GPU Smoke Simulation on Compressed DCT Space

D. Ishida[1], R. Ando[2], and S. Morishima[1]

[1]Waseda University [2] National Institute of Informatics

Abstract

This paper presents a novel GPU-based algorithm for smoke animation. Our primary contribution is the use of Discrete Cosine Transform (DCT) compressed space for efficient simulation. We show that our method runs an order of magnitude faster than a CPU implementation while retaining visual details with a smaller memory usage. The key component of our method is an on-the-fly compression and expansion of velocity, pressure and density fields. Whenever these physical quantities are requested during a simulation, we perform data expansion and compression only where necessary in a loop. As a consequence, our simulation allows us to simulate a large domain without actually allocating full memory space for it. We show that albeit our method comes with some extra cost for DCT manipulations, such cost can be minimized with the aid of a devised shared memory usage.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ *Physical simulation; Graphics processors;*

1. Introduction

Smoke simulation has been the mainstream for simulation of fluid for a couple of decades, and yet it still remains the long-standing problem due to the hefty memory consumption and a long duration of run-time. It is worth noting that smoke is relatively blurry in terms of flow and density field, unlike sharp interfaces of liquid. This suggests that smoke can be efficiently encoded with the basis functions that are spatially smooth. We exploit this property and propose a new fluid simulation method where physical quantities such as pressure, density and velocity are all represented by the linear combination of DCT basis functions.

Although the choice of DCT seems reasonable, in order to further utilize the nature of DCT, we show that only using $1/8$ basis functions (essentially, clamping high-frequency bands) allows us to preserve visual details while reducing the actual memory storage needed. This provides us with a potential to simulate a large-scale domain that is not possible with a full smoke simulation without DCT due to limited GPU memory resources. In our approach, we follow the traditional splitting approach for solving smoke [Sta99, FSJ01]. The only difference from the previous approaches is the way the loop is handled. More specifically, when looping over cells of a physical quantity, we choose to loop per DCT block. When looping over a DCT block, we temporarily expand the block from the DCT space to the full space and perform a manipulation (e.g., the Laplacian operation) over the cells within the block. Once the manipulation is done, we compress the block back to the DCT space again. This way, we only need to compress/uncompress one block at a time, and thereby we enable small memory footprints for all the steps in the simulation. Overall, our contribution is summarized as follows.

- DCT-based smoke simulation on the GPU that runs an order of magnitude faster than the CPU implementation without DCT.
- On-the-fly compression and expansion of the DCT space.
- Reduced DCT basis functions that retain visual details and consume less memory.
- Our method requires a small incremental effort to implement on top of an existing fluid simulation framework.

1.1. Problem Statement

We find that modern GPUs offer high performance in parallel processing, which is suitable for fluid simulation. On the other hand, GPUs often come with limited resources in memory, which hinders memory intensive simulations. Our method is the first to provide a system that enables a GPU simulation of smoke that runs on a compressed memory space. More specifically, we demonstrate that our approach runs more than a magnitude faster than a CPU implementation without DCT while the memory consumption is greatly reduced by the use of our novel DCT compression.

2. Related Work

The field of smoke simulation has flourished since the seminal work of Stam [Sta99]. Later, the staggered grid was popularized by Fedkiew et al. [FSJ01] for discretizing the velocity field. We refer interested readers to the book by Bridson [Bri08] for a comprehensive overview of fluid simulation. Since the literature of efficient smoke simulation is vast, we only summarize relevant works below.

