Efficient and Robust From-Point Visibility

Category: n/a

Paper Type:

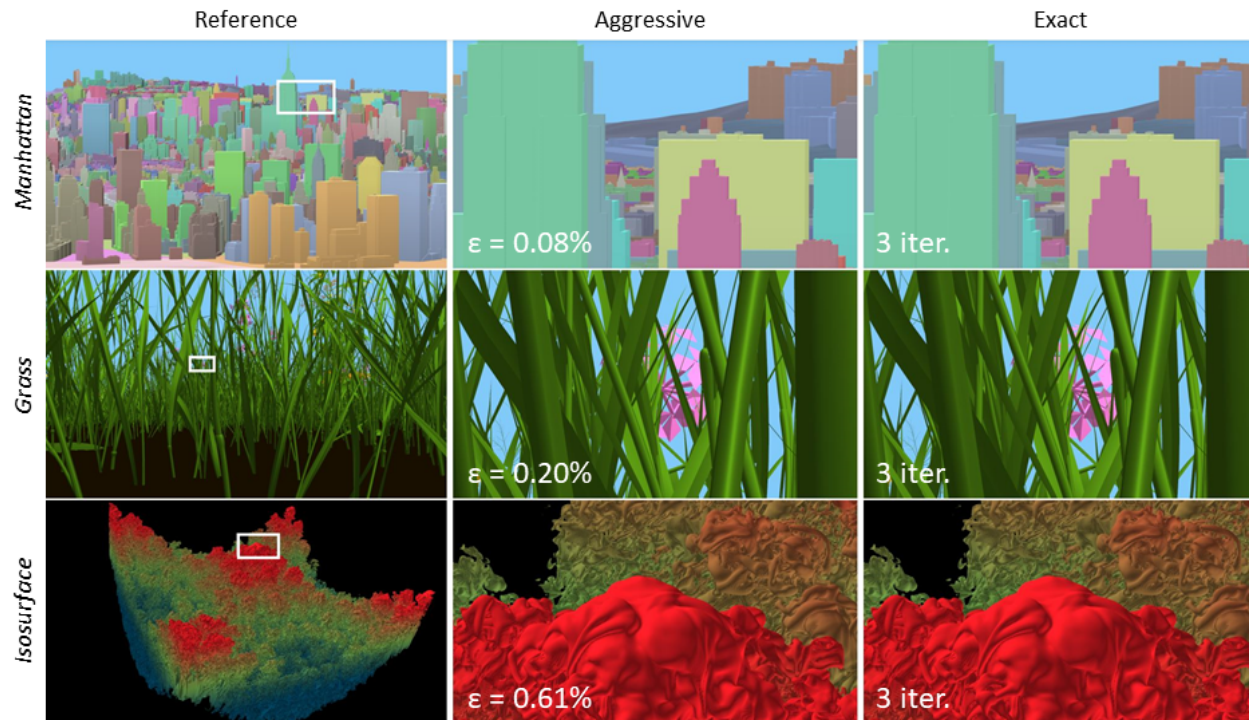

Fig. 1. Results of our from-point visibility algorithms on three datasets with 4, 55, and 500 million triangles (from top to bottom). The left column shows the reference images where our algorithms were run. The middle and the right columns show frames rendered from our aggressive and exact visible sets. Despite the large zoom factors (i.e., 7x, 17x, and 10x), the frames from the aggressive set have only a small percentage ε of incorrect pixels. The frames from the exact set are identical to the ones one would obtain from the entire dataset. Despite the complex occlusion patterns in these examples, our exact algorithm converged in three iterations.

Abstract—This paper presents two from-point visibility algorithms, one aggressive and one exact. The aggressive algorithm computes efficiently a nearly complete visible set, with the *guarantee* of finding *all* triangles of a front surface, no matter how small their image footprint. The exact algorithm starts from the aggressive visible set and finds the remaining visible triangles efficiently and robustly. The algorithms are based on the idea of generalizing the set of sampling locations defined by the pixels of an image. Starting from a conventional image with one sampling location at each pixel's center, the aggressive algorithm adds sampling locations to make sure that a triangle is sampled at all the pixels it touches. Thereby, the aggressive algorithm finds all triangles that are completely visible at a pixel they touch, such as the triangles of a front surface, regardless of geometric level of detail, distance from viewpoint, or view direction. The exact algorithm builds an initial visibility subdivision from the aggressive visible set, which it then uses to find most of the hidden triangles. The triangles whose visibility status is yet to be determined are processed iteratively, with the help of additional sampling locations. Since the initial visible set is almost complete, and since each additional sampling location finds a new visible triangle by construction, the algorithm converges quickly, in at most four iterations for datasets with tens of millions of triangles.

Index Terms—from-point visibility, aggressive visibility, exact visibility, triangle visibility, particle visibility, image generalization.

1 Introduction

Visibility is a fundamental problem in visualization that remains open despite decades of research. Given a 3D dataset and a set of viewpoints, the visibility problem asks which of the dataset's geometric primitives are visible from at least one of the given viewpoints. Visibility algorithms typically work with triangles, which are building blocks of more complex geometric primitives. A triangle t is visible from a viewpoint v if there is a point p in t to which there is line of sight from v, i.e., the line segment vp does not intersect any other triangle in the dataset. As the number of triangles in the visible set is typically a small fraction of the total number of triangles in the dataset, visibility is a powerful tool for reducing dataset complexity.

