

```
ics
& (depth < MAXDEPTH)
= inside ? 1.0f : 0.0f;
nt = nt / nc; ddn = ddn * ddn;
s2t = 1.0f - nnt * nnt;
D, N );
)
at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * ddn);
Fr) R = (D * nnt - N * (ddn *
E * diffuse;
= true;
-
efl + 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 <
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
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 2018 - Lecture 10: "GPGPU (2)"

# Welcome!



# Today's Agenda:

- Practical GPGPU: Verlet Fluid
- (in several steps)

```
...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 > 0 ? 1 : -1));

E * diffuse;
= true;

-
efl + 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
(ive)

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



# Verlet

```

ics
& (depth < MAXDEPTH)
c = inside ? 1.0 : 0.0;
nt = nt / nc, ddn = ddn * c;
cos2t = 1.0f - nnt * ddn;
D, N );
)
at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * c);
Tr) R = (D * nnt - N * (ddn * c));
E * diffuse;
= true;
-
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) * (rad
andom walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```





# Verlet

```

...
    && (depth < MAXDEPTH)
{
    if ( ! inside )
    {
        nt = nt / nc; ddn = ddn * 2.0f - nnt * nnt;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

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

```

# • INGREDIENTS



# Verlet

## Verlet Physics

Motion along a straight line:

$$x_1 = x_0 + v\Delta t$$

For a fixed time step we can express this without explicit velocities:

$$x_2 = x_1 + (x_1 - x_0)$$

Simulation:

- Backup current position:  $x_{current} = x$
- Update positions:  $x = x + (x - x_{previous})$
- Store last position:  $x_{prev} = x_{current}$
- Apply constraints (e.g. walls)

Applying constraints:

- e.g. `if (x < 0) x = 0;`
- ...



# Verlet

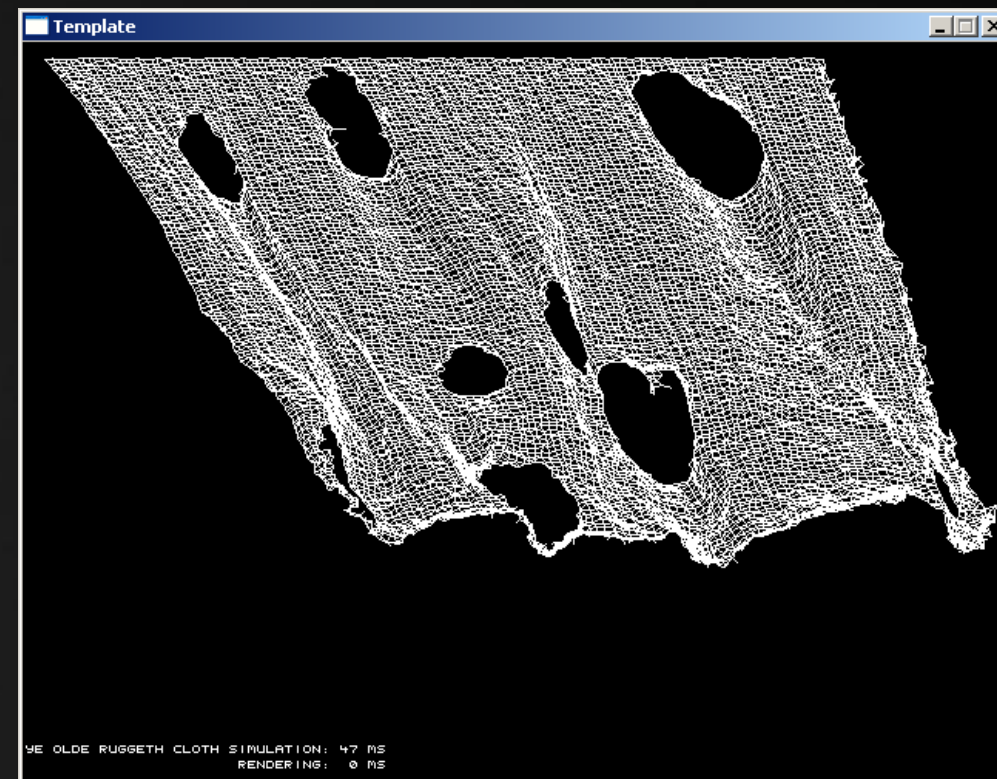
## Verlet Physics

### Cloth:

- Using a grid of vertices
- Forces on all vertices: gravity
- Constraint for top row: fixed position
- Constraint for all vertices: maximum distance to neighbors

### Fluid:

- Using large collection of particles
- Forces on all particles: gravity
- Constraint for all particles: container boundaries
- Constraint for all particles: do not intersect other particles



# Verlet

## Template

### Texture:

- To efficiently display OpenCL output using OpenGL

### Shader:

- As an alternative to OpenCL, e.g. for postprocessing

### Kernel:

- Specifying actual device code:
- Setting and changing arguments:
- Launching the kernel:

