Progressive Rendering of Transparent Integral Surfaces

Xingze Tian and Tobias Günther

Department of Computer Science, ETH Zürich, Switzerland

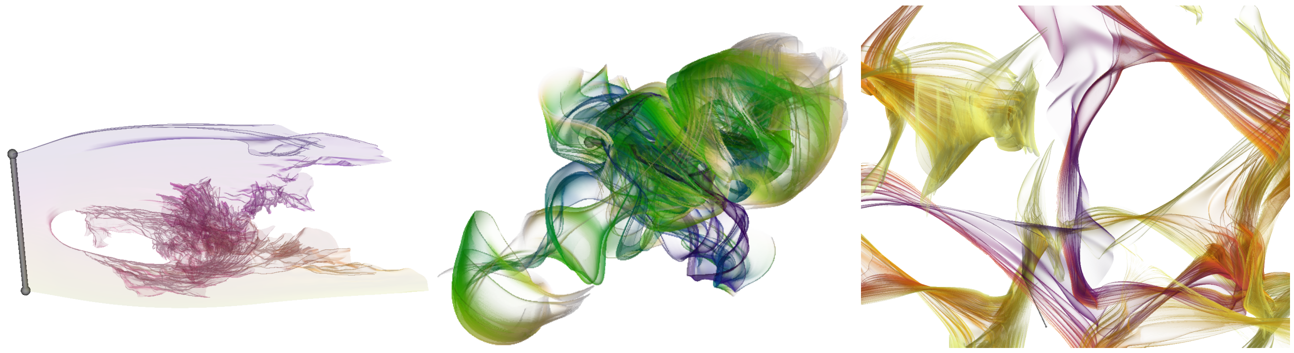

Figure 1: *By composing progressively rendered integral curves into a dynamic per-pixel tree data structure, we can render large and complicated integral surfaces without having to refine their vertex meshes. Here, applied from left to right: the flow past a* WALL-MOUNTED CYLINDER*, the magnetic* BORROMEAN RINGS *vector field and the* ABC FLOW.

Abstract

Integral surfaces are a useful method in illustrative and geometry-based flow visualization, as they convey shading, depth and geometric information better than their line-based counterparts. However, they are not as frequently used as line-based techniques due to the added complexity that arises from their computation. Frontline-based methods, such as stream surfaces and path surfaces require an adaptive subdivision of the frontline, whereas advected surfaces, such as streak surfaces and time surfaces, require refinement and possibly retriangulation of the entire surface after each time step. In this paper, we extend an image-space surface rendering technique to support transparency, which enables the application of illustrative surface rendering techniques without the need to adaptively refine frontlines or entire surfaces. We develop a pixel-based dynamic tree data structure that is progressively filled with integral curves and compactly stores the transparent layers arising in the rendering of the surfaces. We apply the method to the illustrative rendering of path surfaces and streak surfaces in a number of time-dependent vector fields.

1. Introduction

Geometry-based flow visualization methods [MLP*10], such as streamline and pathline rendering, are among the most-commonly used visualization tools, due to their simplicity and expressiveness. Among the geometry-based techniques, surface-based methods [ELC*12] are able to convey shape information better than lines. However, the robust computation of a well-behaved surface mesh is quite challenging, as it requires an adaptive refinement that has a potentially unbound memory consumption at long integration duration, when obstacles are in the flow or when particles get stuck on boundaries. For this reason, most algorithms employ heuristics to terminate or split surfaces when the refinement becomes too extreme [Sta98, SRWS10]. Since these issues already arise with frontline algorithms, Machado et al. [MSE14] introduced an image-based stream surface rendering algorithm that progressively rasterizes many streamlines, which are composed in the backbuffer via depth testing to yield an opaque stream surface. The approach can conceptually be extended to streak surfaces, but it still has a major limitation: the algorithm can only render opaque surfaces, which hinders the applicability of illustrative techniques [BCP*12].

