Efficient Neighbor Search for Particle-based Fluids

GPU-Based Neighbor Search Algorithm for Fluid Simulations

Peter Gergely
FMFI UK, 2011

Outline

- Introduction
- Smoothed Particle Hydrodynamics
- Neighbor search
- Zhao's GPU-Based Neighbor search (GPU-BNS)
- Conclusion, Performance and future work

Introduction

- This presentation serves as a quick introduction into the problematic of efficient fluid simulation
- It is based primarily on the following papers:
  - Efficient Neighbor Search for Particle-based Fluids [Onderik, Ďurikovič; 2007] [1]
  - GPU-Based Neighbor-Search Algorithm for Particle Simulations [Serkan Bayraktar a.o.; 2007] [2]
  - A New GPU-Based Neighbor Search Algorithm for Fluid Simulations [Xiangkun Zhao a.o; 2010] [3]

Introduction – Fluid system

- Can be defined as physic based simulations of natural phenomena like rain, waves, smoke, etc.
- These motions can be described by Navier-Stokes equations
- There are two main different solutions for tracking the motions of fluid:
  - The Eulerian method
  - The Lagrangian method

Introduction – Eulerian method

- The *Eulerian approach looks at fixed points in the* volume and measures how the fluid quantities (density, velocity, pressure etc.) change in time, which corresponds to using a fixed grid that doesn't change in space even as the fluid flows through it.
- Fixed mesh-based computation domain
- Regular / Hierarchical grids, Tetrahedral meshes, etc.
- Suitable for full 3D Navier-Stokes equations
- Commonly used algorithms:
  - Marker and Cell (MAC)
  - Volume of Fluid (VOF)
  - Lattice-Boltzmann Method (LBM)

Introduction – Lagrangian method

- The *lagrangian approach represents the motion as a* finite interpolated system like a given number of particles. The fluid quantities are interpolated at special location by weighted sum contributions from the particles.
- Free mesh-based and mesh-less computation domains
- Particles, tetrahedral meshes
- Suitable for full 3D Navier-Stokes equations
- Commonly used algorithms:
  - Smoothed Particle Hydrodynamics (SPH)
  - Moving Particle Semi-implicit (MPS)

Smoothed Particle Hydrodynamics (SPH)

- An interpolated method for fluid simulation.
- Fluid is represented by a set of particles that carry various fluid properties. These properties are distributed around the particles and locating particle neighbors dominates the real-time of a particle-based simulation system.

Benefits

•Mesh-less (grid-less)
•No convection term
•Inherently mass conserving
•Straightforward multiphase extension
•Simple implementations
•Unlimited simulation space
•Suitable Interactive Applications

Drawbacks

•100% incompressible hard to achieve
•Time consuming Surface extraction.

Smoothed Particle Hydrodynamics (SPH) - Principles

- Represent fluid with finite number of particles
- Store all quantities only on particle positions
- Approximate field quantities by convolution
- Uses Lagrangian formulation of Navies-Stokes equations for particle dynamics

- To find all neighbors of a special particle in certain radius range, space subdivision method is often used.
- Different particles are distributed into different grids.
- Particles in the same grid are recognized as neighbors of each other.

Neighbor search

- Method of collision detection
- Space subdivision is one of the ways to speed up the SPH force computation.
- Search for potential inter-particle contacts is done within the grid cells and between immediate neighbors, thus improving the whole simulation speed.

GPU-Based Neighbor Search

- Why should we use a GPU instead of a CPU?
  - GPU has the ability to process multiple particles in parallel
- What kind of problems while using GPU have to be overcome?
  - Fragment shaders, that are used as the main processing unit, are not capable of scatter.
    - They cannot write a value to a memory location for a computed address since fragment programs run using precomputed texture addresses only, and these addresses cannot be changed by the fragment program itself.
    - This limitation makes several basic algorithmic operations (such as counting, sorting, finding maximum and minimum) difficult.