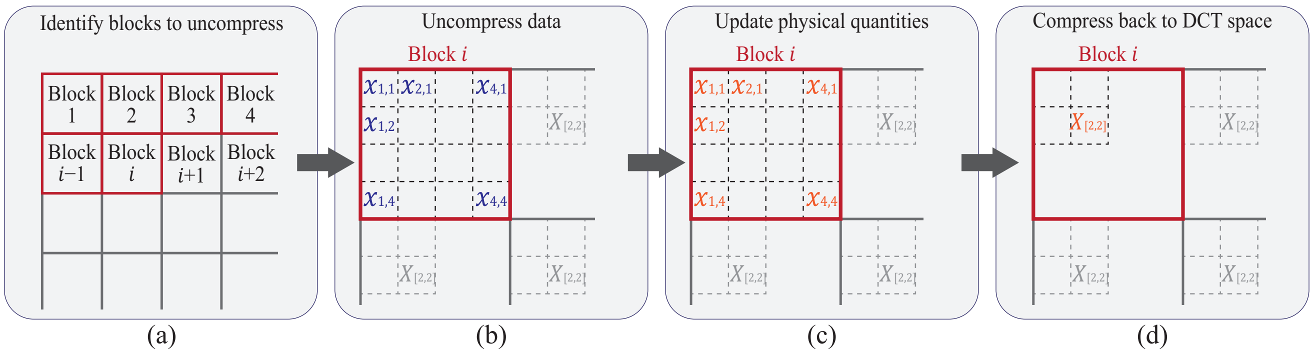
Identify blocks to uncompress
Block 1
Block 2
Block 3
Block 4
Block i−1
Block i
Block i+1
Block i+2
Uncompress data
Block i
$x_{1,1}$ $x_{2,1}$ $x_{4,1}$ $x_{1,2}$ $x_{1,4}$ $x_{4,4}$
$X_{[2,2]}$
Update physical quantities
Compress back to DCT space
(a)
(b)
(c)
(d)

Figure 1: *Our block-wise loop overview. We first identify which DCT blocks to uncompress into the shared memory (a). Next, we uncompress the blocks (b) and manipulate cells within the blocks (c). After the manipulation is done, we compress the blocks back to the DCT space (d). We do this in an "out-of-core" fashion; we never uncompress all the blocks at once.*

Adaptive simulation This group includes methods that make use of octree grid structure [LGF04], Voronoi cells [BBB10, dGWH*15] and tetrahedral meshes [ATW13,BXH10]. These adaptive methods can reduce both the total memory consumption and the computational overhead through the use of adaptive grids where the cell size spatially varies according to a sizing function. While adaptive methods may be tempting, it should be noted that these methods come with an extra cost for mesh generation. It should be also noted that unstructured grids often make discretization complex. Our method does not introduce such complexity.

Reduced space This group includes methods that make use of model reduction [TLP06, WST09], eigen vector analysis [DWLF12] and sub-space simulation [KD13]. Reduced space algorithms start by constructing sub-space basis functions. Reduced space algorithms drastically reduce the degrees of freedom; and thus enable real time simulations. Our method likewise uses reduced space, but a key difference is that our method does not directly compute interactions between the degrees of freedom held by the DCT coefficients. Instead our method simply uses DCT as a data compression mechanism during simulation. Our method is another way of reducing the simulation space which may offer different characteristics compared to the above cited methods. A comprehensive analysis is left for future work.

3. Our Method

3.1. Overview

We use DCT to realize compression of physical quantities:

$$X[k] = \sum_{i=0}^{n-1} x_i \cos\left(\frac{\pi}{2n}(2i+1)k\right), \tag{1}$$

where $X[k]$ is the k-th coefficient in the DCT space and x_i is the i-th data within a block. We extend this idea to three dimensions to realize compression on 3D space. Uncompression of the DCT space is done in a similar fashion. Unless noted, we follow the method of Fedkiew et al. [FSJ01] to proceed our simulation, which consists of advection, pressure projection, and the external force steps along with the staggered discretization. Since the only difference from a regular simulation is how we loop over physical quantities, we focus our discussion of our method around it.