In this paper, we develop a progressive rendering algorithm that computes *transparent* surfaces in image-space by iteratively inserting lines into an image space data structure. Unfortunately, existing order-independent transparency algorithms are not compatible with progressive rendering, since they relate the pixel count to the fragment opacity, resulting in opaque surfaces over time. To compose the pixel fragments of each pixel in the correct depth order, we propose a pixel-wise dynamic tree data structure that is implemented on the GPU. Each frame, we trace a fixed number of integral curves from the seeding curve and construct a fragment linked list for every covered pixel. This way, we stay each frame within a memory budget rather than rendering all lines at once. The fragments are inserted into a dynamic tree data structure that represents the visible surface layers in each pixel in order to properly count how often a line has been rasterized per pixel per surface layer. Once the fragments are inserted into the tree, both the lines and the fragments are no longer needed, since the final image compositing is done with our tree. Our algorithm can render transparent stream, path, and streak surfaces without having to consider adaptive refinement of the front line or surface. Thus, we can render complicated surface geometry in turbulent flow without having to handle particles getting stuck at boundaries, getting trapped behind obstacles, shearing effects, or strong separations. Examples of complicated integral surfaces are shown in Fig. 1.

2. Related Work

Surface-based Flow Visualization. In geometry-based flow visualization [MLP*10], stream surfaces and path surfaces arise when seeding streamlines or pathlines, respectively, from a continuous seeding curve [ELC*12]. The integration of these surfaces requires adaptive refinement of the frontline. A number of recursive [Hul92, Sta98, GTS*04, SRWS10] and GPU-friendly surface construction algorithms were developed [STWE07, GKT*08, SGRT12]. Streak surfaces and time surfaces require refinement of the entire mesh. Point-based and triangle stream-based GPU implementations [BFTW09], geometric edge flip, collapse and split operations [KGJ09], and quad-based CPU implementations [MLZ10] have been proposed, along with the reformulation of streaklines and timelines as tangent curves in a lifted flow [WHT12]. In practice, however, particles can get stuck at boundaries, obstacles or critical points, resulting in an infinite amount of refinement. Further, if the surface passes through a shear flow, the triangles degenerate. Fig. 2 illustrates a few examples, in which the angle between the front line (blue) and the tangent curves (black) vanishes, leading to thin needle triangles. Machado et al. [MSE14] proposed an image-space stream surface rendering method that progressively renders many streamlines into the backbuffer. Since they use the depth buffer to determine the fragment order, their approach is limited to opaque surfaces. In this paper, we extend their method to transparent integral surfaces, which enables the use of illustrative methods [BCP*12] such as silhouette enhancing transparency [HGH*10], emphasis of layer order by halo diffusion [CFM*12], and visibility optimization [GSE*14, BRGG19].

Order-Independent Transparency. The rendering of transparent surface geometry requires an order-dependent compositing either from front-to-back or from back-to-front [MCTB11]. This is either

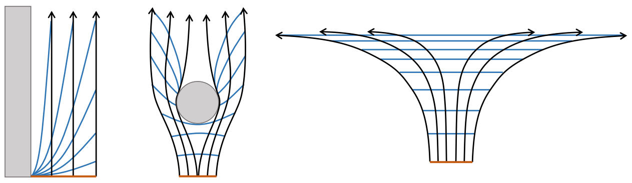

Figure 2: *Parameterizations of integral surfaces quickly deteriorate when particles get stuck at boundaries (left), at obstacles (middle) or when the surface hits a saddle critical point (right).*

done through multi-pass rendering methods, such as depth peeling [Eve01, BM08] or by storing and sorting all fragments [Car84, YHGT10]. The above methods can become slow and therefore approximations have been investigated, storing only k fragments [BCL*07], merging layers [SV14], computing a weighted sum [Mes07], a weighted average [BM08], adding depth dependence and proper background weighting [MB13], using moment representations [MKKP18] or Fourier approximations [BRGG19]. See Kern et al. [KNM*19] for a recent comparison on line data.

