Simulation and Visualization of Fluids in a Real-Time Application

Dr. Ahmad Khalaf

Iraqi University /Engineering Faculty/ Electrical Division

Article Info
Page Number: 1810 - 1820
Publication Issue:
Vol 72 No. 1 (2023)

Article History
Article Received: *15 October 2022*
Revised: *24 November 2022*
Accepted: *18 December 2022*

Abstract
Fluids can be used to capture some of nature's most visually stunning phenomenasuch as fire, water or smoke realistically. The simulation and visualization of fluids has therefore always been of great interest in the past [1]. Fluids are now considered well-researched in computer graphics area [2]. However, fluid simulations have so far tended to be done in offline rendering, for example for animated films or in research and developmentused for flow simulations. Simulate fluid movement in real timewas long considered too expensive. Thanks to the recent development of graphics hardware,the real-time simulation of fluids is not only possible, fluid simulationscan even be used today in real-time applications such as game engines, among othersintegrate complex calculations. The algorithms were adapted in such a way thatthat they optimally utilize the highly parallel architecture of the GPUs[3, 4].

Physical processes such as the development of fluids are becoming a whole Simulated for a while using computer systems. Such simulations have basically two different goals. On the one hand, they are intended for scientific research to be physical correct, on the other hand they are intended for presentation purposes optically plausible for use in computer games, for example, and to be calculated efficiently. A physically plausible and with current Graphics hardware can also be used in real-time simulation of fluids with the help the Navier-Stokes equations suitable for numerical flow simulation.

1. Introduction

The Navier-Stokes equations were developed in the 19th century by George Gabriel Stokes and Claude Louis Marie Henri Navier independently formulated. They are still considered the basis of fluid mechanics, and describe the law of momentum (conservation of momentum) for Newtonian fluids such asWater, gases or oils (viscous liquids) in differential form [5].

For a complete description of a fluid development there are different approaches, one uses theMass Conservation Equation. For the numerical solution of the Navier-Stokes equations(a "nonlinear partial differential equation system of the 2nd order" [6]).All elements of a volume in a fluid using the Navier-Stokes equations it is too complex to calculate to be practical. Therefore, one chooses for the suitable simulation method for each purpose.

The basic idea of the grid-based approach is to divide the scene intoa smooth 3D grid. The Navier-Stokes equations are applied to the valuesapplied in the voxel centers of the 3D grid. Each voxel center has itsown speed, pressure and density value (number of particles). The density valuescan be rendered directly. The values of the volume elements betweenthe voxel centers are interpolated. The flow of a simulation step seesas follows:apply theadvection (fluid motion) to the velocity values andthe density values and move the particles in the fluid,

Reverse the acceleration (external forces, temperature simulation, vorticity conservation), adjust the pressure in the voxel iteratively (incompressibility) [[7]], Render the density using an appropriate volume rendering method.

The grid-based method is mainly used in numerical flow simulation[[8]], because it is physically plausible and delivers precise results. However, the method is very computationally expensive and also throughthe local limitation of the 3D grid can only be used to a limited extent. The useof the process in computer games is therefore best suited for indoor use.

Later on the Smoothed Particle Hydrodynamics (SPH for short) algorithms were developed. It were originally developed by L.B. Lucy, R. A. Gingold and J. J. Monaghan for the simulation of astrophysical Problems. It can also be applied to fluid simulation according to [[9]]. In Smoothed Particle Hydrodynamicsa Lagrange methodis alloied, i.e. the basic idea of the method existsin not storing the values of the elements in the volume in voxels,but to simply let individual elements swim in the fluid. AVolume element in a fluid is treated as a mass point and one choosesa function that makes small drops out of the mass point [10]]. Aroundto be able to determine the acceleration of a volume element, the positionsand accelerations of the neighbouring elements are used. pressure arisesdue to many relatively close neighbouringelements; friction is caused by the differentVelocities of neighbouring elements. Between those in the calculationincluded elements of the volume one can get the values of the remaining volume elements through simply interpolation [[10]].

