207]=10263708;
a[3208]=9211020;
a[3209]=7039851;
a[3210]=5395026;
a[3211]=2697513;
a[3212]=526344;
a[4000]=526344;
a[4001]=2697513;
a[4002]=4868682;
a[4003]=8092539;
a[4004]=10263708;
a[4005]=12434877;
a[4006]=14079702;
a[4007]=13553358;
a[4008]=11908533;
a[4009]=9211020;
a[4010]=7039851;
a[4011]=3750201;
a[4012]=1579032;
a[4800]=1052688;
a[4801]=3223857;
a[4802]=5395026;
a[4803]=8684676;
a[4804]=10855845;
a[4805]=14079702;
a[4806]=15724527;
a[4807]=14606046;
a[4808]=13027014;
a[4809]=9737364;
a[4810]=7566195;
a[4811]=4342338;
a[4812]=2171169;
a[5600]=1052688;
a[5601]=3223857;
a[5602]=5395026;
a[5603]=8092539;
a[5604]=10263708;
a[5605]=13553358;
a[5606]=14606046;
a[5607]=14079702;
a[5608]=12434877;
a[5609]=9211020;
a[5610]=7566195;
a[5611]=4342338;
a[5612]=2171169;
a[6400]=526344;
a[6401]=2697513;
a[6402]=4868682;
a[6403]=7566195;
a[6404]=9211020;
a[6405]=11908533;
a[6406]=13027014;
a[6407]=12434877;
a[6408]=11382189;
a[6409]=8684676;
a[6410]=6513507;
a[6411]=3750201;
a[6412]=1579032;
a[7201]=1052688;
a[7202]=3223857;
a[7203]=5921370;
a[7204]=7039851;
a[7205]=9211020;
a[7206]=9737364;
a[7207]=9211020;
a[7208]=8684676;
a[7209]=6513507;
a[7210]=4868682;
a[7211]=2171169;
a[7212]=526344;
a[8002]=1579032;
a[8003]=4342338;
a[8004]=5395026;
a[8005]=7039851;
a[8006]=7566195;
a[8007]=7566195;
a[8008]=6513507;
a[8009]=4868682;
a[8010]=3223857;
a[8011]=1052688;
a[8803]=1579032;
a[8804]=2697513;
a[8805]=3750201;
a[8806]=4342338;
a[8807]=4342338;
a[8808]=3750201;
a[8809]=2171169;
a[8810]=1052688;
a[9604]=526344;
a[9605]=1579032;
a[9606]=2171169;
a[9607]=2171169;
a[9608]=1579032;
a[9609]=526344;
break;
case 14:
a[5]=526344;
a[6]=1052688;
a[7]=1052688;
a[8]=526344;
a[804]=1579032;
a[805]=2697513;
a[806]=3223857;
a[807]=3223857;
a[808]=2697513;
a[809]=2171169;
a[810]=1052688;
a[1602]=526344;
a[1603]=1579032;
a[1604]=3750201;
a[1605]=4868682;
a[1606]=5395026;
a[1607]=5395026;
a[1608]=4868682;
a[1609]=4342338;
a[1610]=3223857;
a[1611]=1052688;
a[2402]=1579032;
a[2403]=3223857;
a[2404]=5395026;
a[2405]=6513507;
a[2406]=7566195;
a[2407]=7566195;
a[2408]=7039851;
a[2409]=5921370;
a[2410]=4868682;
a[2411]=2697513;
a[2412]=1052688;
a[3201]=1579032;
a[3202]=3750201;
a[3203]=5395026;
a[3204]=8092539;
a[3205]=9211020;
a[3206]=10263708;
a[3207]=10855845;
a[3208]=10263708;
a[3209]=8684676;
a[3210]=7039851;
a[3211]=4868682;
a[3212]=2697513;
a[3213]=526344;
a[4000]=526344;
a[4001]=2697513;
a[4002]=4868682;
a[4003]=6513507;
a[4004]=9211020;
a[4005]=11382189;
a[4006]=12434877;
a[4007]=13027014;
a[4008]=11908533;
a[4009]=10263708;
a[4010]=8684676;
a[4011]=5395026;
a[4012]=3750201;
a[4013]=1579032;
a[4800]=1052688;
a[4801]=3223857;