In a progressive renderer such as ours, more and more geometry is rasterized over time, ultimately sampling the surfaces densely and non-uniformly with an unbound number of transparent fragments. Unfortunately, all the methods above are incompatible with such a progressive rendering method, since they estimate transparency based on the number of rasterized fragments. Even when adding our merge heuristics to multi-layer methods [SV14], the variable sampling density will cause biases in the bins, when the surface layers in a bin were sampled at different rates. Linked-list approaches [YHGT10] still have linear cost at construction when searching for the bin and require atomic synchronization. Transmittance approximations along view rays [MKKP18, BRGG19] would need an additional representation of the line counter, which needs to be accurate rather than approximate to give the correct color. In this paper, we build a transparency approximation specifically to compute the transparency of *progressively* rendered geometry. Note that this is conceptually different to the rendering of many transparent lines, for which many techniques are available [KNM*19].

3. Progressive Rendering of Integral Surfaces

Geometry-based integral surface computation methods need to update and refine the mesh topology whenever integral curves separate, shear or contract. To avoid these issues, we compose the surfaces in image-space from a progressively rendered set of integral curves. Since previous methods [MSE14] are limited to opaque surfaces, we build an image-space data structure on the GPU that allows us to render transparent surfaces progressively. Our data structure merges fragments into layers, such that the memory consumption is determined by the depth complexity of the scene and not by the total number of integral curves that were rendered.

Progressive Surface Sampling. First, we sample a set of integral curves uniformly from the seeding curve using a Halton sequence [Hal60]. Machado et al. [MSE14] explored two other approaches for the sampling of the surface with streamlines: streamline placement and frontline placement, which would likewise be applicable. Following Machado et al., we compute the normal by

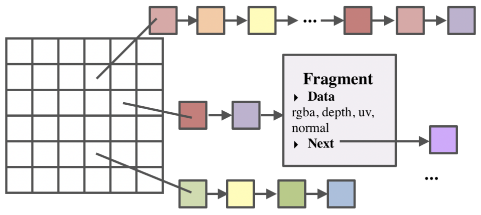

Figure 3: *In the first step, we rasterize all integral curves to form fragment linked lists for each pixel. Each fragment stores color, depth, uv coordinate, a normal and a pointer to the next fragment.*

advecting a ghost particle, which is reset to stay within a threshold to the sampled particle. At the rasterization step, lines are rendered into fragments that store both data (color, depth, uv coordinates and normal) and a pointer to the next fragment. Fig. 3 illustrates the fragment linked lists [YHGT10] for three pixels. Note that the lists do not have to be sorted. The size of the line set is chosen such that the fragment linked list stays within a memory budget.

Per-Pixel Tree Construction. The unsorted fragment-linked lists are then passed to the second step to construct a tree per pixel that maintains the surface layers visible in the pixel. In this step, a full-screen quad is rendered such that each pixel is managed by only one thread. We iterate all fragments in the fragment linked list, and traverse the tree to find their closest nodes for insertion into the tree. Distance is measured as the view space squared depth difference between the fragment to insert and the respective tree node. If we have found a tree node that is close enough to the fragment, i.e., the fragment belongs to a surface layer that has already been rasterized into the pixel, we merge the fragment into the node by averaging its data (color, depth, uv and normal) and update the node with the average. Note that the running average per layer entails a supersampling if multiple lines are rasterized into the same pixel. Unlike the fragment linked lists of the first step, we keep the tree nodes across frames unless the scene or the view is changed. Therefore, the tree data structure is growing over time until every visible surface layer has been represented by at least one fragment. Note that all tree elements are drawn from a single memory pool that is shared by all pixels, which supports a dynamic allocation of deep trees.

Compositing of Transparent Tree Nodes. The per-pixel tree nodes store the average properties of the fragments that have been merged into surface layers. To compose the final transparent image, we apply the front-to-back blending equation [HLSR09]:

$$C = \underbrace{\sum_{i=1}^{n} C_i \alpha_i \prod_{j=1}^{i-1} (1-\alpha_j)}_{C_{src}} + C_{bg} \prod_{i=1}^{n} (1-\alpha_i) \tag{1}$$

where C is the final blended color, C_i and α_i are the color and opacity at fragment i, and C_{bg} is the background color. In this equation, the fragments are sorted from front to back, and C_i's are not premultiplied by the α_i's. Similar to the previous step, this last part of the pipeline is also initiated by rendering a full-screen quad. Since our binary trees are constructed based on the view space depth, a traversal from the left most tree node in the depth-first order intrinsically obtains a sorted list of layers.

