rics & (depth < Notos

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

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

= \* diffuse = true;

efl + refr)) && (depth < MAXDEP

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) 8

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Pourvive 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

J. Bikker - Sep-Nov 2019 - Lecture 11: "Fixed Point Math"

# Welcome!



ics & (depth < ™xxxxx

: = inside ? 1 | | | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
adiance = 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) (rad

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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



ics **& (dept**h < Moc⊡

: = inside ? 1 1 1 3 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

• efl + refr)) && (depth < MANDER)

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

(AXDEPTH)

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

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 Soul /ive)

, t33 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 < MADEFF := inside } 1 : 1 nt = nt / nc, ddn ns2t = 1.0f - nmt D, N ); ≫)

\* diffuse; = true;

at Tr = 1

. efl + refr)) && (depth < ⊨

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

#### AXDEPTH)

survive = SurvivalProbability
estimation - doing it proper
if;
radiance = SampleLight( &rand
2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffusa at3 factor = diffuse \* INVP at weight = Mis2( directPdf at cosThetaOut = dot( N, L) E \* ((weight \* cosThetaOut)

andom walk - done properly, ( /ive)

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

Turing introduces a new processor architecture, the **Turing SM**, that deshading efficiency, achieving 50% improvement in delivered performa compared to the Pascal generation. These improvements are enabled changes. First, the Turing SM adds a new independent integer datapat instructions concurrently with the floating-point math datapath. In preexecuting these instructions would have blocked floating-point instruct the SM memory path has been redesigned to unify shared memory, te load caching into one unit. This translates to 2x more bandwidth and r available for L1 cache for common workloads.



#### float f(vec3 p)

p.z+=iTime;return length(.05\*cos(9.\*p.y\*p.x)+cos(p)-.1\*cos(9.\*(p.z+.3\*p.x-p.y)))-1.;

void mainImage( out vec4 c, vec2 p )

vec3 d=.5-vec3(p,1)/iResolution.x,o=d;for(int i=0;i<128;i++)o+=f(o)\*d; c.xyz = abs(f(o-d)\*vec3(0,1,2)+f(o-.0)\*vec3(2,1,0))\*(1...1\*o.2);

Could we evaluate function f without using floats?



ics Menth & MayDourn

z = inside ? 1 ( 1 ) ht = nt / nc, ddh os2t = 1.0f - nnt ( 1 0, N ); 3)

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

= \* diffuse; = true;

• efl + refr)) && (depth < MAXDEPTH

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) && (dot)

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) (PSUR);

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

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

- Multiplying that by  $10^{10}$  yields 31415926536.
- Adding 1 to  $\pi$  yields 4.1415926536.
- 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 get N digits of fractional precision if we multiply our numbers by 10<sup>N</sup> (and remember where we put that dot).



ics & (depth < Modes)

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

= \* diffuse; = true;

efl + refr)) && (depth < NADEPTH

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.

- $\pi / 2 \equiv 31415926536 / 2000000000 = 1$  (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.

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

efl + refr)) && (depth < )

AXDEPTH)

at a = nt

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

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) (Page 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

nics & (depth < Mocosa }

: = inside ? 1 : 1 : ht = nt / nc, ddh os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPID

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

AXDEPTH)

survive = SurvivalProbability( diffuse estimation - doing it properly. If; radiance = SampleLight( &rand, I, &L, &II 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 following Se /ive)

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

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

- Multiplying by 2<sup>16</sup> yields 205887.
- Adding  $1 \cdot 2^{16}$  to the scaled up version of  $\pi$  yields 271423.

In binary:

- 205887 = 0000000 00000011 00100100 00111111
- $271423 = 00000000 \ 00000100 \ 00100100 \ 00111111$

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

• 0000000000011 (base 2) = 3 (base 10); • 10010000111111 (base 2) = 9279 (base 10);  $\frac{9279}{2^{16}} = 0.141586304$ .



ics & (depth < Moc⊙s

: = inside 7 1 1 1 1 ht = nt / nc, ddn bs2t = 1.0f - nnt ∩ n D, N ); ∂)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - nc fr) R = (D \* nnt - N \* (2

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEP

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

AXDEPTH)

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

w = true; at brdfPdf = EvaluateDiffuse( L, N ) Pour 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, Bpdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

The Concept of Fixed Point Math

Interpolation, base 10:

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



ics & (depth < №000

: = inside ? 1 1 1 2 ht = nt / nc, ddn bs2t = 1.0f - nnt 7 2, N ); 2)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDED

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

AXDEPTH)

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

w = 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) \* (ra

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

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

The Concept of Fixed Point Math

Interpolation, base 2:

```
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 )</pre>
```

```
plot( x / 65536, y / 65536 );
```

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



nics & (depth < NOCCST

: = inside ? l . . . . ht = nt / nc, ddn . . os2t = 1.0f - nnt . . O, N ); 0)

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

= \* diffuse; = true;

• efl + refr)) && (depth < MAXDEPIII

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

AXDEPTH)

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

w = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Psurat3 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, lpdf) urvive; pdf; n = E \* brdf \* (dot( N, R ) / 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  $\frac{1}{2} 10^{-X}$  for x fractional digits.
- A fixed point number with 16 fractional bits has a maximum error of  $\frac{1}{2}2^{-16}$ .

## During interpolation:

If our longest line is Y pixels, the maximum error with X fractional bits is  $\frac{1}{2}Y 2^{-X}$ . 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>



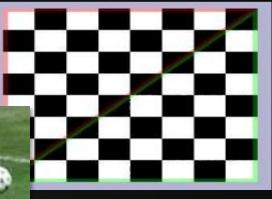
## Practical example

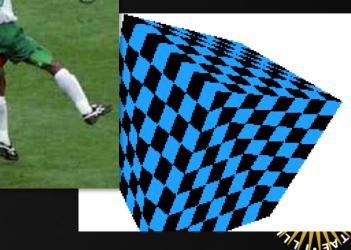
Texture mapping in Quake 1: Perspective Correction

- 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
• refr)) &&	FLD	r/m32/m64	1	0	0	0
); * E * diffu	FLD	m80	3	np	0	0
ue;	FBLD	m80	48-58	np	0	0
	FST(P)	r	1	np	0	0
PTH)	FST(P)	m32/m64	2 m)	np	0	0
ve = Surviv	FST(P)	m80	3 m)	np	0	0
mation - do	FBSTP	m80	148-154	np	0	0
nce = Sampl	FILD	m	3	np	2	2
radiance.y	FIST(P)	m	6	np	0	0
rue; dfPdf = Eva	FLDZ FLD1		2	np	0	0
actor = dif	FLDPI FLDL2E etc.		5 s)	np	2	2
ight = Mis2 sThetaOut =	FNSTSW	AX/m16	6 q)	np	0	0
((weight *	FLDCW	m16	8	np	0	0
walk - don	FNSTCW	m16	2	np	0	0
	FADD(P)	r/m	3	0	2	2
15 5 3	FSUB(R)(P)	r/m	3	0	2	2
rdf = Sampl e;	EMUL(P)	r/m	3	0	2	2 n)
* brdf * (	FDIV(R)(P)	r/m	19/33/39 p)	0	38 o)	2
	ECHS EARS		1		0	







## Practical example

Texture mapping in Quake 1: Perspective Correction

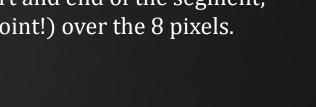
- Affine texture mapping: interpolate u/v linearly over polygon
- Perspective correct texture mapping: interpolate 1/z, u/z and v/z.
- Reconstruct u and v per pixel using the reciprocal of 1/z.

#### Quake's solution:

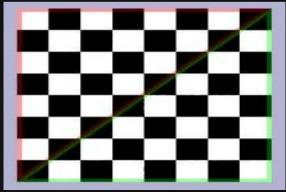
- Divide a horizontal line of pixels in segments of 8 pixels;
- Calculate u and v for the start and end of the segment;
- Interpolate linearly (fixed point!) over the 8 pixels.

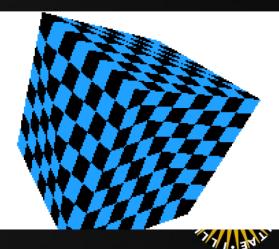
#### And:

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, &R, urvive; pdf; 1 = E \* brdf \* (dot( N, R ) / pdf); sion = 1

efl + refr)) && (depth

survive = SurvivalProbability

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

at brdfPdf = EvaluateDiffuse( L, | 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

), N );

= true;

4AXDEPTH

v = true:

/ive)

ics & (depth < ™xxxxx

: = inside ? 1 | | | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
adiance = 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) (rad

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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



at a = nt

), N );

(AXDEPTH)

v = true:

/ive)

f:

efl + refr)) && (depth

survive = SurvivalProbability( dif

radiance = SampleLight( &rand, I, e.x + radiance.y + radian<u>ce.z) > 0</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 foll

refl \* E \* diffuse; = true; 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

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

Or, get the original float back by casting to float and dividing by 2<sup>fractionalbits</sup>:
 float result = (float)fp\_pi \* (1.0f / 65536.0f);

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

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

#### 15

Practical Things - Considerations

Example: precomputed sin/cos table

```
#define FP_SCALE 65536.0f 1073741824.0f
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).

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)

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I,

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

at a = nt

), N );

(AXDEPTH)

f:

efl + refr)) && (depth

refl \* E \* diffuse;

andom walk - done properly, closely following /ive)

, H33 Brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

tics & (depth < Max∷s

: = inside ? 1 0 0 0 nt = nt / nc, ddn 0 0 ps2t = 1.0f - nnt 0 0, N ); %)

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

= \* diffuse; = true;

. :fl + refr)) && (depth < NAXDE

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

AXDEPTH)

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

v = 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 /ive)

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

**Practical Things - Considerations** 

Example: values in a z-buffer

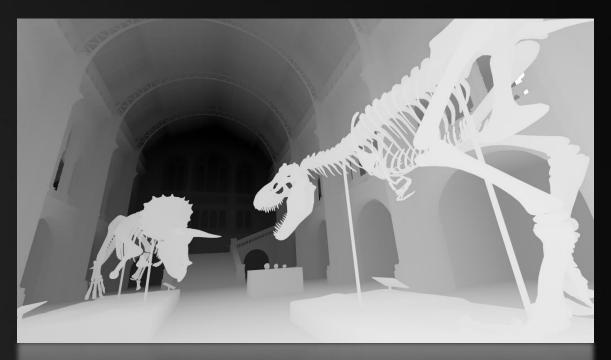
A 3D engine needs to keep track of the depth of pixels on the screen for depth sorting. For this, it uses a z-buffer.

We can make two observations:



. 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.





nics & (depth < NOCCS

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

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

\* diffuse; = true;

efl + refr)) && (depth < MOCO

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly
if;
radiance = SampleLight( &rand, I, &L, &L
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) / directPd

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

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

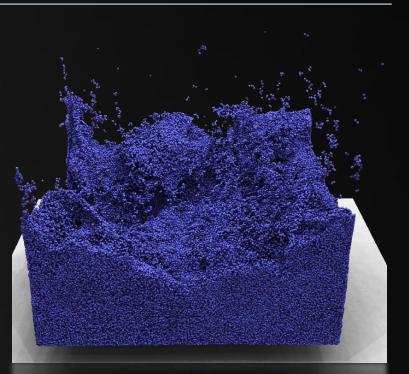
Practical Things - Considerations

Example: particle simulation

Your 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.
- 2. The maximum integer value of 50 fits in 6 bits.
- 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.





ics & (depth < Motos

: = inside ? 1 + 1.0 ht = nt / nc, ddn os2t = 1.0f - nnt D, N ); 2)

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

\* diffuse; = true;

• efl + refr)) && (depth < MOXDEPT

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

AXDEPTH)

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

w = 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 Sea /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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]; → sign bit + 31 fractional bits;

 $\rightarrow$  0:31 signed fixed point.

Storing 1/(z+1): original values [0..1];
→ 32 fractional bits;

 $\rightarrow$  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.



ics & (depth < ™xxxxx

: = inside ? 1 | | | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
adiance = 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) (rad

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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



ics

z = inside } 1 ( ) ) ht = nt / nc, ddh bs2t = 1.0f - nnt ( ), N ); ∂)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXO

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

AXDEPTH)

survive = SurvivalProbability( diffuence estimation - doing it properly if; radiance = SampleLight( &rand, I, &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 following Sec /ive)

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

**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 need to be in the same format.

Example:  $I_A: F_A = 4:28$ ,  $I_B: F_B = 16:16$ 

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 a = nt

), N );

= true;

(AXDEPTH)

f:

efl + refr)) && (depth < MA)

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, & e.x + radiance.y + radiance<u>.z) > 0)</u>

refl \* E \* diffuse;

Basic Operations on Fixed Point Numbers

Operations on mixed fixed point formats:

• A\*B  $(I_A: F_A * I_B: F_B)$ 

We can freely mix fixed point formats for multiplication.

Example:  $I_A: F_A = 18:14$ ,  $I_B: F_B = 14:18$ 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.

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) (rectPdf)

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

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



}

## **Operations**

#### Multiplication

Color scaling, base 2:

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

AXDEPTH)

survive = SurvivalProbability( diff) radiance = Sample e.x + radiance.y

#### v = true;

at brdfPdf = Eva at3 factor = dif at weight = Mis2 at cosThetaOut E \* ((weight \*

andom walk - do /ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R urvive;

pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

#### uint redblue = c & 0x00FF00FF; uint green = c & 0x0000FF00;redblue = (redblue \* x) & 0xFF00FF00; green = (green \* x) & 0x00FF0000; return (redblue + green) >> 8;

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

JeLight( &rand, I, 81, 81) y + radiance.z) > 0) 88 (30)							
<pre>valuateDiffuse( L, N ) Psur iffuse * INVPI; control = = = = = = = = = = = = = = = = = = =</pre>							
2( directPdf, brdfPdf ); = dot( N, L ); * cosThetaOut) / directPdf) *	31	24	23	16	15	8	7 0
		2.4	22	10			
	31	24	23	16	15	8	/



## Multiplication

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

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

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;

Which option we chose depends on the parameters:

fp\_a = PI;
fp\_b = 0.5f \* 2^12;
int fp\_prod = fp\_a >> 1; // ③

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

efl + refr)) && (depth < MAXDE

survive = SurvivalProbability( diff.

radiance = SampleLight( &rand, I,

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)

), N );

AXDEPTH)

v = true;

f:

refl \* E \* diffuse;

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



#### 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;

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

fp\_reci = (1 << 12) / fp\_b;</pre> fp\_prod = (fp\_a \* fp\_reci) >> 12; // or one of the alternatives at weight = Mis2( directPdf, brdfPdf );

andom walk - done properly, closely follow /ive)

efl + refr)) && (depth < MAXDE

survive = SurvivalProbability( diff)

radiance = SampleLight( &rand, I,

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

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

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

at cosThetaOut = dot( N, L );

), N );

AXDEPTH)

v = true;

f:

refl \* E \* diffuse;

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



nics & (depth < NACCSI

: = inside ? 1 : 1 .0 ht = nt / nc, ddn ps2t = 1.0f - nnt 2, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < NAXDEPTH

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

AXDEPTH)

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

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R, 8pdf 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.
→ x86 is designed for 16:16.



## **Multiplication**

Special case: multiply 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
if;
radiance = SampleLight( &rand, I, &L, &I
ext + radiance.y + radiance.z) > 0) &&

efl + refr)) && (depth < MAX

refl \* E \* diffuse;

), N );

AXDEPTH)

w = 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) \* (Psi

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

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



#### nics & (depth < MADS

: = inside ? | | | | ht = nt / nc, ddn os2t = 1.0f - nnt ), N ); ?)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXO

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) && (course)

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) \* (Padd);

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

Square Root

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\*.



\*: Not on the GPU however. Alternative: <u>https://www.coranac.com/2009/07/sines</u>

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

#### Fixed Point & SIMD

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

\_mm\_mul\_epu32
\_mm\_mullo\_epi16
\_mm\_mulhi\_epu16
\_mm\_srl\_epi32
\_mm\_srai\_epi32

See MSDN for more details.

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

), N );

AXDEPTH)

refl \* E \* diffuse;

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

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

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

ics & (depth < ™xxxxx

: = inside ? 1 | | | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
adiance = 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) (rad

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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



ics &∫(depth < NoCDS

: = inside ? 1 1 1 1 ht = nt / nc, ddh = 1 ps2t = 1.0f - nnt = n D, N ); 2)

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

= \* diffuse; = true;

-:fl + refr)) && (depth < MAXDE

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

AXDEPTH)

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

w = 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 Small /ive)

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apdf 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:  $\frac{1}{2^{16}} * 0.5 = \frac{1}{2^{17}}$ . Interpolating a 1024 pixel line, the maximum cumulative error is  $2^{10} \cdot \frac{1}{2^{17}} = \frac{1}{2^7} \approx 0.008$ .



·ics

t = inside ? 1 0 1 7 ht = nt / nc, ddn os2t = 1.0f - nnt 0 n O, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPT

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

AXDEPTH)

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

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 follo ⁄ive)

```
;
t3 Brdf = SampleDiffuse( diffuse, N, rl, r2, 8R, 8pdf
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 bits

•  $2^{23} \approx 8,000,000$ ; floats thus have ~7 digits of decimal precision.



