Interactive visualization of flows using LIC and video streaming techniques

Peter Kipfer, Günther Greiner

University of Erlangen, Computer Graphics Group
Am Weichselgarten 9, 91058 Erlangen, Germany
Email: kipfer@informatik.uni-erlangen.de

Abstract

For the visualization of subtle structures in vector fields, the line integral convolution (LIC) method has proven to be of great value. When trying to display 3D LIC volumes of flow fields, the possibility of comprehensive and convenient interactive exploration of the volume becomes a crucial point, because of the amount of information present in the volume. Previous work has shown how to efficiently make use of 3D texture rendering techniques to enable interactivity. Based on this approach, we present a stream-oriented solution to animated 3D LIC volumes, integrating stationary and non-stationary flow simulations. We further show how particle transport is integrated in order to enhance the spatial understanding and produce animations that also show the absolute value and the sign of velocity in the flow field. Our approach is flexible enough to support multiple implementations and makes efficient use of available hardware. It integrates nicely with other approaches to enhance the visual quality.

Keywords: Flow Visualization, CFD Simulation, Animated LIC, Volume Rendering, Stream Oriented Transport

1 Introduction

The line integral convolution (LIC) approach for visualizing flows has seen many improvements since its introduction by Cabral and Leedom [CL93]. Their original algorithm performed the convolution along curved stream line segments. The improvements introduced by Stalling and Hege [SH95] made the algorithm faster, independent of resolution and added more accuracy to it. Since then, an ever increasing number of applications of LIC for the visualization of flows has shown its importance in computational fluid dynamics (CFD) research.

LIC is capable of visualizing very subtle features of the vector field of a flow. However, most CFD data (measured or simulated) is of intrinsic 3D nature. Volume LIC shares the appealing properties of 2D LIC image computation, but a thorough analysis and interpretation of the vector field requires to be able to approach interior structures of the data volume. The task of visualizing the region of interest with LIC therefore becomes quite challenging, because in contrast to geometry-based display of glyphs, the density of information hinders the user to "see through" uninteresting parts. The main task of 3D LIC display is therefore to give the user interactivity to explore the volume.

There are several approaches to this task. One is to generate flat LIC images on surfaces in three-dimensional space. The surface can be defined freely in position and shape, or it can be computed automatically using boundary conditions. The 2D image is then simply mapped [For94] onto the surface. The problem of distortion of length introduced by this mapping was solved by computing the LIC directly onto a triangulation

of the surface [TGE97] or by using solid texturing [MKFI97].

A different approach is taken by Rezk-Salama et al [RSHTE99]. Their system is based on direct volume rendering technology like it is common in medical imaging applications. Because there is hardware support available for display, the user can interactively explore the volume. This approach however is restricted to a static LIC volume.

Another drawback of the original LIC algorithm, not to visualize the absolute value of velocity, has been addressed in many other papers, using color coding [BR98, SJM94, SH95], asymmetric filter kernels [HS98, SH95] and varying line width [IG98, WG97] approaches. In [SK97] the problems arising from non-stationary flows have been investigated, too.

After a short discussion of pervious work, this paper is based on, we continue with a detailed description of our system in section 2. Section 3 discusses some implementation specific, but crucial details and presents some performance numbers, before we conclude in section 4.

1.1 Previous Work

The contribution this paper makes to the task of 3D LIC visualization, was inspired by the work of Rezk-Salama et al [RSHTE99]. They present two approaches to LIC visualization using 3D textures for direct volume rendering.

The first one is to use color-tables to interactively assign color and opacity values, the second approach is clipping of the LIC volume. The clipping geometries[1] can be specified and interactively manipulated by the user. An OpenInventor testbed [SDWE98] for direct volume rendering capable of exploiting hardware accelerated 3D textures is used as a platform for the implementation.

The 3D LIC can be automatically animated with both approaches using color-table animation and clipping against precalculated geometries obtained by time-surfaces respectively. While color-table animation does not produce good results in every case, the clipping approach is very flexible and gives convincing results. However, it suffers from degraded rendering performance, if the geometry gets complex. Although the clipping approach seems to be capable of handling non-stationary flows, the article does not specify it.