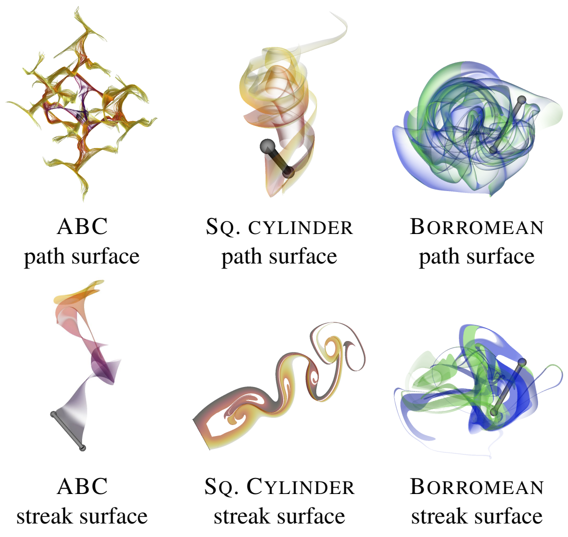

Figure 4: *Visualizations of path surfaces (top row) and streak surfaces (bottom row) rendered with our progressive algorithm in a number of unsteady vector fields. Colors are either mapped by texture coordinates (time and the seeding curve parameterization), or are distinguished between front and back faces.*

4. Implementation

The rendering of a continuous surface potentially requires a large number of integral curves to fill the gaps. Since our rendering algorithm is progressive, users can see in real-time how the surface is formed from the lines, which provides an interactive feedback for the parameter adjustment. For numerical integration, we use a fourth-order Runge-Kutta integration. When rendering streak surfaces, we trace the streaklines with adaptive refinement, where we reseed in time more vertices and trace them to the desired time step. To determine the alpha value, we apply the mapping of Hummel et al. [HGH*10], which adjusts the transparency based on the angle between the normal and the viewer, such that silhouettes are emphasized. Our transparency rendering method is implemented on the GPU with Direct3D 11, and uses default rasterization rules, i.e., diamond tests. Each frame, we need one geometry rendering pass for the first step, and a full-screen pass for the second and third step. In the HLSL code, we use a customized stack to perform tree traversal and compositing. Thereby, the maximum depth of the tree must be specified, which we conservatively set to 200, which is well beyond what was needed in our examples. If the maximum tree depth is exceeded, fragments are blended into the closest tree node, which would then be similar to multi-layer alpha blending [SV14], but with a different merge criterion. We refer to the additional material for schematric illustrations and pseudocode of the algorithm.

5. Results

Examples. We applied our image-based renderer to visualize path and streak surfaces in unsteady vector fields, see Fig. 4. The first example shows integral surfaces in the Arnold-Beltrami-Childress (ABC) flow [GKT16]. This vector field is a worst-case scenario for most surface integrators, since the domain is filled with sepa-

		ABC		Square Cylinder		Borromean Rings		W. Cylinder	
Lines	Op.	Path	Streak	Path	Streak	Path	Streak	Path	Streak
10	I	0.1	0.3	0.1	0.3	0.1	0.3	0.1	0.1
	II	0.6	0.8	0.6	4.2	0.6	3.2	0.5	0.6
	III	55.1	56.4	16.2	119.8	37.7	94.0	4.5	18.5
100	I	0.2	1.3	1.3	1.1	0.2	1.2	0.1	0.2
	II	1.8	2.4	1.6	14.7	2.1	11.6	0.7	2.5
	III	55.0	56.5	16.1	119.7	37.6	94.3	4.6	18.5
1000	I	0.5	12.8	0.6	12.3	10.7	8.9	0.4	0.6
	II	11.3	34.9	12.9	35.8	11.5	88.4	2.5	16.1
	III	55.2	56.5	16.2	119.9	37.8	94.1	4.5	18.7

