nics & (depth < MAXDEF

: = inside } 1 ht = nt / nc, ddh bs2t = 1.0f - nnt D, N ); 3)

at a = nt - nc, b = 00 at Tr = 1 - (R0 + (1 - R0 Tr) R = (D <sup>=</sup> nnt - N <sup>-</sup> 000

= \* diffus = true;

• •fl + refr)) && (depth < MAXDEPTIO

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse )
estimation - doing it properly, closed
if;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y > 0) &&

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

# /INFOMOV/ Optimization & Vectorization

A. Chatzimparmpas - April-June 2025 - Lecture 12: "Fixed Point"

# Welcome!



Slides Courtesy of J. Bikker

ics & (depth k MAXDEFT

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - nc) Fr) R = (D <sup>=</sup> nnt - N - (30)

= \* diffuse = true;

efl + refr)) && (depth < MANDEPTIO

D, N ); refl \* E \* diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, doing if; radiance = SampleLight( &rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

andom walk - done properly, closely following Source /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Today's Agenda:

- Introduction
- Float to Fixed Point and Back
- Operations
- Fixed Point & Accuracy
- Demonstration



tics ≹(depth < NACCSS

: = inside ? | ht = nt / nc, ddn bs2t = 1.0f - nnt / n D, N ); 3)

t a = nt - nc, b = nt nt Tr = 1 - (R0 + (1 - R0 r) R = (D \* nnt - N \* (dom

= \* diffuse; = true;

efl + refr)) && (depth < NOCCEPTIO

D, N ); refl \* E \* diffuse = true;

(AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, loss if; radiance = SampleLight( &rand, I, &L, &L e.x + radiance.v + radiance.z) > 01 & 8

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Psi at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following Soli /ive)

; t3 Brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); Sion = true:

The Concept of Fixed Point Math

Basic idea: emulating floating point math using integers.

Why?

- Not every CPU has a floating point unit.
- Specifically: cheap DSPs do not support floating point.
  - Mixing floating point and integer is Good for the Pipes.
- Some floating point ops have long latencies (div).
  - Data conversion can be a significant part of a task.
  - Fixed point can be more accurate.





fics
{ (depth < PW00e)
t = inside ? 1 \*\*\*
t = nt / nc, ddt
s2t = 1.0f - nnt
0, N );
}
t a = nt - nc, l
</pre>

\* diffuse; = true;

```
efl + refr)) && (depth c
```

```
), N );
refl * E * diffuse;
= true;
```

```
(AXDEPTH)
```

```
survive = SurvivalProbability( diff
estimation - doing it properly, diff
radiance = SampleLight( &rand, I, &
e.x + radiance.y + radiance.z) > 0)
w = true;
at brdfPdf = EvaluateDiffuse( L, N
at3 factor = diffuse * INVPI;
```

```
ats factor = diffuse ~ lnvrl;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) *
```

```
andom walk - done properly, closely following Source
/ive)
```

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

Turing introduces a new processor architecture, the **Turing SM**, that delivers a dramatic boost in shading efficiency, achieving 50% improvement in delivered performance per CUDA Core compared to the Pascal generation. These improvements are enabled by two key architectural changes. First, the Turing SM adds a new independent integer datapath that can execute instructions concurrently with the floating-point math datapath. In previous generations, executing these instructions would have blocked floating-point instructions from issuing. Second, the SM memory path has been redesigned to unify shared memory, texture caching, and memory load caching into one unit. This translates to 2x more bandwidth and more than 2x more capacity available for L1 cache for common workloads.

```
vec3 shade( vec3 V, vec3 R )
{
   float spec = pow( max( dot( V, R), 0 ), 32 );
   return spec * lightColor;
```

Could we evaluate function shade without using floats?



The Concept of Fixed Point Math

Basic idea: we have  $\pi$ : 3.1415926536.

