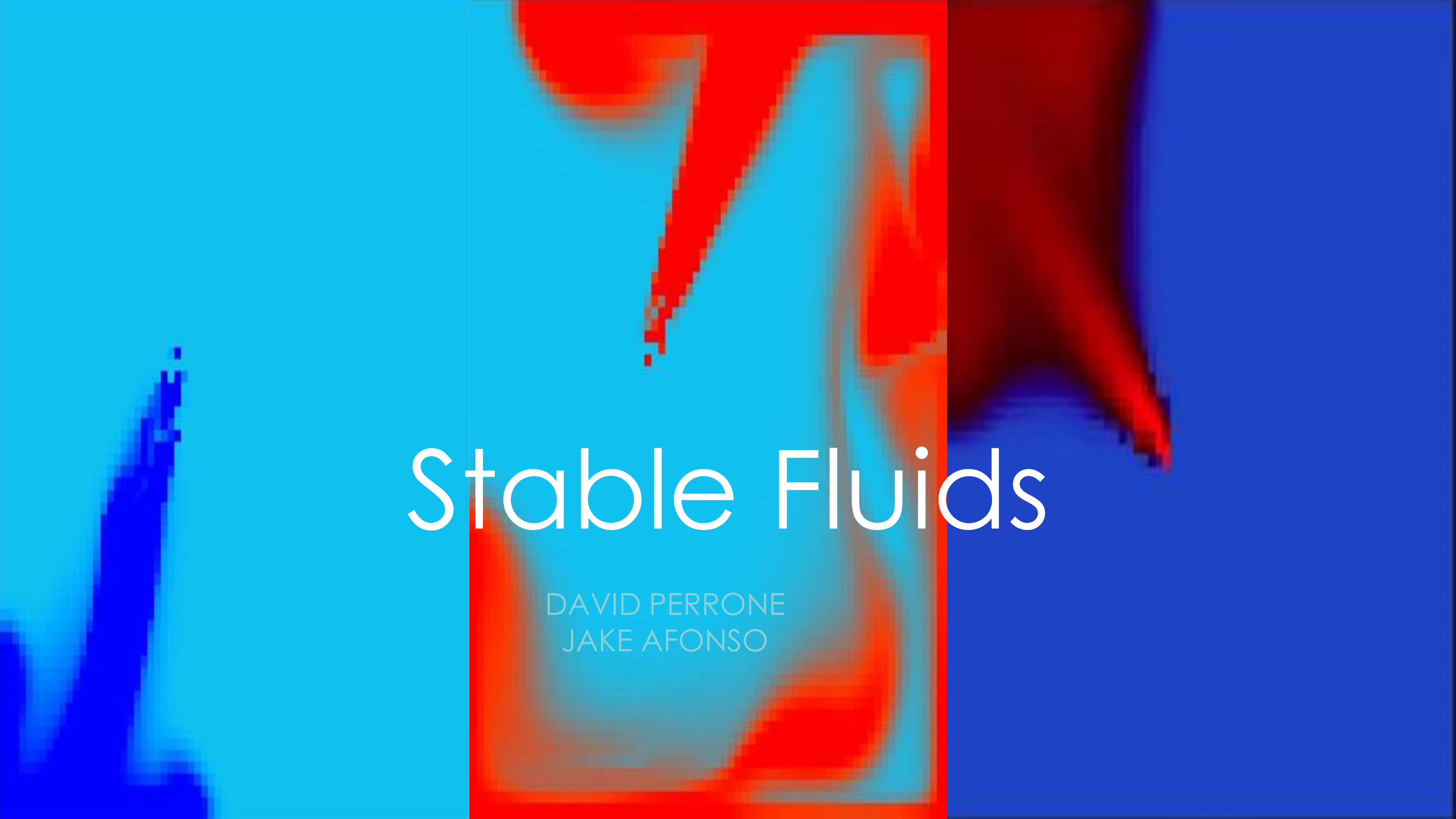


Stable Fluids

DAVID PERRONE
JAKE AFONSO

Presentation overview

- Algorithm
- Parallelization
- Rendering
- Demo

Background

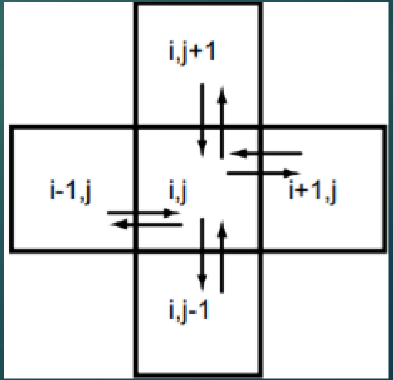
- Simulates a square of a fluid, with a defined viscosity and diffusion
  - Viscosity is the thickness of the fluid
  - Diffusion is how fast density spreads through the fluid
- A “dye” is added at some point of the square with a given velocity
- Dye moves throughout the fluid based on the fluid dynamics defined in the Navier-Stokes equations
- Stable in the fact that there are no bounds on the time steps and the algorithms never “blow up”