[1] Although they are using only clip-planes, other geometries are possible too.

2 Implementation

The main idea of the system presented in this paper is to decouple the LIC computation and the display of the volume, and to enable integrated handling of stationary and non-stationary flows.

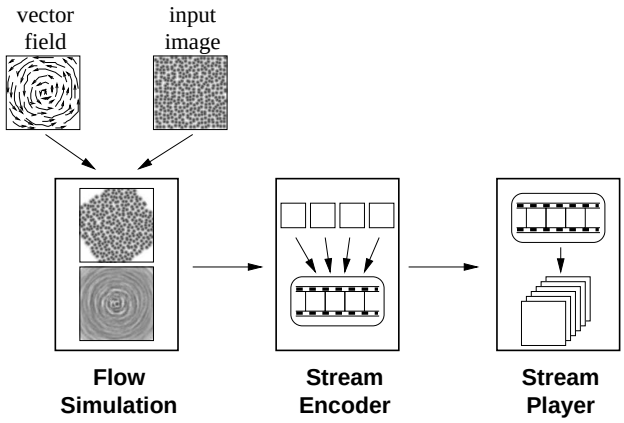

Figure 1: The whole system.

Our approach shares the idea of using volume rendering to display the 3D LIC with the work presented by Rezk-Salama et al. In contrast to them, our system enables the integrated analysis of any kind of flow by not restricting the visualization to a single static volume source. We provide a solution to the problem of keeping the volumes of each timestep in memory by using pixel-oriented video streaming techniques. Additionally, we introduce the smooth integration of 3D LIC and particle transport visualization without additional memory or computational costs.

As depicted in figure 1, our system consists of three major building blocks. The first one

is the actual LIC computation of a flow simulation vector field. This is done by a small C++ program wrapping an optimized Fortran LIC kernel. The kernel supports parallel LIC computation on shared memory machines using a static load balancing scheme based on spatial partitioning of the vector field. Using an input volume and a vector field, it creates two output volumes: One, containing the traditional LIC volume and a second one, that contains the input volume distorted according to the vector field.

The creation of the second volume is folded directly into the LIC computation within the kernel. It requires no extra computation time, because the LIC algorithm is already sampling the motion vectors of the input field. The warped position of a point in the second volume is therefore exactly the motion vector with maximum length at the original position in the input field. Note that if vector field and voxel volume have the same resolution, this reduces to a simple pixel-copy operation.

The user can choose which volume to process further. In the normal case, this will be the computed LIC volume. In addition to the simulated time-step motion, the blurred LIC lines help the spatial understanding (see figure 4 and section 3). On the other hand, using the second volume offers a simple way of doing before—after comparisons.

The biggest benefit of creating two volumes is, that the second volume can be used as input volume for the next time-step, enabling pixel-transport visualization. The vector field can be exchanged after each simulation step. Note that this feature is therefore especially useful when non-stationary flow fields are to be simulated: If the user places some non-zero spots in an input field that is zero elsewhere, the spots will be transported accordingly.

The second part of our system is a configurable stream encoder. It takes the simulated LIC volumes and slices them in each major direction. This produces one video stream for each X, Y and Z. The reason for this is, that it enables the stream player later to easily switch to shear-warp rendering of the volume, if 3D texturing is not available (or not hardware accelerated). The only problem to solve in this case, is to be able to search the target stream for the current frame number, if the shear-warp algorithm needs to switch the slicing direction.