- Multiplying that by 10<sup>10</sup> yields 31415926536 (π).
- Adding 1 to  $\pi$  yields 4.1415926536 ( $\pi$  + 1).
- But, we scale up 1 by 10<sup>10</sup> as well: adding 1·10<sup>10</sup> to the scaled up version of π yields 41415926536.
- → In base 10, we simulate N digits of fractional precision by multiplying our numbers by 10<sup>N</sup> (and remember where we put that dot).

survive = SurvivalProbability( diffuse estimation - doing it properly, closed if; adiance = SampleLight( &rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && (dou .

at a = nt

), N );

(AXDEPTH)

efl + refr)) && (depth <

refl \* E \* diffuse;

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) = (rad);

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:



ics & (depth < Modes)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D <sup>=</sup> nnt - N <sup>-</sup> (d0

= \* diffuse; = true;

• efl + refr)) && (depth < MONDEPTION

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, closed if; radiance = SampleLight( &rand, I, &L, &L e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Psu at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec. /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

#### The Concept of Fixed Point Math

Addition and subtraction are straight-forward with fixed point math.

We can also use it for interpolation:

#### void line( int x1, int y1, int x2, int y2 )



The Concept of Fixed Point Math

For multiplication and division things get a bit more complex.

- $\pi \cdot 2 \equiv 31415926536 * 200000000 = 62831853072000000000$
- $\pi / 2 \equiv 31415926536 / 200000000 = 1$  (or 2, if we use proper rounding).

Multiplying two fixed point numbers yields a result that is  $10^{10}$  too large (in this case). Dividing two fixed point numbers yields a result that is  $10^{10}$  too small.

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

at a = nt

survive = SurvivalProbability( diffuse estimation - doing it properly, closed H; radiance = SampleLight( &rand, I, &L, &light 2.x + radiance.y + radiance.z) > 0) && (doct

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:



The Concept of Fixed Point Math

- Multiplying by 2<sup>16</sup> yields 205887.
- Adding  $1.2^{16}$  to the scaled-up version of  $\pi$  yields 271423.

- efl + refr)) && (depth
- ), N ); refl \* E \* diffuse;

AXDEPTH)

at a = ni

survive = SurvivalProbability( dif radiance = SampleLight( &rand, I, e.x + radiance.y + radiance.z) > 0)

v = true: at brdfPdf = EvaluateDiffuse( L, N

at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf

andom walk - done properly, closely fo. /ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 1 urvive; pdf; 1 = E \* brdf \* (dot( N, R ) / pdf); sion = tru

- In binary:
  - $205887 = 000000000000001100100000111111(\pi)$

Looking at the first number (205887), and splitting in two sets of 16 bit, we get:

00000000000011 (base 2) = 3 (base 10);  $0010010000111111 \text{ (base 2)} = 9279 \text{ (base 10)}; \frac{9279}{2}$ = 0.141586304.



On a computer, we obviously do not use base 10, but base 2. Starting with  $\pi$  again:

The Concept of Fixed Point Math

Interpolation, base 10:

void line( int x1, int y1, int x2, int y2 )



st3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apd urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

efl + refr)) && (depth c

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, 8L e.x + radiance.y + radiance.z) > 0)8

at brdfPdf = EvaluateDiffuse( L, N ) \*
at3 factor = diffuse \* INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf) \* andom walk - done properly, closely followi

refl \* E \* diffuse;

), N );

(AXDEPTH)

v = true;

/ive)

if;

The Concept of Fixed Point Math

Interpolation, base 2:

void line( int x1, int y1, int x2, int y2 )



st3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, dod urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

efl + refr)) && (depth <

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, 8L e.x + radiance.y + radiance.z) > 0) 8

at brdfPdf = EvaluateDiffuse( L, N ) \*
at3 factor = diffuse \* INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf) \* andom walk - done properly, closely followi

refl \* E \* diffuse;

), N );

(AXDEPTH)

v = true;

/ive)

if;

at Tr = 1 - (R0

efl + refr)) && (dep

refl \* E \* diffuse;

survive = SurvivalProbability( d

radiance = SampleLight( &rand, e.x + radiance.y + rad<u>iance.z</u>)

at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf

at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf

andom walk - done properly, closely fol

at3 brdf = SampleDiffuse( diffuse, N, r1,

i = E \* brdf \* (dot( N, R ) / pdf);

), N );

(AXDEPTH)

v = true:

/ive)

urvive; pdf;

sion = true:

The Concept of Fixed Point Math

#### How many bits do we need?

- The number 10.3 (base 10) has a maximum error of 0.05: 10.25 ≤ 10.3 < 10.35.</li>
- So, the error is at most  $\frac{1}{2} 10^{-X}$  (or x + 1) for x fractional digits.
- A fixed-point number with 16 fractional bits has a maximum error of  $\frac{1}{2}2^{-16} = 2^{-17}$ .
- ... or  $2^{-16}$  if we always round down (twice as much!).
- This can be prevented by adding  $\frac{1}{2}2^{-16}$  before flooring:

round(10.7) = floor(10.7 + 0.5) = 11.