```
fractal = new Kernel( "programs/program.cl", "TestFunction" );
fractal->SetArgument( 0, outputBuffer );
fractal->Run( outputBuffer );
```



```

...ics
& (depth < MAXDEPTH)
...
c = inside ? 1 : 0;
nt = nt / nc; ddn = ddn;
ps2t = 1.0f - nnt;
D, N );
0)
...
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, r
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (rand
...
w = 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 S&S
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



# Verlet

## GPU Verlet Fluid

### Input:

- Array of particle positions
- Array of previous particle positions

### Output:

- Visualization of simulation
- Array of particle positions (updated)
- Array of previous particle positions (updated)



# Verlet

## GPU Verlet Fluid

```

...ics
& (depth < MAXDEPTH)
...
= inside ? 1.0f : 0.0f;
nt = nt / nc; ddn = ddn * nc;
ps2t = 1.0f - nnt * ddn;
D, N );
)
...
at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * ps2t);
Tr) R = (D * nnt - N * (ddn * ps2t));
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse, 1);
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
...
andom 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;

```

# .STAGE 1

*Drawing a number of moving particles using OpenCL*

*What if they touch the same pixel?*

Idea:

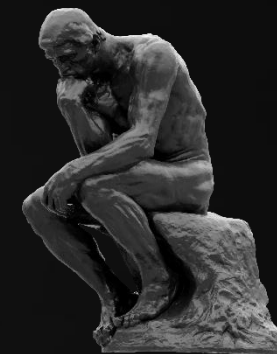
*Let's draw 128 balls, brute force.*

Data:

- Screen buffer, 800x480
- Ball data, 128 records

Procedure:

1. Clear screen
2. Update ball positions
3. Draw balls



Drawing balls, options:

- Loop over balls
- Loop over pixels

*Check 128 balls per pixel*



# Verlet

## GPU Verlet Fluid – Host Code

```

...
    & (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) * ps2t);
    Tr) R = (D * nnt - N * (ddn * ps2t));
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
    survive = SurvivalProbability( diffuse, 1);
    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 ) * 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
    (survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

```

Buffer* balls = new Buffer( BALLCOUNT * 6 );
// put initial ball positions in buffer
float* fb = (float*)balls->GetHostPtr();
for( int i = 0; i < BALLCOUNT; i++ )
{
    fb[i * 6] = Rand( 1 );
    fb[i * 6 + 1] = Rand( 1 );
    fb[i * 6 + 2] = Rand( 0.01f ) - 0.005f;
    fb[i * 6 + 3] = Rand( 0.01f ) - 0.005f;
    fb[i * 6 + 4] = fb[i * 6 + 0];
    fb[i * 6 + 5] = fb[i * 6 + 1];
}
balls->CopyToDevice();

```

position

velocity (for now)



# Verlet

## GPU Verlet Fluid – Device Code

```

__kernel void clear( write_only image2d_t outimg )
{
    int column = get_global_id( 0 );
    int line = get_global_id( 1 );
    if ((column >= 800) || (line >= 480)) return;
    write_imagef( outimg, (int2)(column, line), 0 );
}

```

```

__kernel void update( global float* balls )
{
    int idx = get_global_id( 0 );
    balls[idx * 6 + 0] += balls[idx * 6 + 2];
    balls[idx * 6 + 1] += balls[idx * 6 + 3];
}

```

Task:

- write a single black pixel.

Workset:

- number of pixels.

Task:

- Update the position of one ball.

Workset:

- Number of balls.



# Verlet

## GPU Verlet Fluid – Host Code