Diffusion

- Represents the property of a dye spreading out throughout a fluid.
- Happens for the dye itself and less obviously for the velocities of the fluid
- Can happen differently based on the viscosity/diffusion value of the liquid
  - For example, putting dye in water will spread much faster than in molasses
  - For our simulation, velocity diffusion is based on the viscosity value and density diffusion is based on the diffusion value
- Each particle exchanges density and velocity with its 4 direct neighbors

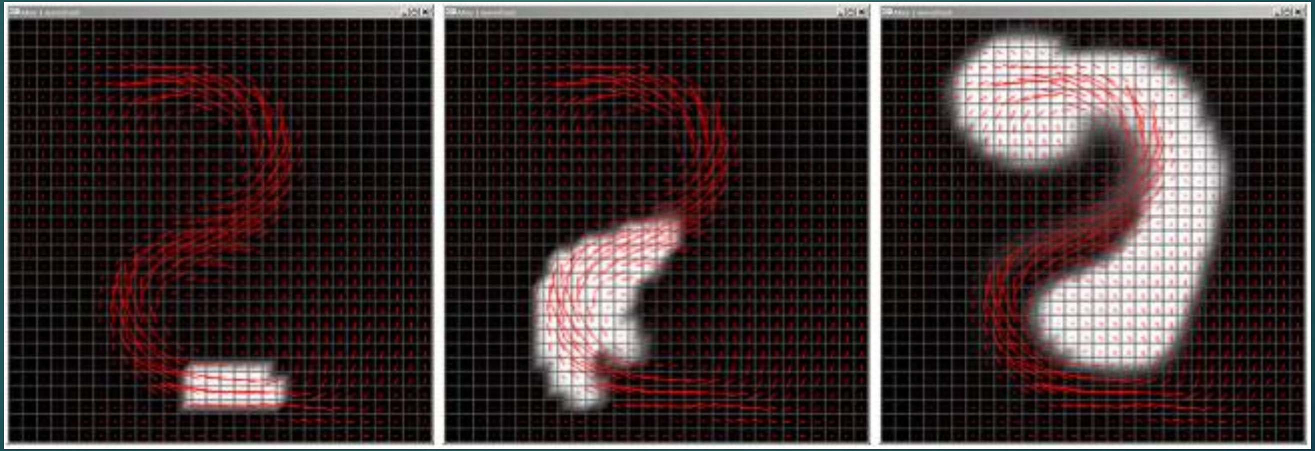
Start

End



Advection

- Represents the movement of the dye based on the velocity field within the fluid
- If a liquid is moving, it will carry the dye in that direction as well
- Similarly to diffusion, this happens for the dye and the velocities themselves



Advection carries the dye through the velocity field

Projection

- This upholds the incompressible property of the fluid
- The amount of fluid coming into the square must equal the amount of fluid leaving the square
- The other 2 functions mess with the equilibrium of the fluid, and project cleans it up after

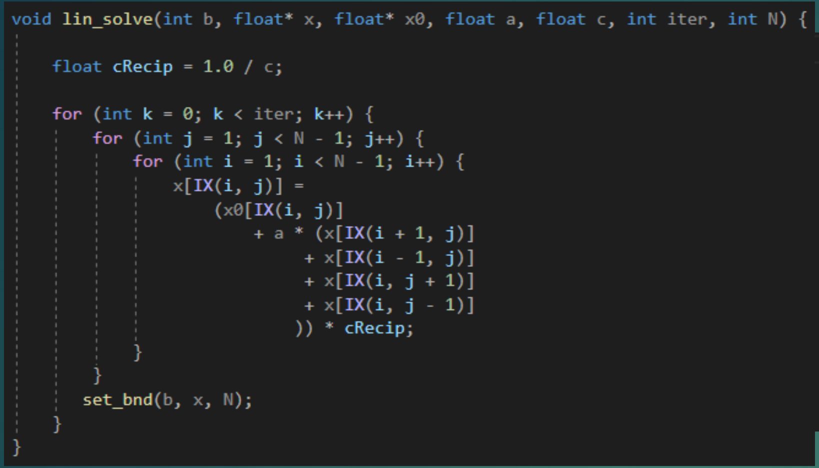
Subroutines

Linear Solve	All 3 of these functions are based on approximating a solution to a linear system
	Implements the Gauss-Seidel method → iterative solver
Set bnd	These functions all only deal with the inner part of the square
	Set boundary sets the edges to be the opposite of the neighboring inner particle
	Gives the effect of bouncing off the walls

