Eurographics Symposium on Parallel Graphics and Visualization (2025)
G. Reina, S. Rizzi, C. Gueunet (Editors)

A Transparent and Efficient Extension of IceT for Parallel Compositing on Non-Convex Volume Domain Decompositions

Paul Hempel*[1], Aryaman Gupta*[1], Ivo F. Sbalzarini[1,2,3] and Stefan Gumhold[1]

[1]TUD | Dresden University of Technology, Faculty of Computer Science, Dresden, Germany
[2]Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany
[3]Center for Systems Biology Dresden (CSBD), Dresden, Germany
* These authors contributed equally.

Abstract

The IceT library is widely used for parallel compositing but does not support non-convex volume domain decompositions. We provide a backward-compatible extension of IceT to handle non-convex domain decompositions of volume data. These are frequently produced in numerical simulations, but it is challenging to render them in parallel due to the non-commutativity of alpha compositing. We enable parallel volume rendering of non-convex domains in IceT by extending its parallel compositing to layered images. Our code follows an embedded design, extending and generalizing IceT's internal functions for image compression, splitting, compositing, and decompression to efficiently handle layered images, while maintaining the existing functionality and API. We perform scalability tests and provide our implementation open-source in a public repository, with in-line documentation and integration tests.

CCS Concepts

• ***Computing methodologies*** → ***Distributed algorithms; Rendering;*** • ***Human-centered computing*** → ***Visualization techniques;***

1. Introduction

IceT – the Image Compositing Engine for Tiles – is a high-performance sort-last parallel rendering library [Mor11]. It supports parallel rendering of volumetric and geometric data using a range of sort-last parallel compositing algorithms [Neu93, MPHK93, PGR*09], and it compresses images to optimize performance [LMPM21]. These features have led to IceT becoming the *de facto* standard for parallel compositing on distributed high-performance computers, and it is used in popular visualization tools like ParaView [AGL05] and VisIt [CBW*12]. However, IceT does not support parallel rendering of volumetric data decomposed into non-convex subdomains.

Non-convex domain decompositions of volumetric data are frequently generated in numerical simulations, e.g., by distributed-computing frameworks like OpenFPM [ILZ*19] and Fun3D [NAS]. *In situ* visualization of such simulations requires parallel rendering on non-convex subdomains, which is challenging for volume rendering because of the non-commutativity of the `over` operator used in α-compositing [PD84]. In a non-convex domain decomposition, rays may leave the domain of a processing element (**PE**) and later re-enter it. In Fig. 1a, e.g., rays $\mathbf{R}_2$ and $\mathbf{R}_4$ intersect the domain of **PE** 2 twice and the domain of **PE** 1 in-between. This implies that **PE**s cannot render their data independently to a single image, as is required for sort-last parallel rendering [MCEF94].

This lack of data independence has been addressed by producing multiple layers of fragments per pixel on each processor [GIB*23, SDW*24]. In these approaches, one fragment is generated for each individually convex subdomain intersected by the ray. Each fragment stores a color together with the associated depth. The depth is then used during parallel compositing to place the colors in the correct order, producing the correctly α-composited result. The popular parallel compositing library IceT, however, currently lacks an implementation of this approach [Mor11].

Here, we extend IceT to support parallel compositing on non-convex volume domain decompositions using a layered image approach. We implement the software extension transparently, ensuring backward compatibility and consistency with IceT's existing design. Specifically, we extend IceT's image compression and parallel compositing methods to support layered images and expose this functionality through the familiar API. We provide the extended library in a public repository called `layered-icet`, contributing, to the best of our knowledge, the first dedicated open-source parallel compositing tool that supports distributed volume rendering on non-convex domain decompositions, supporting different compositing strategies and image formats.