a[4802]=5395026;
a[4803]=7566195;
a[4804]=10263708;
a[4805]=12434877;
a[4806]=14079702;
a[4807]=14606046;
a[4808]=13553358;
a[4809]=11382189;
a[4810]=9211020;
a[4811]=6513507;
a[4812]=4342338;
a[4813]=2171169;
a[5600]=1052688;
a[5601]=3223857;
a[5602]=5395026;
a[5603]=7566195;
a[5604]=10855845;
a[5605]=13027014;
a[5606]=14606046;
a[5607]=15724527;
a[5608]=14079702;
a[5609]=11908533;
a[5610]=9737364;
a[5611]=6513507;
a[5612]=4342338;
a[5613]=2171169;
a[6400]=526344;
a[6401]=2697513;
a[6402]=4868682;
a[6403]=7039851;
a[6404]=10263708;
a[6405]=11908533;
a[6406]=13553358;
a[6407]=14079702;
a[6408]=12434877;
a[6409]=10855845;
a[6410]=9211020;
a[6411]=5921370;
a[6412]=3750201;
a[6413]=1579032;
a[7201]=2171169;
a[7202]=4342338;
a[7203]=5921370;
a[7204]=8684676;
a[7205]=10263708;
a[7206]=11382189;
a[7207]=11908533;
a[7208]=10855845;
a[7209]=9737364;
a[7210]=8092539;
a[7211]=5395026;
a[7212]=3223857;
a[7213]=1052688;
a[8001]=1052688;
a[8002]=3223857;
a[8003]=4868682;
a[8004]=7039851;
a[8005]=8684676;
a[8006]=9211020;
a[8007]=9737364;
a[8008]=9211020;
a[8009]=8092539;
a[8010]=6513507;
a[8011]=3750201;
a[8012]=2171169;
a[8013]=526344;
a[8802]=1052688;
a[8803]=2697513;
a[8804]=4868682;
a[8805]=5395026;
a[8806]=6513507;
a[8807]=6513507;
a[8808]=5921370;
a[8809]=5395026;
a[8810]=3750201;
a[8811]=2171169;
a[8812]=526344;
a[9603]=1052688;
a[9604]=2697513;
a[9605]=3750201;
a[9606]=4342338;
a[9607]=4342338;
a[9608]=3750201;
a[9609]=3223857;
a[9610]=2171169;
a[9611]=526344;
a[10404]=526344;
a[10405]=1579032;
a[10406]=2171169;
a[10407]=2171169;
a[10408]=1579032;
a[10409]=1052688;
a[10410]=526344;
break;
case 15:
a[6]=526344;
a[7]=1052688;
a[8]=1052688;
a[9]=526344;
a[804]=1052688;
a[805]=2171169;
a[806]=2697513;
a[807]=3223857;
a[808]=3223857;
a[809]=2697513;
a[810]=1579032;
a[811]=526344;
a[1602]=526344;
a[1603]=1579032;
a[1604]=3223857;
a[1605]=4342338;
a[1606]=4868682;
a[1607]=5395026;
a[1608]=5395026;
a[1609]=4868682;
a[1610]=3750201;
a[1611]=2697513;
a[1612]=1052688;
a[2402]=1579032;
a[2403]=3223857;
a[2404]=4868682;
a[2405]=5921370;
a[2406]=7039851;
a[2407]=7566195;
a[2408]=7566195;
a[2409]=6513507;
a[2410]=5395026;
a[2411]=4342338;
a[2412]=2697513;
a[2413]=1052688;
a[3201]=1052688;
a[3202]=3223857;
a[3203]=4868682;
a[3204]=6513507;
a[3205]=8092539;
a[3206]=9211020;
a[3207]=9737364;
a[3208]=9211020;
a[3209]=8684676;
a[3210]=7039851;
a[3211]=5921370;
a[3212]=3750201;
a[3213]=2171169;
a[3214]=526344;
a[4001]=2171169;
a[4002]=4342338;
a[4003]=5921370;
a[4004]=8092539;
a[4005]=9737364;
a[4006]=10855845;
a[4007]=11908533;
a[4008]=11382189;
a[4009]=10263708;
a[4010]=8684676;
a[4011]=7039851;
a[4012]=5395026;