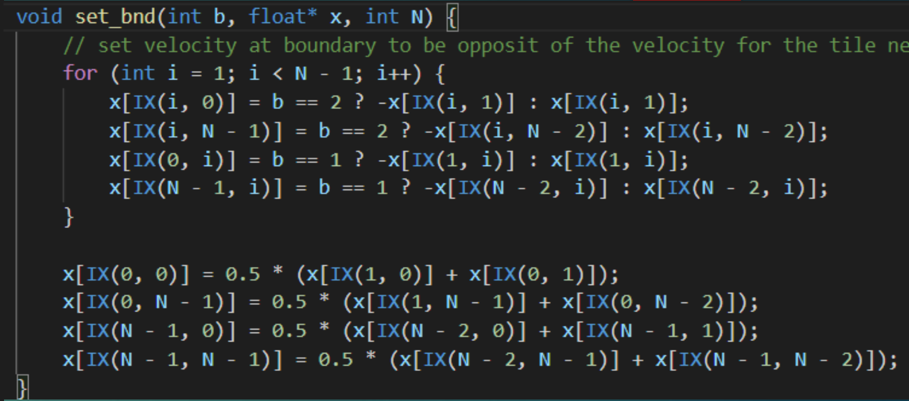
Parallelization

- Used cuda to implement a parallel version of the simulation
- Defined num_threads/num_blks to have each thread be assigned a single particle
- Diffuse, advect, and project functions into __global__ functions to launch as kernels
- Lin_solve and set_bnd into __device__ functions to be called from within the kernels
- Performed slower than serial for n < 6400 particles, but then started to outperform for n > 6400.
    - Overall close to constant time for increasing size, while serial was exponential

Serial:

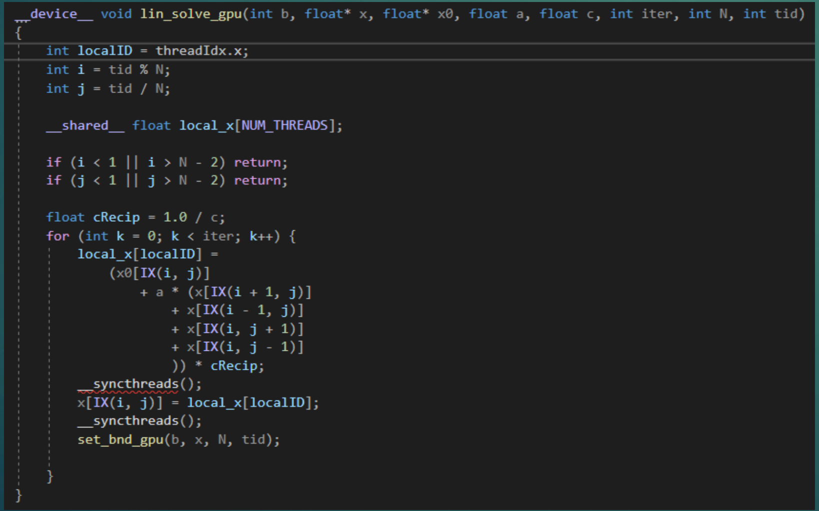


```
void lin_solve(int b, float* x, float* x0, float a, float c, int iter, int N) {

    float cRecip = 1.0 / c;

    for (int k = 0; k < iter; k++) {
        for (int j = 1; j < N - 1; j++) {
            for (int i = 1; i < N - 1; i++) {
                x[IX(i, j)] =
                    (x0[IX(i, j)]
                        + a * (x[IX(i + 1, j)]
                             + x[IX(i - 1, j)]
                             + x[IX(i, j + 1)]
                             + x[IX(i, j - 1)]
                           )) * cRecip;
            }
        }
        set_bnd(b, x, N);
    }
}
```

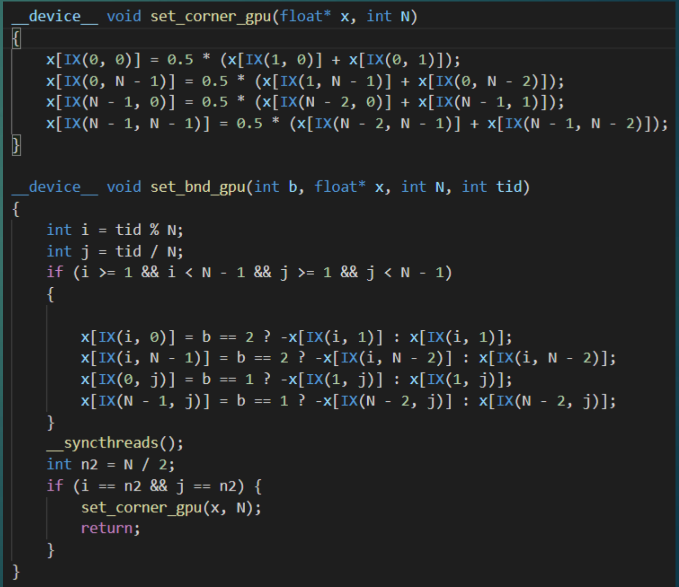


```
void set_bnd(int b, float* x, int N) {
    // set velocity at boundary to be opposit of the velocity for the tile ne
    for (int i = 1; i < N - 1; i++) {
        x[IX(i, 0)] = b == 2 ? -x[IX(i, 1)] : x[IX(i, 1)];
        x[IX(i, N - 1)] = b == 2 ? -x[IX(i, N - 2)] : x[IX(i, N - 2)];
        x[IX(0, i)] = b == 1 ? -x[IX(1, i)] : x[IX(1, i)];
        x[IX(N - 1, i)] = b == 1 ? -x[IX(N - 2, i)] : x[IX(N - 2, i)];
    }

    x[IX(0, 0)] = 0.5 * (x[IX(1, 0)] + x[IX(0, 1)]);
    x[IX(0, N - 1)] = 0.5 * (x[IX(1, N - 1)] + x[IX(0, N - 2)]);
    x[IX(N - 1, 0)] = 0.5 * (x[IX(N - 2, 0)] + x[IX(N - 1, 1)]);
    x[IX(N - 1, N - 1)] = 0.5 * (x[IX(N - 2, N - 1)] + x[IX(N - 1, N - 2)]);
}
```