DOI: 10.2312/pgv.20251151

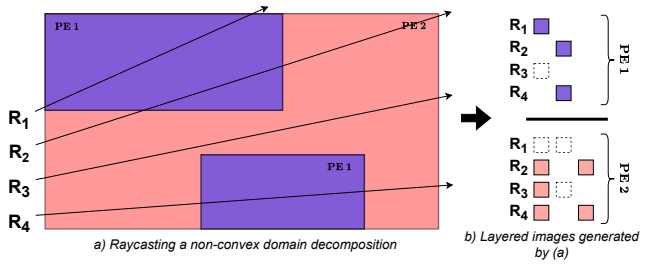


Figure 1: *(a) Raycasting on a non-convex domain decomposition. The subdomains are colored by processing elements (**PE**). (b) The layered image produced by raycasting. **PE** 1 produces an image with a single layer, while **PE** 2 requires two layers to represent each subdomain intersection. The horizontal spacing of fragments within a pixel represents their relative distances to the camera. White fragments with dashed outline represent empty fragments generated to maintain the resolution of the layered image.*

2. Related Work and Background

We review existing techniques for parallel compositing of non-convex volume domain decompositions and provide background on the IceT library.

2.1. Compositing of Non-Convex Domain Decompositions

Parallel volume rendering on non-convex decompositions has generally been enabled by rendering fragments over individually convex subdomains, gathering all fragments for a given pixel onto a single **PE**, and compositing them in visibility order. Ma [Ma95] applied this approach to unstructured data with non-convex boundaries, communicating fragments as they are generated, overlapping communication with the rendering of other fragments for parallel efficiency. To achieve better load balance, Childs et al. [CDM06], and later Binyahib et al. [BPL*19], adopted a hybrid object- and image-order approach, sending fragments for small data elements between **PE**s and transferring larger elements directly. The redistribution was performed such that all fragments along a pixel reside on the same **PE** and can be composited in correct visibility order.

Gupta et al. [GIB*23] avoid replicating or moving data between **PE**s, proposing a method potentially more suitable for *in situ* visualization. During rendering, layers of fragments are generated for each pixel, at least one for each individually convex subdomain. These are combined in visibility order during parallel compositing based on the depth values stored in the fragments. They applied this method to the parallel compositing of Volumetric Depth Images [FSE13], while Sahistan et al. [SDW*24] used a similar technique for regular images, applying it to non-convex distributions of unstructured volumes. Here, we contribute an implementation of this approach for the widely used parallel compositing library IceT.

2.2. The IceT Parallel Compositing Library

The IceT library performs sort-last parallel compositing for both tiled and single displays. For the application-specific rendering of each **PE**'s input image, users can either register a callback with IceT or pass color and depth buffers to the function `icetCompositeImage`.

IceT performs several optimizations for parallel compositing, including active pixel encoding [Mor11], which is a form of run-length encoding to compress background pixels. The image is thus compressed into contiguous regions of active (foreground) and inactive (background) pixels, yielding a so-called *sparse image*. This compression reduces network communication during compositing, which is crucial to IceT's scalability [LMPM21].

For parallel volume rendering with α-compositing, IceT requires defining a strict "visibility order" of **PE**s. If **PE** 1 is placed before **PE** 2 in the visibility order, all parts of the first subdomain must be in front of the second subdomain with respect to the camera. In non-convex domain decompositions, such as the one shown in Fig. 1a, visibility ordering based on PEs is not possible, such that traditional use of IceT produces incorrect results.

Therefore, we extend IceT to parallel compositing of layered images. The order of rendered fragments along a ray can then be determined by per-pixel depths, supporting correct α-compositing also for non-convex domains.

3. Design Criteria

We extend IceT in such a way that all existing functionality, as well as the API, remain unchanged. Existing applications using IceT therefore require no change.

We follow an embedded design approach, extending IceT's internal functions for active pixel encoding and parallel compositing to operate on layered images. This leverages the template-and-macro-based code extensibility and re-usability features present in the library. IceT uses C macros for specialization of code templates to, e.g., support images with different bit depths. We define custom macros and extend existing templates to generalize the code to handle layered images.

4. Layered Images in IceT

We introduce layered images to IceT as a new image format. Pixels contain layers of fragments, each storing color and depth. The number of layers is constant across the pixels and is declared in the image header. To ensure that the header size for existing image types does not change, we store the number of layers as the first entry of the data, effectively creating an extended header.