The particle-based approach is good for real-time computer graphics simulationssuitable (e.g. computer games) since you can have a variable number of particlescan be determined, which must be included in the calculations of the simulation. A good performance is achieved with a smaller number of particles. According to, the method is also very dispersive, i.e., similar to diffusion, only more strongly, there is a balance between the concentration differencesin the fluid. Nevertheless, one can also use this method in combination with the vortex particles from plausible visual results [[11]]. The Smoothed Particle Hydrodynamics form the basis for fluid simulationsin Nvidia's PhysX engine.

This work provides an insight into the technology behind real-time simulationof fluids on the GPU using Shader Model 3.0 in OpenGL2.1. In addition, an OpenGL renderer was designed as a real-time application,about the possibility of integrating fluid simulation into such an applicationto demonstrate. To run the simulation, at least one DirectX9GPU with shader model 3.0 required[12, 13].

2. Solving the Navier-Stokes equations

Navier-Stokes equations are for some simplified physicsanalytically solvable. An incremental numerical solution allows us to show the development of a flow over time. The one presented in ref [14]algorithm for solving the Navier-Stokes equations consists of several steps in which each equation is solved. The NavierStokes' equations for conservation of momentum contain four in the three-dimensional casescalar equations with four unknowns$\vec{U}(u, v, w)$[15]].

2.1.volume rendering

Volume rendering allows visualization of the structures inside an object. This differs from the usual rendering of polygon surfaceshollow objects with no internal structure. Without considering these internal structures, for example, it is not possible to correctly describe lighting phenomenarepresent, which can only arise through this inner structure. Especially in area of medical image processing in which the actual recording data should not be manipulated for medical analysisfalsification-free representation methods. However, they come through when rendering polygons,the triangulation of the data reveals falsifications. Therefore, this procedure is notsuitable. this problem can be solved with the help of the volume rendering method,because the methods do not change the data sets on the way to presentation.In addition to medical image processing, volume rendering methods are also used forrealistic visualization of volumetric fog, smoke or fire.There are two different types of volume rendering methods: ray casting and volume slicing. Since this work uses a volume slicing method, it is not detailed herethe raycasting method described in reference [[16]].

2.2.Volume slicing

Volume slicing creates a volume within a bounding boxrepresented by Slices. These slices are quad primitives that form a stackordered by the volume in the bounding box. The single one

Layer in the volume, which is present as a 3D texture, for exampleplaced on the quads as a 2D texture. using an alpha blending function, translucent regionsfade out or become transparent.

The position of the camera matters when rendering the stack. The alpha blending also depends on the order of the objects to be rendered,it can happen that the layers are rendered with an incorrect order and are no longer visible. An incorrect render order occurs e.g. at a position of the camera behind the stack when the layers of the stack from behindbe drawn forward.

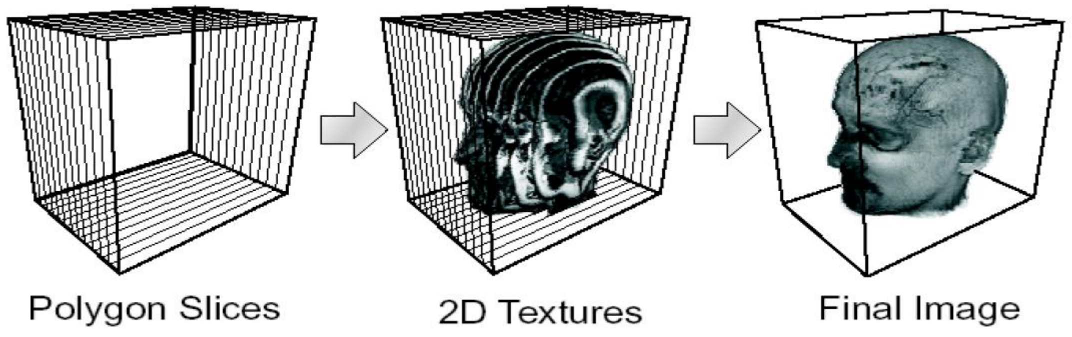

Figure 1: Object-aligned volume slicing

To avoid these unwanted effects in object-aligned volume slicing (see Fig. 1), the Cartesian coordinate system$\vec{x}; \vec{y}; \vec{z}$, a is laid through the volume texture stack (seeFig 3 for illustration). The plane normal of such a texture stack are parallelto the respective axis. The texture stack whose normal is most parallelto the camera's line of sight is rendered. In Fig. 2, for example

the texture stack toggled between steps C and D. Additionallythe order is taken into account for alpha blending to work correctly,in which the individual layers of a stack are rendered. For example, camera behind the stack, the layers are front-to-backdrawn on the back. If it is in front of the stack, it will move backwardsdrawn in front. Object-aligned volume slicing has the advantage of being easy to implement.

to be.

