

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * ddn);
    (Tr) R = (D * nnt - N * (ddn *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light,
    e.x + radiance.y + radiance.z) > 0) && (depth <
    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) * (radiance
    random walk - done properly, closely following
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```

/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2018 - Lecture 9: "GPGPU (1)"

Welcome!



Global Agenda:

1. GPGPU (1) : Introduction, architecture, concepts
2. GPGPU (2) : Practical Code using GPGPU
3. GPGPU (3) : Parallel Algorithms, Optimizing for GPU



Today's Agenda:

- Introduction to GPGPU
- Example: Voronoi Noise
- GPGPU Programming Model
- OpenCL Template



“If you were plowing a field, which would you rather use? Two strong oxen, or 1024 chickens?”

- Seymour Cray



Introduction

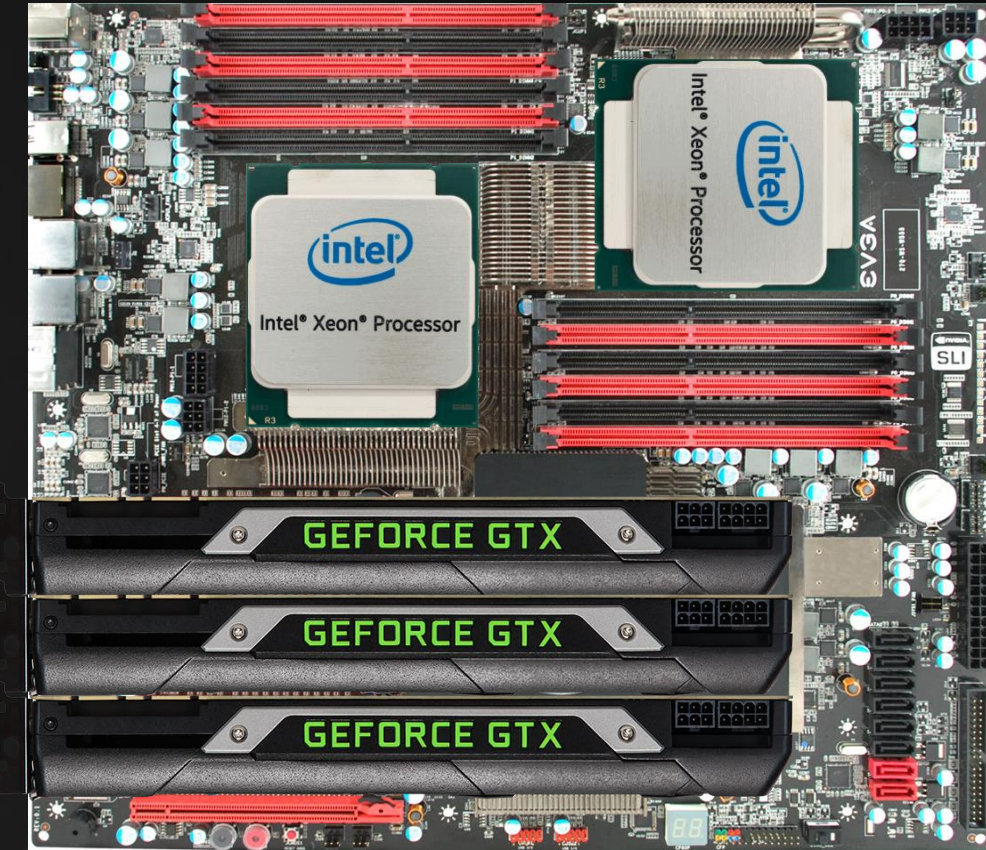
Heterogeneous Processing

The average computer contains:

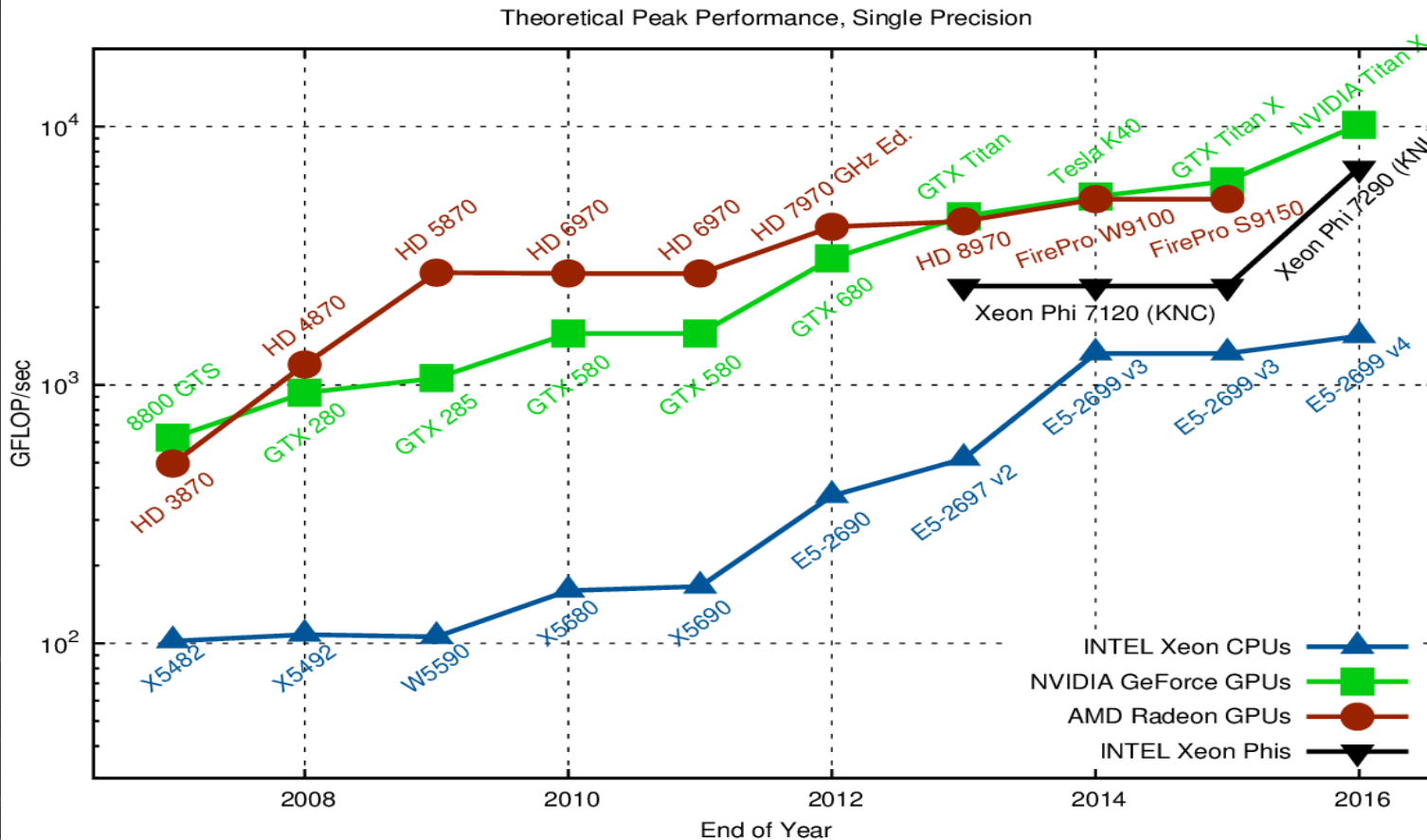
- 1 or more CPUs;
- 1 or more GPUs.

We have been optimizing CPU code.
A vast source of compute power has remained unused:

The Graphics Processing Unit.