a[4013]=3223857;
a[4014]=1052688;
a[4800]=526344;
a[4801]=2697513;
a[4802]=4868682;
a[4803]=7039851;
a[4804]=9211020;
a[4805]=10855845;
a[4806]=12434877;
a[4807]=14079702;
a[4808]=13553358;
a[4809]=11908533;
a[4810]=10263708;
a[4811]=8092539;
a[4812]=5921370;
a[4813]=3750201;
a[4814]=1579032;
a[5600]=1052688;
a[5601]=3223857;
a[5602]=5395026;
a[5603]=7566195;
a[5604]=9737364;
a[5605]=11908533;
a[5606]=14079702;
a[5607]=15724527;
a[5608]=14606046;
a[5609]=13027014;
a[5610]=10855845;
a[5611]=8684676;
a[5612]=6513507;
a[5613]=4342338;
a[5614]=2171169;
a[6400]=1052688;
a[6401]=3223857;
a[6402]=5395026;
a[6403]=7566195;
a[6404]=9211020;
a[6405]=11382189;
a[6406]=13553358;
a[6407]=14606046;
a[6408]=14079702;
a[6409]=12434877;
a[6410]=10263708;
a[6411]=8092539;
a[6412]=6513507;
a[6413]=4342338;
a[6414]=2171169;
a[7200]=526344;
a[7201]=2697513;
a[7202]=4868682;
a[7203]=6513507;
a[7204]=8684676;
a[7205]=10263708;
a[7206]=11908533;
a[7207]=13027014;
a[7208]=12434877;
a[7209]=11382189;
a[7210]=9211020;
a[7211]=7566195;
a[7212]=5395026;
a[7213]=3750201;
a[7214]=1579032;
a[8001]=1579032;
a[8002]=3750201;
a[8003]=5395026;
a[8004]=7039851;
a[8005]=8684676;
a[8006]=10263708;
a[8007]=10855845;
a[8008]=10263708;
a[8009]=9211020;
a[8010]=8092539;
a[8011]=6513507;
a[8012]=4868682;
a[8013]=2697513;
a[8014]=526344;
a[8801]=526344;
a[8802]=2697513;
a[8803]=4342338;
a[8804]=5921370;
a[8805]=7039851;
a[8806]=8092539;
a[8807]=8684676;
a[8808]=8092539;
a[8809]=7566195;
a[8810]=6513507;
a[8811]=4868682;
a[8812]=3223857;
a[8813]=1579032;
a[9602]=1052688;
a[9603]=2697513;
a[9604]=3750201;
a[9605]=5395026;
a[9606]=5921370;
a[9607]=6513507;
a[9608]=6513507;
a[9609]=5395026;
a[9610]=4868682;
a[9611]=3223857;
a[9612]=2171169;
a[9613]=526344;
a[10403]=1052688;
a[10404]=2171169;
a[10405]=3223857;
a[10406]=3750201;
a[10407]=4342338;
a[10408]=4342338;
a[10409]=3750201;
a[10410]=2697513;
a[10411]=1579032;
a[10412]=526344;
a[11204]=526344;
a[11205]=1052688;
a[11206]=1579032;
a[11207]=2171169;
a[11208]=2171169;
a[11209]=1579032;
a[11210]=526344;
break;
case 16:
a[6]=526344;
a[7]=1052688;
a[8]=1052688;
a[9]=526344;
a[804]=1052688;
a[805]=2171169;
a[806]=2697513;
a[807]=3223857;
a[808]=3223857;
a[809]=2697513;
a[810]=2171169;
a[811]=1052688;
a[1602]=526344;
a[1603]=1579032;
a[1604]=3223857;
a[1605]=4342338;
a[1606]=4868682;
a[1607]=5395026;
a[1608]=5395026;
a[1609]=4868682;
a[1610]=4342338;
a[1611]=3223857;
a[1612]=1579032;
a[1613]=526344;
a[2402]=1579032;
a[2403]=3223857;
a[2404]=4868682;
a[2405]=5921370;
a[2406]=7039851;
a[2407]=7566195;
a[2408]=7566195;
a[2409]=7039851;
a[2410]=5921370;
a[2411]=4868682;
a[2412]=3223857;