Table 1: *Computation time in ms of our pipeline steps: linked list creation (I), tree construction (II) and compositing (III). Timings are listed for all scenes for a varying number of integral curves rasterized per iteration, for both path surfaces and streak surfaces.*

rating structures that prompt the frontline to expand and refine in a space-filling manner. Since we do not need refinement, our surface is without artifacts. The SQUARE CYLINDER flow [CSBI05] is a classic test bed for geometry-based visualization techniques, due to its low Reynolds number. Path surfaces are mostly well-behaved, except for the recirculation area [WRT19] behind the obstacle, in which vortices are formed before shedding. In this region, particles can get trapped while others escape, causing a significant amount of refinement. The BORROMEAN RINGS data set [CB11] describes a turbulent magnetic field, in which interlocked rings undergo a topological reconnection. In our visualizations, the front face and back face are mapped to different colors. An example with a continuous color map, encoding the seeding curve parameterization, was shown in Fig. 1 (middle). Our last unsteady vector field contains the turbulent flow around a WALLMOUNTED CYLINDER [FWT08], see Fig. 1 (left). We deliberately place our seeding curve such that the streak surface hits the obstacle. Note that the obstacle geometry is omitted in the visualization, as it would cause occlusion. Due to the permanent trapping of particles in the wake of the obstacles and the high degree of turbulence, this would lead to an unlimited degree of refinement with traditional surface integrators.

Performance. Finally, we report the time taken for each individual pipeline steps for all the data sets used in this paper. For the measurements, we used an Intel(R) Core(TM) i7-4770 CPU with 3.40 GHZ and 32 GB RAM, and an Nvidia GeForce GTX 680 GPU with 2 GB VRAM. All images were rendered at a resolution of 960 × 720 pixels. The runtime per frame of each rendering pass for both converged path surfaces and streak surfaces is listed in Table 1. Suppose M fragments are added to a binary tree of depth N, the time for creating the linked lists ($\mathcal{O}(M)$) and inserting into the tree ($M \cdot \mathcal{O}(\log N)$) scales with the number of fragments. Once the tree converges, the number of nodes is fixed, and hence the composition time is the same for all three cases in the table. This time, however, is the current bottleneck of the system. For the future, it is imaginable to parallelize the computation with a parallel reduction. Fig. 5 shows the memory consumption of our per-pixel tree over time. For each pixel, we pre-allocate an expected number of 16 nodes, which in total sums up to 530MB for our 960 × 720 image. As the number of tree nodes converges, the memory required also converges to a constant, which is the upper bound for each scene configuration and is dependent on the depth complexity of the current view. Note that the node memory pool is shared among pixels. Since the number of tree nodes converges to a constant, the time to perform the blending of the transparent surfaces also converges.

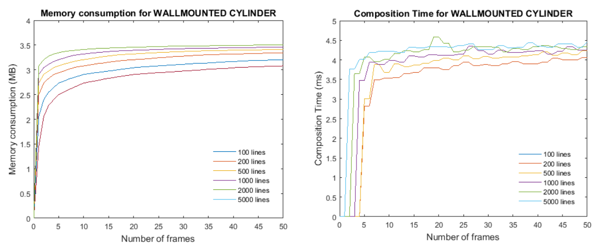

Figure 5: *Convergence of number of tree nodes over time.*

Further Results. We refer to the additional material for further evaluations of our method, including:

- A comparison with existing surface integrators, including no refinement, frontline refinement and the Hultquist [Hul92] algorithm, showing that our method matches the ground truth.
- A comparison with existing approximating OIT algorithms, including weighted sum [Mes07], weighted average [BM08], its extension for a correct background blending [MB13], a binning approach [BCL*07], and the ground truth [YHGT10].
- A parameter study of the depth threshold ε for merging layers.
- Intermediate frames of the progressive computation.

6. Discussion

CPU Tracing. We traced the integral curves on the CPU on a single machine. The parallel and distributed tracing of trajectories in large data sets is an interesting, but orthogonal problem [BPNC19]. Our CPU implementation is a proof of concept mainly suited for the rendering of still images. Interactive rendering of closed surfaces would require GPU tracing and adaptive seeding [MSE14].