One approach for solving visibility is to probe for visible triangles along rays that originate from the given set of viewpoints. Such sample-based visibility algorithms are called *aggressive*, in the sense that they find *some* but not *all* visible triangles. Fundamentally, a sample-based visibility algorithm cannot verify that a triangle is hidden, as that would require an infinite number of rays to verify that there is no line of sight to any of the triangle points. Therefore, a sample-based visibility algorithm cannot know whether the set it found is complete, because it cannot verify that the triangles that are not in the set are indeed hidden. A potential advantage of such sample-based algorithms is efficiency, as they *could* find all visible triangles at the cost of one

ray per visible triangle, but deciding which rays to use is challenging. Aggressive visibility algorithms strive to minimize the number of rays while maximizing the number of visible triangles found.

Another approach to visibility is to analyze continuously the space of visualization rays originating from the given viewpoints, subdividing it into regions where a single triangle is visible. Such continuous visibility algorithms can provide the *exact* visible set, containing *all* visible triangles. One challenge is that continuous visibility algorithms are computationally expensive, as the dataset complexity is compounded by the dimensionality of the space of visualization rays. Another challenge is that continuous visibility algorithms are prone to robustness problems, meaning that even tiny rounding errors can cause large errors in the visibility set that they output.

From-point visibility is the problem of finding all triangles that are visible from a *single* viewpoint. The problem is important for computing *accurate, trustworthy* visualizations of complex datasets by taking into account the contributions of *all* triangles visible at each pixel, and not just those selected haphazardly by palliative conventional antialiasing schemes such as 4x4 supersampling. Other applications include bandwidth and latency reduction in remote visualization, avoiding light leaks in hard shadow computation, simulating sound propagation accurately, and serving as a building block for solving higher-order visibility problems, such as from-region visibility.

Although it deals with the simplest possible set of viewpoints–that of a single viewpoint–from-point visibility does not yet have an efficient and robust solution. A promising aggressive approach for from-point visibility is to render the dataset from the given viewpoint into an image that records one visible triangle per pixel. The advantage is efficiency, as visibility is probed along a large number of rays at a small computational cost, in feed forward fashion, by projection followed by rasterization, which has a small amortized per ray cost. The challenge is that, in the case of complex datasets, many visible triangles have a small image footprint and they are not found by any of the pixels of the image. Improving the aggressive visible set by uniformly increasing the resolution of the image is prohibitively expensive. Continuous from-point visibility algorithms have to build a subdivision of the 2D space of rays originating at the viewpoint, which is computationally expensive if they have to process all triangles of a complex dataset.

In this paper we present two from-point visibility algorithms, one aggressive and one exact. Our aggressive algorithm computes efficiently a nearly complete visible set, with the *guarantee* of finding *all* triangles of a front surface, regardless how small their image footprint. Our exact algorithm starts from the aggressive visible set and finds the remaining visible triangles efficiently and robustly. The algorithms are based on the idea of generalizing the set of sampling locations defined by the pixels of an image. Our aggressive algorithm starts from a conventional image with sampling locations at pixel centers, and adds sampling locations to make sure that each triangle is sampled at all the pixels that it touches. If a triangle t is completely visible at a pixel p, t is guaranteed to be found by its sampling location in p. Since the set of triangles that are completely visible at one pixel subsumes the set of completely visible triangles, our aggressive algorithm finds all triangles of a front surface, regardless of geometric level of detail, distance from viewpoint, or view direction.

Our exact algorithm builds an initial visibility subdivision from the aggressive visible set, which it then uses to find most of the hidden triangles. The triangles whose visibility status is yet to be determined are processed iteratively, by generating additional sampling locations where these triangles are not hidden by the current visibility subdivision. The additional sampling locations find new visible triangles which are added to the visibility subdivision. Since the initial visible set is almost complete, and since, by construction, each additional sampling location finds a new visible triangle, the algorithm converges quickly. Only visible triangles are added to the visibility subdivision, which makes our exact algorithm output sensitive.

The robustness requirement is that the visibility subdivision be correct. The subdivision is a polygonal partition of the image plane, with one visible triangle per polygon. Correctness is challenging because the polygonal regions are computed by evaluating millions of predicates, i.e., numerical expressions on whose sign the algorithm execution branches. A single incorrect predicate, due to floating point rounding error, can corrupt the entire subdivision because of global dependencies. We achieve robustness using the Exact Geometric Computation (EGC) [33] strategy of implementing predicates that are correct despite numerical error. EGC evaluates a predicate using floating point arithmetic, it tests if rounding error might cause an error, and if so it reevaluates the predicate using rational arithmetic. The strategy is efficient because the expensive rational arithmetic evaluation is limited to the small fraction of predicates where it is needed.

Figure 1 illustrates our from-point visibility algorithms. The aggressive visible set is nearly complete, as shown by the small errors obtained even in frames with a large zoom factor (middle column). The few incorrect pixels are at surface boundaries, so the frames are comparable to the truth frames obtained from the exact set. Another measure of the completeness of the aggressive set is the percentage of the reference image where visibility is solved correctly, which is 99.98%, 99.93%, and 99.96% for the three datasets, from top to bottom. Our exact visibility algorithm extends the aggressive set to the exact set in a small number of iterations. We also refer the reader to the accompanying video. We will distribute our exact algorithm via the public domain, to be included in applications and to serve as reference for bench-marking heuristic visibility algorithms.

In summary, our paper makes the following contributions:

- The first aggressive visibility algorithm with a quality guarantee.
- The first output sensitive exact from-point visibility algorithm.
- The first robust implementation of exact from-point visibility.

2 Prior Work

Visibility algorithms are classified based on the visible sets that they compute. Conservative algorithms overestimate visibility, so no visible triangle is omitted. The benefit is an accurate image, but the number of hidden triangles in the output can be substantial [8, 11]. Aggressive algorithms underestimate the set of visible triangles, which leads to image errors. The goal of aggressive visibility research is to reduce and control the error [26, 32]. Exact algorithms find only and all visible triangles, which avoids the cost of rendering unnecessary triangles as well as any image error.