GPU-Based Neighbor Search

- How to overcome the outlined problems?
  - One of the common methods is to use a uniform grid to subdivide the simulation space.
  - A stencil buffer can be used for dealing with multiple photons residing in the same cell.
  - Bucket textures can be used to represent a 3D grid structure
- Zhao's [3] method being described next is based on Bayraktar's method [2]

Zhao's GPU-Based Neighbor Search

- Summary of Zhao's algorithm[3] as shown in Figure 1:
  1. Construct a grid map from the position attribute texture
  2. Backup grid map texture from the previous step as grid map source
  3. Sort grid map texture from the first step
  4. Construct neighbor map using the output from the position attribute texture, grid map source and sorted grid map

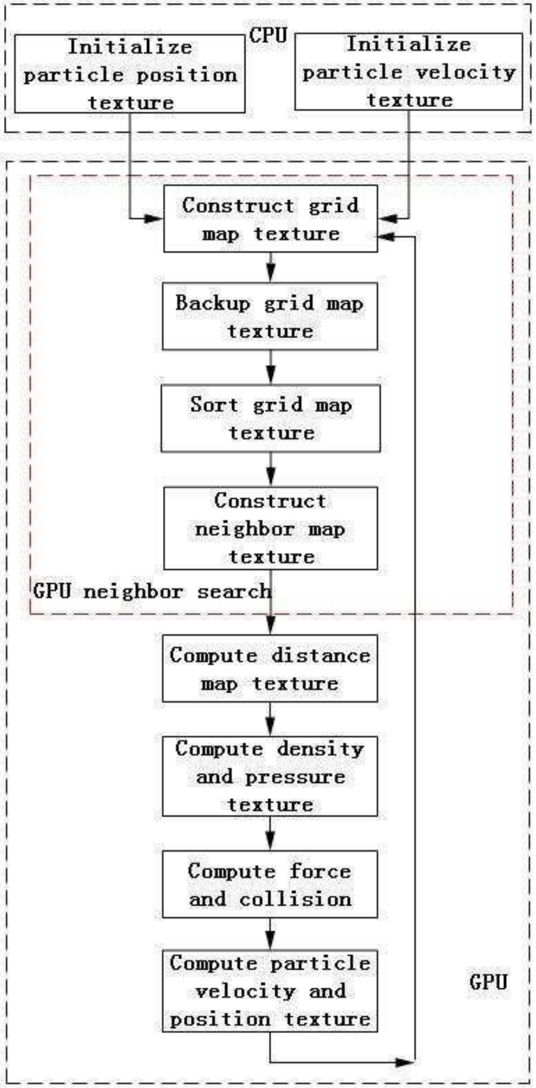

Figure 1

1. Construct a Grid Map

- Compute a one dimensional grid coordinate of each particle
- Discretize particle coordinates with respect to a virtual grid of cell size h and obtain integral positions (ix,iy,iz)
- Convert to 1D coordinates by:
  $gridIdx = ix + iy \times sizeX + iz \times sizeX \times sizeY$

  (sizeX, sizeY are the total number of grids in the horizontal and vertical direction)

1. Construct Grid Map

- The shader then calculates the grid index of each particle
  - The particle index is stored in the red channel
  - The grid index is stored in the green channel

- The output:
  - A texture called SortMap[source]
    - SortMap is a texture array of two members for next GPU sorting
    - Source in initialized as source=0

2. Backup Grid Map

- Backup before sorting
- Step 3 will use the grid map texture source as input and sorts it as ascending with respect to the grid index
- Step 4 uses the grid map texture which is arranged as ascending according to the particle index

3. Sort Grid Map

- Sort the grid map texture to aggregate the particles within the same grid.
- Odd-even merge sort algorithm used
- Sorting the grid map texture with respect to grid coordinates stored in the green channel
- In the sorted grid map texture, the particles within the same grid are arranged in adjacent texels

(A texel, or texture element is the fundamental unit of texture space; textures are arrays of texels)

4. Construct Neighbor Map

- What is known:
  - The particles within in the same grid cell are now adjacent to each other
  - Neighbors of a particle in a given radius include also particles within other advanced 26 grid cells
- What needs to be done:
  - The intention is to create a neighbor map texture whose width is the number of particles neighbors and height is the number of total particles
  - In this arrangement all the neighbors of a particle will be arranged in the same row.
  - Because the neighbor map must be a rectangle, we assume that all particles have the same number of neighbors

4. Construct Neighbor Map

- In the neighbor map texture the
  - texture coordinate y means the particle index
  - the texture coordinate x stores the current particles neighbor index
- Let us assume that one neighbor texture holds 4096 particles. Then the particle index equals the sum of page number multiplied by 4096 and texture coordinate y:

$$partSrcIdx = y + itNum \times 4096$$

4. Construct Neighbor Map

- Using the particle index, we can get particle's affiliated grid cell from grid map source texture of step 2.