```

__kernel void render( write_only image2d_t outimg, global float* balls )
{
    int column = get_global_id( 0 );
    int line = get_global_id( 1 );
    float2 uv = { (float)column / 800.0, (float)line / 480.0 };
    for( int i = 0; i < BALLCOUNT; i++ )
    {
        float2 pos = { balls[i * 6], balls[i * 6 + 1] };
        float dist = length( pos - uv );
        if (dist > 0.02f) continue;
        write_imagef( outimg, (int2)(column, 479 - line), (float4)(1,0,0,1) );
        break;
    }
}

```



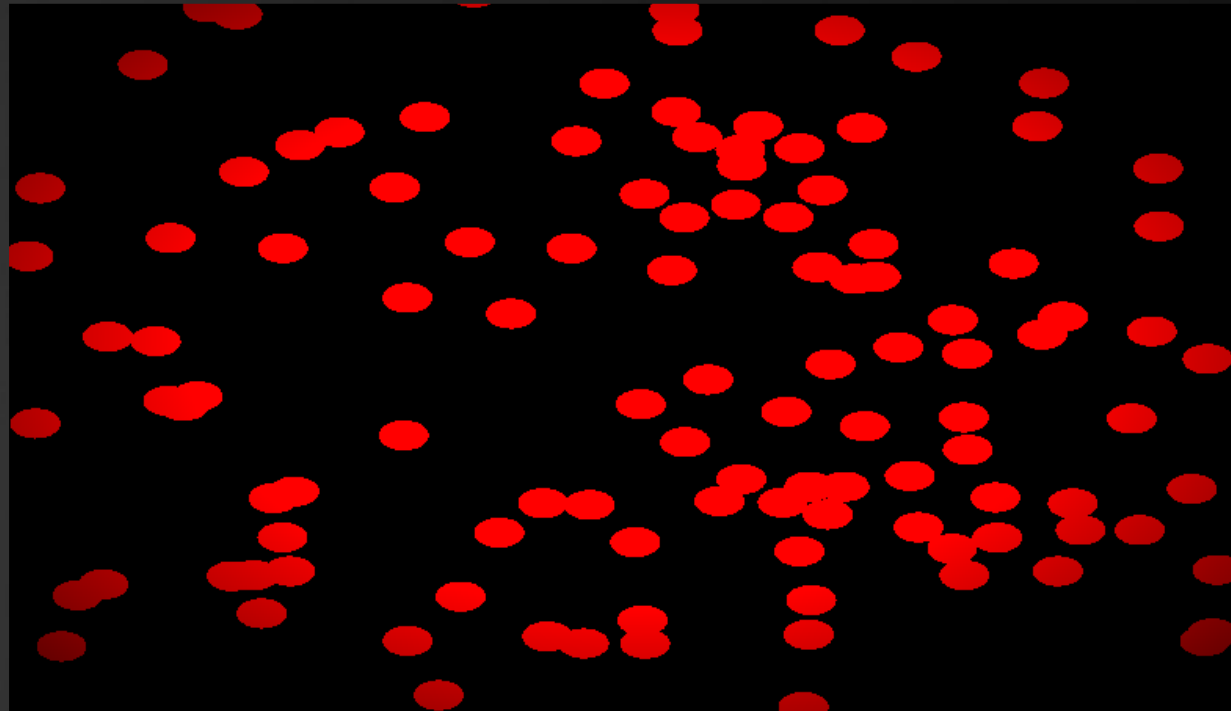
# Verlet

## GPU Verlet Fluid – Result

```

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) * cos2t);
        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
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, $pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



# Verlet

## GPU Verlet Fluid

```

...ics
& (depth < MAXDEPTH)
...
= inside ? 1.0 : 0.0;
nt = nt / nc; ddn = ddn * nc;
os2t = 1.0f - nnt * nnt;
D, N );
)
...
at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (ddn
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse, 1.0f -
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light;
e.x + radiance.y + radiance.z) > 0) && (rand <
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
...
andom walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

# • STAGE 2

*Rendering many particles efficiently*

Idea:

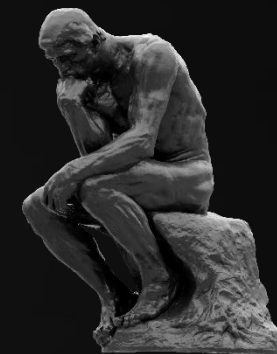
*Let's use a grid to reduce the number of balls we check per pixel.*

Data:

- Grid, custom resolution
- Fixed room per cell for N balls

Procedure:

1. Clear grid
2. Add balls to grid
3. Render pixels.



# Verlet

## GPU Verlet Fluid – Grid

Host:

```
grid = new Buffer( GRIDX * GRIDY * (BALLSPERCELL + 1) *
                 sizeof( unsigned int ) );
```

Device:

```
__kernel void clearGrid( global unsigned int* grid )
{
    int idx = get_global_id( 0 );
    int baseIdx = idx * (BALLSPERCELL + 1);
    grid[baseIdx] = 0;
}
```

Data layout:

- [0]: ball count for cell
- [1..N]: ball indices

Task:

- Reset a grid cell by setting ball count to 0.

Workset:

- Number of cells.



# Verlet

## GPU Verlet Fluid – Grid

```

__kernel void fillGrid( global float* balls, global unsigned int* grid )
{
    int ballIdx = get_global_id( 0 );
    int gx = balls[ballIdx * 6 + 0] * GRIDX;
    int gy = balls[ballIdx * 6 + 1] * GRIDY;
    if ((gx < 0) || (gy < 0) || (gx >= GRIDX) || (gy >= GRIDY)) return;
    int baseIdx = (gx + gy * GRIDX) * (BALLSPERCELL + 1);
    int count = grid[baseIdx]++;
    grid[baseIdx + count + 1] = ballIdx;
}

```

### Task:

- Add a single ball to the correct grid cell.

### Workset:

- Number of balls.



# Verlet

## GPU Verlet Fluid – Grid

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        ps2t = 1.0f - nnt * nnt;
        D, N );
    }
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * ps2t);
    Tr) R = (D * nnt - N * (ddn *
    E * diffuse;
    = true;
    refl + refr) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, I,
    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;

```

```

__kernel void fillGrid( global float* balls, global unsigned int* grid )
{
    int ballIdx = get_global_id( 0 );
    int gx = balls[ballIdx * 6 + 0] * GRIDX;
    int gy = balls[ballIdx * 6 + 1] * GRIDY;
    if ((gx < 0) || (gy < 0) || (gx >= GRIDX) || (gy >= GRIDY)) return;
    int baseIdx = (gx + gy * GRIDX) * (BALLSPERCELL + 1);
    unsigned int count = atomic_inc( grid + baseIdx );
    if (count < BALLSPERCELL) grid[baseIdx + count + 1] = idx; else
    {
        balls[ballIdx * 6 + 1] = balls[ballIdx * 6 + 5] = 0.1;
        grid[baseIdx] = BALLSPERCELL;
    }
}

```



# Verlet

## GPU Verlet Fluid – Grid

```

__kernel void render( write_only image2d_t outimg, global float* balls,
                    global unsigned int* grid )
{
    int column = get_global_id( 0 );
    int line = get_global_id( 1 );
    if ((column >= 800) || (line >= 480)) return;
    float2 uv = { (float)column / 800.0, (float)line / 480.0 };
    // draw balls using grid
    int gx = uv.x * GRIDX;
    int gy = uv.y * GRIDY;
    int gx1 = max( 0, gx - 1 ), gx2 = min( GRIDX - 1, gx + 1 );
    int gy1 = max( 0, gy - 1 ), gy2 = min( GRIDY - 1, gy + 1 );
    ...
}

```



# Verlet

## GPU Verlet Fluid – Grid

```

...
    & (depth < MAXDEPTH)
...
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        rns2t = 1.0f - nnt * rns2t;
        D, N );
    }
...
    at a = nt - nc; b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * rns2t);
    Tr) R = (D * nnt - N * (ddn * rns2t));
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, rns2t );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light );
e.x + radiance.y + radiance.z) > 0) && (rand < survive)
...
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;

```

...

```

for( int y = gy1; y <= gy2; y++ ) for( int x = gx1; x <= gx2; x++ )
{
    unsigned int baseIdx = (x + y * GRIDX) * (BALLSPERCELL + 1);
    unsigned int count = grid[baseIdx];
    for( int i = 0; i < count; i++ )
    {
        unsigned int ballIdx = grid[baseIdx + i + 1];
        float2 pos = { balls[ballIdx * 6], balls[ballIdx * 6 + 1] };
        float dist = length( pos - uv );
        if (dist > 0.01f) continue;
        write_imagef( outimg, (int2)(column, 479 - line), (float4)(1,0,0,1) );
    }
}