For compositing, fragments within each pixel are assumed to be in visibility order. If a fragment is empty, all subsequent fragments must also be empty. Fig. 1b shows examples of layered images. Importantly, our implementation can handle images with different numbers of layers on each **PE**, which enables rendering applications to handle imbalanced domain decompositions without a need for synchronization. **PE** 1 in Fig. 1b, for example, generates a layered image with one layer, while **PE** 2 utilizes two layers to represent ray intersections with non-convex domains.

The task of generating the layered image is left to the rendering application using IceT. Possible strategies include using a two-pass raycasting approach to count the number of individually convex subdomains intersected to determine the number of layers.

Figure 2: *An example result of active pixel encoding applied to the layered images produced in Fig. 1b, showing inactive runs (underlined grey numbers) and active runs (dashed rectangular boxes). Active runs contain the number of active fragments in a pixel (regular black numbers) followed by the contents of the fragments, and are preceded by the number of pixels (red italics) and the number of fragments (blue bold) contained in the run.*

5. Parallel Compositing of Layered Images

Layered parallel compositing functionality is exposed via the new `icetCompositeLayeredImage` function. The parameters to this function are pointers to memory buffers containing color and depth data, similar to the existing `icetCompositeImage` function, as well as the number of layers in the image.

For parallel compositing, we extend each phase of IceT's pipeline: active pixel encoding is performed to compress empty pixels and fragments, the compressed image is efficiently split for parallel compositing, communicated via MPI, and then composited. This process repeats if the compositing strategy requires multiple passes. Finally, the image is decompressed to form the final output.

5.1. Active pixel encoding

IceT contains a code template for active pixel encoding (`compress_template_body.h`). For regular images, runs of active pixels are represented by the number of consecutive active pixels, followed by the color and/or depth values of the respective pixels. Runs of inactive pixels are represented by the number of consecutive inactive pixels. Thus encoded images are called *sparse images*. The representation alternates between active and inactive runs, always beginning with an inactive run, which may be of length `0` if the first pixel is active.

We extend active pixel encoding to layered images using the same code template with custom macros. Inactive pixels are found by checking whether the first fragment in a pixel is empty. In addition to compressing inactive pixels, our implementation also compresses inactive fragments within active pixels. For this, a custom macro counts the number of active fragments in a pixel and writes the value to the encoded output. The number of active fragments in an active pixel is stored using one Byte to minimize memory. If more than 255 layers are required, this can be changed by redefining the CMake cache variable `ICET_LAYER_COUNT_T`. Fig. S1 in the Supplement shows the custom macro.

In addition to storing the number of active pixels in a run, we also store the total number of active fragments in each run. While this information is not required for decompression, it is useful for splitting the encoded image. Figure 2 shows an example of a resulting active-pixel-encoded layered image.

5.2. Splitting sparse images for compositing

For parallel compositing, the sparse image is split into divisions containing equal numbers of pixels. Since this splitting happens at each stage of compositing – the binary-swap [MPHK93] and radix-k [PGR*09] compositing strategies require multiple compositing stages – it is performed directly on the sparse images to avoid encoding and decompressing multiple times.

Consider the example of splitting the encoded layered image in Fig. 2 into two divisions of two pixels each. For **PE** 1, since the first inactive run is empty and the first active run contains exactly two pixels, we select both for the first two-pixel division. Here, storing the total number of fragments in the active run becomes handy, as we can compute the offset for the end of the active run without traversing it. For **PE** 2, the first active run contains more pixels than required for the first division, so it must be traversed pixel-by-pixel to perform splitting.

5.3. MPI communication

The split images are communicated among the **PE**s according to the policy of the respective parallel compositing strategy. Since a **PE** cannot know the size of the encoded sparse image it is going to receive, allocating a receive buffer is difficult. For regular images, IceT pre-allocates a buffer that is guaranteed to be larger than any sparse image. This is done on the basis of the window dimensions. For layered images, this is not feasible because the number of layers can vary across **PE**s (see Sec. 4). We instead use an `MPI_Probe` command to determine the size of the incoming buffer and allocate the receive buffer accordingly.