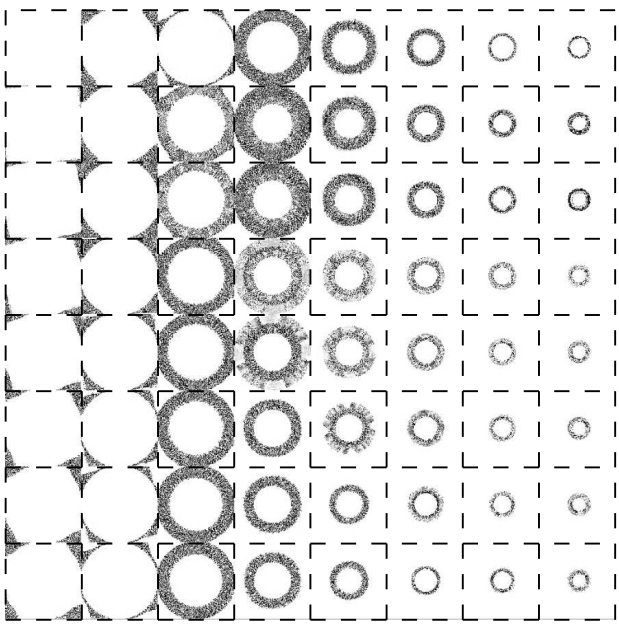

Figure 2: Multiple volume slices are stored in one image to optimize codec usage.

In order to take full advantage of the capabilities of the encoder, it is in general not a good idea to encode every volume slice separately. The stream encoder therefore tries to tile adjacent slices, using a well defined enumeration scheme, into a larger image (see figure 2). The image then gets encoded and is appended to the stream. The encoder takes care to tile the slices in such a way, that there is an equivalent number of images per time-step of the LIC volume in each of the shear-warp streams, so it's easy for the player to predict the position of the next simulated time-step within the stream. If the stream player can use 3D textures, it's enough to display only one of the streams.

The third part of the system is the interactive stream player. It's layout is shown in detail in figure 6. The pipeline design with ring-buffers between the most CPU consuming components enables efficient multi-

threaded asynchronous processing. Additionally, each component can have multiple implementations which take advantage of available hardware support or provide a software solution on other systems.

The main processing steps of the pipeline are (compare figure 6 from bottom to top):

- **Reader:** It is responsible for providing the raw stream data. This can be a simple file reader (see section 3) or a subsystem for network access. It's main task is to ensure the continuous data feed to the pipeline. This includes quality-of-service negotiation or caching in the networked case.
- **Stream decoder:** This component interprets the byte stream from the reader by splitting it into content streams. Currently, there is a stream of image data and a stream of configuration commands for the successor components.
- **Image decoder:** It's first task is to decode the image data. This includes decompression and re-assembly of the image that has perhaps been split into multiple chunks by the streams format definition. The second task is to perform pixel-oriented image manipulation. This is mostly used for conversion between color-spaces.
- **Mapper:** Here, the packing of multiple volume slices into a larger image is reverted. Because this component has knowledge about display properties, this is the place to decide how to extract the slices from the image according to the rendering method (3D textures vs. shear-warp) to be used. This includes pixel-oriented operations on the slice textures to modify opacity, if the stream does not provide opacity information.

While the reader and the stream decoder are tightly coupled for optimized stream-oriented transport, the two decoders and the mapper are connected by a ring-buffer component. This decouples the writing and reading routines of the connected components and allows input processing at a different granularity than the block size intrinsic to the data. For example, the stream decoder can choose to process the stream as soon as it is available from the reader. This can be useful in case of memory-mapped file access. Alternatively it can block until a certain amount of data has been received and process it in one step in order to avoid frequent context switches when running on a single-processor machine. Note that this strategy also helps the image decoder in case of split-field or interlaced streams, or when the player is connected via a network.

3 Results

The system described above has been implemented on a SGI O2 workstation. All measurements have been done on a machine with R10000 195 MHz processor and MVP Vice TRE video board. The timings and numbers throughout the remainder of this paper refer to a 128^3 3D LIC simulation with 100 time-steps.