Aggressive Visibility. We distinguish between probing visibility by casting individual rays and by rendering entire images. Algorithms in the first category use heuristics to shoot rays that are likely to find visible triangles, and subsequent sampling is guided by what the initial rays find [3, 23, 32]. The advantage is the flexibility to cast precisely the rays deemed necessary, which limits sampling redundancy, and allows supporting effects such as lens distortion, foveation, or depth of field [19]. However, it is difficult to place error bounds on the results, and, as we show in the results section, our aggressive visible set has a quality advantage over guided visibility sampling [3, 23].

Algorithms in the second category leverage the fact that the amortized cost of rays in an image is lower than that of individual rays. Our algorithms fall in this category. An image only captures samples visible from its viewpoint. One option is to use images from additional viewpoints [25], which are highly redundant, or to eliminate redundancy as a pre-process [24, 29]. The challenge of these approaches is to decide which images are needed for a sufficient sampling of the visibility parameter space. The usual strategy is to sample uniformly as densely as possible, and thus the visibility error is not bounded. Multiperspective images capture in a single shot more than what is visible from a single viewpoint through innovation at the camera model level [9, 34], but there is no visible set quality guarantee.

Specialized visibility algorithms have been developed for many contexts. The algorithms are typically aggressive, focusing on finding the visible triangles of highest relevance in the particular context. The semi-analytical visibility algorithm [16], developed for motion blur, samples the image with lines as opposed to points, an idea borrowed from temporal antialiasing [21]. Visibility is analyzed continuously over time for each line sample. The algorithm is aggressive because

the analysis is restricted to a uniform grid of image lines. Line samples are a brute force approach for improving uniform point sampling. The line parameter adds an expensive second dimension to the 1D motion blur visibility problem. The uniform line sample pattern is heuristic, so even after solving the higher-dimension visibility problem, there is still no guarantee for the quality of the solution. We propose deterministic point sampling that guarantees a quality visible set without increasing the visibility problem dimensionality.

Recent work analyzes visibility in the camera offset space defined by viewpoint translations [18]; the visible set is exact at pixel centers under camera translations, which means that visible triangles that are not visible at a pixel center are missed; in order to capture additional visible triangles, such as those visible under camera rotations, additional sampling locations are added heuristically. Our aggressive algorithm guarantees quality by generalizing the set of sampling locations on the image plane, beyond the predefined set of pixel centers.

Exact Visibility. Early work focuses on from-point visibility for antialiasing. The solution was to compute a visibility subdivision for each pixel, defined by the triangle fragments visible at each pixel [5, 6, 31]. The solution is inefficient because fragments of hidden triangles are added and then removed from the visibility subdivision. We compute the visibility subdivision exclusively from visible triangles, which amounts to an efficient way of computing an accurate image that takes into account the contributions of all visible triangles, no matter how small their footprint. Pixel-free from-point visibility algorithms are also inefficient because they compute occluded intersections [15]; typical running times are $O((n+k)\log n)$ or $O(n\log n+k+t)$ for n triangles with k edge intersections and t triangle intersections on the image plane. Output sensitive algorithms are restricted to special input [22, 30]. From-point visibility was implemented on the GPU [2], but with a running time quadratic in the number of triangles.

Beam tracing [17] analyzes from-point visibility continuously by using conical or frustum-like beams. The unsampled gaps between rays are avoided, but beam-triangle intersection is costly. Beam-tracing has also been used for shadow [1,7] and sound [27] rendering, using acceleration schemes based on adaptive beam splitting. Beam tracing has been recently revisited for its ability to integrate visibility, which supports the differentiable rendering used for example in inverse graphics [36]; the complexity of the polygonal regions of the visibility subdivision of the beam is capped at four vertices, but that increases the number of regions; most importantly, the visibility subdivisions are computed from all triangles, unnecessarily adding and then removing the contributions of triangles that are only visible with respect to the triangles considered so far, and that ultimately turn out to be hidden. We bypass the need for beams, replacing them with the smallest number of rays needed to capture the visible triangles over a solid angle; we do not trace rays, but rather evaluate visibility along them by projection onto their sampling locations; finally, we compute the visibility subdivision exclusively from visible triangles.

Conservative Visibility algorithms are exact algorithms that run on a visibility problem that was conservatively simplified, e.g. through extended projections [12], or occluder erosion [10]. Our aggressive algorithm produces a visible set that is almost complete, so adding the triangles that are not hidden by the aggressive set yields a good conservative visible set. Per-frame occlusion culling improves rendering performance by batch discarding triangles that are hidden in the current output frame [4]. Triangles are grouped inside containers with simple geometry, the containers are rendered on a partial z-buffer of the output frame obtained from known big blockers, and the triangles of hidden containers are discarded. Occlusion culling methods can also be aggressive by fusing blockers heuristically [35].

Irregular Framebuffers. We advocate abandoning the uniform sampling of conventional images in favor of adding sampling locations deterministically to guarantee that all visible triangles are found. The benefits of irregular framebuffers have been noted before in contexts that include: pixel-accurate shadow mapping [20], where the shadow map estimates light visibility precisely at the point samples captured by the output image; point-based rendering [28], where projected reference image samples are not clamped to the output image pixel grid but rather located precisely within the output image pixel using a pair of offsets; and focus plus context visualization where focus regions are sampled at a higher rate [14].