5.4. Compositing

Compositing layered images is similar to the existing process in IceT for compositing regular images. The only difference is that instead of blending pixels, the fragments along the pixel are ordered according to their depths. Images at this stage are still in the *sparse* format, so pixels can contain differing numbers of fragments. Since fragments contain only a single depth value, it is not possible to determine whether they are adjacent. Therefore, no fragments are blended at this stage.

5.5. Decompression

After all stages of parallel compositing are complete, IceT decompresses the composited sparse image. At this stage, the image is still distributed across **PE**s, but each pixel is guaranteed to have all of its fragments on the same **PE**. We can thus at this stage blend active fragments back-to-front by α-compositing. Inactive pixels are assigned the predefined background color. The blending operation across all pixels is implicitly parallel over the **PE**s since the pixels were distributed among the **PE**s after the final compositing step.

IceT then gathers the α-composited pixels onto the root **PE** in the final step.

6. Code Availability

We release our extension open-source in a public GitHub repository: `https://github.com/plhempel/layered-icet`, under the BSD-3 license. In addition, we provide a line-by-line `diff` relative to the original IceT repository in the supplementary material, both as a `git patch` and in an HTML file.

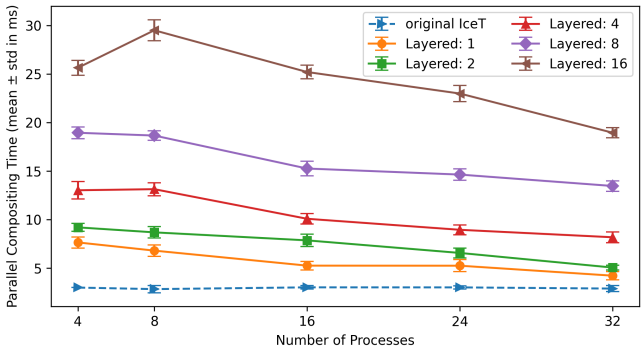
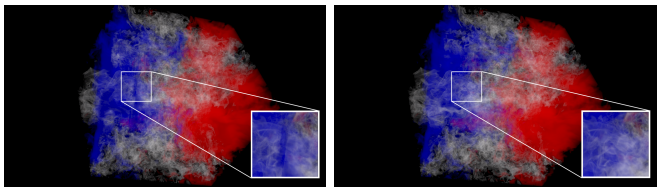


Figure 3: *Parallel compositing times (mean ± standard deviation over a 360° orbit) over non-convex decompositions of the Rayleigh-Taylor dataset with 1 non-adjacent block per **PE** (Layered: 1) up until 16 non-adjacent blocks per **PE** (Layered: 16), compared with the original, non-layered IceT as a baseline (dashed).*



(a) *Original IceT* **(b)** *Layered IceT*

Figure 4: *Visual comparison of compositing output by the original IceT (a) and Layered IceT (b) on a non-convex decomposition of the Rayleigh-Taylor dataset [CCM04] over 4 **PE**s. Inset highlights artifacts in original IceT.*

7. Results and Benchmarks

We test the performance of our implementation on the `Barnard` high-performance computer of TU Dresden. A block distribution is used for MPI ranks across compute nodes, with a maximum of 8 ranks per node. Each node consists of 2 Intel Xeon Platinum 8470 CPUs with 52 cores and 512 GB of RAM. IceT uses CPUs for parallel compositing.

We conduct microbenchmarks to assess scalability across **PE** counts and number of layers. We use the Rayleigh-Taylor [CCM04] and the Richtmyer-Meshkov [CDD*02] datasets, both of which are regular grids, and decompose them into cuboidal blocks. Each **PE** gets multiple non-adjacent blocks, resulting in a non-convex domain decomposition. Images of resolution 1920×1080 are composited, and the radix-k [PGR*09] strategy in IceT is used.