```



# Verlet

## GPU Verlet Fluid – Grid - Result

```

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

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

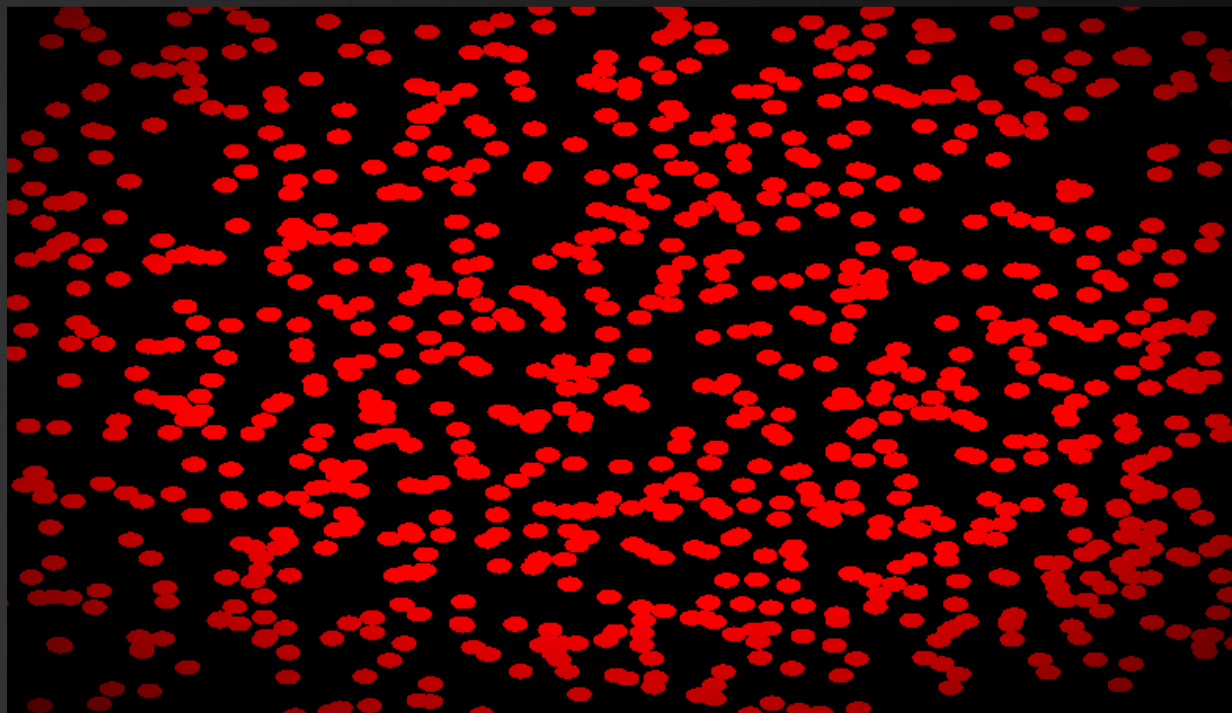
    E * diffuse;
    = true;

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

    MAXDEPTH)

    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, closely 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;
    }
    }
}

```



# Verlet

## GPU Verlet Fluid

# .STAGE 3

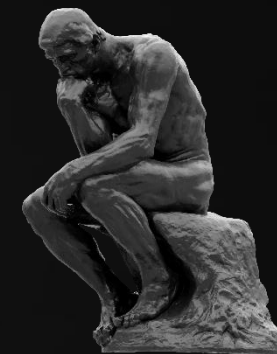
### Implementing simulation

Idea:

*Basics work; let's add some physics.*

Procedure:

1. Move particles
2. Satisfy constraints



# Verlet

## GPU Verlet Fluid – Simulation

```

__kernel void simulate1( global float* balls )
{
    int idx = get_global_id( 0 );
    float2 prevPos = { balls[idx * 6 + 0], balls[idx * 6 + 1] };
    float2 delta = { balls[idx * 6 + 0] - balls[idx * 6 + 4],
                    balls[idx * 6 + 1] - balls[idx * 6 + 5] + 0.0002 };
    float speed = length( delta );
    if (speed > 0.01f) delta = 0.01f * normalize( delta );
    balls[idx * 6 + 0] += delta.x;
    balls[idx * 6 + 1] += delta.y;
    balls[idx * 6 + 4] = prevPos.x;
    balls[idx * 6 + 5] = prevPos.y;
}