1. Transfer an integer array with 27 member which stored intervals of any grid cell's adjacent neighbor cell to GPU
   - Use binary search (GPU) to get all the neighbor grid cells of a particle's affiliated grid cell
2. Use binary search (GPU) on the sorted grid map to get the start position of neighbors grid cell of the particle's affiliated grid cell
3. Use the start position of the neighbor grid cell to get the first neighbor particle index stored in the red channel

4. Construct Neighbor Map

- The part where we overcome the GPU incapability of scatter.
- We assume that the number of particles in all grid cells are the same

4. Divide texture coordinate *x* corresponding to the 27 grid cells

4. Construct Neighbor Map

5. Using *gridStep* to determine the neighbor grid cell of the current particle's affiliated grid cell
   $gridStep = (int)\ x\ /\ (int)\ maxPartNum$

6. Adding the partStep to the start position of neighbor cell, we can determine which particle index in current neighbor cell is selected
   $partStep=(int)x-\leftarrow gridStep*\leftarrow(int)maxPartNum$
   (*maxPartNum* means the max number of particles within the same grid cell)

4. Construct Neighbor Map

7. Calculate the distance between the pairs of the particle and its neighbor particle, those pairs whose distance is less than the smooth radius can be written to texture.

8. Write the pairs of the particle index to red channel and its neighbor particle index to green

Zhao's GPU-Based algorithm

- The before mentioned algorithm does not transfer data between the GPU and CPU except when being initialized as shown in Figure 1

- Therefore the algorithm is called GPU-based and not GPU-accelerated

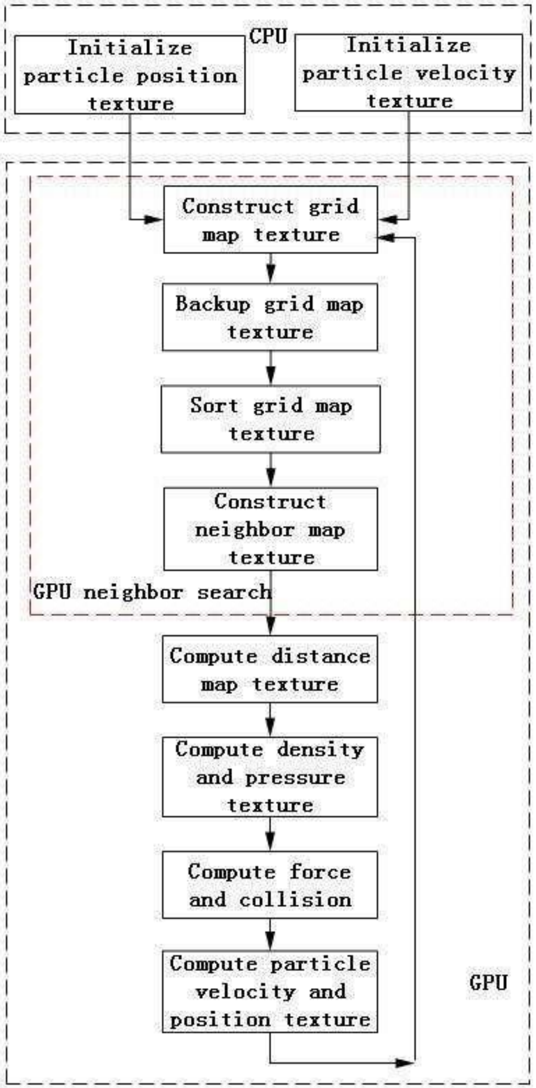

Figure 1

Performance
Waving pool simulation

- Relevant PC specifications:
  - GPU: NVIDIA™ 9600GT
  - CPU: Intel core2 duo CPU E4400

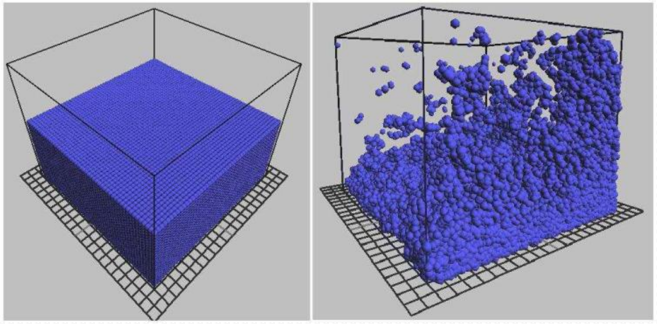

Table 1 The frame-rate of different methods for different number of particles on NVIDIA 9600GT graphics card.