a[2413]=1579032;
a[3201]=1052688;
a[3202]=3223857;
a[3203]=4868682;
a[3204]=6513507;
a[3205]=8092539;
a[3206]=9211020;
a[3207]=9737364;
a[3208]=9737364;
a[3209]=9211020;
a[3210]=8092539;
a[3211]=6513507;
a[3212]=4868682;
a[3213]=3223857;
a[3214]=1052688;
a[4001]=2171169;
a[4002]=4342338;
a[4003]=5921370;
a[4004]=8092539;
a[4005]=9737364;
a[4006]=10855845;
a[4007]=11908533;
a[4008]=11908533;
a[4009]=10855845;
a[4010]=9737364;
a[4011]=8092539;
a[4012]=5921370;
a[4013]=4342338;
a[4014]=2171169;
a[4800]=526344;
a[4801]=2697513;
a[4802]=4868682;
a[4803]=7039851;
a[4804]=9211020;
a[4805]=10855845;
a[4806]=12434877;
a[4807]=14079702;
a[4808]=14079702;
a[4809]=12434877;
a[4810]=10855845;
a[4811]=9211020;
a[4812]=7039851;
a[4813]=4868682;
a[4814]=2697513;
a[4815]=526344;
a[5600]=1052688;
a[5601]=3223857;
a[5602]=5395026;
a[5603]=7566195;
a[5604]=9737364;
a[5605]=11908533;
a[5606]=14079702;
a[5607]=15724527;
a[5608]=15724527;
a[5609]=14079702;
a[5610]=11908533;
a[5611]=9737364;
a[5612]=7566195;
a[5613]=5395026;
a[5614]=3223857;
a[5615]=1052688;
a[6400]=1052688;
a[6401]=3223857;
a[6402]=5395026;
a[6403]=7566195;
a[6404]=9737364;
a[6405]=11908533;
a[6406]=14079702;
a[6407]=15724527;
a[6408]=15724527;
a[6409]=14079702;
a[6410]=11908533;
a[6411]=9737364;
a[6412]=7566195;
a[6413]=5395026;
a[6414]=3223857;
a[6415]=1052688;
a[7200]=526344;
a[7201]=2697513;
a[7202]=4868682;
a[7203]=7039851;
a[7204]=9211020;
a[7205]=10855845;
a[7206]=12434877;
a[7207]=14079702;
a[7208]=14079702;
a[7209]=12434877;
a[7210]=10855845;
a[7211]=9211020;
a[7212]=7039851;
a[7213]=4868682;
a[7214]=2697513;
a[7215]=526344;
a[8001]=2171169;
a[8002]=4342338;
a[8003]=5921370;
a[8004]=8092539;
a[8005]=9737364;
a[8006]=10855845;
a[8007]=11908533;
a[8008]=11908533;
a[8009]=10855845;
a[8010]=9737364;
a[8011]=8092539;
a[8012]=5921370;
a[8013]=4342338;
a[8014]=2171169;
a[8801]=1052688;
a[8802]=3223857;
a[8803]=4868682;
a[8804]=6513507;
a[8805]=8092539;
a[8806]=9211020;
a[8807]=9737364;
a[8808]=9737364;
a[8809]=9211020;
a[8810]=8092539;
a[8811]=6513507;
a[8812]=4868682;
a[8813]=3223857;
a[8814]=1052688;
a[9602]=1579032;
a[9603]=3223857;
a[9604]=4868682;
a[9605]=5921370;
a[9606]=7039851;
a[9607]=7566195;
a[9608]=7566195;
a[9609]=7039851;
a[9610]=5921370;
a[9611]=4868682;
a[9612]=3223857;
a[9613]=1579032;
a[10402]=526344;
a[10403]=1579032;
a[10404]=3223857;
a[10405]=4342338;
a[10406]=4868682;
a[10407]=5395026;
a[10408]=5395026;
a[10409]=4868682;
a[10410]=4342338;
a[10411]=3223857;
a[10412]=1579032;
a[10413]=526344;
a[11204]=1052688;
a[11205]=2171169;
a[11206]=2697513;
a[11207]=3223857;
a[11208]=3223857;
a[11209]=2697513;
a[11210]=2171169;
a[11211]=1052688;