3.2. Preprocessing

To accommodate all the simulation data on a GPU, we first make sure that the total memory size fits within the available GPU memory storage. For this purpose, we partition the simulation domain into rectangular blocks. In principle, we prefer larger block sizes since the maximal achievable compression ratio increases this way. Since the maximal number of assignable threads per "work block" is 1024 on CUDA, we choose $8 \times 8 \times 8$ grids as our block size. Finally, we only carry $1/2$ DCT coefficients for each axis to achieve $1/8$ compression rate in the end.

After we determine the size of DCT blocks, initial simulation data are arranged according to the block-wise index order and they are encoded to the reduced DCT space. Once the arrangements and the encoding are done, the initial velocity and the density field are transferred to the global memory on the GPU.

3.3. Simulation Loop

Once the initial data transfer is completed, we enter a simulation loop that consists of the following four steps: (a) identify the maximal number of blocks that can be uncompressed within the shared memory on the GPU; (b) uncompress the data onto the shared memory for calculation; (c) update the physical quantities on the shared memory; (d) compress the updated data back to the DCT space and store it to the global memory. We illustrate the procedure of our block-wise loop in Figure 1. We emphasize that we do not uncompress all the simulation data simultaneously; we only uncompress a few number of blocks at the same time so that we never need to allocate the full memory space for simulation. In the following, we use the word "activate" to mean the identification of which block to uncompress into the shared memory.

3.3.1. Activating Blocks

The key idea of our method is to perform updates of a physical quantities per DCT block, while performing block-wise data uncompression and compression as needed. Recall that blocks to be activated are decided depending on the shared memory capacity on the GPU. For clarification, we illustrate an example of a $256 \times 256 \times 256$ resolution. First, we partition the domain into $32 \times 32 \times 32$ blocks, each having an 8x8x8 grid. Next, suppose that a GPU can only allocate a shared memory of up to a $128 \times 128 \times 128$ resolution, then the number of blocks to be activated will be $16 \times 16 \times 16$ because it is the maximal number of grids to uncompress within the shared memory. When all the calculations in the active blocks are done, we compress the blocks back to the DCT space and then move on to the next set of DCT blocks.

A careful attention must be paid to cells on block boundaries. For example, when updating the velocity values on faces with the pressure gradient, pressure samples outside the block are needed. Performing the semi-Lagrangian advection also encounters similar situations where a back-traced location falls outside the block. We address this problem in the following way: when a cell outside a uncompressed block is requested, we perform an on-the-fly uncompression for the cell. The overhead of this operation is relatively small because such cases only arise on elements near boundaries.

3.3.2. Pressure Projection

We choose the conjugate gradient (CG) method for the pressure solver. For a linear equation $A\boldsymbol{x} = \boldsymbol{b}$, the CG method only requires the knowledge of how to apply the matrix A on the vector $\boldsymbol{x}$ to complete an iteration. To perform CG iterations on our DCT-compressed pressure field, we perform a matrix-free vector multiplication on a subset of the pressure vector $\boldsymbol{x}$ corresponding to each block obtained by uncompression. A block is uncompressed when a matrix multiplication loop enters the block and compressed again when the loop leaves the block. This way, we can complete a full step of the CG without uncompressing the whole vector $\boldsymbol{x}$.

4. Results

We performed and measured a series of benchmarks, including actual runtime and memory footprints during simulation. We also compared our method with one without using our DCT compression. We implemented our method with CUDA 10.0 and ran on an Intel Core i7 6900K CPU and an NVIDIA GTX 1060.

As can be seen in Table 1, our method achieves more than an order of magnitude speedup than the CPU simulation without our DCT compression. Also, we point out that when compared to a GPU simulation without our DCT compression, the overhead of our DCT compression is acceptably small. We highlight that in terms of the whole simulation time, our method greatly benefits from parallel calculation on the GPU with a significantly reduced memory footprint. Figure 2 compares a smoke simulation without our DCT compression and a simulation using our memory compression at the rates of $1/8$ and $1/16$. It qualitatively shows that our method with $1/8$ ratio well retains the visual details despite compression.