```



# Verlet

## GPU Verlet Fluid – Simulation

```

...
    & (depth < MAXDEPTH)
...
    = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn * nc;
    rns2t = 1.0f - nnt * nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * rns2t);
    Tr) R = (D * nnt - N * (ddn * nnt));
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, 1.0f );
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;

```

```

__kernel void simulate2( global float* balls, global float* balls2,
                        global unsigned int* grid )
{
    int cellIdx = get_global_id( 0 );
    int baseIdx = cellIdx * (BALLSPERCELL + 1);
    int count = grid[baseIdx];
    if (count == 0) return;
    int gx = idx % GRIDX;
    int gy = idx / GRIDX;
    // determine 3x3 block around current cell
    int gx1 = max( 0, gx - 1 ), gx2 = min( GRIDX - 1, gx + 1 );
    int gy1 = max( 0, gy - 1 ), gy2 = min( GRIDY - 1, gy + 1 );
    for( int i = 0; i < count; i++ )
    {

```



# Verlet

## GPU Verlet Fluid – Simulation

```

// get active ball
int idx1 = grid[baseIdx + i + 1];
float2 ball1Pos = { balls[idx1 * 6 + 0], balls[idx1 * 6 + 1] };
// evade other balls
for( int y = gy1; y <= gy2; y++ ) for( int x = gx1; x <= gx2; x++ )
{
    int baseIdx = (x + y * GRIDX) * (BALLSPERCELL + 1);
    int count2 = min( (unsigned int)BALLSPERCELL, grid[baseIdx] );
    for( int j = 0; j < count2; j++ )
    {
        int idx2 = grid[baseIdx + j + 1];
        if (idx2 != idx1)
        {
            float2 ball2Pos = { balls2[idx2 * 6 + 0], balls2[idx2 * 6 + 1] };
            ...
        }
    }
}