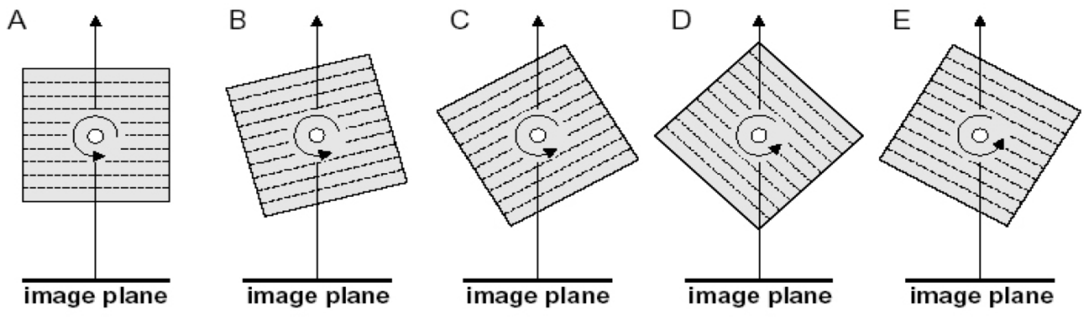

Figure 2: Depending on the camera view, you can switch between the three texture stacks

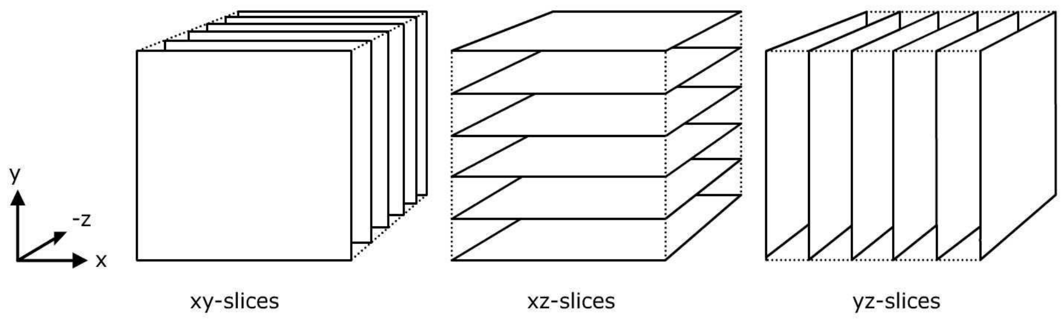

Figure 3: A stack for each axis x,y,z

Object-aligned volume slicing is easy to implement and preventsdisturbing effects such as looking through between the layers or that complete volume disappearance due to incorrect render order. However, looking through the layers is only prevented as long as the opening angle of the camera is set to approx. 45 -50 degrees. However, a larger opening angle for a better overview such as 90 degrees is used,it can again lead to looking through the individual layersof a stack, e.g. when the volume is located at the edge of the field of view. At the moment of switching between the stacks arises at Objectaligned, volume slicing also an annoying visually perceptible "jump"in the display (popping effect). These effects can be compared with the one shown in Fig. 4 illustrated view-aligned volume slicing avoid and the visual qualityincrease

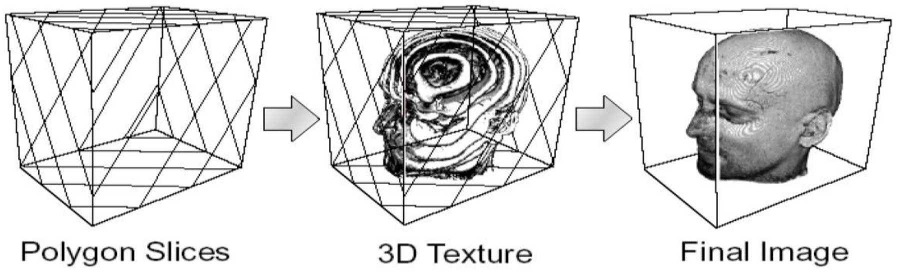

Figure 4. View-aligned Volume Slicing [Will]

However, this method is more complex to implement. As in Fig. 4, slices are placed through the volume in such a way that their normalsare always aligned parallel to the viewing direction of the camera.