It may appear contradictory that the memory consumption in Table 1 is actually not $1/8$. This is because we chose to uncompress multiple DCT blocks to fill all the shared memory to maximize performance (minimizing the occurrence of the on-the-fly uncompression of elements near DCT block boundaries). Therefore, at the end of a simulation loop, our memory consumption will be $1/8$. If the amount of an available shared memory was smaller, our peak memory consumption would approach $1/8$ at the cost of a slightly poor cache hit rate.

5. Discussion

Convergence on Pressure Solver Our pressure solver does not converge at a desired precision (e.g., relative residual of 10^{-4}) due to the error introduced by our DCT compression. Hence, we set the maximum number of iterations for the CG method to the average of CG iterations needed by a regular simulation without using our DCT compression. In practice, we found this technique does not introduce apparent visual artifacts.

Loss of High Frequency Detail Since our method clamps high frequency DCT bands, small vorticities may be filtered out during compression. Nevertheless, our method may be more efficient than a simulation of low resolutions in the sense that our method is capable of expressing a field of high resolutions using multiple basis functions. We demonstrate an example of a two dimensional comparison in the supplemental video.

Higher Compression Rates We chose a uniform $1/8$ compression rate for obtaining good visual quality. More aggressive compression rates can introduce severe visual artifacts as shown in both Figure 2 and the supplemental video. An interesting avenue for future work would be to make our algorithm adaptive such that unimportant areas get compressed more aggressively.

Mosquito Noise Like a highly compressed JPEG image, when an initial density field contains sharp interfaces, our method can exhibit so-called "mosquito noise" near boundaries. Although such noise does not persist as simulation advances since the density field gets diffused by advection, one may need to carefully distribute the initial density filed to prevent the artifacts.

6. Conclusions

This paper proposed a novel GPU smoke simulation on a compressed DCT space. Building on the standard smoke simulation framework, we devised the order of loops over cells of physical quantities. Particularly, we chose an "out-of-core" block-wise compression/uncompression which allows us to segregate our computation per block. To minimize performance overhead, we chose to uncompress blocks into shared memory. As a consequence, we were able to achieve an order of magnitude faster performance than a CPU implementation without DCT while the memory consumption is reduced down to $1/8$. In our future work, we would like to apply our DCT-based compression approach to develop a multi-grid pressure solver to further speed up the simulation. We also would like to adaptively select the number of DCT coefficients to achieve a more memory efficient simulation.

	Time [sec]	Memory [MB]
CPU Simulation	310.8	3523
GPU Simulation w/o our method	16.7	2051
GPU Simulation w/ our method (1/8 compression)	27.6	933
GPU Simulation w/ our method (1/64 compression)	18.2	931

Table 1: *The benchmark of our method with and without the DCT compression shown in Figure 2. Note that the difference in memory between the CPU simulation and the GPU simulation without the DCT comes from our implementation differences (e.g., difference in a linear solver library).*