```



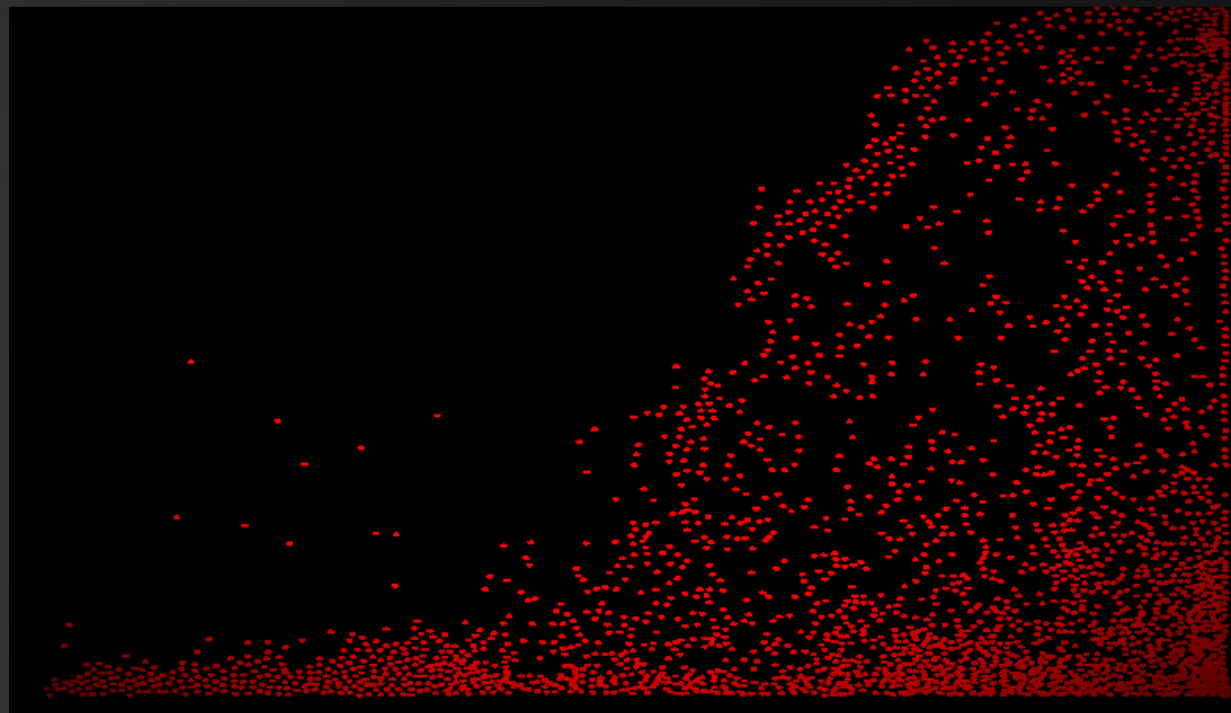
# Verlet

## GPU Verlet Fluid – Simulation

```

...
    & (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) * cos2t);
    Tr) R = (D * nnt - N * (ddn * cos2t));
...
    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
(ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



# Verlet

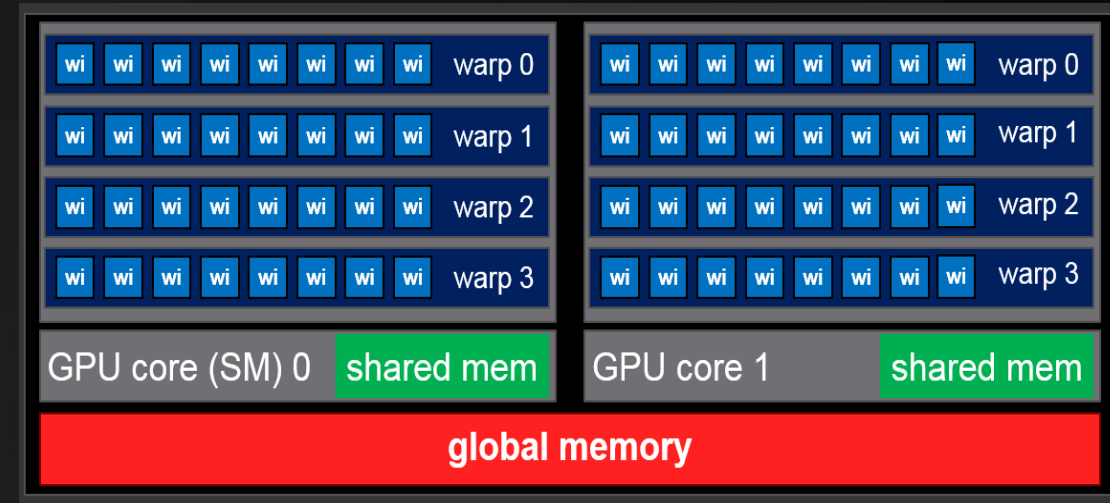
## GPU Verlet Fluid

What causes the poor performance?

```

ics
& (depth < MAXDEPTH)
    if (inside ? 1 : 0) {
        nt = nt / nc; ddn = ddn * nc;
        ps2t = 1.0f - nnt * nnt;
        D, N );
    }
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * ps2t);
    Tr) R = (D * nnt - N * (ddn * nnt));
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, L );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    e.x + radiance.y + radiance.z) > 0) && (rand < psurvive)
    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;
    
```

- Simulation handles one grid cell *per thread*
- Grid cell workload is highly irregular
- Do we even have enough grid cells?



# Verlet

## GPU Verlet Fluid

```

...
    & (depth < MAXDEPTH)
...
    c = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn * c;
    cos2t = 1.0f - nnt * nnt;
    D, N );
...
    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)

```

# • STAGE 4

## Improving performance

```

survive = SurvivalProbability( diffuse, I, &L, &light);
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (cosThetaOut > 0)
...
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
(ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Idea:

*Grid cells are filled irregularly; loop over balls for simulation.*

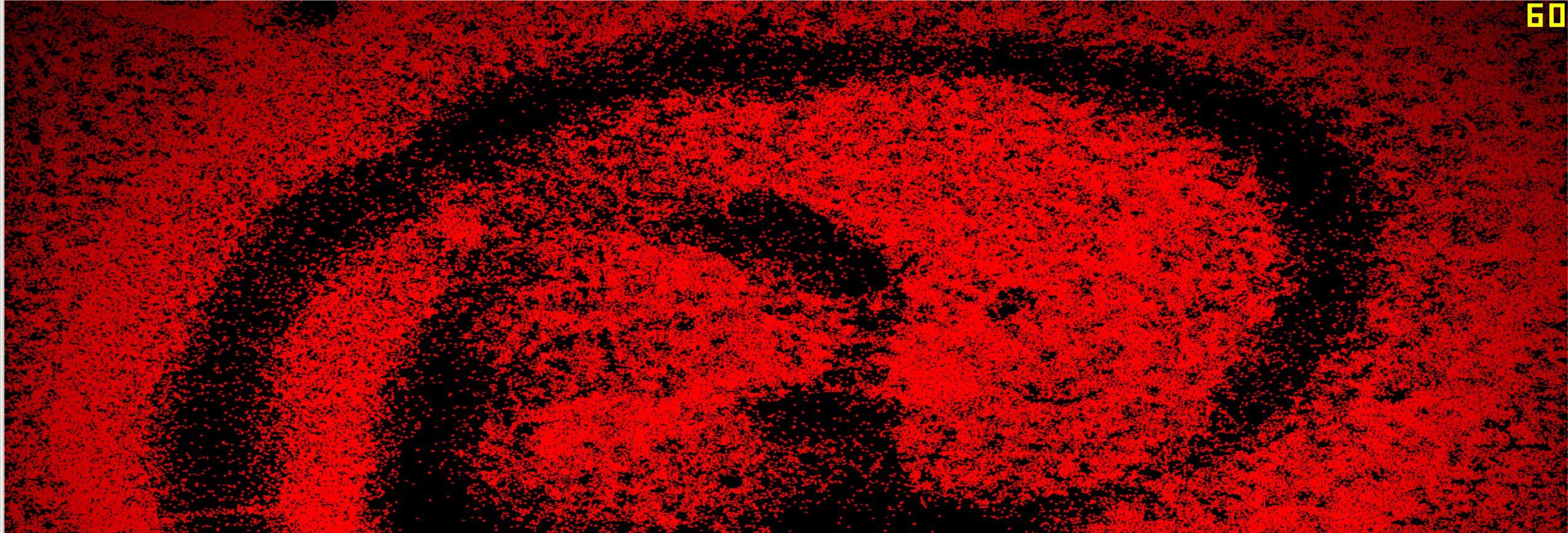
Procedure, simulation:

1. A ball checks its surroundings in the grid.

Procedure, rendering (new):

- For rendering we loop over balls too. If two balls fight for the same pixel, we ignore that.





FRAPS movies

General **99** FPS Movie

Folder to save movies in  
C:\Fraps\Movies

Video Capture Hotkey  
F9 Disable

Video Capture Settings  
 60 fps  Full-size  
 50 fps  Half-size  
 30 fps  
 29.97

Loop buffer length  seconds

Sound Cap...  
 Recor...  
 Recor...  
Device n...  
 Onl...  
 Hide n...  
 Lock f...

# Verlet

## GPU Verlet Fluid - TakeAway

GPGPU is a bit different:

- We have ‘host’ and ‘device’ code
- We need many small identical tasks
- Each task has an ‘identity’ (1D, 2D or 3D index in the workset)
- Some tasks may be outside the workset (check for this!)
- Ideally, each of those tasks should do a similar amount of work (if, for)
- The tasks run in parallel: mind concurrency issues! (atomic)
- Data transfer from CPU to GPU is expensive (avoid this)



*In this example, OpenCL directly plotted to an OpenGL texture (which is then drawn on a quad, using a shader). It is probably more efficient to let OpenCL prepare a vertex buffer for drawing point sprites.*

```

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) *
Tr) R = (D * nnt - N * (ddn
E * diffuse;
= true;
efl + 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) && (rand
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:

- Practical GPGPU: Verlet Fluid
- (in several steps)

```
...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) * cos2t);
Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));

E * diffuse;
= true;

-
efl + 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
(ive)

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



/INFOMOV/

END of “GPGPU (2)”

next lecture: GPGPU (3)

```
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) * ddn);
Tr) R = (D * nnt - N * (ddn * nnt));

E * diffuse;
= true;

efl + 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
ive)

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