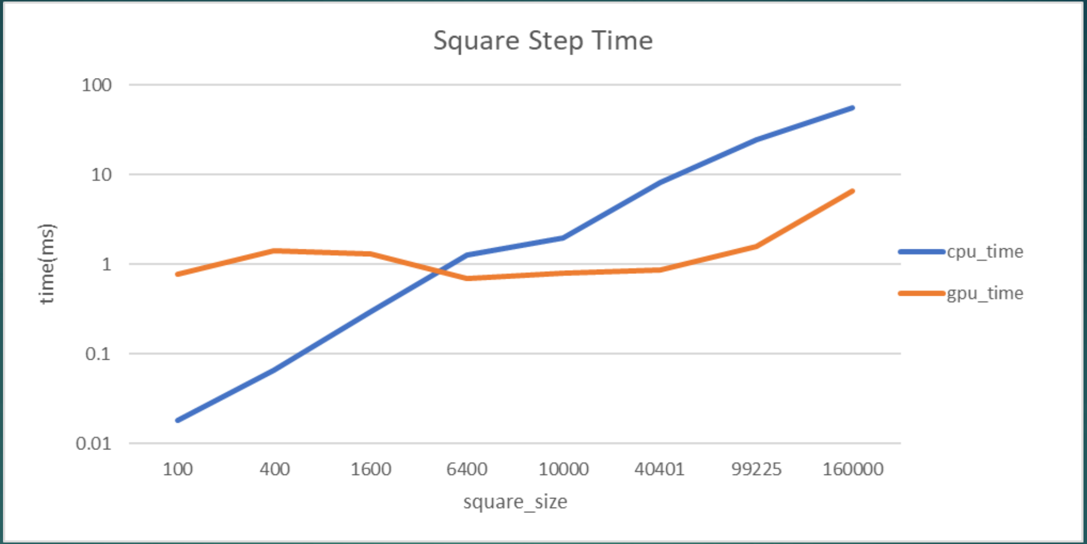
Cuda:

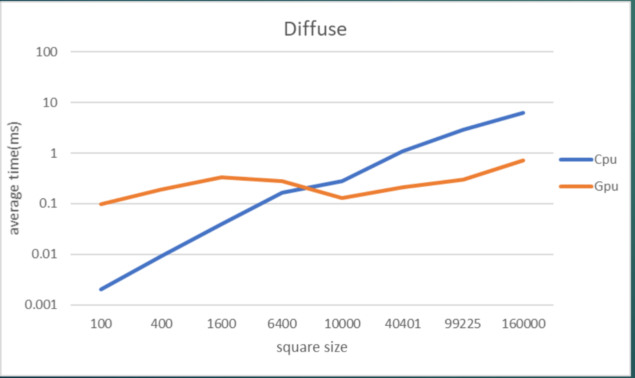


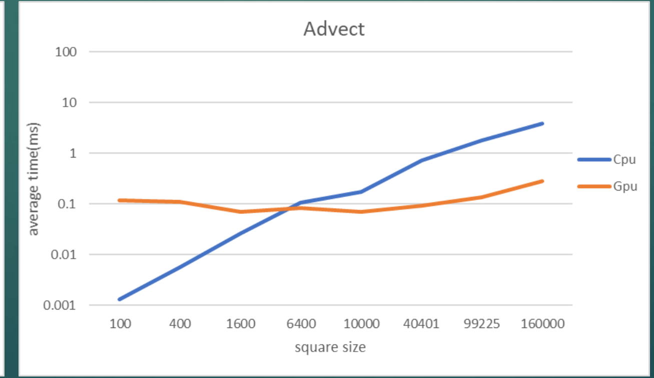
```
__device__ void lin_solve_gpu(int b, float* x, float* x0, float a, float c, int iter, int N, int tid)
{
    int localID = threadIdx.x;
    int i = tid % N;
    int j = tid / N;

    __shared__ float local_x[NUM_THREADS];

    if (i < 1 || i > N - 2) return;
    if (j < 1 || j > N - 2) return;

    float cRecip = 1.0 / c;
    for (int k = 0; k < iter; k++) {
        local_x[localID] =
            (x0[IX(i, j)]
                + a * (x[IX(i + 1, j)]
                    + x[IX(i - 1, j)]
                    + x[IX(i, j + 1)]
                    + x[IX(i, j - 1)]
                    )) * cRecip;
        __syncthreads();
        x[IX(i, j)] = local_x[localID];
        __syncthreads();
        set_bnd_gpu(b, x, N, tid);

    }
}
```