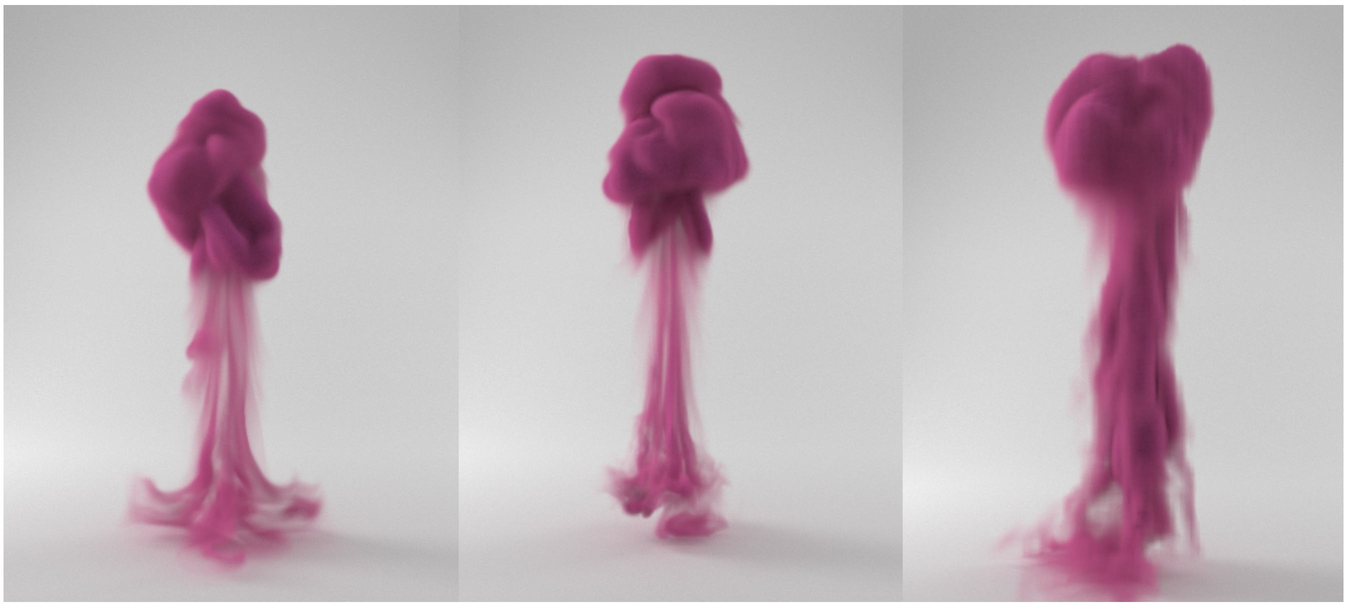

Figure 2: *Smoke simulation without our DCT compression (left). Our method with a* $1/8$ *DCT compression (middle). Our method with a* $1/64$ *DCT compression (right). Resolution:* $256 \times 512 \times 256$.

References

[ATW13] Ando R., Thürey N., Wojtan C.: Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32*, 4 (July 2013), 103:1–103:10. 2

[BBB10] Brochu T., Batty C., Bridson R.: Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph. 29*, 4 (2010), 1–9. 2

[Bri08] Bridson R.: *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press, Sept. 2008. 1

[BXH10] Batty C., Xenos S., Houston B.: Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In *Proceedings of Eurographics* (2010). 2

[dGWH*15] de Goes F., Wallez C., Huang J., Pavlov D., Desbrun M.: Power particles: An incompressible fluid solver based on power diagrams. *ACM Trans. Graph. 34*, 4 (July 2015), 50:1–50:11. 2

[DWLF12] De Witt T., Lessig C., Fiume E.: Fluid simulation using laplacian eigenfunctions. *ACM Trans. Graph. 31*, 1 (Feb. 2012), 10:1–10:11. 2

[FSJ01] Fedkiw R., Stam J., Jensen H. W.: Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, pp. 15–22. 1, 2

[KD13] Kim T., Delaney J.: Subspace fluid re-simulation. *ACM Trans. Graph. 32*, 4 (July 2013), 62:1–62:9. 2

[LGF04] Losasso F., Gibou F., Fedkiw R.: Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers* (2004), SIGGRAPH '04, pp. 457–462. 2

[Sta99] Stam J.: Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), SIGGRAPH '99, pp. 121–128. 1

[TLP06] Treuille A., Lewis A., Popović Z.: Model reduction for real-time fluids. In *ACM SIGGRAPH 2006 Papers* (2006), SIGGRAPH '06, pp. 826–834. 2

[WST09] Wicke M., Stanton M., Treuille A.: Modular bases for fluid dynamics. *ACM Trans. Graph. 28*, 3 (July 2009), 39:1–39:8. 2