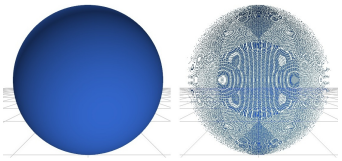

Fig. 2. Image of a finely tessellated sphere (left) and frame rendered from the visible triangles found by the image, using the same viewpoint, but a slightly different view direction (right). The frame shows that the visible set is far from complete.

3 Guaranteed-Quality Aggressive Visibility

An image is an appealing tool for computing visibility. Rendering an image amounts to probing dataset visibility efficiently with millions of rays, one for each pixel. However, the visible set found by an image can be incomplete because triangles can have small footprints due to high dataset complexity, to large distances to the eye, or to grazing viewing angles. Fig. 2 shows that a conventional image misses most visible triangles of the front surface of a finely tessellated sphere. A slightly different view direction reveals the many gaps in the aggressive visible set. A single sampling location per pixel, e.g., at the center of the pixel, captures only one of the many visible triangles whose projections overlap with the pixel (Fig. 3, left). Alleviating the problem by increasing image resolution is inefficient, as some triangles will be sampled multiple times, and only palliative, as no matter the resolution, the image cannot guarantee finding all triangles.

3.1 Approach

We improve the quality of the visible set found by an image by enhancing the image with additional sampling locations where visibility is probed. The goal is to minimize the number of additional sampling locations and to maximize the number of triangles found. Furthermore, the additional sampling locations should guarantee finding all triangles of a front surface, no matter how small their image footprint.

Sampling locations are added in greedy fashion with a pass over the dataset triangles, to make sure that all *triangle fragments* are sampled. We define a triangle fragment as the intersection between the image plane projection of a triangle and a pixel that the projection touches. If a triangle fragment does not contain any of the sampling locations already defined at the pixel, the list of sampling locations at the pixel is extended with the fragment center. The resulting set of sampling locations samples all triangle fragments (Fig. 3, left). The dataset is then rendered over the sampling locations to obtain the visible set.

Our approach *guarantees* finding all triangles with a completely visible fragment, which includes all completely visible triangles such as those of a front surface. Consider a fragment of a triangle of a front face. The first pass of the algorithm guarantees that the fragment contains a sampling location, and, since the triangle is completely visible, the triangle will win the visibility test at the sampling location, which guarantees finding the triangle. Our approach is *efficient*. Whereas the conventional approach of uniformly increasing the image resolution adds sampling locations blindly, in the hope of finding more triangles, our approach adds a sampling location only when it encounters a fragment that is not sampled by any of the existing sampling locations, and the new sampling location is guaranteed to sample the fragment by construction. Our approach increases the image resolution locally, as needed to sample the visibility of dataset regions with higher complexity. Our approach finds all the triangles of the front face of the sphere in Fig. 2 at the cost of one sampling location per fragment.

3.2 Algorithm

Our aggressive from-point visibility algorithm (Algorithm 1) takes as input the dataset geometry D, the point for which to estimate the

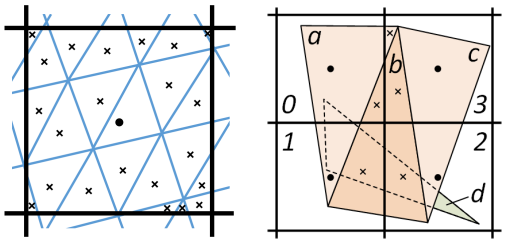

Fig. 3. **(Left)** Finding all front-surface triangles. The triangles (blue lines) projecting at a pixel (black lines) are shown in wireframe. A sampling location at the pixel center (dot) captures only one of the many visible triangles. Our aggressive algorithm adds sampling locations (crosses) to sample all triangle fragments at the pixel. **(Right)** Aggressive visibility and extension to exact visibility. The aggressive algorithm processes triangles a–d, in that order, finding the visible triangles a, b, c. Triangle d is missed because d does not have a completely visible fragment, and because its only partially visible fragment (in pixel 2), is sampled where d is hidden by b. The exact algorithm finds that the fragment of d in pixel 2 is not hidden by the earlier triangles a–c, and adds a sampling location in the visible part of the fragment (gray).

visibility o, and a uniform 2D grid of pixels G. G corresponds to the reference image where visibility is computed.

Algorithm 1 Aggressive from-point visibility

Input: dataset triangles D, viewpoint o, grid of pixels G
Output: aggressive set of visible triangles V_0

1: **for all** pixels $p \in G$ **do** $p.S = \{Center(p)\}$
2: **Render** D from o over G // PASS 1
3: **for all** triangles $t \in D$ **do** // PASS 2
4: $t' = Project(t, o, G)$
5: **for all** pixels $p \in t'$ **do**
6: **if** $\nexists\ s \in p.S$ such that $s \in t'$ **then**
7: fragment $f = t' \cap p$
8: sampling location $s = Centroid(f)$
9: $t_0 = p.center.triangle$
10: **if** t is closer to o than t_0 at s **then**
11: $p.S = p.S \cup \{s\}$
12: **Render** D from o over G // PASS 3
13: **for all** sampling locations s of all pixels in G **do**
14: $V_0 = V_0 \cup \{s.triangle\}$
15: **return** V_0

(Line 1) Each pixel of G stores a set of sampling locations S, which initially contains only the pixel center. In addition to its 2D coordinates, a sampling location also stores the depth to the closest triangle sampled so far, and that triangle's index, which are used to find the triangle visible at that sampling location.