3. Software technical concept and implementation

Since the entire system architecture of the renderer is very complex,only the parts of the system that are important for the fluid simulation are described.

using a flow chart process of the initializations shown in Fig. 5 and function calls, the software-technical concept should be explained.

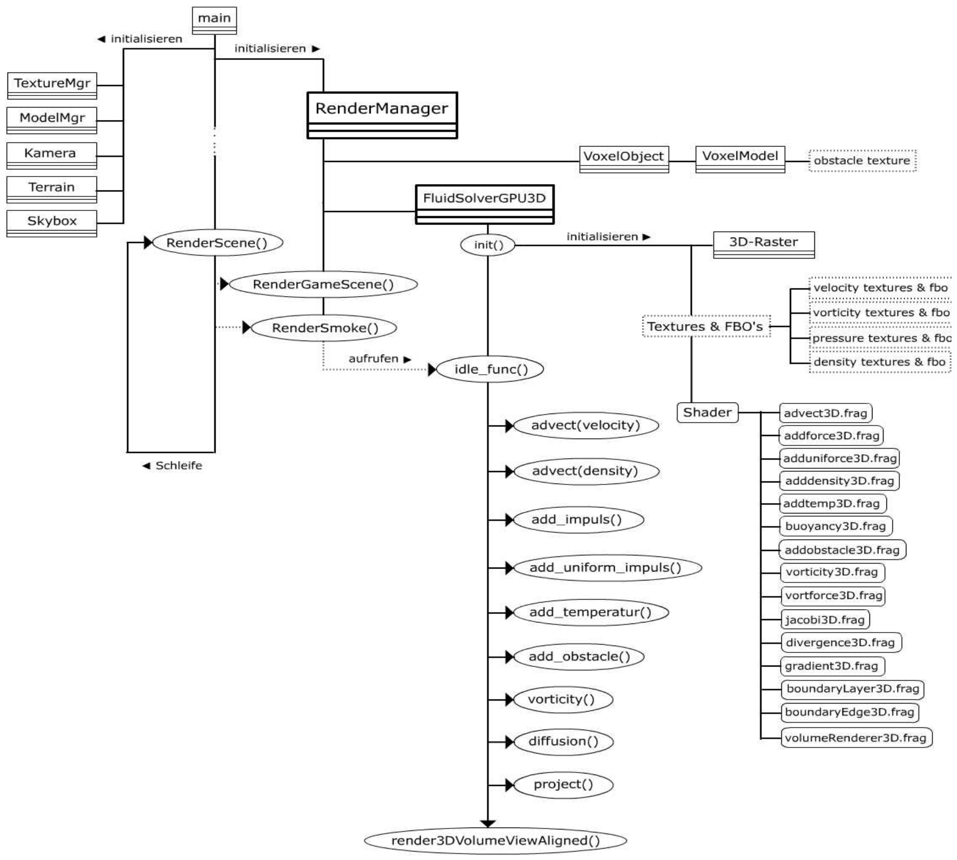

Figure 5: Flow chart of a section of the application

- **Main**

The starting point of all processes is main.cpp. From here the most importantObjects and variables created globally. In the main.cpp all necessaryOpenGL functions for initializing the renderer as well as the functionRenderScene() is called. In RenderScene() it is used in each pass

the scene is recalculated, i.e. from here the functions of the render manager is called.

- **Render Manager**

In the Init() function of the render manager, all textures called once from main()

, models, the skybox, the terrain, some shadersfor the lighting, the octree, the fluid solver and the voxelizer are loaded. the Flat 3D texture for the objects Of theVoxelizer generates located in the fluid volume and from which the smoke particles should bounce off and hands them over to thefluid solver.