#### During interpolation:

- If our longest line is Y pixels,
- → the maximum accumulated error with x fractional bits is: Y  $2^{-(x+1)}$ .
  - \*Only\* if the maximum error exceeds 1, the line may differ from 'ground truth'.

void line( int x1, int y1, int x2, int y2 )
{
 int dx = (x2 - x1) \* 65536;
 int dy = (y2 - y1) \* 65536;
 int pixels = max( abs( x2 - x1 ), abs( y2 - y1 ) );
 dx /= pixels;
 dy /= pixels;
 int x = x1 \* 65536, y = y1 \* 65536;
 for( int i = 0; i < pixels; i++, x += dx, y += dy )
 plot( x / 65536, y / 65536 );
}</pre>



u = u\_over\_z / z rz = 1 / z v = v\_over\_z / z u = u\_over\_z \* rz v = v\_over\_z \* rz

#### Practical example

Texture mapping in Quake 1: Perspective Correction

- Affine texture mapping: interpolate u/v linearly over polygon
- Perspective correct texture mapping: interpolate 1/z, u/z and v/z.
- Reconstruct u and v per pixel using the reciprocal of 1/z.

	Instruction	Operand	Clock cycles	Pairability	i-ov	fp-ov	
retr)) &&	FLD	r/m32/m64	1	0	0	0	l
; = * d\$ff.,	FLD	m80	3	np	0	0	l
2 uinu 2;	FBLD	m80	48-58	np	0	0	l
	FST(P)	r	1	np	0	0	l
TH)	FST(P)	m32/m64	2 m)	np	0	0	l
e = Surviv	FST(P)	m80	3 m)	np	0	0	l
ation - do	FBSTP	m80	148-154	np	0	0	l
e = Sampl	FILD	m	3	np	2	2	l
radiance.y	FIST(P)	m	6	np	0	0	l
ie; FPdf = Eva	FLDZ FLD1		2	np	0	0	l
tor = dif	FLDPI FLDL2E etc.		5 s)	np	2	2	l
ght = Mis2 ThetaOut =	FNSTSW	AX/m16	6 q)	np	0	0	l
weight *	FLDCW	m16	8	np	0	0	l
valk - don	FNSTCW	m16	2	np	0	0	l
	FADD(P)	r/m	3	0	2	2	l
uf c1	FSUB(R)(P)	r/m	3	0	2	2	l
i = Sampi	EMUL(P)	r/m	3	0	2	2 n)	
* brdf * (	FDIV(R)(P)	r/m	19/33/39 p)	0	38 o)	2	
true:	ECHS EARS		1		0		L





= true:

), N );

AXDEPTH)

v = true:

/ive)

efl + refr)) && (dept)

survive = SurvivalProbability(

radiance = SampleLight( &rand, I **x + radiance.v +** radiance.z)

at brdfPdf = EvaluateDiffuse( L, N at3 factor = diffuse \* INVPI at weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPd andom walk - done properly, closely f

refl \* E \* diffuse;

#### Practical example

Texture mapping in Quake 1: Perspective Correction

- Affine texture mapping: interpolate u/v linearly over polygon
- Perspective correct texture mapping: interpolate 1/z, u/z and v/z.
- Reconstruct u and v per pixel using the reciprocal of 1/z.

#### Quake's solution:

- Divide a horizontal line of pixels in segments of 8 pixels;
- Calculate u and v for the start and end of the segment;
- Interpolate linearly (fixed point!) over the 8 pixels.