a[12006]=526344;
a[12007]=1052688;
a[12008]=1052688;
a[12009]=526344;
break;
case 17:
a[6]=526344;
a[7]=1052688;
a[8]=1052688;
a[9]=1052688;
a[10]=526344;
a[805]=526344;
a[806]=1579032;
a[807]=2171169;
a[808]=2171169;
a[809]=2171169;
a[810]=1579032;
a[811]=1052688;
a[812]=526344;
a[1603]=526344;
a[1604]=1579032;
a[1605]=2697513;
a[1606]=3750201;
a[1607]=4342338;
a[1608]=4342338;
a[1609]=4342338;
a[1610]=3750201;
a[1611]=3223857;
a[1612]=2171169;
a[1613]=1052688;
a[2402]=526344;
a[2403]=2171169;
a[2404]=3223857;
a[2405]=4868682;
a[2406]=5395026;
a[2407]=6513507;
a[2408]=6513507;
a[2409]=6513507;
a[2410]=5921370;
a[2411]=5395026;
a[2412]=3750201;
a[2413]=2697513;
a[2414]=1052688;
a[3202]=1579032;
a[3203]=3223857;
a[3204]=4868682;
a[3205]=6513507;
a[3206]=7566195;
a[3207]=8092539;
a[3208]=8684676;
a[3209]=8684676;
a[3210]=8092539;
a[3211]=7039851;
a[3212]=5921370;
a[3213]=4342338;
a[3214]=2697513;
a[3215]=526344;
a[4001]=526344;
a[4002]=2697513;
a[4003]=4868682;
a[4004]=6513507;
a[4005]=8092539;
a[4006]=9211020;
a[4007]=10263708;
a[4008]=10855845;
a[4009]=10855845;
a[4010]=10263708;
a[4011]=8684676;
a[4012]=7039851;
a[4013]=5395026;
a[4014]=3750201;
a[4015]=1579032;
a[4800]=526344;
a[4801]=1579032;
a[4802]=3750201;
a[4803]=5395026;
a[4804]=7566195;
a[4805]=9211020;
a[4806]=11382189;
a[4807]=12434877;
a[4808]=13027014;
a[4809]=13027014;
a[4810]=11908533;
a[4811]=10263708;
a[4812]=8684676;
a[4813]=6513507;
a[4814]=4868682;
a[4815]=2697513;
a[4816]=526344;
a[5600]=1052688;
a[5601]=2171169;
a[5602]=4342338;
a[5603]=6513507;
a[5604]=8092539;
a[5605]=10263708;
a[5606]=12434877;
a[5607]=14079702;
a[5608]=15198183;
a[5609]=14606046;
a[5610]=13553358;
a[5611]=11382189;
a[5612]=9211020;
a[5613]=7566195;
a[5614]=5395026;
a[5615]=3223857;
a[5616]=1052688;
a[6400]=1052688;
a[6401]=2171169;
a[6402]=4342338;
a[6403]=6513507;
a[6404]=8684676;
a[6405]=10855845;
a[6406]=13027014;
a[6407]=15198183;
a[6408]=16250871;
a[6409]=16250871;
a[6410]=14079702;
a[6411]=11908533;
a[6412]=9737364;
a[6413]=7566195;
a[6414]=5395026;
a[6415]=3223857;
a[6416]=1052688;
a[7200]=1052688;
a[7201]=2171169;
a[7202]=4342338;
a[7203]=6513507;
a[7204]=8684676;
a[7205]=10855845;
a[7206]=13027014;
a[7207]=14606046;
a[7208]=16250871;
a[7209]=15724527;
a[7210]=14079702;
a[7211]=11908533;
a[7212]=9737364;
a[7213]=7566195;
a[7214]=5395026;
a[7215]=3223857;
a[7216]=1052688;
a[8000]=526344;
a[8001]=1579032;
a[8002]=3750201;
a[8003]=5921370;
a[8004]=8092539;
a[8005]=10263708;
a[8006]=11908533;
a[8007]=13553358;
a[8008]=14079702;
a[8009]=14079702;
a[8010]=12434877;
a[8011]=10855845;
a[8012]=9211020;
a[8013]=7039851;
a[8014]=4868682;
a[8015]=2697513;