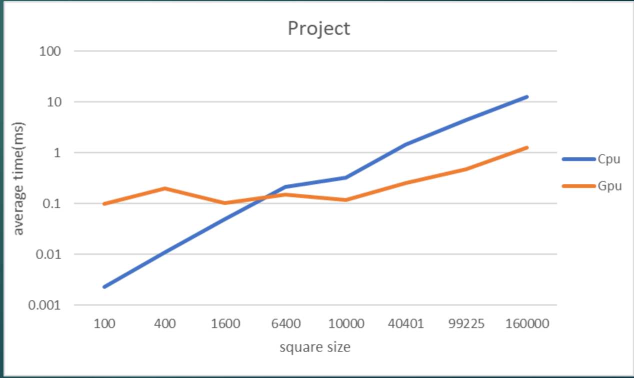


```
__device__ void set_corner_gpu(float* x, int N)
{
    x[IX(0, 0)] = 0.5 * (x[IX(1, 0)] + x[IX(0, 1)]);
    x[IX(0, N - 1)] = 0.5 * (x[IX(1, N - 1)] + x[IX(0, N - 2)]);
    x[IX(N - 1, 0)] = 0.5 * (x[IX(N - 2, 0)] + x[IX(N - 1, 1)]);
    x[IX(N - 1, N - 1)] = 0.5 * (x[IX(N - 2, N - 1)] + x[IX(N - 1, N - 2)]);
}

__device__ void set_bnd_gpu(int b, float* x, int N, int tid)
{
    int i = tid % N;
    int j = tid / N;
    if (i >= 1 && i < N - 1 && j >= 1 && j < N - 1)
    {

        x[IX(i, 0)] = b == 2 ? -x[IX(i, 1)] : x[IX(i, 1)];
        x[IX(i, N - 1)] = b == 2 ? -x[IX(i, N - 2)] : x[IX(i, N - 2)];
        x[IX(0, j)] = b == 1 ? -x[IX(1, j)] : x[IX(1, j)];
        x[IX(N - 1, j)] = b == 1 ? -x[IX(N - 2, j)] : x[IX(N - 2, j)];
    }
    __syncthreads();
    int n2 = N / 2;
    if (i == n2 && j == n2) {
        set_corner_gpu(x, N);
        return;
    }
}
```


Square Step Time
time(ms)
100
10
1
0.1
0.01
100
400
1600
6400
10000
40401
99225
160000
square_size
cpu_time
gpu_time


Diffuse
average time(ms)
100
10
1
0.1
0.01
0.001
100
400
1600
6400
10000
40401
99225
160000
square size
Cpu
Gpu


Advect
average time(ms)
100
10
1
0.1
0.01
0.001
100
400
1600
6400
10000
40401
99225
160000
square size
Cpu
Gpu

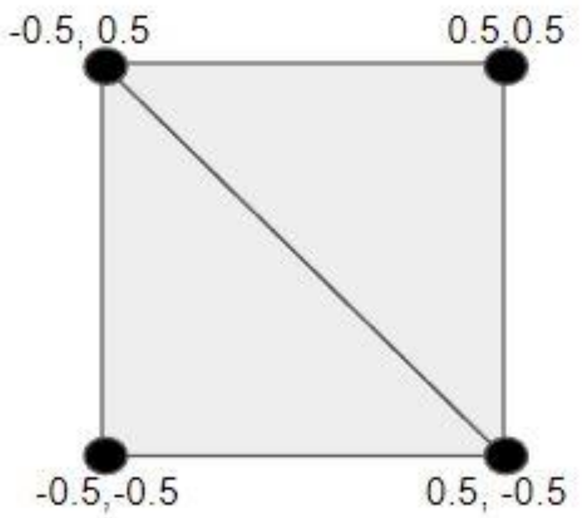

Project
average time(ms)
100
10
1
0.1
0.01
0.001
100
400
1600
6400
10000
40401
99225
160000
square size
Cpu
Gpu

Rendering

- Vertices
- Instanced Rendering
- Cuda OpenGL Interop
- OpenGL Memory layout
- Matrix Multiplication

Rendering in OpenGL

- The fluid square is made up of several smaller squares
- Each square is rendered as two triangles
- Multiply each vertex of the square with a transformation matrix to move/scale
- The color within the square is represented by 4 floats (RGBA)



Instanced Rendering

1 draw call for 1000s of objects

Requires:

- Shape vertices
- Transformation matrices per instance
- Colors per instance

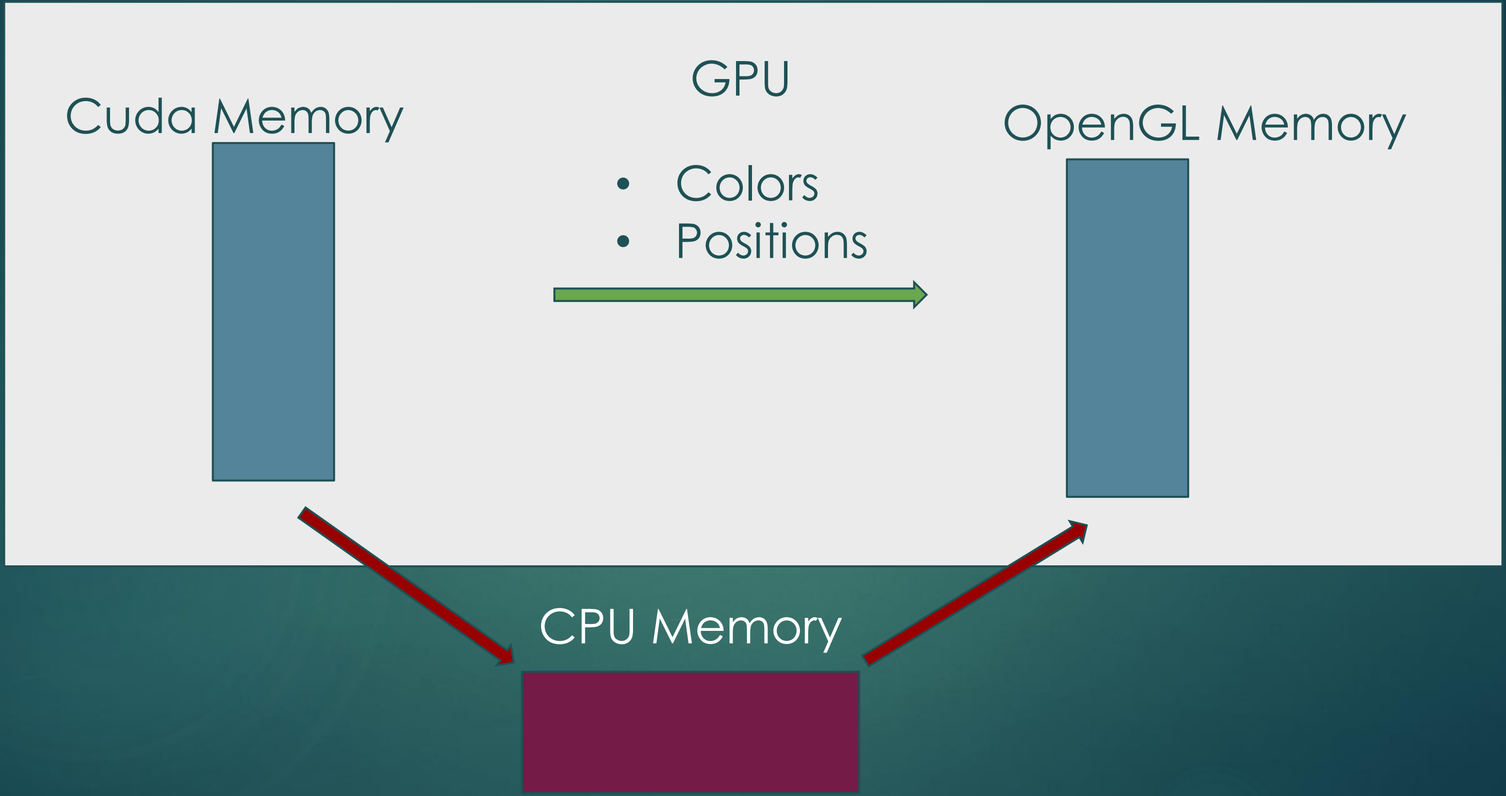
Memory

Cuda performs the square calculations

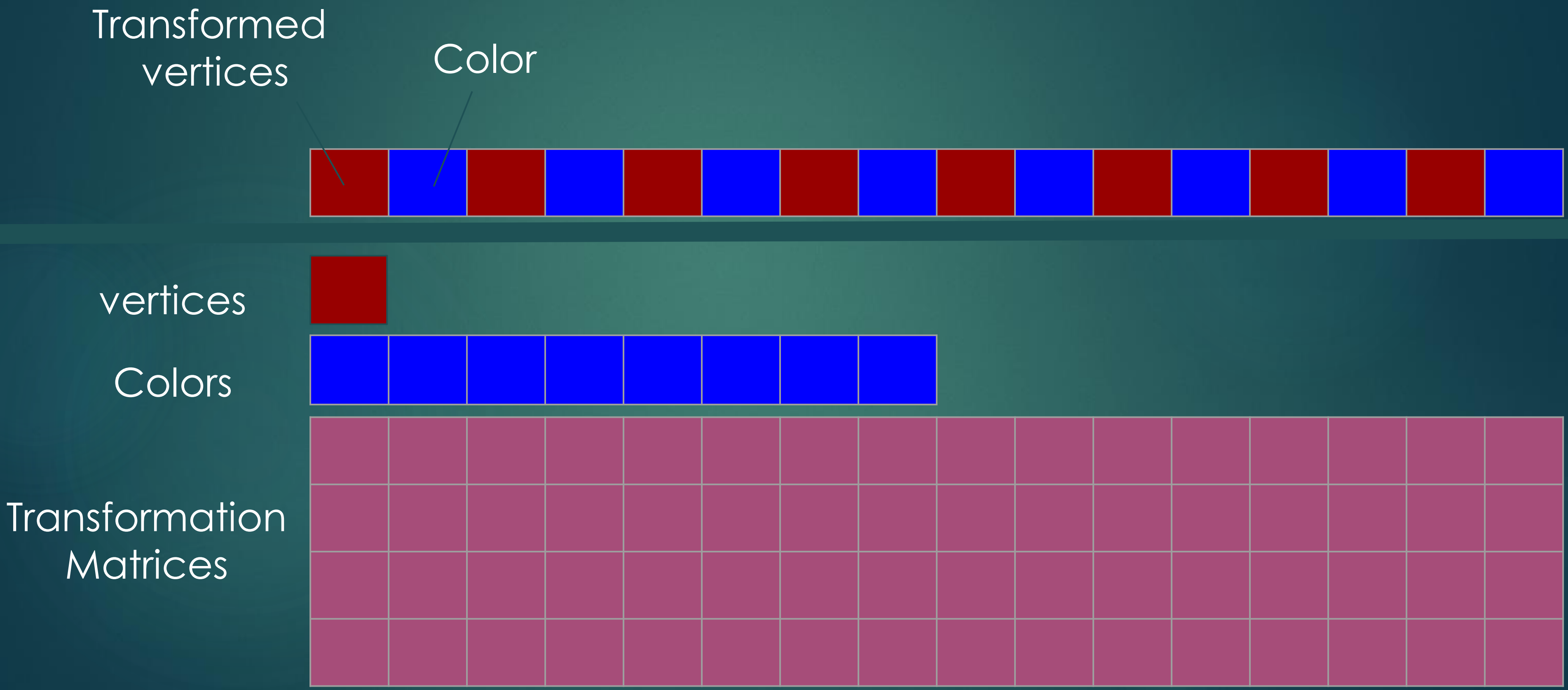
- Stores the location for each square
- Stores the density (color) for each square

OpenGL renders the updated data each frame

Data Transfer



Rearranging the memory

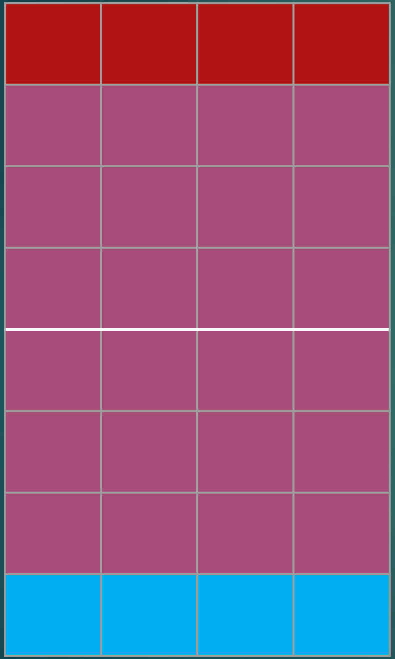


Cuda - OpenGL Communication

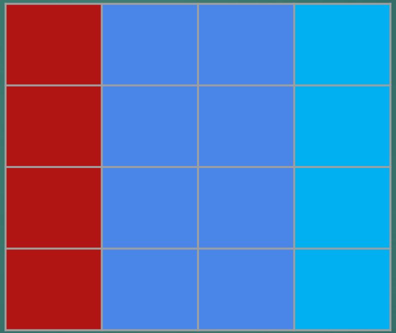
1. Create CudaGraphicsResource
2. Create OpenGL Vertex Array Object
3. Create OpenGL vertex buffer - fill with 4 vertices of square
4. Create OpenGL Matrix Buffer
5. Register CudaGraphicsResource with Matrix Buffer
6. Create translation matrices, multiply with one scale matrix - store into Buffer
7. UnRegister Cuda
8. Repeat with OpenGL Color Buffer
9. NOTE: OpenGL and Cuda cannot use the same buffer simultaneously