Introduction



AMD:
RX Vega 64
 € 525

NVidia:
GTX2080Ti
 \$1200

Intel:
i9-7980XE
 € 1978

Xeon Phi 7120P
 € 3167

484 GB/s
13.7 TFLOPS

616 GB/s
14 TFLOPS

50 GB/s
1.1 TFLOPS

352 GB/s
~6 TFLOPS



Introduction

A Brief History of GPGPU

```

...ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    else
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * r);
        Tr) R = (D * nnt - N * (ddn * r));
        E * diffuse;
        = true;
    }
    else
    {
        refl + refr)) && (depth < MAXDEPTH)
        D, N );
        refl * E * diffuse;
        = true;
    }
}
MAXDEPTH)
survive = SurvivalProbability( diffuse, r);
estimation - doing it properly, closely following
df;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (depth <
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) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Introduction

A Brief History of GPGPU

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

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

    E * diffuse;
    = true;

    -
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;

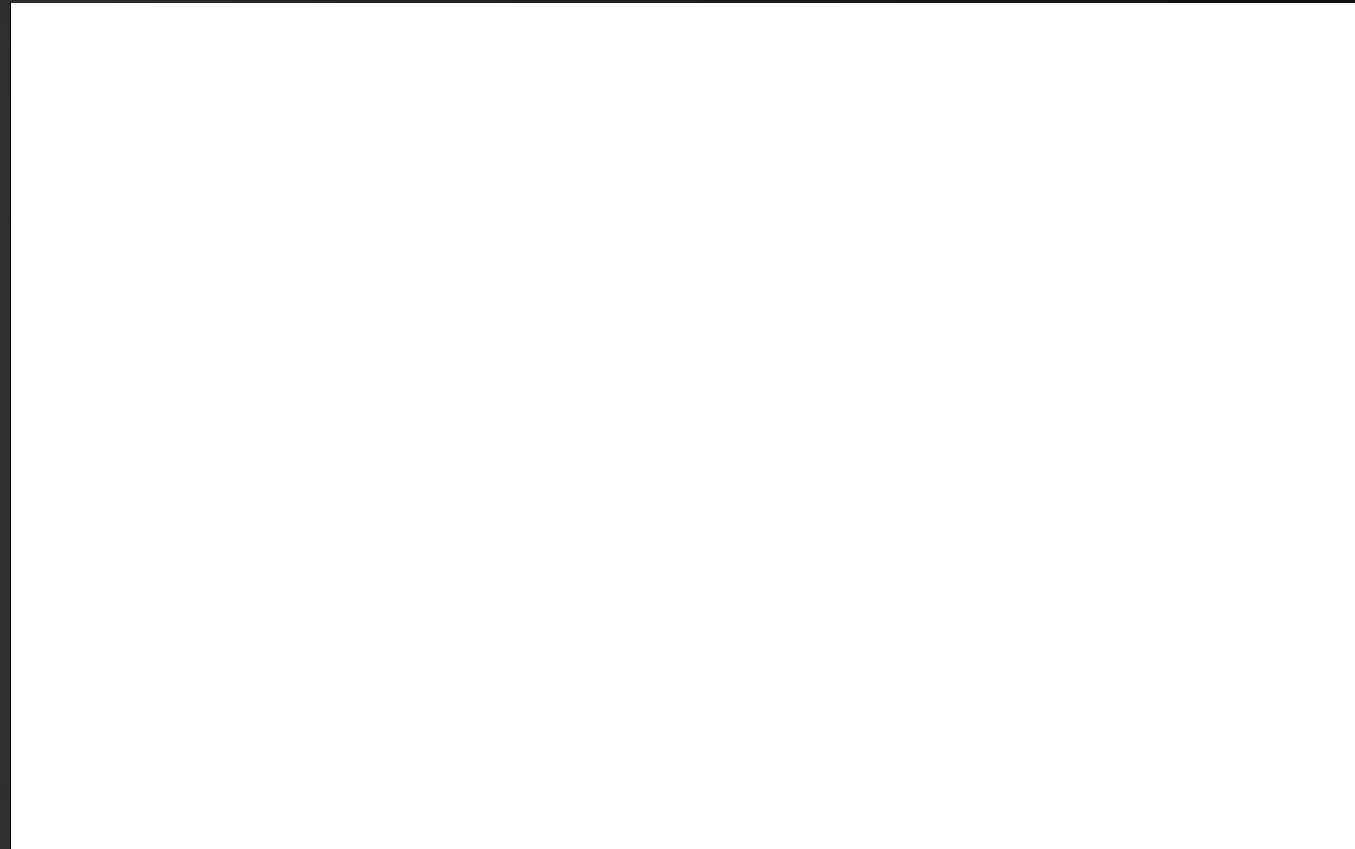
    MAXDEPTH)

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

    random walk - done properly, closely following Section 2.6.1 (survive)
    ve);

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