(Line 2) The algorithm takes three passes over the dataset. The first pass is a conventional rendering over the pixel grid with one sampling location at each pixel center. The motivation for this first pass is efficiency. The first pass finds nearby triangles with a large footprint, which hide a large number of triangles. These triangles are used in the second pass to avoid creating unnecessary sampling locations, i.e. sampling locations that cannot provide additional visibility information. In Figure 3 (right), the first pass finds triangle a as visible at the centers of pixels 0 and 1, and c at the centers of pixels 2 and 3.

(Lines 3–11) The second pass adds sampling locations to ensure that all triangle fragments are sampled. For each pixel p covered by the projection t' of a triangle t (line 5), the algorithm checks whether p already has a sampling location s that samples t (line 6). Since s must be inside p, this is equivalent to checking whether p has a sampling location inside the fragment f of t at p. If not, f is computed and a sampling location s is created at its centroid (lines 7–8). s is added to the set of sampling locations of p, unless t is hidden at s by the triangle

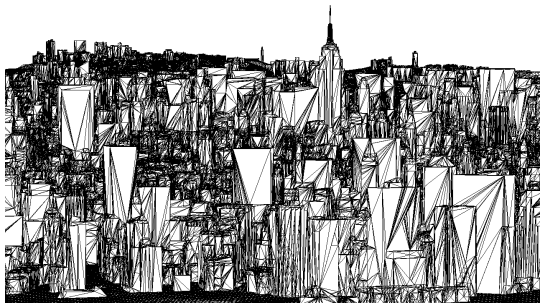

Fig. 4. Visibility subdivision for top-left image in Figure 1.

t_0 found by the first pass as visible at the center of p (lines 9–11). The test in line 10 prevents a futile attempt to probe the visibility of t with a sampling location where it is already known that t_0 is closer than t. The test implements occlusion culling efficiently by precluding the generation of sampling locations for any triangle completely hidden by the visible triangles found at the first pass.

In Figure 3 (right), a generates one additional sampling location in pixel 3 (cross), and b generates one in each of the four pixels. Triangle d generates no sampling locations, as the fragments of d in pixels 1 and 2 already contain sampling locations that were added for b. The fragment of d in pixel 0 does not contain a sampling location because d is hidden at the centroid of its fragment by a, which was found at step 1. A sampling location at the centroid of the fragment of d in pixel 0 would be wasteful as it would only reconfirm that a is visible, with no chance of elucidating the visibility status of d.

(Line 12) The third pass renders the dataset triangles over the sampling locations defined by the second pass. For this, each triangle is projected onto the pixel grid G; for each pixel touched by the triangle projection, the triangle is z-buffered over all the pixel's sampling locations that are inside the triangle projection.

(Lines 13–15) The closest triangles recorded by each sampling location after the third pass are collected to form the visible set.

4 Efficient and Robust Exact Visibility

Exact from-point visibility algorithms have to partition the 2D space of rays through the viewpoint into regions from which a single triangle is visible. This visibility subdivision can be specified as a polygonal subdivision of a continuous image plane (Figure 4). One challenge is efficiency. Whereas it is well understood how to build the visibility subdivision incrementally, one triangle at a time, it is inefficient to do so by considering all triangles in the dataset, including the hidden ones. It is unnecessary to update the visibility subdivision for a hidden triangle t, as the updates will be undone once the triangles hiding t are processed. Since in a complex dataset the overwhelming majority of triangles are hidden, updating the visibility subdivision unnecessarily for all hidden triangles results in a significant amount of wasted computation. Another challenge is robustness. We have seen that the key to a robust implementation is correct predicate evaluation. Floating point evaluation is fast, but rounding error can yield an incorrect sign, which can cause a large error in the visibility subdivision or even a program crash. Sign errors are most likely for *degenerate* predicates whose true sign is zero. Predicates can be evaluated correctly using arbitrary precision integer arithmetic, but this is slow. A second robustness problem is that degenerate predicates complicate the visibility subdivision algorithm by introducing a third case at every branch.

4.1 Approach

For our exact from-point visibility algorithm to be efficient, we construct the visibility subdivision exclusively from visible triangles. For this we introduce a *hybrid* approach to exact visibility computation that combines sample-based with continuous visibility analysis (Figure 5).

The aggressive visibility algorithm is run first (step 0). Then the aggressive visible set is extended iteratively to the exact set (steps

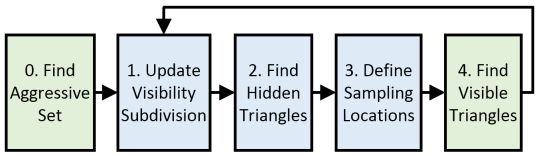

Fig. 5. Iterative from-point visibility approach combining sample-based (green) and continuous (blue) visibility analysis.

1-4). At each iteration, the visibility subdivision is updated based on the newly found visible triangles (step 1). For the first iteration, an initial visibility subdivision is built from the aggressive visible set. Then the triangles yet to be decided as visible or hidden are tested against the updated visibility subdivision in search of additional hidden triangles (2). Since all completely visible triangles have already been found by the aggressive algorithm, a triangle cannot be visible without its projection intersecting an edge of the visibility subdivision. Therefore a triangle is hidden if its projection does not intersect any subdivision edge, or if it is hidden at all such intersections. At the first iteration, all dataset triangles not in the aggressive set are undecided. Since the aggressive set is almost complete, the visibility subdivision is almost complete, so the first iteration finds most hidden triangles. Additional sampling locations are defined where an undecided triangle is not hidden by the current visibility subdivision (3). The undecided triangles are rendered over the additional sampling locations to reveal additional visible triangles (4). The iterative process stops if after step 2 no undecided triangles remain. A sampling location created at step 3 is guaranteed to find a new visible triangle because it is inside an undecided triangle and outside the current visibility subdivision. Therefore each iteration reduces the undecided set and the algorithm is guaranteed to converge.