#### And:

Start the floating point division (39 cycles) for the next segment, so it can complete while we execute integer code for the linear interpolation.



at3 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 1 urvive; pdf; 1 = E \* brdf \* (dot( N, R ) / pdf); sion = true

#### 13

ics & (depth < Moorente

: = inside ? 1 | 1 | 1 ht = nt / nc, ddh bs2t = 1.0f - nnt D, N ); 3)

at a = nt - nc, b = n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N

= \* diffuse = true;

efl + refr)) && (depth < MAGEPTIO

D, N ); refl \* E \* diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, doing if; radiance = SampleLight( &rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Psurvice at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (radi

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Today's Agenda:

- Introduction
- Float to Fixed Point and Back
- Operations
- Fixed Point & Accuracy
- Demonstration



#### Practical Things

Converting a floating point number to fixed point:

Multiply the float by a power of 2 <u>represented by a floating point value</u>, and cast the result to an integer. E.g.:

int fp\_pi = (int)(3.141593f \* 65536.0f); // 16 bits fractional
(rookie mistake: int fp\_pi = (int)3.141593f \* 65536.0f;

After calculations, cast the result to int by discarding the fractional bits. E.g.: int result = fp\_pi >> 16; // divide by 65536

```
estimation - doing it properly,

if;

radiance = SampleLight( &rand, I, &L,

e.x + radiance.y + radiance.z) > 0) &&
```

survive = SurvivalProbability( di

```
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * P
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf
```

andom walk - done properly, closely foll /ive)

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

at a = nt

), N );

AXDEPTH)

efl + refr)) && (depth

refl \* E \* diffuse;

Or, get the original float back by casting to float and dividing by  $2^{\text{fractionalbits}}$ :

```
float result = (float)fp_pi * (1.0f / 65536.0f);
```

Note that this last option has significant overhead, which should be outweighed by the gains.



at a = nt

), N );

(AXDEPTH)

v = true:

/ive)

ff:

efl + refr)) && (depth

survive = SurvivalProbability( dif

radiance = SampleLight( &rand, I

e.x + radiance.y + radiance.z) > 0

at brdfPdf = EvaluateDiffuse( L, |

E \* ((weight \* cosThetaOut) / directPdf) andom walk - done properly, closely folic

at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L );

refl \* E \* diffuse;

Practical Things - Considerations

Example: precomputed sin/cos table

```
#define FP_SCALE 65536.0f 1073741824.0f (or 2^30)
int sintab[256], costab[256];
for( int i = 0; i < 256; i++ )
    sintab[i] = (int)(FP_SCALE * sinf( (float)i / 128.0f * PI )),
    costab[i] = (int)(FP_SCALE * cosf( (float)i / 128.0f * PI ));</pre>
```

What is the best value for FP\_SCALE in this case? And should we use int or unsigned int for the table?

Sine/cosine: range is [-1, 1]. In this case, we need 1 sign bit, and 1 bit for the whole part of the number. So:

→ We use 30 bits for fractional precision, 1 for sign, 1 for range. In base 10, the fractional precision is ~10 digits (float has 7).



. t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, so urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

nics & (depth < ⊅00000

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt " ∩ 2, N ); 3)

nt a = nt - nc, b = 00 nt Tr = 1 - (R0 + (1 - 80 r) R = (D = nnt - 8 - 6

= \* diffuse; = true;

efl + refr)) && (depth < NACCEPT

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly ff; radiance = SampleLight( &rand, I, &L, e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse( L, N ) \* P at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf

andom walk - done properly, closely fo /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Bpdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

**Practical Things - Considerations** 

Example: values in a z-buffer

A 3D engine needs to keep track of the depth of pixels on the screen for depth sorting. For this, it uses a z-buffer.

We can make two observations:

- 1. All values are positive (no objects behind the camera are drawn);
- 2. Further away we need less precision.

By adding 1 to z, we guarantee that z is in the range [1..infinity]. The reciprocal of z is then in the range [0..1].