```



Introduction

NVidia NV-1
(Diamond Edge 3D)
1995

A Brief History of GPGPU



3Dfx –
Diamond Monster 3D
1996

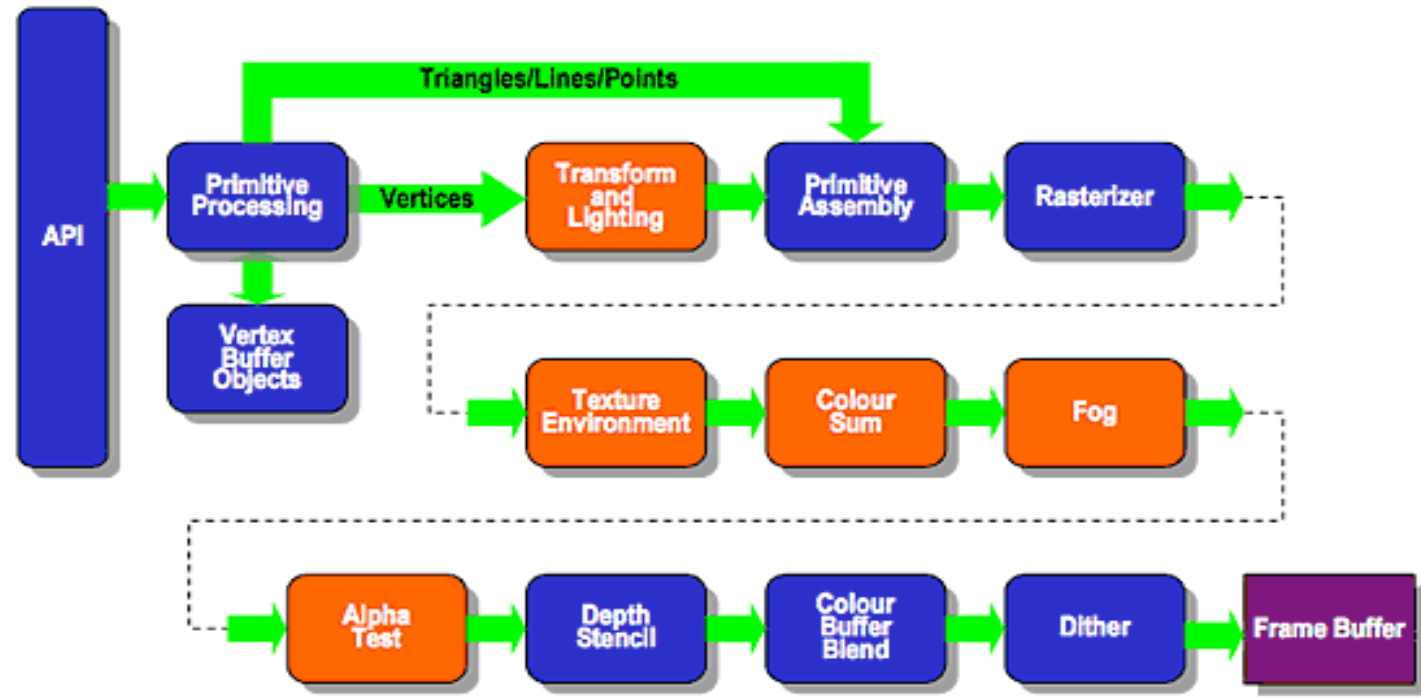


Introduction

A Brief History of GPGPU

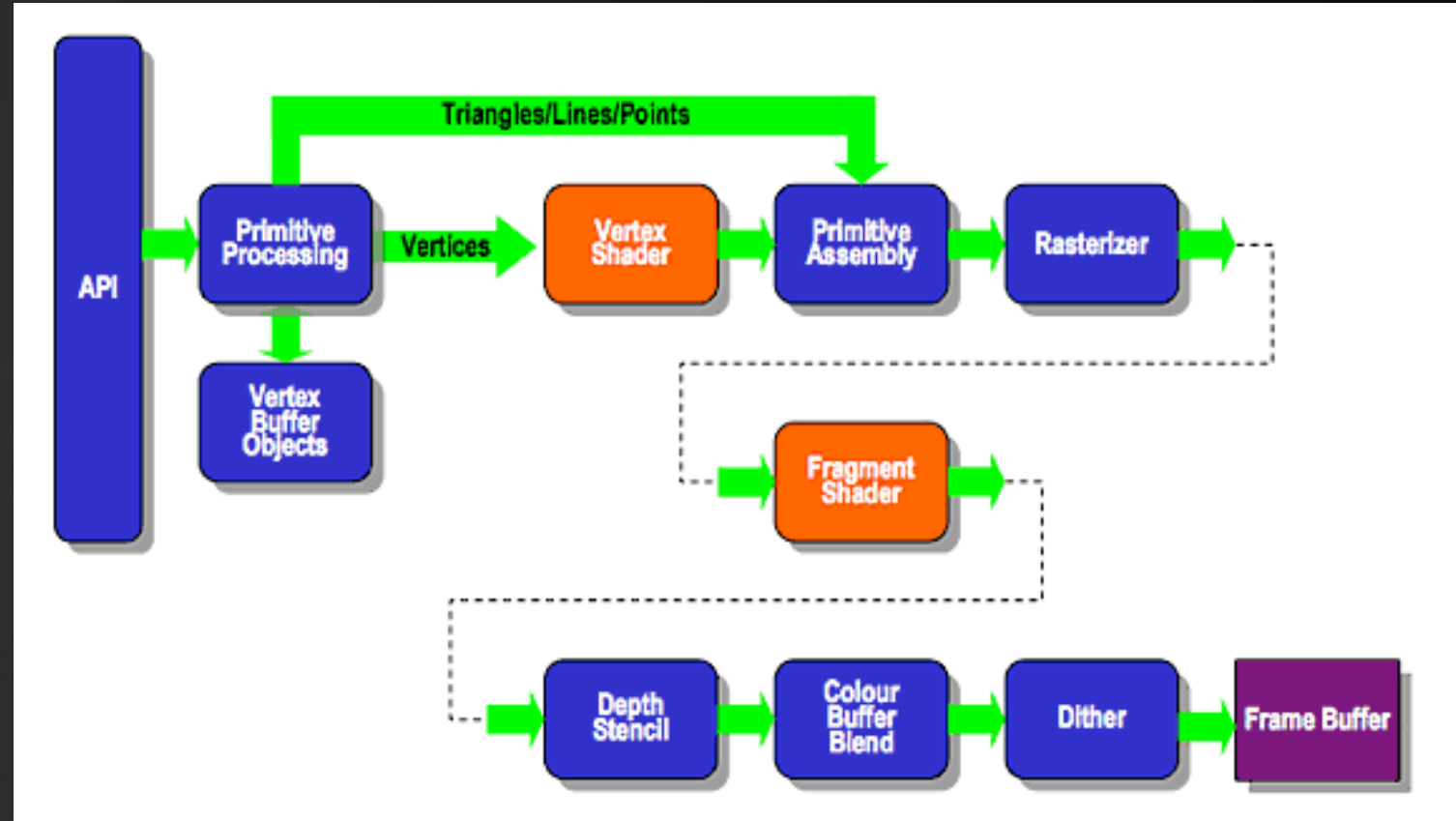
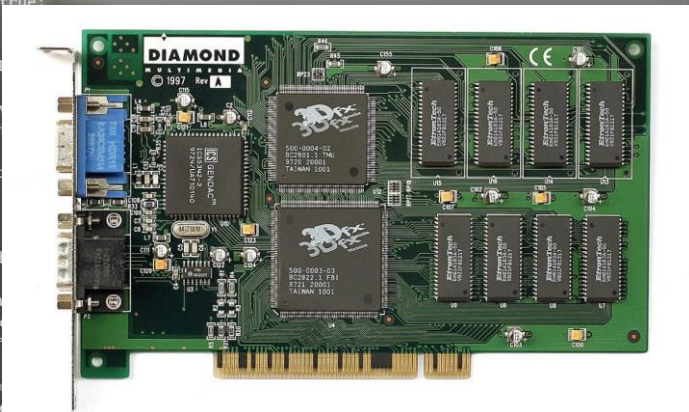


Existing Fixed Function Pipeline



Introduction

A Brief History of GPGPU



Introduction

A Brief History of GPGPU

```

Ok
load"cas:GUSANO"
Skip :RANA
Found:GUSANO
Ok
list 1-100
65 OPEN "GRP:" FOR OUTPUT AS#1
70 STOP ON:ON STOP GOSUB 4000
71 COLOR15,15,15:SCREEN2
72 FORF=1TO8
73 READA:S#=S#+CHR$(A)
74 NEXTF
75 SPRITE$(1)=S$
76 COLOR15,4,13:S#=""
80 SOUND 7,255:SOUND8,15:RR=25
90 SCREEN2,2:PRESET(130,5):PRINT #1,"
MAX.:";PEEK(-1200)
100 DIMX3(500),Y3(500):PSET(35,5),4:P
RINT#1,"PUNTOS: 0":PSET(200,5),4:PRIN
T#1,"@":PSET(212,5),4:PRINT#1,"@":XB=
200:Q=1:RR=25
Ok
color auto goto list run

```

POTATO SALAD

Some good cooks sprinkle grated pimienta cheese on this

- 4 cups diced cooked potatoes
- 1 cup sliced celery
- 3 hard-cooked eggs, cut up
- ½ cup finely cut onion or sliced green onions
- ¼ cup sliced radishes
- 1 cup mayonnaise
- 1 tablespoon vinegar
- 1 teaspoon prepared mustard
- 1½ to 2 teaspoons salt
- ⅛ teaspoon pepper

Lettuce

Mix all the ingredients in a bowl. Cover and refrigerate several hours so flavors can blend. Serve on crisp lettuce.

Makes 6 servings.



Introduction

A Brief History of GPGPU

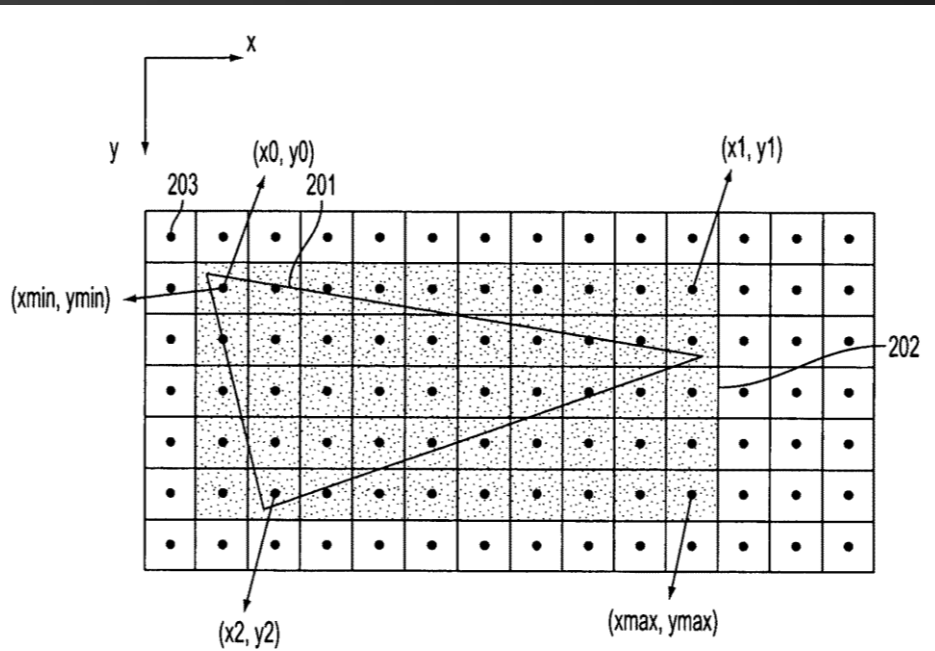


FIG. 4

GPU - conveyor belt:

input = vertices + connectivity

step 1: transform

step 2: rasterize

step 3: shade

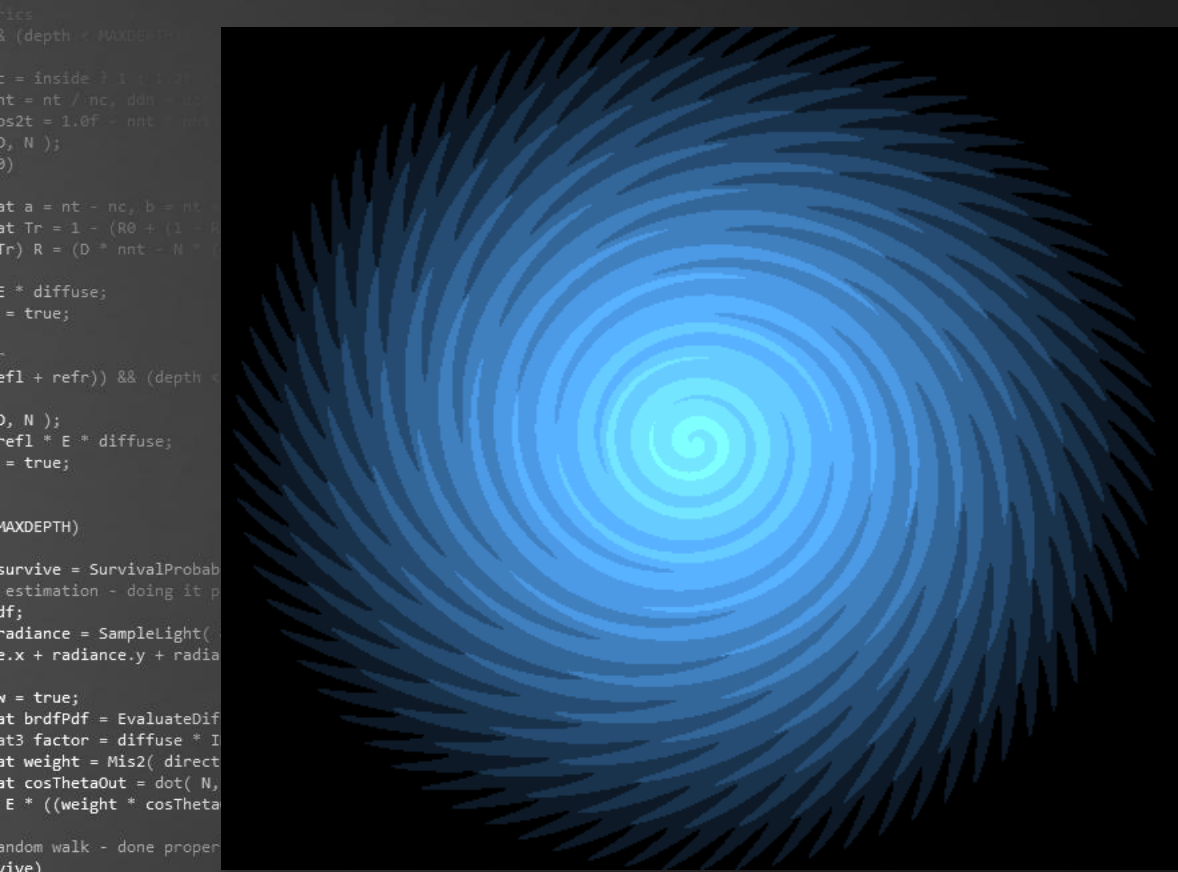
step 4: z-test

output = pixels



Introduction

A Brief History of GPGPU



```

void main(void)
{
    float t = iGlobalTime;
    vec2 uv = gl_FragCoord.xy / iResolution.y;
    float r = length(uv), a = atan(uv.y,uv.x);
    float i = floor(r*10);
    a *= floor(pow(128,i/10));
    a += 20.*sin(0.5*t)+123.34*i-100.*
        (r*i/10)*cos(0.5*t);
    r += (0.5+0.5*cos(a)) / 10;
    r = floor(N*r)/10;
    gl_FragColor = (1-r)*vec4(0.5,1,1.5,1);
}
    
```

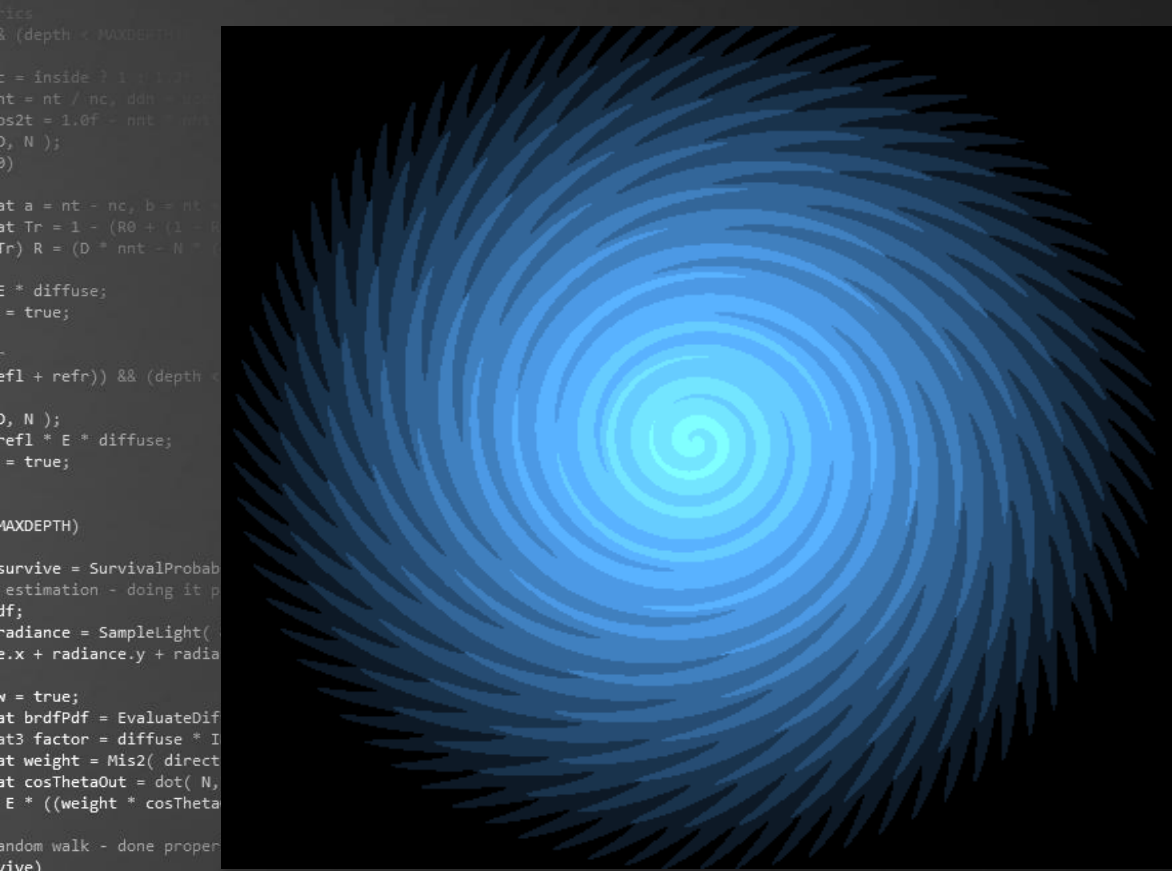
GLSL ES code

<https://www.shadertoy.com/view/4sjSRt>



Introduction

A Brief History of GPGPU



GPUs perform well because they have a constrained execution model, based on massive parallelism.

CPU: Designed to run one thread as fast as possible.

- Use caches to minimize memory latency
- Use pipelines and branch prediction
- Multi-core processing: *task parallelism*

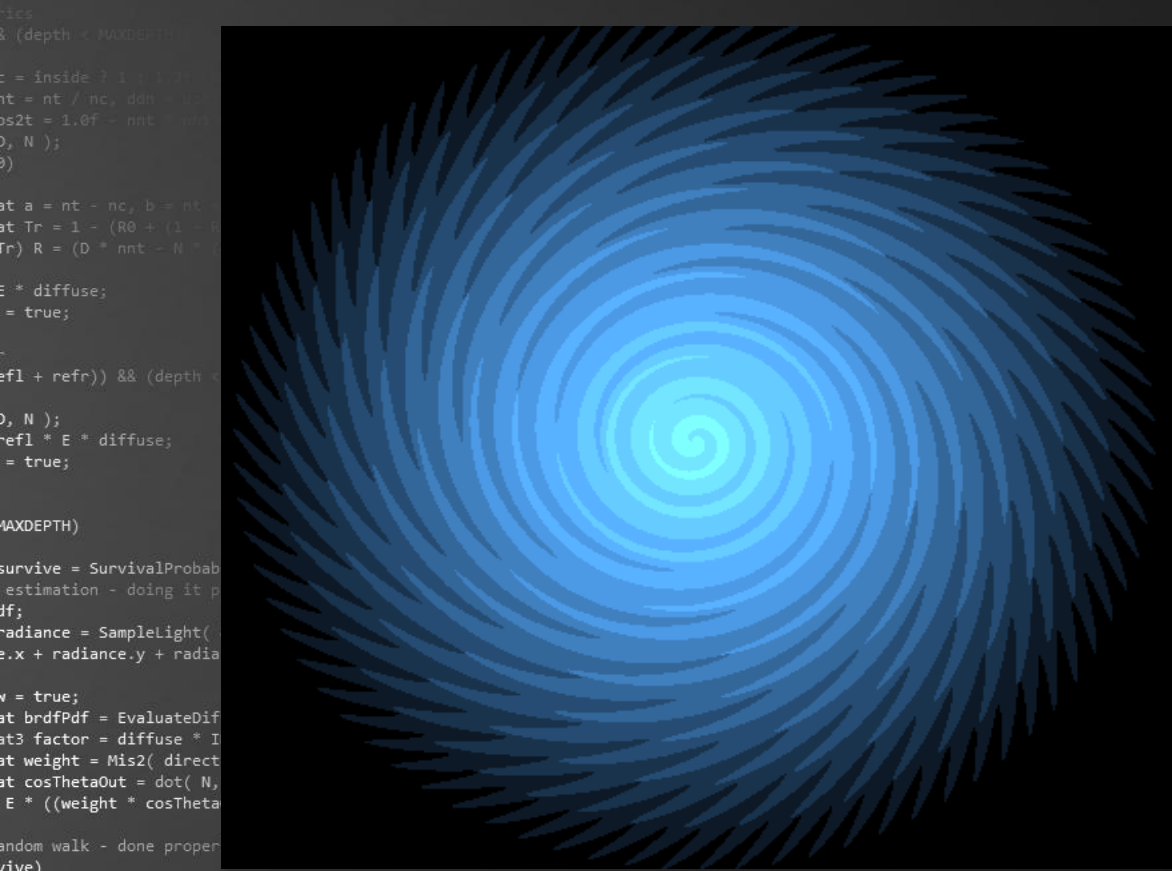
Tricks:

- SIMD
- “Hyperthreading”



Introduction

A Brief History of GPGPU



GPUs perform well because they have a constrained execution model, based on massive parallelism.