4.2 Algorithm

The set of triangles found by the aggressive algorithm contains only but not all visible triangles. Most dataset triangles not in the aggressive visible set are likely to be hidden, but none is confirmed to be hidden. We have developed an exact from-point visibility algorithm that leverages the aggressive visible set to find the remaining visible triangles, confirming that all other triangles are hidden (Algorithm 2).

Algorithm 2 Exact from-point visibility

Input: dataset triangles D, viewpoint o, reference image G, aggressive visible set V_0
Output: exact set of visible triangles V

```
1:  V = V_0, U = D − V_0
2:  VS = ConstructVisibilitySubvision(V_0, o, G)
3:  while U ≠ ∅ do
4:      for all pixels p ∈ G do p.S = ∅
5:      for all triangles t ∈ U do
6:          if t is hidden by VS then
7:              U = U − {t}
8:          else
9:              AddSamplingLocations(t, G)
10:     Render U from o over G
11:     for all sampling locations s of all pixels in G do
12:         t = s.triangle, V = V ∪ {t}, U = U − {t}
13:         VS = AddTriangle(VS, t)
14: return V
```

(Line 1) Initialize the visible set V to the aggressive visible set V_0 and place the other triangles in the undecided set U.

(Line 2) Construct an initial visibility subdivision *VS* from the aggressive visible set V_0 by projecting triangles from o onto the rectangular frame of G. *VS* is a subdivision of the image frame into polygonal regions where a single triangle is visible. The region boundaries are the visible segments of the projected triangle edges. *VS* is constructed

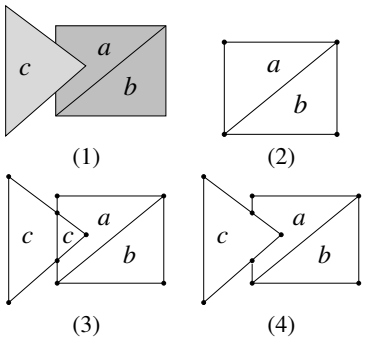

Fig. 6. Incremental construction of visibility subdivision.

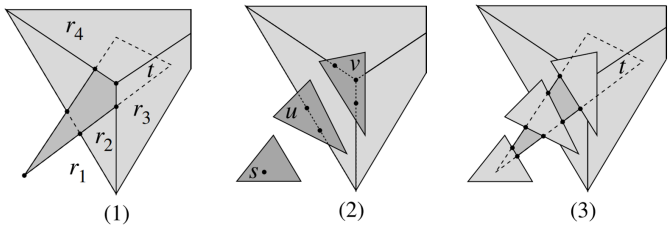

Fig. 7. (1) Six sampling locations (dots) created for undecided triangle t at first iteration, (2) triangles s, u, and v visible at those sampling locations, and (3) eight sampling locations (dots) created for t at second iteration.

incrementally by adding one visible triangle at a time. Consider the dataset with three triangles a, b, c from Figure 6 (1). After a and b are added, *VS* has two regions, one for a and one for b (2). *VS* is updated with triangle c in two steps. First, *VS* is intersected with c, which shrinks the region for a to a concave hexagon, and creates two regions for c (3). Second, the two regions for c are merged (4).

(Lines 3–13) Iterate until there are no more undecided triangles. Each iteration first clears the sets of sampling locations S stored at each pixel p in G (line 4), as a sampling location is not useful beyond the iteration when it was created and found its visible triangle. Then the undecided triangles are processed one at a time (lines 5–9). If an undecided triangle t is hidden by the visibility subdivision *VS*, t is removed from further consideration (line 7). If t is not hidden by *VS*, sampling locations are created for each region r of *VS* where t is visible (line 9). For a region r and a triangle t, the algorithm creates sampling locations at the vertices of t that project in r, at the vertices of r inside the projection of t, and at the intersection points of the edges of r with the projected edges of t. The sampling locations are created inside t, albeit close to its edges or vertices. In Figure 7 (1), t is hidden in regions r_3 and r_4, but not in r_1 and r_2, and six sampling locations are created. After the undecided triangles are processed, the remaining undecided triangles are rendered over the new sampling locations (line 10). The newly found visible triangles are added to the visible set, and are used to update the visibility subdivision (lines 11–13).

Dataset visibility is determined over several iterations. A sampling location q generated for an undecided triangle t might be covered by another undecided triangle that is closer than t at q. If all sampling locations generated for an undecided triangle t are won by other undecided triangles, the visibility of t will not be determined at the current iteration. In Figure 7, none of the six sampling locations generated for triangle t at the first iteration are won by t–they are won by triangles s, u, v (2). Therefore, t remains undecided after the first iteration. The updated visibility subdivision does not hide t, so another eight sampling locations are generated for t (3).

The algorithm is guaranteed to converge because every iteration creates a sampling location at which an undecided triangle is visible, thereby shrinking the undecided set.

The algorithm is fast for several reasons.

(1) The initial construction and subsequent updates of the visibility

subdivision only process visible triangles, which saves the unnecessary cost of adding a hidden triangle to the visibility subdivision just to remove it later as the triangles that hide it are discovered. In other words, the expensive construction of the visibility subdivision is shielded from the full dataset complexity and only has to run on the visible set which is a small fraction of the total number of triangles.

(2) Almost all triangles are decided by the first iteration, so little effort is wasted on failed classification attempts. Since the starting aggressive visible set is almost complete, the visibility subdivision is almost complete, so the first iteration finds almost all hidden triangles, and few triangles remain to be decided through additional iterations.