We store 1/(z+1) as a 0:32 unsigned fixed point number for

maximum precision.





tics & (depth < ⊅0000

: = inside ? 1 | 1 | 1 ht = nt / nc, ddn - 1 ps2t = 1.0f - nnt - 1 D, N ); 3)

nt a = nt - nc, b = nt nt Tr = 1 - (R0 + 11 - nc) ir) R = (D <sup>=</sup> nnt - N

\* diffuse; = true;

efl + refr)) && (depth < MODEPT

), N ); refl \* E \* diffuse; = true;

AXDEPTH)

w = true; at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / direct

andom walk - done properly, closely following Sm /ive)

, H33 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Bpdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

**Practical Things - Considerations** 

Example: particle simulation

A particle simulation operates on particles inside a 100x100x100 box centered around the origin. What fixed point format do you use for the coordinates of the particles?

1. Since all coordinates are in the range [-50,50], we need a sign.

- The maximum integer value of 50 fits in 6 bits  $(2^6 1 = 0.63)$ .
- 3. This leaves 25 bits fractional precision (a bit more than 8 decimal digits).

#### → We use a 6:25 signed fixed point representation.

Better: scale the simulation to a box of 127x127x127 for better use of the full range; this gets you ~8.5 decimal digits of precision.





18

ics € (depth < MocDar)

: = inside ? 1 : 1 : ht = nt / nc, ddn 1 : bs2t = 1.0f - nnt n: D, N ); ≷)

at a = nt - nc, b = 00 at Tr = 1 - (R0 + (1 - 00) Tr) R = (D = nnt - N = (30)

\* diffuse; = true;

. efl + refr)) && (depth < MAXDEP⊺)

), N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, close ff; radiance = SampleLight( &rand, I, &L, & 2.x + radiance.y + radiance.z) > 0) &#

w = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pst at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following set /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); Sion = true:

Practical Things - Considerations

We pick the right precision based on the problem at hand.

Sin/cos: original values [-1..1];

 $\rightarrow$  sign bit + 1 range/integer bit + 30 fractional bits;

 $\rightarrow$  1:30 signed fixed point.

Storing 1/(z+1): original values [0..1];

- $\rightarrow$  32 fractional bits;
- ➔ 0:32 unsigned fixed point.

Particles: original values [-50..50];

 $\rightarrow$  sign bit + 6 integer bits, 32-7=25 fractional bits;

 $\rightarrow$  6:25 signed fixed point.

In general:

- first determine if we need a sign;
- then, determine how many bits are need to represent the integer range;
- use the remainder as fractional bits.
- If too imprecise: use 64-bit integers (use sparsely on platforms that do not support this natively!).



ics & (depth k MAXDEFT-

: = inside ? 1 1 1 0 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N ); 3)

at a = nt - nc, b = 11 at Tr = 1 - (R0 + 1 - 10 Fr) R = (D = nnt - N

= \* diffuse = true;

• •fl + refr)) && (depth < MODEPTIO

D, N ); refl \* E \* diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, doing if; radiance = SampleLight( &rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Today's Agenda:

- Introduction
- Float to Fixed Point and Back
- <u>Operations</u>
- Fixed Point & Accuracy
- Demonstration



at a = nt

), N );

(AXDEPTH)

v = true:

/ive)

f:

efl + refr)) && (depth

survive = SurvivalProbability( di

radiance = SampleLight( &rand, I,

e.x + radiance.y + radiance.z) > 0

at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf )

refl \* E \* diffuse;

Basic Operations on Fixed Point Numbers

Operations on mixed fixed point formats:

• A+B  $(I_A: F_A + I_B: F_B)$ 

To be able to add the numbers, they must be in the same format.

Example: *I*<sub>*A*</sub>: *F*<sub>*A*</sub>=4:28, *I*<sub>*B*</sub>: *F*<sub>*B*</sub>=16:16

Option 1: A >>= 12 (to make it 16:16) Option 2: B <<= 12 (to make it 4:28)

Problem with option 2: we do not get 4:28, we get 16:28! Problem with option 1: we drop 12 bits from A...



at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) andom walk - done properly, closely follow

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

21

at a = ni

), N );

AXDEPTH)

v = true:

/ive)

at Tr = 1 - (R0

efl + refr)) && (dept)

survive = SurvivalProbability(

radiance = SampleLight( &rand, : e.x + radiance.v + radiance.z)

at brdfPdf = EvaluateDiffuse( L, N ) \* at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf) andom walk - done properly, closely follo

refl \* E \* diffuse;

**Basic Operations on Fixed Point Numbers** 

Operations on mixed fixed point formats:

• A\*B  $(I_A: F_A * I_B: F_B) \rightarrow$  yields  $(I_A+I_B): (F_A+F_B)$ 

We can however freely mix fixed point formats for multiplication.