GPU: Designed to combat latency using many threads.

- Hide latency by computation
- Maximize parallelism
- Streaming processing → Data parallelism → SIMT

Tricks:

- Use typical GPU hardware (filtering etc.)
- Cache anyway



Introduction

GPU Architecture

```

...
    & (depth < MAXDEPTH)
...
    if ( ! inside )
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    )
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) *
    (Fr) R = (D * nnt - N * (ddn
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)

```

CPU

- Multiple tasks = multiple threads
- Tasks run different instructions
- 10s of complex threads execute on a few cores
- Thread execution managed explicitly

GPU

- SIMD: same instructions on multiple data
- 10.000s of light-weight threads on 100s of cores
- Threads are managed and scheduled by hardware

```

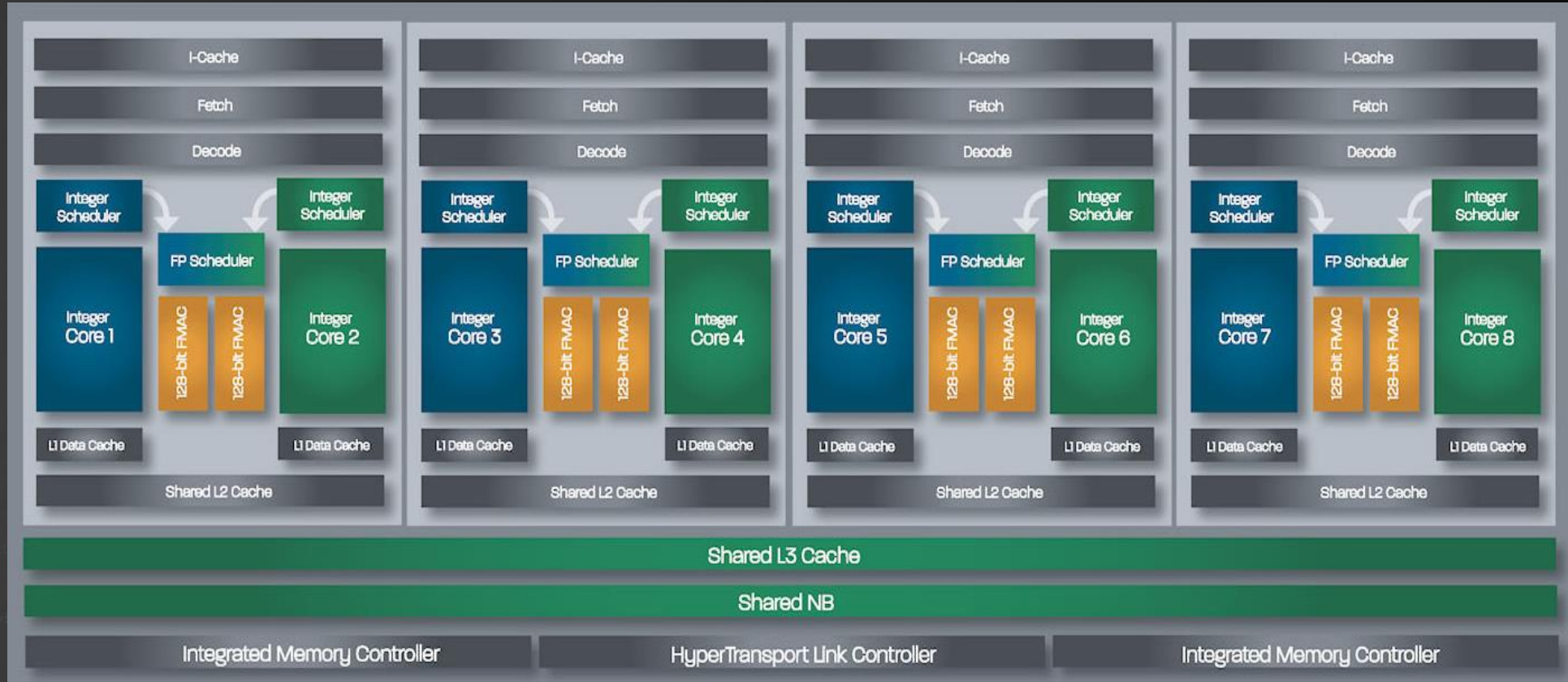
survive = SurvivalProbability( diffuse, r1, r2, &R, &pdf );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light );
e.x + radiance.y + radiance.z) && (cosThetaOut > 0)
...
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
survive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Introduction

CPU Architecture...



Introduction

versus GPU Architecture:



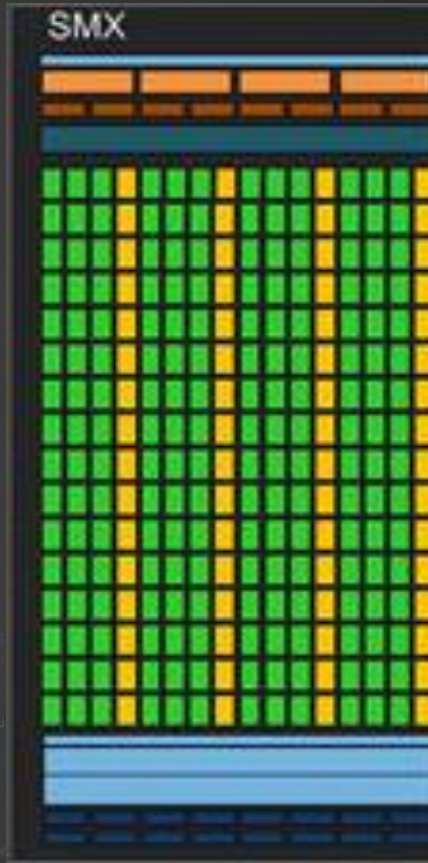
```

ics
& (depth < MAXDEPTH)
    c = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn / ddn;
    ps2t = 1.0f / nnt;
    D, N );
    0)
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (ddn *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, closely
    df;
    radiance = SampleLight( &rand, I, &L, &light;
    e.x + radiance.y + radiance.z) > 0) && (depth <
    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) * (radiance
    random walk - done properly, closely following 3rd
    vive)
    ,
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```



Introduction

GPU Architecture



SIMT Thread execution:

- Group 32 threads (vertices, pixels, primitives) into warps
- Each warp executes the same instruction
- In case of latency, switch to different warp (thus: switch out 32 threads for 32 different threads)
- Flow control: ...