Fig. 3 shows the results for the Rayleigh-Taylor dataset. We observe a general reduction in compositing time with increasing numbers of **PE**s. This is because each **PE** composites increasingly sparse images, which reduces the image splitting time (Sec. 5.2). When using 24 **PE**s, for example, the time for compressing and splitting images reduces by 24% relative to 4 **PE**s for the 8-layered images. As a baseline, the performance of the original IceT on the same dataset is reported (dashed line). The '`Layered: 1`' configuration performs the same compositing operation as the baseline but incurs an overhead ranging between 1.5× and 2.5×. This is because sparse layered images additionally contain depth data, and store the number of fragments in active runs (Sec. 5.1), both of which increase image compression, splitting, and merging times. This is also responsible for the relatively smaller overhead as the number of layers increases: compositing 8-layered images, for example, only incurs an overhead between 1.7× and 2.5× over 2-layered images. In addition, our extended image encoding (Sec. 5.1) compresses inactive fragments within active pixels. This results in the total data communicated via MPI for compositing 8-layered images being only 1.5× the data for 2-layered images when using 8 **PE**s. Complete profiling results, along with results on the Richtmyer-Meshkov dataset, are provided in the Supplement.

Fig. 4 shows an example of the artifacts produced by using the original IceT for compositing on non-convex decompositions. Full-resolution images, as well as reference images for the dataset, are provided in the Supplement.

8. Conclusion

We have presented an open-source extension of IceT for parallel compositing of layered images, enabling distributed volume rendering of non-convex domain decompositions. We exposed the functionality through the familiar API of IceT, leveraged IceT's code templates to reuse code and logic, and ensured that all existing functionality and API are unaffected. We therefore call our implementation "transparent".

Embedding our code into the internal functions of IceT also means that our implementation can take advantage of existing features and flexibility in IceT. While the implementation of Sahistan et al. [SDW*24] benefits from GPU-accelerated compositing, it relies on CUDA. IceT performs compositing on the CPU and is platform agnostic. Existing implementations [CDM06, BPL*19, GIB*23, SDW*24] supporting parallel rendering on non-convex data are limited to the direct-send compositing strategy [Neu93]. Our implementation can take full advantage of the binary-swap [MPHK93] and radix-k [PGR*09] compositing strategies available in IceT, which have been shown to scale better [PGR*09]. Our implementation also supports images of different bit resolutions.

We presented microbenchmarks to assess the scalability of our implementation and its dependence on the number of convex subdomains per **PE**. Future work could consider running tests at scale, where MPI communication is likely to become a bottleneck, especially since layered images require accumulation of fragments during compositing instead of blending (Sec. 5.4). "Early blending" of fragments could be considered as an optimization to alleviate the bottleneck, weighing the advantage against the cost of using an additional depth layer.

Overall, we believe our contribution adds useful functionality to IceT, and we see it finding use especially in *in situ* visualization.

9. Acknowledgments

This work was supported by the competence center "ScaDS.AI Dresden/Leipzig" funded by the German Federal Ministry BMBF (SCADS22B) and the Saxon State Ministry SMWK.

References

[AGL05] AHRENS J., GEVECI B., LAW C.: ParaView: An end-user tool for large data visualization. *Visualization Handbook 717* (2005), 50038–1. 1

[BPL*19] BINYAHIB R., PETERKA T., LARSEN M., MA K.-L., CHILDS H.: A scalable hybrid scheme for ray-casting of unstructured volume data. *IEEE Transactions on Visualization and Computer Graphics 25*, 7 (2019), 2349–2361. doi:10.1109/TVCG.2018.2833113. 2, 4

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., PUGMIRE D., BIAGAS K., MILLER M., HARRISON C., WEBER G. H., KRISHNAN H., FOGAL T., SANDERSON A., GARTH C., BETHEL E. W., CAMP D., RÜBEL O., DURANT M., FAVRE J. M., NAVRÁTIL P.: VisIt: An end-user tool for visualizing and analyzing very large data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. 10 2012, pp. 357–372. doi:10.1201/b12985. 1

[CCM04] COOK A. W., CABOT W., MILLER P. L.: The mixing transition in Rayleigh-Taylor instability. *Journal of Fluid Mechanics 511* (2004), 333–362. doi:10.1017/S0022112004009681. 4