#### Example:

```
I_A: F_A = 16:16, I_B: F_B = 16:16
Result: 32:32, shift to the right by 32 to get a <u>32</u>:16 number.
```

#### $I_A: F_A = 18:14, I_B: F_B = 14:18$

Result: 32:32, shift to the right by 18 to get a ..:14 number, or by 14 to get a ..:18 number.

#### Problem:

- The intermediate result doesn't fit in a 32-bit register.
- The answer may not fit in a 32-bit register.

# NOS SOL

#### , t33 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd urvive; .pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

22

#### **Multiplication**

Color scaling, base 2:

uint redblue = c & 0x00FF00FF; uint green = c & 0x0000FF00;

return (redblue + green) >> 8;

redblue = (redblue \* x) & 0xFF00FF00;

green = (green \* x) & 0x00FF0000;

= inside ? 1 t = nt / nc, ddn s2t = 1.0f - nnt , N ); )

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D <sup>=</sup> nnt - N = (d0

= \* diffuse; = true;

efl + refr)) && (depth < NACEPTIC

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, close H; radiance = SampleLight( &rand, I, &L, e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse( L, N ) \* at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPd

andom walk - done properly, closely follow /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, SR, Spdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

We did 0:8 \* 0:8 fixed point multiplication here, which yields 0:16 results. Fore more details, check: <u>https://jacco.ompf2.com/2020/05/12/opt3simd-part-1-of-2/</u>

31	24	23	16	15	8	7	0
31	24	23	16	15	8	7	

uint ScaleColor( const uint c, const uint x ) // x = 0..255



#### **Multiplication**

fp\_a = PI;

• "Ensure that intermediate results never exceed 32 bits."

Suppose we want to multiply two 20:12 unsigned fixed point numbers:

Which option we chose depends on the parameters. For example:

fp\_b = 0.5f \* 2^12; // 0.5 in fixed point

int fp\_prod = fp\_a >> 1; // divide by 2 ③

1.	(fp_a * fp_b) >> 12; /	//	good	if	fp_a	and fp_b are very small
2.	(fp_a >> 12) * fp_b; /	//	good	if	fp_a	is a whole number
3.	(fp_a >> 6) * (fp_b >> 6); /	//	good	if	fp_a	and fp_b are large
4.	((fp_a >> 3) * (fp_b >> 3))	>>	6; /	′/ h	ybrid	: good for medium values

#### MAXDEPTH)

refl \* E \* diffuse;

), N );

efl + refr)) && (depth < )

survive = SurvivalProbability( diffuse estimation - doing it properly, close Hf; radiance = SampleLight( &rand, I, &L, e.x + radiance.y + radiance.z) > 0) & A

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Ps at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following Service)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:



at a = nt - nc

), N );

(AXDEPTH)

v = true;

ff:

refl \* E \* diffuse;

efl + refr)) && (depth < MAXD

survive = SurvivalProbability( diff)

radiance = SampleLight( &rand, I,

e.x + radiance.y + radiance.z) > 0)

at brdfPdf = EvaluateDiffuse( L, N ) \*
at3 factor = diffuse \* INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf)

#### Division

• "Ensure that intermediate results never exceed 32 bits."

Dividing two 20:12 fixed point numbers:

1. (fp\_a << 12) / fp\_b; // good if fp\_a and fp\_b are very small
2. fp\_a / (fp\_b >> 12); // good if fp\_b is a whole number
3. (fp\_a << 6) / (fp\_b >> 6); // good if fp\_a and fp\_b are large
4. ((fp\_a << 3) / (fp\_b >> 3)) << 6; // hybrid: good for medium values</pre>

#### Note that a division by a constant can be replaced by a multiplication by its reciprocal:

fp\_reci = (1 << 12) / fp\_b;
fp\_prod = (fp\_a \* fp\_reci) >> 12; // or one of the alternatives

andom walk - done properly, closely following Samma /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apd prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