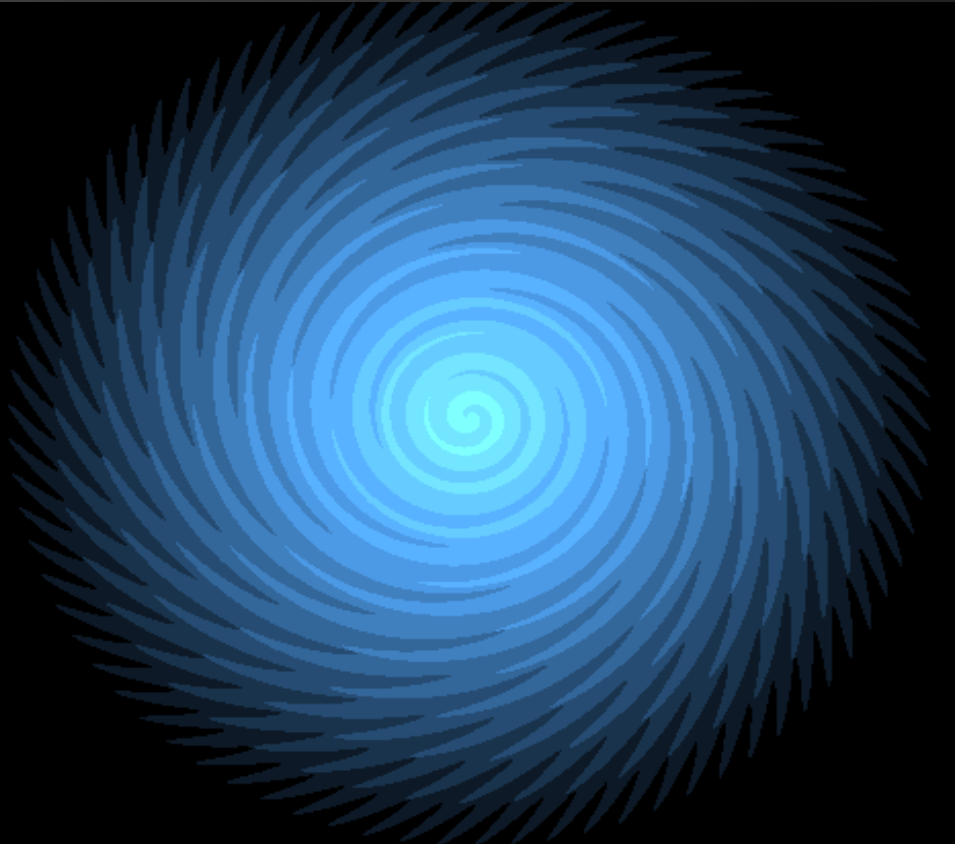
Introduction

GPGPU Programming

```

...
    & (depth < MAXDEPTH)
...
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt;
        D, N );
        t = 0;
    }

    at a = nt - nc; b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    (Fr) R = (D * nnt - N *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth <
    D, N );
    refl * E * diffuse;
    = true;
    -
    MAXDEPTH)
    survive = SurvivalProbab
    estimation - doing it p
    if;
    radiance = SampleLight(
    e.x + radiance.y + radia
    w = true;
    at brdfPdf = EvaluateDif
    at3 factor = diffuse * I
    at weight = Mis2( direct
    at cosThetaOut = dot( N,
    E * ((weight * cosTheta
    random walk - done proper
    vive)
    -
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    -
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    -
    
```



```

void main(void)
{
    float t = iGlobalTime;
    vec2 uv = gl_FragCoord.xy / iResolution.y;
    float r = length(uv), a = atan(uv.y,uv.x);
    float i = floor(r*10);
    a *= floor(pow(128,i/10));
    a += 20.*sin(0.5*t)+123.34*i-100.*
        (r*i/10)*cos(0.5*t);
    r += (0.5+0.5*cos(a)) / 10;
    r = floor(N*r)/10;
    gl_FragColor = (1-r)*vec4(0.5,1,1.5,1);
}

```

<https://www.shadertoy.com/view/4sjSRt>



Introduction

GPGPU Programming

```
...
    & (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nt;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
}

float a = nt - nc, b = nt;
float Tr = 1 - (R0 + (1 - R0) * a);
float R = (D * nnt - N * a) > 0 ? 1 : 0;

E * diffuse;
    = true;
}

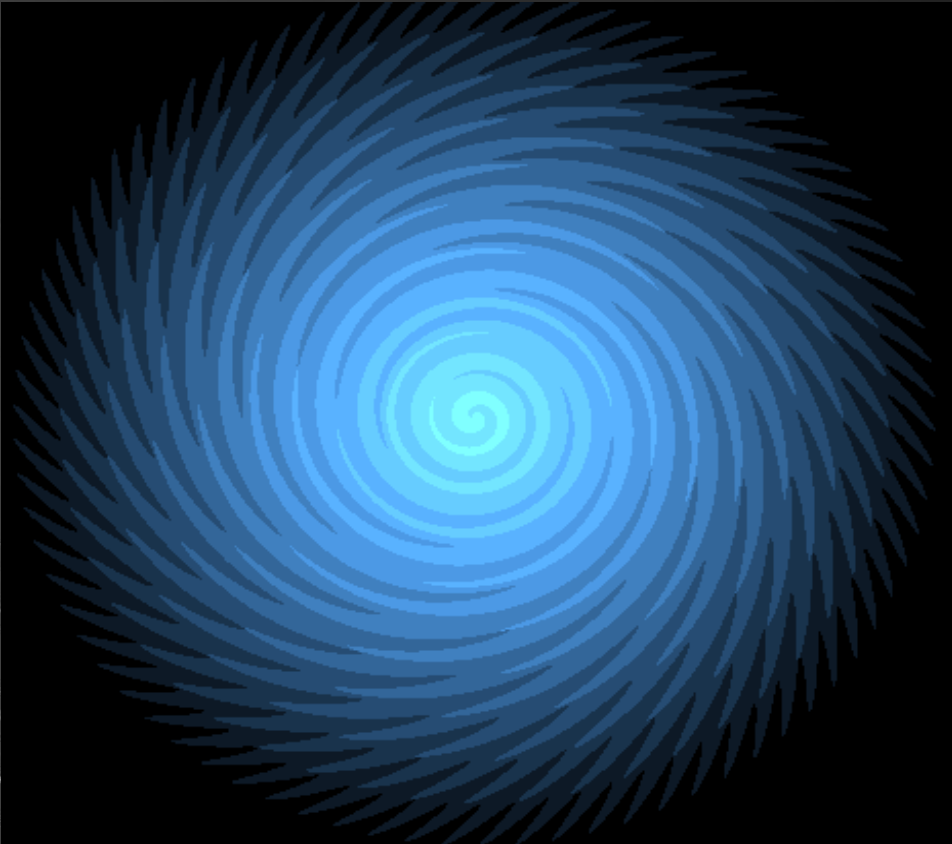
...
    refl + refr)) && (depth <
    D, N );
    refl * E * diffuse;
    = true;
}

MAXDEPTH)

survive = SurvivalProbab
estimation - doing it p
if;
radiance = SampleLight(
e.x + radiance.y + radia

w = true;
at brdfPdf = EvaluateDif
at3 factor = diffuse * I
at weight = Mis2( direct
at cosThetaOut = dot( N,
E * ((weight * cosTheta

random walk - done proper
ive)
```



Easy to port to GPU:

- Image postprocessing
- Particle effects
- Ray tracing
- ...



Today's Agenda:

- Introduction to GPGPU
- Example: Voronoi Noise
- GPGPU Programming Model
- OpenCL Template



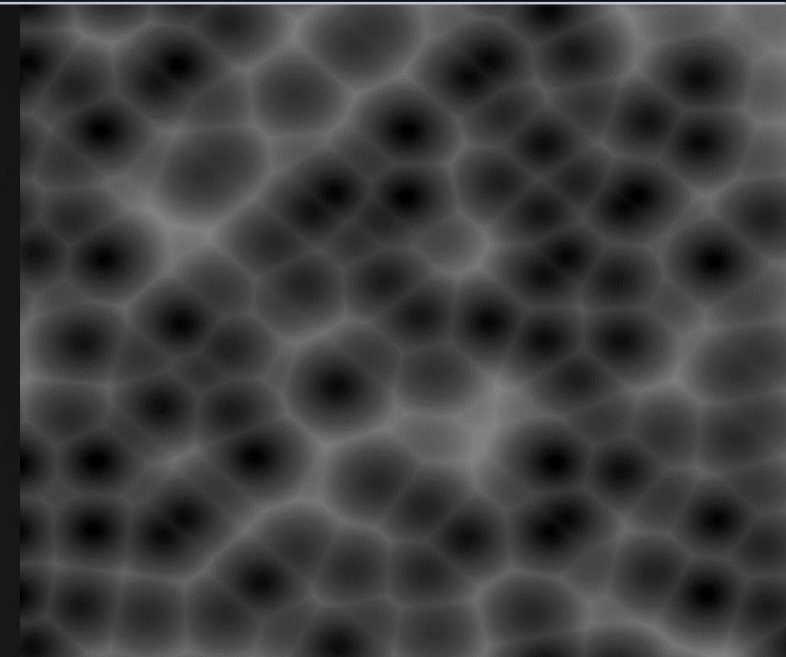
Example

Voronoi Noise / Worley Noise*

Given a set of points, and a position x in \mathbb{R}^2 ,
 $F_1(x)$ = distance of x to closest point.

For Worley noise, we use a Poisson distribution for the points.
 In a lattice, we can generate this as follows:

1. The expected number of points in a region is constant (Poisson);
2. The probability of each point count in a region is computed using the discrete Poisson distribution function;
3. The point count and coordinates of each point can be determined using a random seed based on the coordinates of the region in the lattice.