Matrix Multiplication

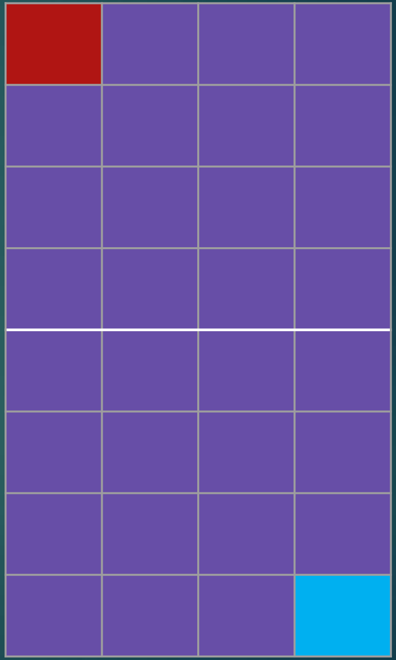
Transformation Matrices



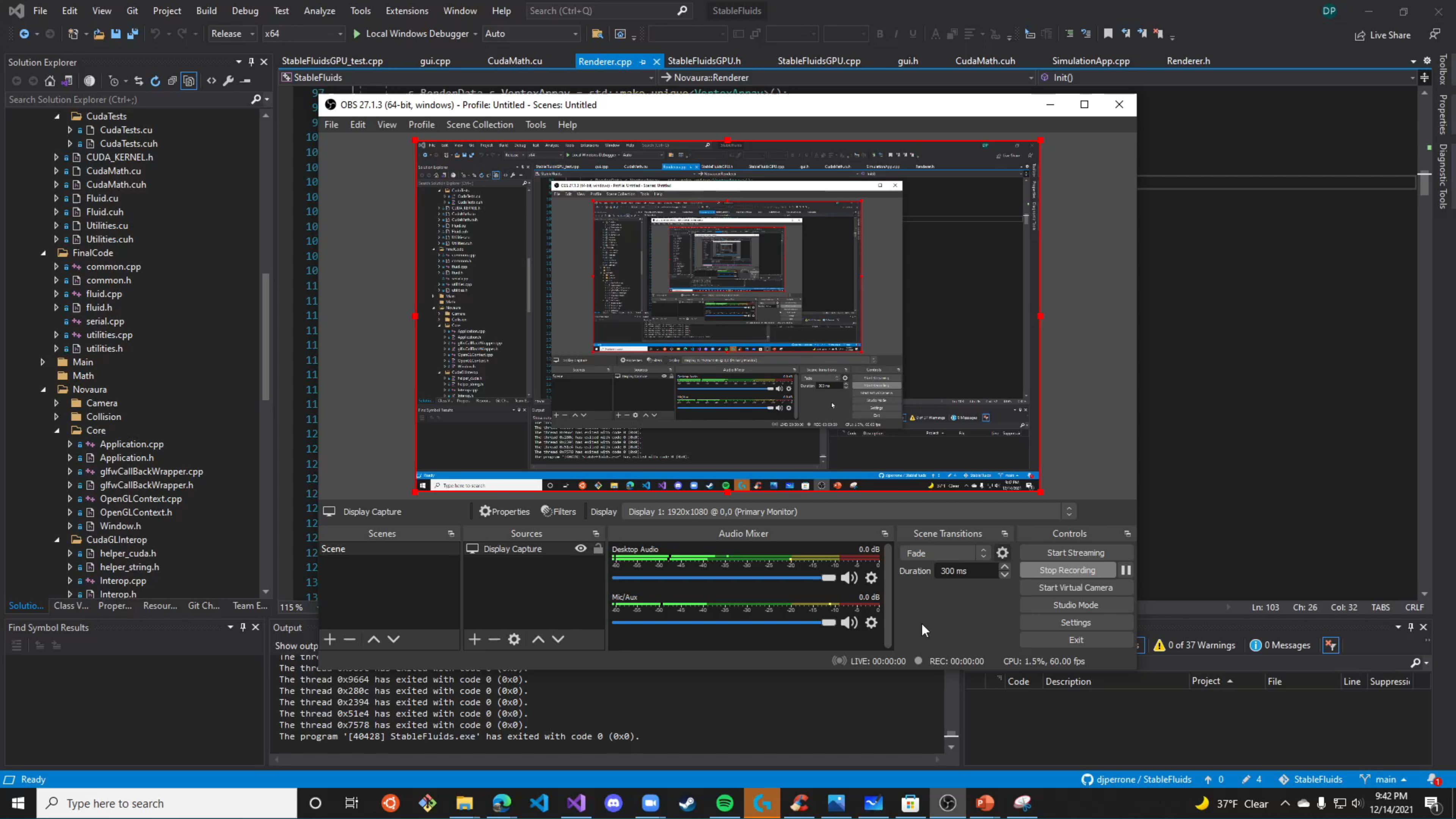
Scale Matrix



OpenGL Matrices



Demo (Mute Audio)



Display 1: 1920x1080 @ 0,0 (Primary Monitor)

Desktop Audio 0.0 dB

Mic/Aux 0.0 dB

Fade

Duration 300 ms

LIVE: 00:00:00 REC: 00:00:00 CPU: 1.5%, 60.00 fps

Release x64 Local Windows Debugger Auto

115 % Ln: 103 Ch: 26 Col: 32 TABS CRLF

0 of 37 Warnings 0 Messages

Output

```
The thread 0x9664 has exited with code 0 (0x0).
The thread 0x280c has exited with code 0 (0x0).
The thread 0x2394 has exited with code 0 (0x0).
The thread 0x51e4 has exited with code 0 (0x0).
The thread 0x7578 has exited with code 0 (0x0).
The program '[40428] StableFluids.exe' has exited with code 0 (0x0).
```

Ready djperrone / StableFluids StableFluids main

37°F Clear 9:42 PM 12/14/2021

Questions?

Github

- djperrone/StableFluids (github.com)