a[8016]=526344;
a[8801]=1052688;
a[8802]=3223857;
a[8803]=5395026;
a[8804]=7039851;
a[8805]=8684676;
a[8806]=10263708;
a[8807]=11382189;
a[8808]=11908533;
a[8809]=11908533;
a[8810]=10855845;
a[8811]=9737364;
a[8812]=8092539;
a[8813]=5921370;
a[8814]=4342338;
a[8815]=2171169;
a[9601]=526344;
a[9602]=2171169;
a[9603]=3750201;
a[9604]=5921370;
a[9605]=7039851;
a[9606]=8684676;
a[9607]=9211020;
a[9608]=9737364;
a[9609]=9737364;
a[9610]=9211020;
a[9611]=8092539;
a[9612]=6513507;
a[9613]=4868682;
a[9614]=3223857;
a[9615]=1052688;
a[10402]=1052688;
a[10403]=2697513;
a[10404]=4342338;
a[10405]=5395026;
a[10406]=6513507;
a[10407]=7566195;
a[10408]=7566195;
a[10409]=7566195;
a[10410]=7039851;
a[10411]=5921370;
a[10412]=4868682;
a[10413]=3223857;
a[10414]=1579032;
a[11203]=1052688;
a[11204]=2697513;
a[11205]=3750201;
a[11206]=4868682;
a[11207]=5395026;
a[11208]=5395026;
a[11209]=5395026;
a[11210]=4868682;
a[11211]=4342338;
a[11212]=3223857;
a[11213]=1579032;
a[11214]=526344;
a[12004]=526344;
a[12005]=1579032;
a[12006]=2697513;
a[12007]=3223857;
a[12008]=3223857;
a[12009]=3223857;
a[12010]=2697513;
a[12011]=2171169;
a[12012]=1052688;
a[12806]=526344;
a[12807]=1052688;
a[12808]=1052688;
a[12809]=1052688;
a[12810]=526344;
break;
case 18:
a[7]=526344;
a[8]=1052688;
a[9]=1052688;
a[10]=1052688;
a[11]=526344;
a[805]=526344;
a[806]=1052688;
a[807]=1579032;
a[808]=2171169;
a[809]=2171169;
a[810]=2171169;
a[811]=1579032;
a[812]=526344;
a[1603]=526344;
a[1604]=1579032;
a[1605]=2171169;
a[1606]=3223857;
a[1607]=3750201;
a[1608]=4342338;
a[1609]=4342338;
a[1610]=4342338;
a[1611]=3750201;
a[1612]=2697513;
a[1613]=1579032;
a[1614]=1052688;
a[2402]=526344;
a[2403]=2171169;
a[2404]=3223857;
a[2405]=3750201;
a[2406]=5395026;
a[2407]=5921370;
a[2408]=6513507;
a[2409]=6513507;
a[2410]=6513507;
a[2411]=5395026;
a[2412]=4868682;
a[2413]=3223857;
a[2414]=2697513;
a[2415]=1052688;
a[3202]=1579032;
a[3203]=3223857;
a[3204]=4868682;
a[3205]=5921370;
a[3206]=7039851;
a[3207]=8092539;
a[3208]=8684676;
a[3209]=8684676;
a[3210]=8092539;
a[3211]=7566195;
a[3212]=6513507;
a[3213]=4868682;
a[3214]=4342338;
a[3215]=2697513;
a[3216]=526344;
a[4001]=526344;
a[4002]=2171169;
a[4003]=3750201;
a[4004]=5921370;
a[4005]=6513507;
a[4006]=8092539;
a[4007]=9211020;
a[4008]=9737364;
a[4009]=9737364;
a[4010]=9211020;
a[4011]=8684676;
a[4012]=7039851;
a[4013]=5921370;
a[4014]=4868682;
a[4015]=3223857;
a[4016]=1052688;
a[4801]=1052688;
a[4802]=3223857;
a[4803]=5395026;
a[4804]=7039851;
a[4805]=8092539;
a[4806]=9737364;
a[4807]=10855845;
a[4808]=11908533;
```

“Madness!? This is INFOMOV!”

/INFOMOV/

END of “Practical”

next lecture: “Grand Recap”