In each pass of the application, the RenderScene() function is called in the main.cpp, it becomesthe RenderGameScene() function of the render manager.

In RenderGameScene() the sorted objectsis called, RenderScene() also becomes the functionRenderSmoke() of the render manager. Into RenderSmoke()the models located in the fluid volume are drawn. It Also willcalled the idle_func() function from the FluidSolverGU3D object.

idle_func() then contains all other necessary function calls to calculate Flow development of the fluid per time step and its visualization.

- **FluidSolverGPU3D**

In the Init() function of the fluid solver, after assigning all variables, initializes a grid structure as a GridLayer object is required to setof coordinates from a 3D data set into a 2D data set.The 2D data set is a texture. Textures become similar to arrays for CPU programsused for GPU programs as data storage. The grid structureshelps us to sort and find the 3D data in a flat3D texture. Next all the texturesand associated frame buffer objects are generated. One FBO are assigned, and it can have severaltextures. We need at least two textures herefor each update of a field by a fragment shader, since we can't write to a texture when it's being read. A texture like e.gtmp fileserves as a buffer. It would also have been possible for everyoneUsing an FBO for textures, which would have saved memory. Howevermore copying processes would be necessary, which in turn was at the expense of the overall performance.

After creating the flat 3D textures, the shaders are initialized on the textures:

- advect3D.frag

advection program.

- addforce3D.frag

Adding external forces, e.g. a fan.

- adduniforce3D.frag

Adding uniform external forces like gravity.

- addtemp3D.frag

Adding heat sources to the temperature texture.

- buoyancy3D.frag

Calculate and add the buoyancy force.

- addobstacle3D.frag

Adding obstacles in the fluid volume.

- vorticity3D.frag

Shader for calculating the vorticity.

- vortforce3D.frag

Shader for calculating the vorticity force and adding itforce to the velocity field.

- jacobi3D.frag

Calculates diffusion using the viscosity factor. Can at very smallViscosity factor can be omitted. The program will alsoProjection step used to turn the divergence field into the pressure fieldto calculate.

- divergence3D.frag

Calculates the divergence field from the divergent velocity field.

- gradient3D.frag

Calculated from the pressure field and the gradient and subtracts it from the divergent

speed field. After that, the velocity field is divergence-free.

- boundaryLayer3D.frag

Calculates the values at the boundaries of the fluid volume.

- boundaryEdge3D.frag

Calculates the values at the outer corners of the fluid volume.

- volumeRender3D.frag

Provides visualization of the fluid in a three-dimensional volume.

- idle_func ()

The idle_func() is called in each pass. It contains all sub-stepsthe fluid calculation. First, the viewport is matched to the resolution of the flatAdjusted 3D textures and aligned the camera so that it is orthogonal to theentire texture looks. Now the shaders can read the textures and with helpwrite the FBO's in the textures. This is followed by all the functions that thecontain individual calculation steps of the fluid development.In Fig. 6 the flow of the function calls in the idle_func and in theshader called by the respective function.