*A Cellular Texture Basis Function, Worley, 1996



Example

Voronoi Noise / Worley Noise*

```

vec2 Hash2( vec2 p, float t )
{
    float r = 523.0f * sinf( dot( p, vec2(53.3158f, 43.6143f) ) );
    return vec2( frac( 15.32354f * r + t ), frac( 17.25865f * r + t ) );
}

float Noise( vec2 p, float t )
{
    p *= 16;
    float d = 1.0e10;
    vec2 fp = floor( p );
    for( int xo = -1; xo <= 1; xo++ ) for (int yo = -1; yo <= 1; yo++)
    {
        vec2 tp = fp + vec2(xo, yo);
        tp = p - tp - Hash2( vec2( fmod( tp.x, 16.0f ), fmod( tp.y, 16.0f ) ), t ), d = min( d, dot( tp, tp ) );
    }
    return sqrtf( d );
}

```

* <https://www.shadertoy.com/view/4djGRh>

Characteristics of this code:

- Pixels are independent, and can be calculated in arbitrary order;
- No access to data (other than function arguments and local variables);
- Very compute-intensive;
- Very little input data required.



Example

Voronoi Noise / Worley Noise*

Timing of the Voronoi code in C++:

~750ms per image (800 x 512 pixels).

Executing the same code in OpenCL (GPU: GTX480):

~12ms (62x faster).

```

...
    & (depth < MAXDEPTH)
...
    if (inside) {
        nt = nt / nc; ddn = ddn / nc;
        ns2t = 1.0f - nnt * nnt;
        D, N );
    }
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * nnt);
    Tr) R = (D * nnt - N * (ddn * nnt));
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (abs(radiance.x) > 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) * (radiance
...
random walk - done properly, closely following
survive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Example

Voronoi Noise / Worley Noise

GPGPU allows for efficient execution of tasks that expose a lot of potential parallelism.

- Tasks must be independent;
- Tasks must come in great numbers;
- Tasks must require little data from CPU.

Notice that these requirements are met for rasterization:

- For thousands of pixels,
- fetch a pixel from a texture,
- apply illumination from a few light sources,
- and draw the pixel to the screen.

```

...
    & (depth < MAXDEPTH)
...
    = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn * 2.0f;
    cos2t = 1.0f - nnt * ddn;
    D, N );
    )
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * r);
    Tr) R = (D * nnt - N * (ddn *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, r);
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (survive
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psum;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following 3rd
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Today's Agenda:

- Introduction to GPGPU
- Example: Voronoi Noise
- GPGPU Programming Model
- OpenCL Template



Programming Model

GPU Architecture

A typical GPU:

- Has a small number of ‘shading multiprocessors’ (comparable to CPU cores);
- Each core runs a small number of ‘warps’ (comparable to hyperthreading);
- Each warp consists of 32 ‘threads’ that run in lockstep (comparable to SIMD).



Core 0



Core 1

```

...
    & (depth < MAXDEPTH)
...
    inside ? 1 : 0;
    nt = nt / nc; ddn = ddn * ddn;
    cos2t = 1.0f - nnt * ddn;
    D, N );
    )
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &lig
e.x + radiance.y + radiance.z) > 0) && (co
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pdf;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following 3rd order
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Programming Model

GPU Architecture

Multiple warps on a core:

The core will switch between warps whenever there is a stall in the warp (e.g., the warp is waiting for memory). Latencies are thus hidden by having many tasks.

This is only possible if you feed the GPU enough tasks: $cores \times warps \times 32$.



Core 0

Core 1



Programming Model

GPU Architecture

Threads in a warp running in lockstep:

At each cycle, all ‘threads’ in a warp must execute the same instruction. Conditional code is handled by temporarily disabling threads for which the condition is not true. If-then-else is handled by sequentially executing the ‘if’ and ‘else’ branches. Conditional code thus reduces the number of active threads (occupancy). Note the similarity to SIMD code!



Core 0

Core 1



Programming Model

SIMT

The GPU execution model is referred to as SIMT: Single Instruction, Multiple Threads.

A GPU is therefore a very wide vector processor.

Converting code to GPGPU is similar to vectorizing code on the CPU.

```

...ics
& (depth < MAXDEPTH)
...
= inside ? 1.0f : 0.0f;
nt = nt / nc; ddn = ddn * ddn;
ps2t = 1.0f - nnt * ddn;
D, N );
)
...
at a = nt - nc, b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * ddn);
Tr) R = (D * nnt - N * (ddn * ddn));
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH)
...
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse, r1, r2, &R, Spdf );
estimation - doing it properly, closely following the
df;
radiance = SampleLight( &rand, I, &L, &N, &R, &Spdf );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pdf;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance * Pdf);
...
random walk - done properly, closely following the
vive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Spdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Core 0