The user first chooses the vector field and defines an initial input field configuration. In figure 4, the choice was to place a block of 10 slices of white noise in the volume, in order to see how these "particles" get distributed over time. Consequently, the LIC kernel was configured to use the second output volume (the distorted input volume) as input volume for the following time-step. The first output volume (the LIC volume) is sent to the stream encoder, giving some kind of "LIC transport" visualization. Then, the LIC computation of the flow simulation is started on the local machine. Simulating 100 time-steps takes about 22 minutes at full resolution. Note that this is in sharp contrast to the work of Rezk-Salama et al [RSHTE99]. However from the point of view of a CFD engineer, interactivity in visualization is the key point. Our system can provide that for visualization methods, for which is was not available previously without loss of accuracy.

The calculated LIC volumes get encoded into Motion-JPEG streams which are written to disk. The encoding of the images is done

using the hardware codec of the video board. This produces three shear-warp streams with a size of ≈ 18.5 MByte each, which equals a good compression ratio of $1 : 10.8$.

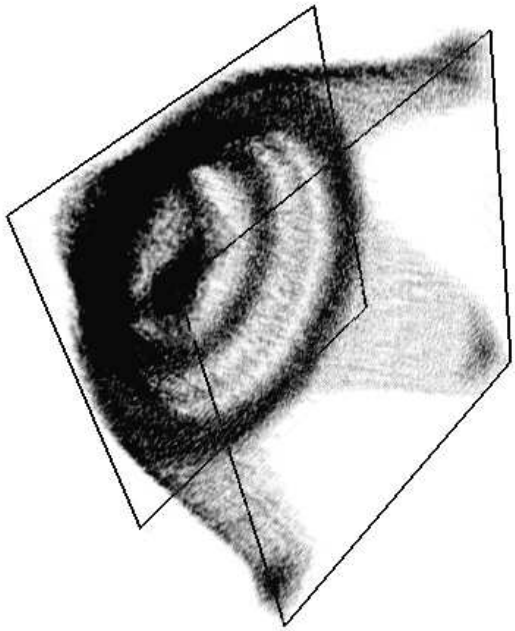

Figure 3: Rendering 3D LIC volumes with more than 1024^2 voxels: The border marks the first slice of each stream frame.

After completion of the stream creation, the user can playback the simulation with the help of the interactive stream player. It has a very efficient implementation that makes in every stage of the processing pipeline (figure 6) use of hardware acceleration: First, the reader memory-maps the first stream, so the stream decoder can easily hand over the pixel substream to the first ring-buffer. The image decoder again uses the hardware codec to decompress the MJPEG fields and renders two of them into one image in the second ring-buffer. Using a SGI OpenGL extension, the pixels get at the same time converted from YCrCb to RGB values. Then, the mapper takes one image from the ring-buffer and renders it into texture memory. It can define an OpenGL color map to modify opacity adaptively for each pixel by using the SGI color map extension. Because the O2 hardware does not support 3D textures, the volume is finally rendered on screen using a shear-warp approach.

The color conversion also enables the mapping of additional scalar values onto the LIC volume. Also note that all the possibilities of convenient visual access to interior structures of the LIC volume presented by Rezk-Salama et al can also be used within our system: The image decoder and the mapper are using color-table mapping which can easily be extended with the functionality described in their paper without a framerate penalty. The clipping approach also integrates nicely by introducing a geometry-description-stream to be used for clipping during rendering. This however will degrade the rendering performance dramatically and lead to inacceptable framerates, like Rezk-Salama et al have experienced. The color-table-based pixel manipulation therefore clearly is the more efficient way to optimize the visual quality during rendering. The clipping approach however is a very powerful tool during stream creation. The encoder can use it to remove unwanted parts of the volume prior to tiling the slices into one stream image.

Also please note that for the setup described above, the stream player draws 101 time-steps of the volume (initial configuration + 100 simulated steps). Because of hardware limitations of the MJPEG codec, a image in the stream may not exceed 1024×1024 pixel. If the LIC volume can't be tiled into this area, multi-pass rendering is used. Figure 3 shows a volume with $128^3 = 1024^2 \times 2$ voxels, that has been rendered from two consecutive stream frames. The first slice of each pass is marked by a border. Because each of the frames within the stream is stored in interlaced mode, the player effectively draws 101 time-steps using 202 stream frames composed from 404 fields. This automatically breaks down the block size the codec has to handle to sizes that allow to put the ring-buffers into framebuffer memory. On the test machine, we obtain rates of 9–10 frames per second[2].