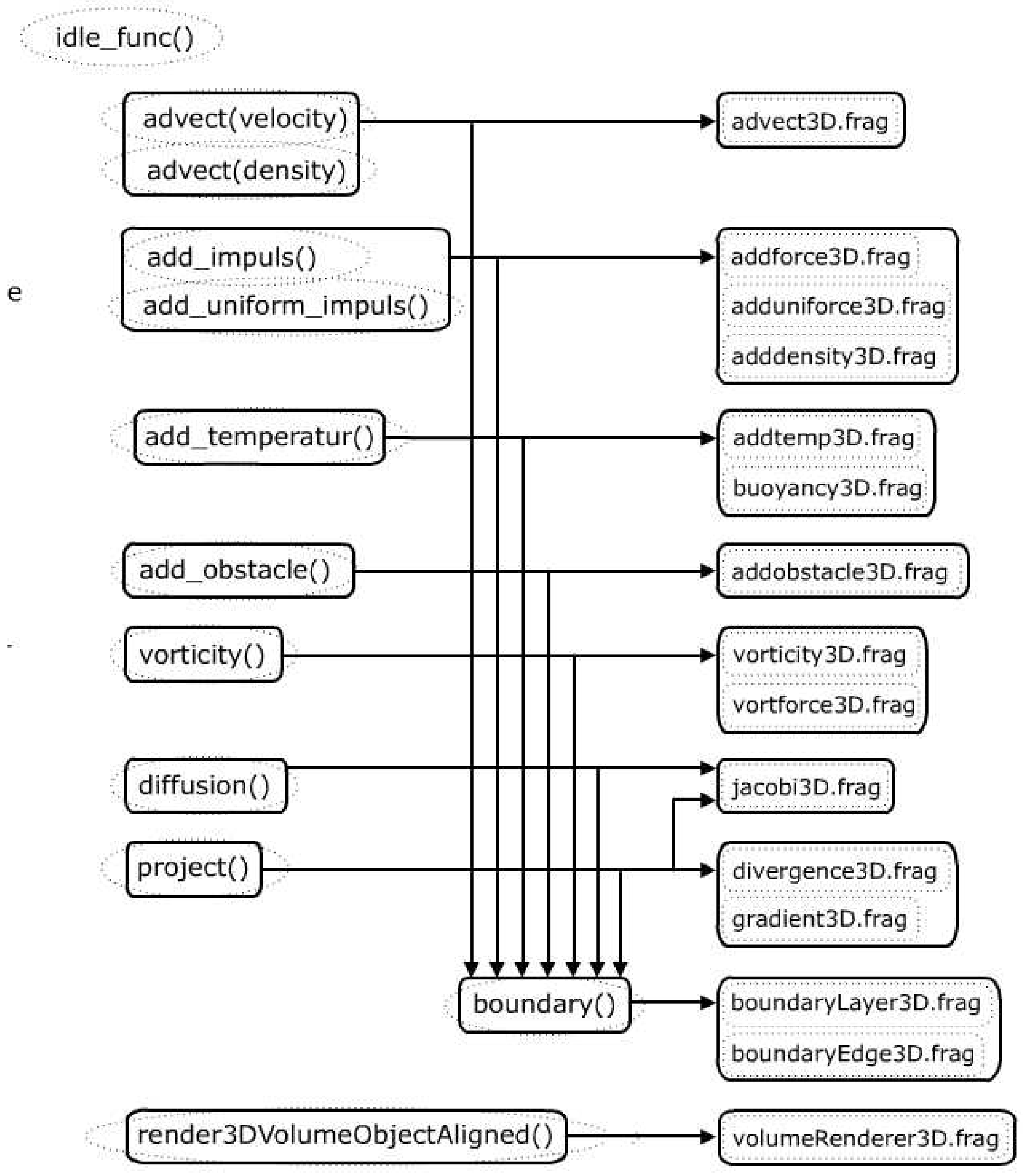

Figure 6: Process idle_func () : functions and shaders

2.3.Implementation

The renderer uses the C++ programming language and the OpenGL graphics library2.1 with some extensions17. For the shaders, the one that can be easily combined with OpenGLand GLSL language supported by ATI and Nvidia.

To calculate the individual development steps of the fluid on the GPU,as can be seen in Fig. 7, textures as data storage and fragment shaders asLoops working on texture pixels areused. That's comparablewith a CPU program in which loops work on array entries. Of the big advantage of the GPU is that the pixels are processed in parallel instead ofin order, as would be the case with a CPU program. It canHowever, only as many pixels can be processed in parallel as there are shader unitsbe available for this.According to [[17]], such a texture update takes place in two steps:

1. Render to texture: A texture is used as a framebuffer. The GPU can thus directly write into thetexture.

2. Copy to texture: The framebufferis copied into a texture (when using an FBOthis step is omitted)

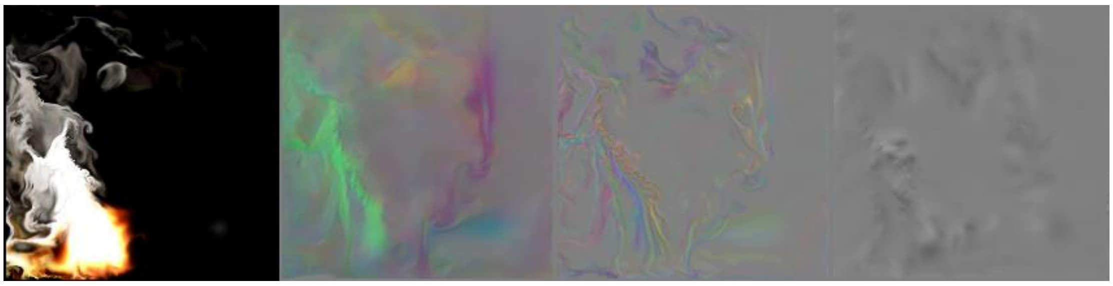

Figure 7. The individual status fields of the fluid saved in textures. Excerptsof textures from left to right: particle field, velocity field(gray is no speed, green is speed in positivey-direction, purple in negative y-direction), vorticity conservation,Pressure field (grey is no pressure, white is high pressure and black is lowerPrint).

The temperature texture is created using the fragment shader whichfirst wrote the heat sources. Then the buoyancy force calculated in the fragment shaderusing the values from temperature texture and density textureand added to the velocity field (see Fig. 8). The ambient temperatureand the scaling factors will be changed at runtime.

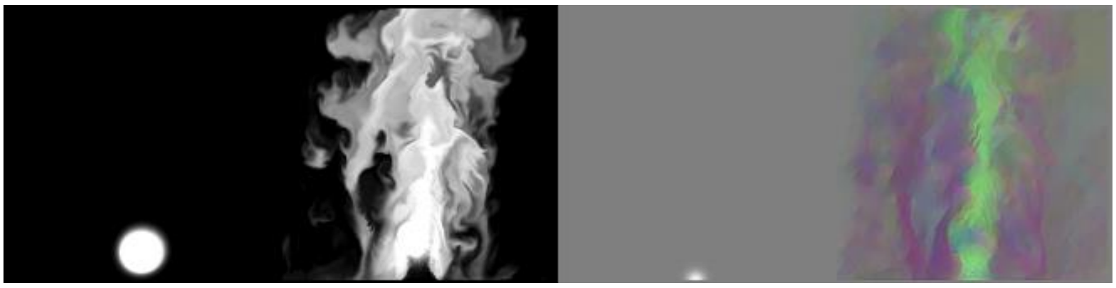

Figure 8: From left to right: particle source, density (particle), heat source, velocity field. The lift can be seen in green in the velocity field. Pixels coloured purple mark a downward movement caused by the gravity acting on the particles.

To calculate the pressure and diffusion, we need to solve the Poisson equation. It can be transformed and presented as discretized Poisson equations. of which can also be written as a linear system of equations of the form$A\overrightarrow{x} = \vec{b}$.

where A is a matrix, *x* is the vector of unknowns, and *b* is the vector with the already known values. We used the simple Jacobian iteration as a technique for solving the systems of equations. an iterative oneStep to solve the two equations for pressure and diffusioncan be as in the form of equation :

$$\vec{x}_{i,j,k}^{n+1} = \frac{\vec{x}_{i-1,j,k}^{n} + \vec{x}_{i+1,j,k}^{n} + \vec{x}_{i,j-1,k}^{n} + \vec{x}_{i,j+1,k}^{n} + \vec{x}_{i,j,k-1}^{n} + \vec{x}_{i,j,k+1}^{n} + \alpha\vec{b}_{i,j,k}}{\beta}$$

In the pressure equation we put fo$\vec{x}$= p.t, for b =$\nabla.\vec{w}$, for $\alpha = (\delta x)^2$andβ= 6 into the diffusion equation. For the velocity field $\vec{w}$ is used for $\vec{x}$ and$\overrightarrow{b}$, and $\alpha\, Becomes\, {}^{(\delta x)^\wedge 2}/_{v\delta t}$ and β=α+6[18].