#### INFOMOV – Lecture 12 – "Fixed Point Math"

### Operations

#### nics & (depth < Modestin

at a = nt - nc, b = Mt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D <sup>#</sup> nnt - N <sup>#</sup> (d0

\* diffuse; = true;

efl + refr)) && (depth < MAXDEPTIOL

), N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, closed
if;
radiance = SampleLight( &rand, I, &L, &light)
ext + radiance.y + radiance.z) > 0) && (doing)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Psurvise at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* [rad

andom walk - done properly, closely following Source /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, %pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

Multiplication, Take 2

"Use a 64-bit intermediate result."

 $A*B (I_A:F_A*I_B:F_B)$ 

Example:  $I_A$ :  $F_A$ =16:16,  $I_B$ :  $F_B$ =16:16 Result: 32:32

Calculate a 64-bit result (with enough room for 32:32), throw out 32 bits afterwards.

x86 MUL instruction:
MUL EDX
Functionality:
multiplies EDX by EAX, stores the result in EDX:EAX.
→ Tossing 32 bits: ignore EAX.

 $\rightarrow$  x86 is designed for 16:16.



#### Multiplication

Special case: multiply or divide by a 32:0 number.

We did this in the line function:

dx /= pixels; // dx is 16:16, pixels is 32:0
dy /= pixels;

survive = SurvivalProbability( diffuse estimation - doing it properly, closed ff; radiance = SampleLight( &rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dot) w = true;

efl + refr)) && (depth < MAX

refl \* E \* diffuse;

), N );

AXDEPTH)

st brdfPdf = EvaluateDiffuse( L, N ) Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) (rest

andom walk - done properly, closely following Same /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:



#### Square Root

tics ⊈(depth < Mooos

: = inside ? | : : : : ht = nt / nc, ddn ss2t = 1.0f - nnt 2, N ); ≥)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Tr) R = (D = nnt - N = (dd

= \* diffuse; = true;

• efl + refr)) && (depth < MAXOE

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly, closed If; radiance = SampleLight( &rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && (double)

w = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (r;

andom walk - done properly, closely following Soli /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

For square roots of fixed point numbers, optimal performance is achieved via \_mm\_rsqrt\_ps (via float). If precision is of little concern, use a lookup table, optionally combined with interpolation and / or a Newton-Raphson iteration.

Sine / Cosine / Log / Pow / etc.

Almost always a LUT is the best option\*.

...But, if you must: <a href="https://github.com/chmike/fpsqrt">https://github.com/chmike/fpsqrt</a>

#### \*: Not on the GPU however. Alternative:

https://www.reddit.com/r/programming/comments/1vbb5l/fast\_fixedpoint\_sine\_approximation





#### Fixed Point & SIMD

\_mm\_mul\_epu32

\_mm\_srl\_epi32

\_mm\_srai\_epi32

\_mm\_mullo\_epi16

\_mm\_mulhi\_epu16

For a delicious world of hurt, combine SIMD and fixed point:

= inside / 1 : = nt / nc, ddn :2t = 1.0f - nnt N );

nt a = nt - nc, b = 11 nt Tr = 1 - (R0 + (1 - 14 Tr) R = (D = nnt - 14

= \* diffuse; = true;

• efl + refr)) && (depth < MacCorr

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, closed
if;
radiance = SampleLight( &rand, I, &L, \lightarrow
e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot(N, L); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 6pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:



See MSDN / Intel Intrinsic list for more details.

ics & (depth < Moorente

: = inside ? 1 1 1 0 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N ); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (000

= \* diffuse = true;

efl + refr)) && (depth < MANDEPTIO

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse)
estimation - doing it properly.
if;
radiance = SampleLight( &rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (000)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 6pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Today's Agenda:

- Introduction
- Float to Fixed Point and Back
- Operations
- Fixed Point & Accuracy
- Demonstration



### Accuracy

ics ≹(depth < Notis

at a = nt - nc, b =  $(R0 + (1 - R0)^2)$ at Tr = 1 - (R0 + (1 - R0)^2) Tr) R = (D = nnt - N = (1 - R0)^2)

= \* diffuse; = true;