[2]Note that this is independent of the complexity of the LIC volume, compared to clipping [RSHTE99]

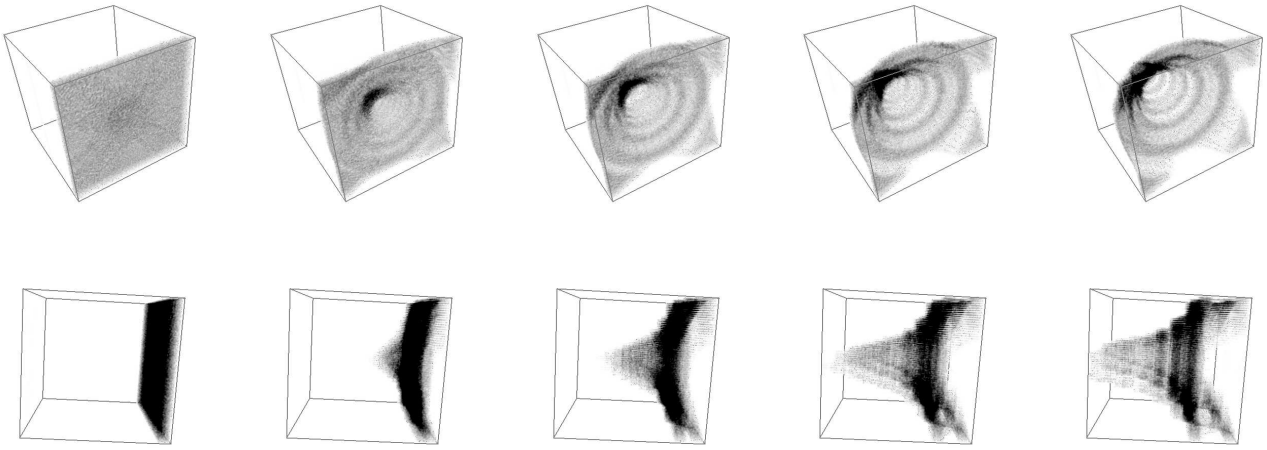

Figure 4: Two views of the same animation from different perspectives at the same time-steps.

The switching of the texture drawing sequence necessary for the shear-warp rendering technique is supported by the stream decoder component (compare figure 6): It has three readers to choose from, each providing a stream for X, Y or Z major rendering direction. Because the seek time for a particular frame in this configuration never exceeds 3 msec when doing jumps at random points in time, the switching of the streams is hardly noticeable to the user. The fast search time is possible, because from the playback, the direction in which to start searching is obvious[3]. Note that this can be accelerated with motion prediction: The stream decoder can be instructed by the mapper to start filling in frames from the target stream to the pipeline in advance to compensate the predicted preroll time.

During the implementation it became evident, that for many configurations, the I/O or memory bandwidth was not the limiting factor. Instead, especially the player is suffering from context switching overhead. Therefore we use OpenGL capable SGI pbuffers for implementing the ring-buffers, which reside in offscreen framebuffer memory. This makes it possible to share them with the rendering context of the screen. However, using the hardware codec generally enforces copying of the data. Codecs supporting overlapping compression and in-place decompression like [LZO98] can be an alternative that will be investigated in future research. Additionally, the O2 hardware seems to have a rather limited pixel fillrate. As figure 5 shows, the framerate dramatically drops when resizing the rendering window (using 2D textured shear-warp rendering).