[CDD*02] COHEN R. H., DANNEVIK W. P., DIMITS A. M., ELIASON D. E., MIRIN A. A., ZHOU Y., PORTER D. H., WOODWARD P. R.: Three-dimensional simulation of a Richtmyer–Meshkov instability with a two-scale initial perturbation. *Physics of Fluids 14*, 10 (2002), 3692–3709. doi:10.1063/1.1504452. 4

[CDM06] CHILDS H., DUCHAINEAU M., MA K.-L.: A scalable, hybrid scheme for volume rendering massive data sets. In *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization* (Goslar, DEU, 2006), EGPGV '06, Eurographics Association, p. 153–161. doi:10.5555/2386124.2386151. 2, 4

[FSE13] FREY S., SADLO F., ERTL T.: Explorable Volumetric Depth Images from raycasting. In *2013 XXVI Conference on Graphics, Patterns and Images* (2013), pp. 123–130. doi:10.1109/SIBGRAPI.2013.26. 2

[GIB*23] GUPTA A., INCARDONA P., BROCK A., REINA G., FREY S., GUMHOLD S., GÜNTHER U., SBALZARINI I. F.: Parallel Compositing of Volumetric Depth Images for Interactive Visualization of Distributed Volumes at High Frame Rates. In *Eurographics Symposium on Parallel Graphics and Visualization* (2023), Bujack R., Pugmire D., Reina G., (Eds.), The Eurographics Association. doi:10.2312/pgv.20231082. 1, 2, 4

[ILZ*19] INCARDONA P., LEO A., ZALUZHNYI Y., RAMASWAMY R., SBALZARINI I. F.: OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers. *Computer Physics Communications 241* (2019), 155–177. doi:https://doi.org/10.1016/j.cpc.2019.03.007. 1

[LMPM21] LIPINKSI R., MORELAND K., PAPKA M. E., MARRINAN T.: GPU-based image compression for efficient compositing in distributed rendering applications. In *2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV)* (2021), pp. 43–52. doi:10.1109/LDAV53230.2021.00012. 1, 2

[Ma95] MA K.-L.: Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. In *Proceedings of the IEEE Symposium on Parallel Rendering* (New York, NY, USA, 1995), PRS '95, Association for Computing Machinery, p. 23–30. doi:10.1145/218327.218333. 2

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14*, 4 (1994), 23–32. doi:10.1109/38.291528. 1

[Mor11] MORELAND K. D.: IceT users' guide and reference. doi:10.2172/1005031. 1, 2

[MPHK93] MA K.-L., PAINTER J., HANSEN C., KROGH M.: A data distributed, parallel algorithm for ray-traced volume rendering. In *Proceedings of 1993 IEEE Parallel Rendering Symposium* (1993), pp. 15–22. doi:10.1109/PRS.1993.586080. 1, 3, 4

[NAS] NASA LANGLEY RESEARCH CENTER: FUN3D Software Suite. https://fun3d.larc.nasa.gov/. Accessed: 2025-02-25. 1

[Neu93] NEUMANN U.: Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *Proceedings of the 1993 Symposium on Parallel Rendering* (New York, NY, USA, 1993), PRS '93, Association for Computing Machinery, p. 97–104. doi:10.1145/166181.166196. 1, 4

[PD84] PORTER T., DUFF T.: Compositing digital images. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 253–259. doi:10.1145/964965.808606. 1

[PGR*09] PETERKA T., GOODELL D., ROSS R., SHEN H.-W., THAKUR R.: A configurable algorithm for parallel image-compositing applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, Association for Computing Machinery. doi:10.1145/1654059.1654064. 1, 3, 4

[SDW*24] SAHISTAN A., DEMIRCI S., WALD I., ZELLMANN S., BARBOSA J., MORRICAL N., GÜDÜKBAY U.: Visualization of large non-trivially partitioned unstructured data with native distribution on high-performance computing systems. *IEEE Transactions on Visualization and Computer Graphics* (2024), 1–14. doi:10.1109/TVCG.2024.3427335. 1, 2, 4