View Dependence. Due to the view dependence, the memory consumption can be small even for potentially large scenes. Data structures, however, are rebuilt when the camera moves. The supplemental material contains a video that shows user interaction and an animation of a camera rotation, in which each frame was rendered to convergence. Reprojection methods from real-time rendering could be used to reuse screen-space information from a previous frame.

7. Conclusion

In geometry-based flow visualization, the calculation of integral surfaces usually requires adaptive refinement and retriangulation of front lines or full surfaces, which often needs heuristics to handle boundaries or obstacles. A previous image-based approach avoided these issues by composing the surface with many integral curves. However, the method only supported opaque surface geometry. In this work, we extended this image-based rendering approach to also support the depiction of transparent integral surfaces, which opens the possibility to apply illustrative rendering methods. For this, we developed a per-pixel tree data structure that maintains a representation of surface elements by merging the rasterized fragments of progressively generated integral curves. In the future, we would like to move the computation of integral curves to the GPU, and we would like to apply a parallel gathering approach to the final blending step in order to improve the performance further.

Acknowledgements. This work was supported by the Swiss National Science Foundation Ambizione grant no. PZ00P2_180114.

References

[BCL*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA J. L., SILVA C. T.: Multi-fragment effects on the GPU using the k-buffer. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (2007), ACM, pp. 97–104. 2, 4

[BCP*12] BRAMBILLA A., CARNECKY R., PEIKERT R., VIOLA I., HAUSER H.: Illustrative flow visualization: State of the art, trends and challenges. *Visibility-oriented Visualization Design for Flow Illustration* (2012). 1, 2

[BFTW09] BÜRGER K., FERSTL F., THEISEL H., WESTERMANN R.: Interactive streak surface visualization on the GPU. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov. 2009), 1259–1266. 2

[BM08] BAVOIL L., MYERS K.: Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK* (2008), 1–12. 2, 4

[BPNC19] BINYAHIB R., PUGMIRE D., NORRIS B., CHILDS H.: A lifeline-based approach for work requesting and parallel particle advection. In *Proc. IEEE Large Data Analysis and Visualization* (2019). 4

[BRGG19] BAEZA ROJO I., GROSS M., GÜNTHER T.: Fourier opacity optimization for scalable exploration. *IEEE Transactions on Visualization and Computer Graphics* (2019). 2

[Car84] CARPENTER L.: The A-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 103–108. 2

[CB11] CANDELARESI S., BRANDENBURG A.: Decay of helical and nonhelical magnetic knots. *Phys. Rev. E 84* (2011), 016406. 4

[CFM*12] CARNECKY R., FUCHS R., MEHL S., JANG Y., PEIKERT R.: Smart transparency for illustrative visualization of complex flow surfaces. *IEEE Transactions on Visualization and Computer Graphics 19*, 5 (2012), 838–851. 2

[CSBI05] CAMARRI S., SALVETTI M.-V., BUFFONI M., IOLLO A.: Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate Reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata* (2005). 4

[ELC*12] EDMUNDS M., LARAMEE R. S., CHEN G., MAX N., ZHANG E., WARE C.: Surface-based flow visualization. *Computers & Graphics 36*, 8 (2012), 974 – 990. Graphics Interaction Virtual Environments and Applications 2012. 1, 2

[Eve01] EVERITT C.: Interactive order-independent transparency. *White paper, nVIDIA 2*, 6 (2001), 7. 2

[FWT08] FREDERICH O., WASSEN E., THIELE F.: Prediction of the flow around a short wall-mounted cylinder using LES and DES. *Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM) 3*, 3-4 (2008), 231–247. 4

[GKT*08] GARTH C., KRISHNAN H., TRICOCHE X., TRICOCHE T., JOY K. I.: Generation of accurate integral surfaces in time-dependent vector fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (Nov 2008), 1404–1411. 2

[GKT16] GÜNTHER T., KUHN A., THEISEL H.: MCFTLE: Monte carlo rendering of finite-time lyapunov exponent fields. *Computer Graphics Forum (Proc. EuroVis) 35*, 3 (2016), 381–390. 3