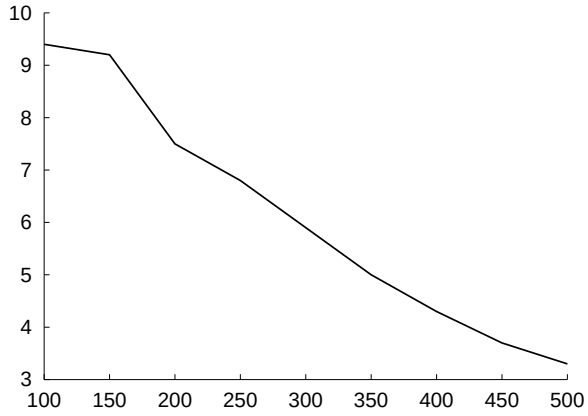

Figure 5: Framerates of the stream player at different window sizes. The X-axis refers to a square $N \times N$ window.

[3]In the case of input from a file, the field indices can be cached upon player start[Ben75].

4 Conclusion

The system presented in this paper shows a new approach to interactive visualization of three dimensional flow simulation both stationary and non-stationary. By generating a video stream, which is configurable to the capabilities of the target machine, the user is also able to save simulation experiments for instant redisplay at a later time. The image-based nature of the video furthermore offers a large variety of possibilities for postprocessing during playback and does not preclude further enhancements of the rendering by other techniques like clipping.

A slight modification of the LIC computing kernel enables particle transport visualization to be integrated smoothly. Especially in combination with 3D LIC computed on each time-step of the simulation, the user gets a convincing impression of the properties of the flow.

While the current solution is particularly useful for standard workstations, future research will investigate into implementing the core routines on a high-end graphics machine. The dedicated texture memory available on these machines offers hardware support for 3D textures. In combination with hardware color-map manipulations, this will reduce the overhead for switching graphics contexts. In conjunction with 3D LIC computation on multi-processor shared memory machines attached to the visualization frontend by streaming-capable middleware, we expect to get a system response time to parameter changes that allows interactive visualization and steering of flow simulations.

4.1 Acknowledgments

We would like to thank Christof Rezk-Salama for providing crucial insight into LIC and direct volume rendering topics.

References

[Ben75] Jon L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BR98] J. Becker and M. Rumpf. Visualization of time-dependent velocity fields by texture transport. In *Visualization in Scientific Computing '98*, pages 91–101, Proceedings of Eurographics Workshop in Blaubeuren, Germany, 1998. Springer Verlag.

[CL93] B. Cabral and L. Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH Conference Proceedings*, pages 263–270, 1993.

[For94] L. Forssell. Visualizing flow over curvilinear grid surfaces using line integral convolution. In *Visualization Conference Proceedings*, pages 240–247. IEEE Computer Society Press, 1994.

[FvDFH90] James A. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics.* Addison-Wesley, Reading, MA, second edition, 1990.

[HS98] H.-C. Hege and D. Stalling. *Fast LIC with Piecewise Polynomial Filter Kernels.* Springer Verlag, 1998.

[IG98] V. Interrante and C. Grosch. Visualizing 3D flow. *IEEE Computer Graphics and Applications*, 18(4):47–53, 1998.

[LZO98] Markus Franz Xaver Johannes Oberhumer. *LZO – a real-time data compression library*, 1998. `http://wildsau.idv.uni-linz.ac.at/mfx/lzo.html`.

[MKFI97] X. Mao, M. Kikukawa, N. Fujita, and A. Imamiya. Line integral convolution for 3D surfaces. In *Visualization in Scientific Computing '97*, pages 57–69, Proceedings of Eurographics Workshop in Boulogne-sur-Mer, France, 1997. Springer Verlag.

[OGL97] OpenGL Architecture Board. *OpenGL — The Industry's Foundation for High Performance Graphics*, 1997. `http://www.opengl.org`.

[Rog98] David F. Rogers. *Procedural Elements for Computer Graphics.* McGraw–Hill, second edition, 1998.

[RSHTE99] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive animation of volume line integral convolution based on 3D texture mapping. In *Visualization '99 Conference Proceedings*, Vienna, Austria, 1999. IEEE Computer Society Press.

[SDWE98] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An interactive visualization and navigation tool for medical volume data. In *WSCG '98 Conference Proceedings*, Plzen, Czech Republic, 1998.