Particle Number	Zhao's Method	Bayraktar's method
20k	25.5	6.1
40k	17.1	4.0
60k	11.6	2.8

Performance + Future work

- The algorithm yields optimum performance if the neighbors of a particle is scattered in large range and particles within a grid cell is fewer.
- Zhao' algorithm can be used for ray-tracing and global illumination
- The method in itself proposes the means to overcome the inability of fragment shaders to do scatter operations

Thank you for your attention

Appendix 1.

- S. Bayraktar, U. Güdükbay, and B. Özgüc.; Gpu-based neighbor-search algorithm for particle simulations. *Journal of graphics, gpu, and game tools*, 14(1):31–42, 2007.
- Onderik J., Ďurikovič R.; Efficient Neighbor Search for Particle-based Fluids; *Journal of the Applied Mathematics, Statistics and Informatics, Faculty of Natural Sciences, UCM Press, Trnava, Slovakia, 2008. To Appear*
- Xiangkun Zhao Fengxia Li Shouyi Zhan; A New GPU-Based Neighbor Search Algorithm for Fluid Simulations; *Beijing Laboratory of Intelligent Information Technology, School of Computer Science and TechnologyBeijing Institute of Technology, Beijing 100081, PRC*

Appendix 2.

- Pixel shader for constructing neighbor map texture

```
void main( float2 OwnPos:TEXCOORD0,
   uniform samplerRECT GridMapSrc : TEXUNIT0,
   uniform samplerRECT SortedGridMap: TEXUNIT1,
   uniform samplerRECT pos_rect : TEXUNIT2,    // prev position
   uniform float itNum,
   uniform float  texWidth,
   uniform float  texHeight,
   uniform float logn,
   uniform float h,       // smoothing length
   uniform float maxPartNum,
   uniform float  stepArray[27],
   out float3 result : COLOR)
{
   float x =floor( OwnPos.x);
   float y =floor( OwnPos.y);
   float partIdxSrc = y + itNum * 4096;
   float2 posTex =convertInd(partIdxSrc, texWidth);
   float4 posSrcTmp= f4texRECT(pos_rect, posTex);
   if(posSrcTmp.w != -1)
   {
     float3   gridDataSrc = f3texRECT(GridMapSrc, posTex);
     float    gridIdx  = gridDataSrc.y;

     int stepIdx = (int)x / (int)maxPartNum;
     gridIdx  = gridIdx + stepArray[stepIdx];

     if(gridIdx  > 0)
     {
        float  start = binarySearch(SortedGridMap, texWidth,texHeight, logn,0.0,  gridIdx);
        float2    texGrid = convertInd(start, texWidth);
        float3    gridDataStart =  f3texRECT(SortedGridMap, texGrid);

        int  step  =(int) x - stepIdx *(int) maxPartNum;

        float2    texGridNew = convertInd(start + step, texWidth);
        float3    gridDataStartNew =  f3texRECT(SortedGridMap, texGridNew);

        float     neighborIdx  = gridDataStartNew.x;
        float     neighbourGridIdx = gridDataStartNew.y;

        if( neighbourGridIdx == gridIdx  && neighborIdx !=  partIdxSrc)
        {
          float4 posNeighborTmp = f4texRECT(pos_rect, convertInd(neighborIdx, texWidth));
          float3 posSrc =posSrcTmp.xyz;
          float3 posNeighbor = posNeighborTmp.xyz;

          float3 dist_vec = posSrc - posNeighbor;

          if(length(dist_vec) < h)// dist < h
              result = float3(partIdxSrc, neighborIdx, -1);
          else
              result = float3(partIdxSrc, -1, -5);
        }
        else//not the adjacent grid
        {
          result = float3(partIdxSrc, -1, -4);
        }
     }
     else//particle not exist
     {
        result = float3(partIdxSrc, -1, -2);
     }
}
```