[GSE*14] GÜNTHER T., SCHULZE M., ESTURO J. M., RÖSSL C., THEISEL H.: Opacity optimization for surfaces. *Computer Graphics Forum 33*, 3 (2014), 11–20. 2

[GTS*04] GARTH C., TRICOCHE X., SALZBRUNN T., BOBACH T., SCHEUERMANN G.: Surface techniques for vortex visualization. In *VisSym* (2004), vol. 4, pp. 155–164. 2

[Hal60] HALTON J.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik 2*, 1 (1960), 84–90. 2

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: Iris: Illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1319–1328. 2, 3

[HLSR09] HADWIGER M., LJUNG P., SALAMA C. R., ROPINSKI T.: Advanced illumination techniques for gpu-based volume raycasting. In *ACM SIGGRAPH 2009 Courses* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 2:1–2:166. 3

[Hul92] HULTQUIST J. P. M.: Constructing stream surfaces in steady 3D vector fields. In *Proceedings Visualization '92* (Oct 1992), pp. 171–178. 2, 4

[KGJ09] KRISHNAN H., GARTH C., JOY K.: Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1267–1274. 2

[KNM*19] KERN M., NEUHAUSER C., MAACK T., HAN M., USHER W., WESTERMANN R.: A comparison of rendering techniques for large 3d line sets with transparency. *arXiv preprint arXiv:1912.08485* (2019). 2

[MB13] MCGUIRE M., BAVOIL L.: Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques* (2013). 2, 4

[MCTB11] MAULE M., COMBA J. L., TORCHELSEN R. P., BASTOS R.: A survey of raster-based transparency techniques. *Computers & Graphics 35*, 6 (2011), 1023–1034. 2

[Mes07] MESHKIN H.: Sort-independent alpha blending. *GDC Talk* (2007). 2, 4

[MKKP18] MÜNSTERMANN C., KRUMPEN S., KLEIN R., PETERS C.: Moment-based order-independent transparency. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 1 (2018), 7. 2

[MLP*10] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829. 1, 2

[MLZ10] MCLOUGHLIN T., LARAMEE R. S., ZHANG E.: Constructing streak surfaces for 3D unsteady vector fields. In *Proceedings of the 26th Spring Conference on Computer Graphics* (New York, NY, USA, 2010), SCCG '10, ACM, pp. 17–26. 2

[MSE14] MACHADO G. M., SADLO F., ERTL T.: Image-based stream-surfaces. In *2014 27th SIBGRAPI Conference on Graphics, Patterns and Images* (2014), IEEE, pp. 343–350. 1, 2, 4

[SGRT12] SCHULZE M., GERMER T., RÖSSL C., THEISEL H.: Stream surface parametrization by flow-orthogonal front lines. *Computer Graphics Forum 31*, 5 (2012), 1725–1734. 2

[SRWS10] SCHNEIDER D., REICH W., WIEBEL A., SCHEUERMANN G.: Topology aware stream surfaces. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1153–1161. 1, 2

[Sta98] STALLING D.: *Fast texture-based algorithms for vector field visualization*. PhD thesis, Freie Universität Berlin, 1998. 1, 2

[STWE07] SCHAFHITZEL T., TEJADA E., WEISKOPF D., ERTL T.: Point-based stream surfaces and path surfaces. In *Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), GI '07, ACM, pp. 289–296. 2

[SV14] SALVI M., VAIDYANATHAN K.: Multi-layer alpha blending. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D '14, Association for Computing Machinery, p. 151–158. 2, 3

[WHT12] WEINKAUF T., HEGE H.-C., THEISEL H.: Advected tangent curves: A general scheme for characteristic curves of flow fields. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 825–834. 2

[WRT19] WILDE T., RÖSSL C., THEISEL H.: Recirculation surfaces for flow visualization. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization 2018) 25*, 1 (2019), 946–955. 4

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. *Computer Graphics Forum 29*, 4 (2010), 1297–1304. 2, 3, 4