(3) A triangle that fails to be decided at the current iteration is sampled in a way that increases the likelihood that it will be classified at the next iteration. An undecided triangle t is sampled with at least three sampling locations close to the contour of the part of t that is not hidden by the current visibility subdivision. It is unlikely that t be visible, yet hidden at all of these sampling locations. These sampling locations either reveal that t is visible, or they reveal visible triangles that complete the visibility subdivision over t, which allows the next iteration to determine that t is hidden.

4.3 Robustness

The ACP robustness technique consists of an input perturbation prior to running the algorithm and of a predicate evaluation algorithm. The input perturbation adds to each vertex coordinate a random number uniformly selected in $[-\delta,\delta]$, with $\delta = 10^{-8}$. This perturbation is negligible in terms of visibility, prevents degenerate predicates with high probability, and changes the value of most predicates by $O(\delta)$.

The algorithm for visibility subdivision construction and update (lines 2 and 13 in Algorithm 2) employs two predicates. The first predicate is $LT(a,b,c)$ for 2D points a,b,c. It equals -1 or 1 when the path abc is a right or left turn, and is degenerate when the points are collinear. The predicate expression is $(c-b)\times(a-b)$ with $u\times v = u_x v_y - u_y v_x$. The 2D points are the projections of dataset vertices, hence are rational expressions in the 3D vertex coordinates and in the camera parameters. The predicate is used to implement two geometric tests. A point p is inside a triangle abc if $LT(a,b,p)=1$, $LT(b,c,p)=1$, and $LT(c,a,p)=1$. Line segments ab and cd intersect if $LT(a,b,c) = -LT(a,b,d)$ and $LT(c,d,a) = -LT(c,d,b)$. The second predicate determines the order of 3D points a and b along a ray with direction vector u. The predicate expression is $(a-b)\cdot u$ with $u\cdot v = u_x v_x + u_y v_y + u_z v_z$.

We evaluate predicates with double precision floating point interval arithmetic, which provides an interval that contains the true value of the predicate. The sign is determined unless the interval contains zero. Ambiguity is rare because the interval width is on the order of the floating point rounding unit 1.1×10^{-16}, whereas the exact value is $O(\delta)$. We resolve ambiguous cases by increasing the precision of the interval arithmetic, and thus shrinking the interval. We start with a precision of 212 bits and increase it in increments of 53 bits until zero is excluded. The extended precision arithmetic uses the MPFR library [13]. The overall efficiency is high because ambiguity is rare.

Perturbation prevents degeneracy due to input in special position, such as three collinear triangle vertices. However, perturbation cannot prevent degeneracy due to algebraic relations among derived quantities. For example, suppose an input triangle abc intersects an input edge uv at p. Let $\hat{u}$, $\hat{v}$, and $\hat{p}$ be the projections of u, v, and p. The predicate $LT(\hat{u},\hat{v},\hat{p})$ is degenerate for all inputs. We call this type of degeneracy an *identity*. We use structural information to detect identities. For example, we label intersection points with their defining edges and triangles. The most complicated case is three 2D edges that intersect at a point because they are the projections of the intersection edges of three triangles that intersect at a point. Once an identity is detected, it is handled with special-case logic.

The robust implementation of the exact visibility algorithm ensures a correct output at a moderate computational cost. Visible triangles are never omitted from the visible set, hidden triangles are never included, and the program never crashes.

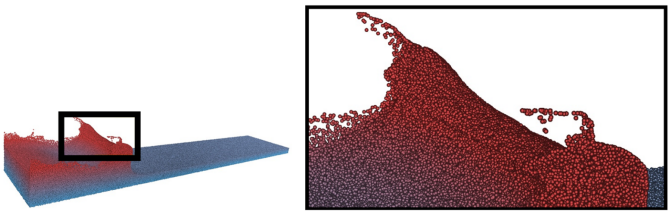

Fig. 8. *Left*: reference view on which our aggressive from-point particle visibility algorithm was run. *Right*: frame rendered from the visible particles; the zoom factor is 4x and there are 0.09% incorrect pixels

5 Spherical Particles

We have described our visibility algorithms for datasets modeled with triangles. The algorithms support any geometric primitive that can be tessellated. For example, the spherical particles used in a smoothed particle hydrodynamics (SPH) simulation can be tessellated and the resulting triangle meshes can be processed with our algorithms. However, for some applications it might be sufficient to determine visibility at particle level, which saves the cost of computing partial particle visibility. For this, we have extended our aggressive algorithm to support spherical particles directly. This reduces substantially the number of geometric primitives considered by the algorithm, as even a coarse a regular octahedron tessellation implies eight triangles for every particle. Also, a coarse tessellation would reduce the accuracy of the visible set.

The aggressive algorithm now has to ensure that all *particles* are sampled. In order to work with particles, Algorithm 1 requires three changes. First, the algorithm has to determine the grid pixels that are covered by the projection of the particle, as needed for each of the three rendering passes (lines 2, 5, and 12). This is done with a conservative circular approximation of the elliptic projection of the particle, centered at the particle center projection. Second, the algorithm has to tell whether a sampling location, i.e. a point on the image plane, is inside the projection of the particle, as needed for each of the three rendering passes (lines 2, 6, 12). This is done accurately, in 3D, by checking whether the distance from the particle center to the sampling location ray is less than the particle radius. Third, the algorithm has to find the centroid of a particle fragment, which is the projection of the particle center if the center projects inside the pixel, or else the average of the intersection points between the pixel frame and the circular approximation of the particle projection. Our aggressive from-point particle visibility algorithm, produces high-quality frames, even for zoomed-in views Figure 8.