3. Conclusion

This fluid simulation using GPU offers a variety of expansion options.For example, for improved volume visualization, the raycasting methods are implemented. A bigger challenge would bethe search for a suitable lighting method. With view-aligned volume slicing, at the pointswhere the slices intersect the geometry of the scene, artifacts such as unnaturalsharp edges can be seen. To avoid these artifacts one could use theUse "Soft Particles" method from [[19]].

In addition to simulating smoke and gas effects, fluid simulation can alsocan be used as a basis for the realization of fire or water effects[20]. In the acceleration step one can also add a temperature simulation. After the acceleration and before the projection can be used for theSimulation of Dynamic Clouds a thermodynamic step as in described inref [20].. For deformable objects, an improvedVoxelization method using the Geometry Shaderis realized. The renderer of the application also offers numerous optionsfor the use of fluids in real-time applications.

the scalar pressure values can be vectorizedusing smaller textures. To do this, four scalar values are always combined into oneRGBA vector packed. In This way,one get fewer fragments that the shader duringthe Jacobi iteration has to calculate.

References

[1] Chen F. The simulation and visualization of multi-phase fluid. 2012.
[2] Ihmsen M, Orthmann J, Solenthaler B, Kolb A, Teschner M. SPH Fluids in Computer Graphics - Eurographics State-of-the-art report. 2014.
[3] Niemeyer KE, Sung C-J. Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. The Journal of Supercomputing 2014;67(2):528-64.

[4] Harris MJ. Fast fluid dynamics simulation on the GPU. SIGGRAPH Courses 2005;220(10.1145):1198555-790.
[5] Munson BR, Okiishi TH, Huebsch WW, Rothmayer AP. Fluid mechanics. Wiley Singapore; 2013.
[6] Chorin AJ. The numerical solution of the Navier-Stokes equations for an incompressible fluid. Bulletin of the American Mathematical Society 1967;73(6):928-31.
[7] Nowak Ł, Bąk A, Czajkowski T, Wojciechowski K. Modeling and rendering of volumetric clouds in real-time with Unreal Engine 4. Springer:68-78.
[8] Nilsson F. 3D Cloud Visualization In Real-Time. 2022.
[9] Atencio YP, Ibarra MJ, Cerrón JJO, Quispe RQ, Condori RF, Marín JH, et al. Particle-Based Physics for Interactive Applications. Springer:411-25.
[10] Ge W, Chang Q, Li C, Wang J. Multiscale structures in particle–fluid systems: Characterization, modeling, and simulation. Chemical Engineering Science 2019;198:198-223.
[11] Wen J, Ma H. Real-time smoke simulation based on vorticity preserving lattice Boltzmann method. The Visual Computer 2019;35(9):1279-92.
[12] Ferraz O, Menezes P, Silva V, Falcao G. Benchmarking Vulkan vs OpenGL Rendering on Low-Power Edge GPUs. IEEE:1-8.
[13] Segal M, Akeley K. The OpenGL Graphics System: A Specification (Version 4.6 (Core Profile)-October 22, 2019). The Khronos Group Inc[cit 2020-12-05] Dostupné z: https://www khronos org/registry/0penGL/specs/gl/glspec46 core pdf 2021.
[14] Yang B, Corse W, Lu J, Wolper J, Jiang C-F. Real-time fluid simulation on the surface of a sphere. Proceedings of the ACM on Computer Graphics and Interactive Techniques 2019;2(1):1-17.
[15] Mortezazadeh M, Wang LL, Albettar M, Yang S. CityFFD–City fast fluid dynamics for urban microclimate simulations on graphics processing units. Urban Climate 2022;41:101063.
[16] Rivera Pinto A. Real-Time Simulation and Rendering of 3D Fluids. 2018.
[17] Harris MJ. Fast fluid dynamics simulation on the GPU. ACM SIGGRAPH 2005 Courses 2005.
[18] Umenhoffer T, Szirmay-Kalos L. Interactive Distributed Fluid Simulation on the GPU. 2008;1.
[19] Akenine-Moller T, Haines E, Hoffman N. Real-time rendering. AK Peters/crc Press; 2019.
[20] Harris MJ, Lastra A. Real-Time Cloud Rendering. Computer Graphics Forum 2001;20(3):76-85.