Range versus Precision: Error

During some operations, precision may suffer greatly:

x = y/z

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

= \* diffuse; = true;

efl + refr)) && (depth < MOO

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

AXDEPTH)

survive = SurvivalProbability( diffuse .estimation - doing it properly, if; radiance = SampleLight( &rand, I, &L, &L) 2.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 following Soli /ive)

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

# $fp_x = (fp_y << 8) / (fp_z >> 8)$

Assuming 16:16 input,  $fp_z$  briefly becomes 16:8, with a precision of only 2 decimal digits. Similarly:

$$fp_x = (fp_y >> 8) * (fp_z >> 8)$$

Here, both  $fp_y$  and  $fp_z$  become 16:8, and the cumulative error may exceed  $1/2^9$ .



#### Error

tics ≰ (depth < NAXOS

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

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

= \* diffuse; = true;

efl + refr)) && (depth < NOCCEPTI

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) && (dot)

w = 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) \* (red

andom walk - done properly, closely following Small /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 < ™xxxxx

: = inside ? 1 | | | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
adiance = 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) (rad

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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



## Demonstration

nics & (depth < NOCCS

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

: \* diffuse; = true;

efl + refr)) && (depth < 100

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

#### AXDEPTH)

survive = SurvivalProbability(
.estimation - doing it proper)
if;
radiance = SampleLight( &rand,
2.x + radiance.z)

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

andom walk - done properly, c /ive)

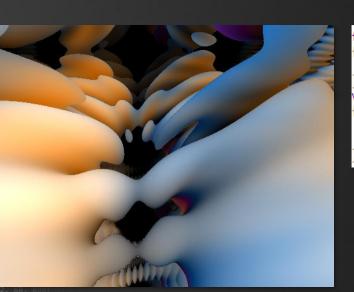
; at3 brdf = SampleDiffuse( diffuse, N, r urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## That Shader

Could it be done?

- Length means sqrt
- Cos, sin (LUT on GPU?)
- Vectors

...



2009-07-16 22:19

← Previous Next →

## Another fast fixed-point sine approximation

Documents

#### Gaddammit!

So here I am, looking forward to a nice quiet weekend; hang back, watch some telly and maybe read a bit – but *NNnnneeeEEEEEUUUuuuuuuuu!! Someone* had to write an interesting article about sine approximation. With a *challenge* at the end. *And* using an inefficient kind of approximation. And so now, instead of just relaxing, I have to spend my entire weekend *and* most of the week figuring out a better way of doing it. I hate it when this happens >\_<.

Links

Projects

Okay, maybe not.

Sarcasm aside, it is an interesting read. While the standard way of calculating a sine – via a look-up table – works and works well, there's just something unsatisfying about it. The LUT-based approach is just ... dull. Uninspired. Cowardly. *Inelegant*. In contrast, finding a suitable algorithm for it requires effort and a modicum of creativity, so something like that always piques my interest.

In this case it's sine approximation. I'd been wondering about that when I did my arctan article, but figured it would require too many terms to really be worth the effort. But looking at Mr Schraut's post (whose site you should be visiting from time to time too; there's good stuff there) it seems you can get a decent version quite

#### float f(vec3 p)

p.z+=iTime;return length(.05\*cos(9.\*p.y\*p.x)+cos(p)-.1\*cos(9.\*(p.z+.3\*p.x-p.y)))-1.;

void mainImage( out vec4 c, vec2 p )

vec3 d=.5-vec3(p,1)/iResolution.x,o=d;for(int i=0;i<128;i++)o+=f(o)\*d; c.xyz = abs(f(o-d)\*vec3(0,1,2)+f(o-.0)\*vec3(2,1,0))\*(1...1\*o.2);



ics & (depth < ™xxxxx

: = inside ? 1 | | | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N ); 3)

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

= \* diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
adiance = 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) (rad

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

; at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, 8pdf 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



nics & (depth < NOCCS

: = inside ? 1 | | | | ht = nt / nc, ddn | | os2t = 1.0f - nnt | nn 0, N ); 3)

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

= \* diffuse = true;

efl + refr)) && (depth < MAXDEPILL

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

AXDEPTH)

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

radiance = SampleLight( &rand, I, 81, 81) e.x + radiance.y + radiance.z) > 0) 88

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) (real

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

; at3 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: "Snippets"