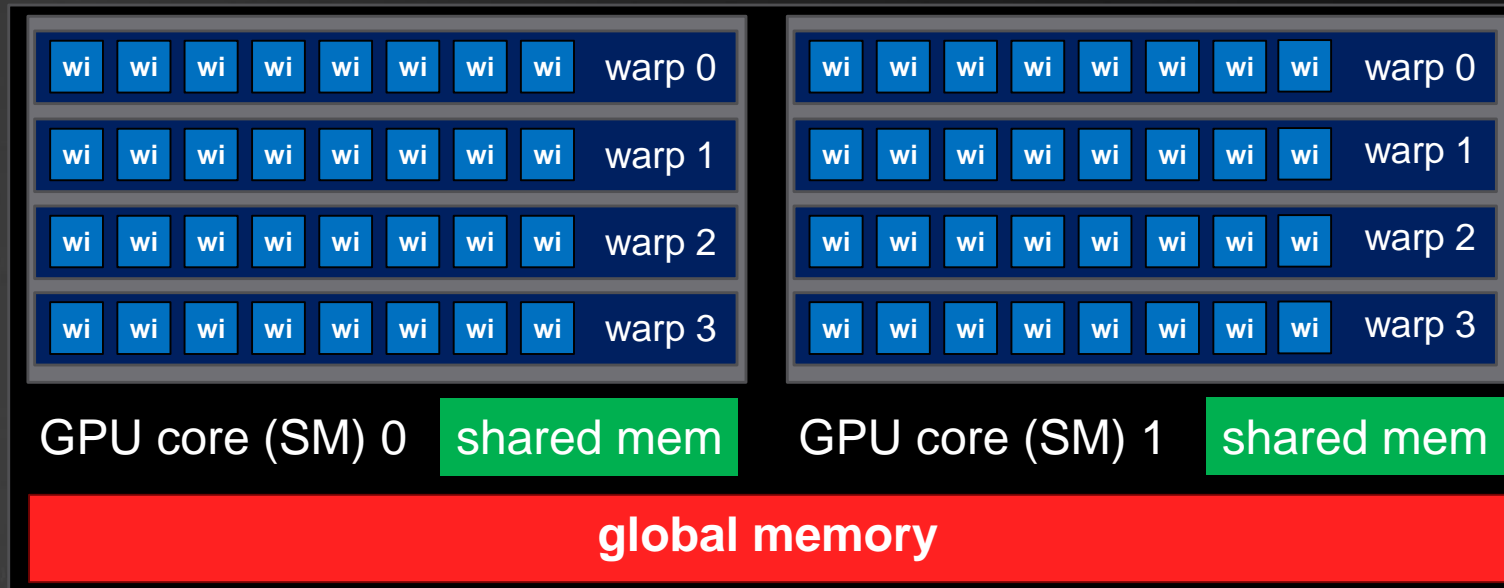


Core 1



Programming Model

GPU Memory Model

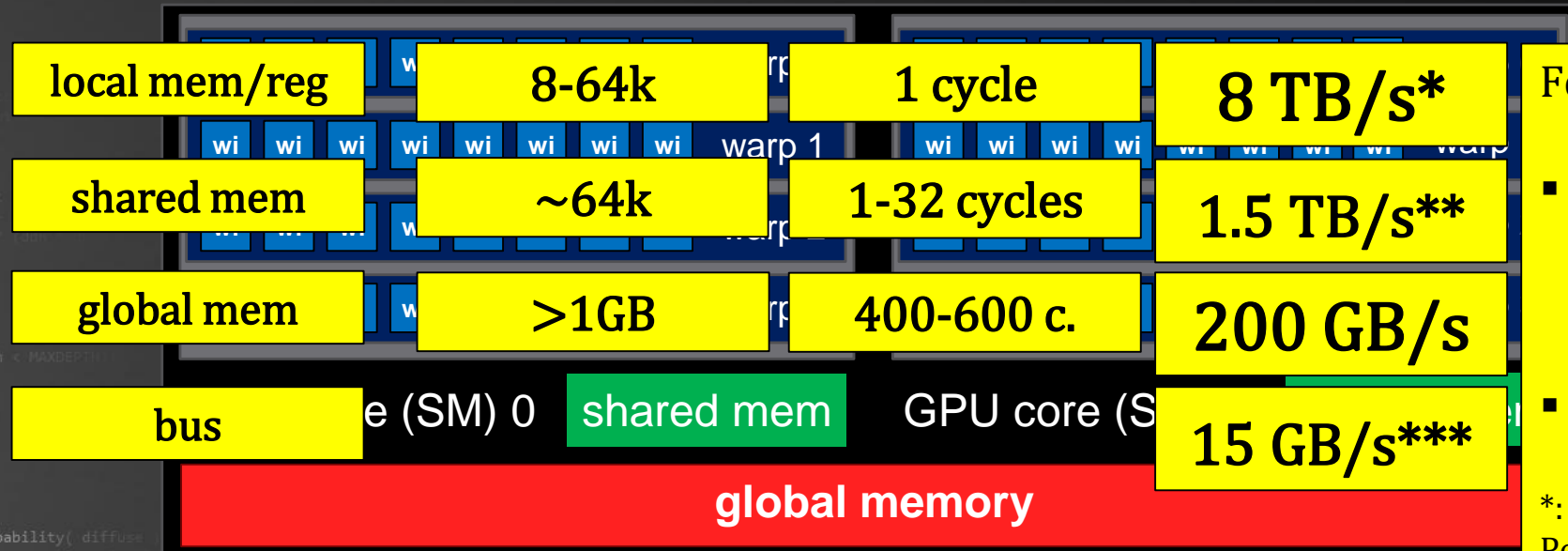


- Each SM has a large number of registers, which is shared between the warps.
- Each SM has shared memory, comparable to L1 cache on a CPU.
- The GPU has global memory, comparable to CPU RAM.
- The GPU communicates with the ‘host’ over a bus.



Programming Model

GPU Memory Model



For reference, Core i7-3960X:

- RAM bandwidth for quad-channel DDR3-1866 memory: **18.1GB/s**
- L2 bandwidth: **46.8GB/s***

*: Molka et al., Main Memory and Cache Performance of Intel Sandy Bridge and AMD Bulldozer. 2014.

* Values for NVidia G80 (Tesla)
 ** Fermi uses L1 cache
 *** PCIe 3.0



Programming Model

GPU Memory Model

There appear to be many similarities between a CPU and a GPU:

- Cores, with hyperthreading
- A memory hierarchy
- SIMD

However, there are fundamental differences in each of these.

- One GPU core will execute up to 64 warps (instead of 2 on the CPU);
- The memory hierarchy is explicit on the GPU, rather than implicit on the CPU;
- GPU SIMD on the other hand is implicit (SIMT model).

```

...
    & (depth < MAXDEPTH)
...
    if (inside ? 1 : 0)
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    )
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
    radiance = SampleLight( &rand, I, &L, &light
    e.x + radiance.y + radiance.z) > 0) && (survive
...
    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) * (radiance
...
random walk - done properly, closely following 3e-10
(ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Programming Model

GPGPU Programming Model

A number of APIs is available to run general purpose GPU code:

Pixel shaders:

- Executed as part of the rendering pipeline
- The number of tasks is equal to the number of pixels

Compute shaders:

- Executed as part of the rendering pipeline
- More control over the number of tasks

OpenCL / CUDA:

- Executed independent of rendering pipeline
- Full control over memory hierarchy and division of tasks over hardware

Graphics-centric work:
Shading, postprocessing (using a full-screen quad)

Graphics-centric work:
Preparing data, output to textures / vertex buffers / ...

General Purpose



Programming Model

GPGPU Programming Model

APIs like CUDA and OpenCL may look like C, but are in fact heavily influenced by the underlying hardware model.

```

__kernel void task( write_only image2d_t outimg, __global uint* logBuffer )
{
    float t = 1;
    int column = get_global_id( 0 );
    int line = get_global_id( 1 );
    float c = Cells( (float2)((float)column / 500, (float)line / 500), t );
    write_imagef( outimg, (int2)(column, line), c );
}
    
```

- Kernel: one task (of which we need thousands to run efficiently);
- get_global(0,1): identifies a single task from a 2D array of tasks.

Many threads will execute the same kernel. We can not execute different code in parallel.



Programming Model

GPGPU Programming Model

Kernels are invoked from the host:

```
size_t workSize[2] = { SCRWIDTH, SCRHEIGHT };
void Kernel::Run( cl_mem* buffers, int count )
{
    ...
    clEnqueueNDRangeKernel( queue, kernel, 2, 0, workSize, NULL, 0, 0, 0 );
    ...
}
```

Device code:

```
__kernel void main( write_only image2d_t outimg )
{
    int column = get_global_id( 0 );
    int line = get_global_id( 1 );
    float red = column / 800.;
    float green = line / 480.;
    float4 color = { red, green, 0, 1 };
    write_imagef( outimg, (int2)(column, line), color );
}
```



Programming Model

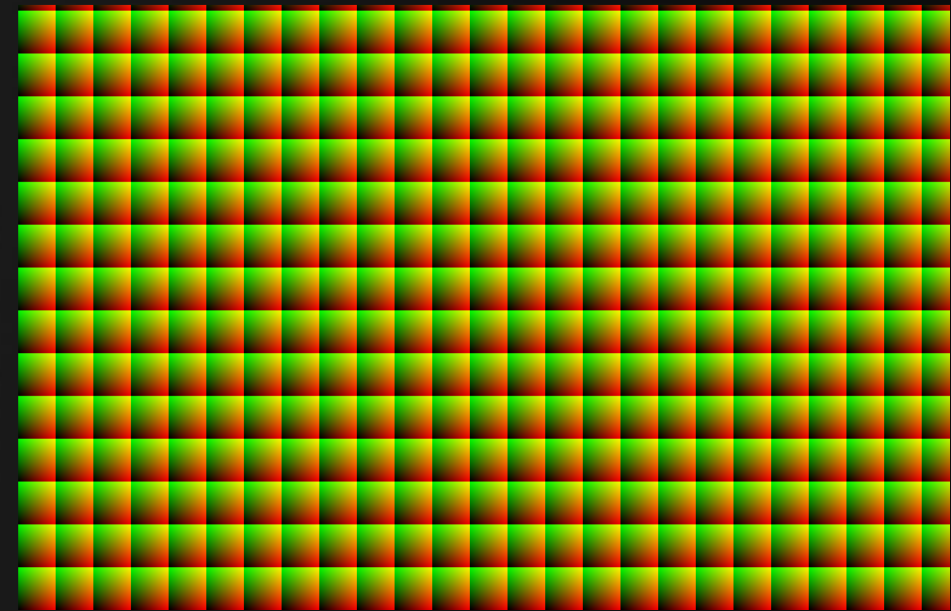
GPGPU Programming Model

Kernels are invoked from the host:

```
size_t workSize[2] = { SCRWIDTH, SCRHEIGHT }, localSize[2] = { 32, 32 };
void Kernel::Run( cl_mem* buffers, int count )
{
    ...
    clEnqueueNDRangeKernel( queue, kernel, 2, 0, workSize, localSize, 0, 0, 0 );
    ...
}
```

Device code:

```
__kernel void main( write_only image2d_t outimg )
{
    int column = get_global_id( 0 );
    int line = get_global_id( 1 );
    float red = get_local_id( 0 ) / 32.;
    float green = get_local_id( 1 ) / 32.;
    float4 color = { red, green, 0, 1 };
    write_imagef( outimg, (int2)(column, line), color );
}
```



Today's Agenda:

- Introduction to GPGPU
- Example: Voronoi Noise
- GPGPU Programming Model
- OpenCL Template



Template

OCL_Lab: The Familiar Template

The OpenCL template is a basic experimentation framework for OpenCL. Game::Tick implements the following functionality:

1. Set arguments for the OpenCL kernel;
2. Execute the OpenCL kernel (which stores output in an OpenGL texture);
3. Draw a full-screen quad using a shader.

You can find the OpenCL code in `program.cl`;
The shader is defined in `vignette.frag`.



/INFOMOV/

END of “GPGPU (1)”

next lecture: “GPGPU (2)”

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

    at a = nt - nc; b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * r);
    (Tr) R = (D * nnt - N * (ddn * ddn));

    E * diffuse;
    = true;

    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;

    MAXDEPTH)

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

    random walk - done properly, closely following
    (survive)

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