• efl + refr)) && (depth < MAXDEPTI

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

```
survive = SurvivalProbability( diffuse
estimation - doing it properly, closed
if;
adiance = SampleLight( &rand, I, &L, &II
e.x + radiance.y + radiance.z) > 0) && (d)
```

v = true; at brdfPdf = EvaluateDiffuse( L, N ) = Psur at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf) = (r

andom walk - done properly, closely following sould /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Bpdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

Range versus Precision

Looking at the line code once more:

void line( int x1, int y1, int x2, int y2 )

precision: 16 bits, maximum error:  $\frac{1}{216} * 0.5 = \frac{1}{217}$ . Interpolating a 1024 pixel line, the maximum cumulative error is  $2^{10} \cdot \frac{1}{217} = \frac{1}{27} \approx 0.008$ .



### Accuracy

nics & (depth < Max00)

z = inside } 1 ht = nt / nc, ddh os2t = 1.0f - nnt 0, N ); ∂)

at a = nt - nc, b = 11 at Tr = 1 - (R0 + (1 - 10) Tr) R = (D = nnt - N = (1

= \* diffuse; = true;

. efl + refr)) && (depth < MacCESTIN

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly) if; adiance = SampleLight( &rand, I, &L, &L e.x + radiance.y + radiance.z) > 0) && ()

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Ps at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follo /ive)

```
,
t33 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 6pdf
urvive;
.pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

Range versus Precision: Error

In base 10, error is clear:

PI = 3.14 means: 3.145 > PI > 3.135The maximum error is thus  $\frac{1}{2} \frac{1}{10^2} = 0.005$ .

In base 2, we apply the same principle:

16:16 fixed point numbers have a maximum error of  $\frac{1}{2} \frac{1}{2^{16}} = \frac{1}{2^{17}} \approx 7.6 \cdot 10^{-6}$ . → We get slightly more than 5 digits of decimal precision.

For reference: 32-bit floating point numbers:

- 1 sign bit, 8 exponent bits, 23 mantissa/fractional bits
- $2^{23} \approx 8,000,000$ ; floats thus have ~7 digits of decimal precision.



### Accuracy

#### Error

tics (depth < MOCOC

c = inside / 1 ht = nt / nc, ddh ps2t = 1.0f - nnt 2, N ); 2)

at a = nt - nc, b = nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D <sup>=</sup> nnt - N

= \* diffuse; = true;

efl + refr)) && (depth < NOCCOT

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (rad

andom walk - done properly, closely following Same /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

Careful balancing of range and precision in fixed point calculations can reduce this problem.

Note that accuracy problems also occur in float calculations; they are just exposed more clearly in fixed point. And: this time we can do something about it.



ics & (depth < Modern

: = inside ? 1 | 1 | 1 ht = nt / nc, ddh os2t = 1.0f - nnt 0, N ); ∂)

at a = nt - nc, b = 1 at Tr = 1 - (R0 + 1 - 1 Fr) R = (D = nnt - N

= \* diffuse = true;

efl + refr)) && (depth < MAXDEPTIO

D, N ); refl \* E \* diffuse; = true;

(AXDEPTH)

survive = SurvivalProbability( diffuse .estimation - doing it properly, closed if; radiance = SampleLight( &rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && closed

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Psurvive at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Source /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Today's Agenda:

- Introduction
- Float to Fixed Point and Back
- Operations
- Fixed Point & Accuracy
- <u>Demonstration</u>



Bilinear Interpolation





tics & (depth < Notica

: = inside ? 1 ht = nt / nc, ddn = 1 os2t = 1.0f - nnt = nn O, N ); ð)

at a = nt - nc, b = 80 + 40 at Tr = 1 - (R0 + (1 - 80 Fr) R = (D = nnt - N = 600

= \* diffuse = true;

efl + refr)) && (depth < MADEPINI

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( diffuse)
estimation - doing it properly, closed
f;

radiance = SampleLight( &rand, I, &L, &lis e.x + radiance.y + radiance.z) > 0) && (down

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pounding at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (nad

andom walk - done properly, closely following Sec. /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

# /INFOMOV/

## END of "Fixed Point Math"

next lecture: "Recap"