[SH95] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *SIGGRAPH Conference Proceedings*, pages 249–256, 1995.

[SJM94] H.-W. Shen, C. Johnson, and K. Ma. Visualizing vector fields using line integral convolution and dye advection. In *Symposium on Volume Visualization*, pages 63–70. ACM SIGGRAPH, 1994.

[SK97] H.-W. Shen and D. Kao. UFLIC line integral convolution algorithm for visualizing unsteady flows. In *Visualization Proceedings*, pages 317–322. IEEE Computer Society Press, 1997.

[TGE97] C. Teitzel, R. Grosso, and T. Ertl. Line integral convolution on triangulated surfaces. In *International Conference in Central Europe on Computer Graphics and Visualization*, volume III, pages 572–581, Plzen, Czech Republic, 1997. University of West Bohemia Press.

[WG97] R. Wegenkittl and E. Gröller. Fast oriented line integral convolution for vector field visualization via the internet. In *Visualization Proceedings*, pages 309–316. IEEE Computer Society Press, 1997.

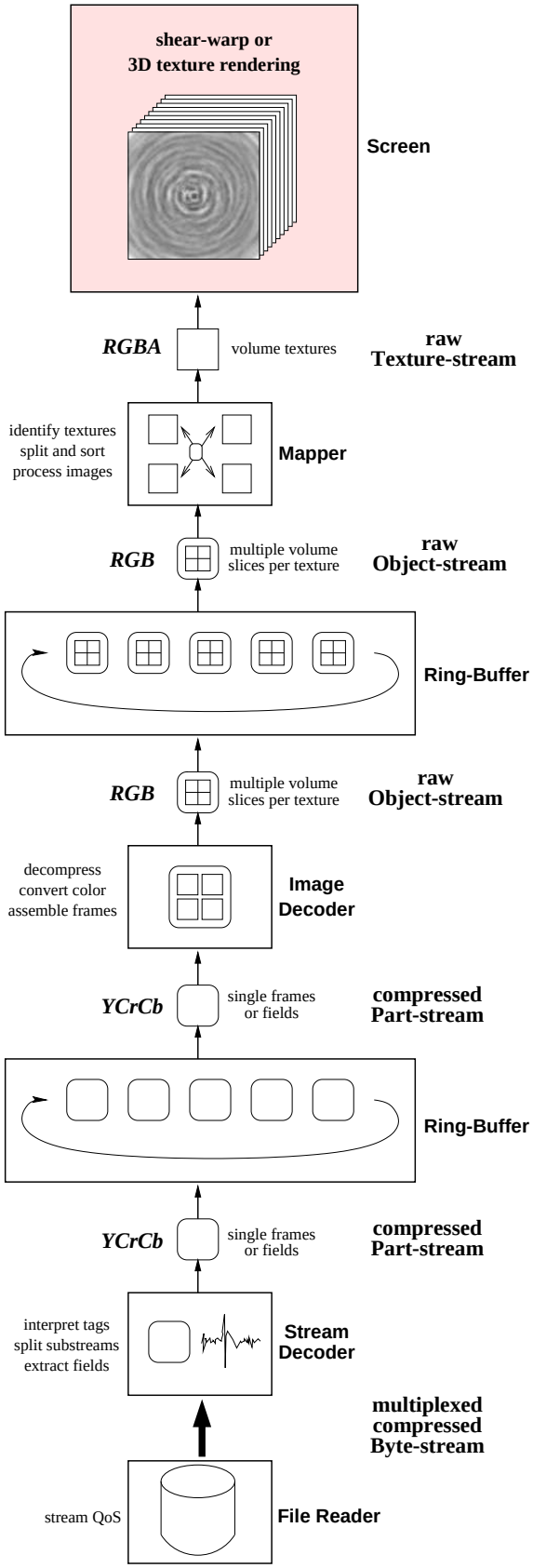

Figure 6: The stream processing pipeline of the interactive stream player (pixel data only).