6 Results and Discussion

We have tested our algorithm on several datasets and viewpoints (Table 1). All datasets are modeled with triangles except for *Water* and *Fusion*, which are modeled with spherical particles. The *Impact* dataset was generated by a finite element analysis (FEA) simulation that modeled the dataset with beam, thin shell, thick shell, and hexahedral elements, which were then triangulated for visualization; the liquid was simulated with the Arbitrary Lagrangian Eulerian (ALE) method and visualized with an isosurface triangle mesh. The *Isosurface* dataset is a triangle mesh. The resolution of the image where our visibility algorithms were run was the same as the resolution of the output frames, i.e. 720×1280. For the Manhattan midtown view we computed visibility in all directions using a cubemap, with a resolution of $6\times1024\times1024$.

6.1 Quality

The quality of the visibility solution computed by our aggressive algorithm is given in rows 4 to 8 of Table 1. Row 4 gives the percentage of visible triangles found. However, this metric is not an accurate estimate of the quality of the visible set, as it does not take into account the size of the visible part of a triangle–missing triangles whose visible parts have areas of 10 or 1.0e-10 pixels are counted the same. For an accurate estimate of the quality of the visible set, we provide the visibility subdivision completeness (row 5), defined as the percentage

Fig. 9. Reference view on which our algorithms were run (*left*) and sample output frames from aggressive visible set (*right*).

Fig. 10. Error pixels highlighted in red for a frame with a 17x zoom factor, rendered from our aggressive set (Figure 1, row 2 & col. 2).

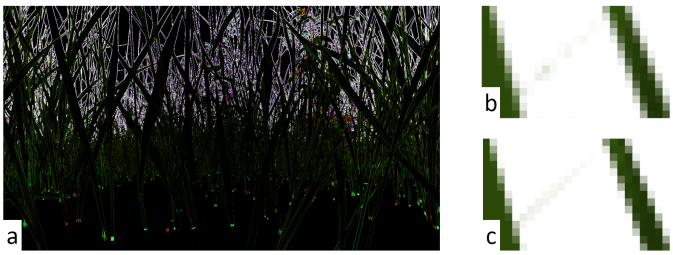

Fig. 11. (a) Difference between conventional 4x4 antialiasing and accurate antialiasing using the exact visible set computed using our from-point visibility algorithm, for the reference *Grass* image in Figure 1; 45.5% of the pixels of the conventional image have an incorrect color value; the average and maximum per pixel color channel errors are 2.96 and 117.65 on the [0, 255] scale. Magnified detail of the conventional (b) and exact (c) images, showing two thick grass blades (left and right) connected by a thin grass blade (diagonally across); the conventional image exaggerates large dataset features, e.g., the thick grass blades, and undersamples fine dataset features, e.g., the thin grass blade.

of the reference image area where the aggressive algorithm computes visibility correctly. Given a reference image point p, visibility is computed correctly at p if the triangle visible at p is part of the aggressive set. We compute the visibility subdivision completeness by comparing the visibility subdivisions built from the aggressive and the exact visible sets. The completeness is 97.55% and 98.14% for *Grass high* and *Forest*, which have tiny triangles and high depth complexity, and 99.66% or better in all the other cases.

We verify the quality of our aggressive visibility algorithm in a second way by rendering sequences of thousands of frames from the aggressive visible sets. The frames are rendered from the reference viewpoint with view direction and zoom factor changes. Row 6 gives the average zoom factor over the sequence. Rows 7 and 8 give the average and maximum percentages of incorrect pixels per frame over the sequence. A pixel is incorrect if it not set by the same triangle that sets it when the frame is rendered from the entire dataset. The errors are small even though the sequences include zooming in. The maximum zoom factors for *Manhattan*, *Grass*, and *Forest* are 7x, 17x, and 10x. As our aggressive algorithm guarantees finding all triangles of a front surface, the few incorrect pixels occur between surfaces, where they are less noticeable. The aggressive algorithm avoids holes in a visible surface efficiently in a purely sample-based fashion, without computing triangle mesh continuity. For example, our aggressive set might omit a grass blade that is barely visible at the silhouette of a front grass blade, but it will never leave an objectionable hole in the middle of the front grass blade (Figure 10).

Our exact algorithm finds all visible triangles, from which accurate frames are rendered. Compared to rendering the entire dataset, finding the exact set brings not only the efficiency advantage of not rendering hidden triangles, but also the quality advantage of avoiding aliasing artifacts no matter how high the dataset complexity. Indeed, the visibility subdivision computed by the exact algorithm allows aggregating an output pixel color correctly from the contributions of *all* triangles visible at the pixel, appropriately weighted by their fractional pixel coverage [5]. By comparison, even when the dataset is rendered with high levels of conventional antialiasing, severe artifacts remain (Figure 11).

6.2 Efficiency

The efficiency of the aggressive algorithm is summarized in rows 9 to 11 of Table 1. The number of sampling locations per pixel (rows 9 and 10) depends on the average image footprint of the triangles and on the presence of large blockers, so it is small for datasets like *Manhattan*, and larger for datasets like *Isosurface*. The optimal set of sampling locations for a sample-based visibility algorithm has exactly one sampling location per visible triangle. The set of sampling locations constructed by our algorithm is missing some sampling locations, i.e., for the few visible triangles the algorithm does not find, and it also has some unnecessary sampling locations, i.e., for visible triangles that are found by multiple sampling locations. Our algorithm can create unnecessary sampling locations in several scenarios: (1) a visible triangle covers several pixels and a sampling location is created at each pixel; (2) a sampling location is constructed for each of several hidden triangles, and all find the same visible triangle; (3) two overlapping triangles are sampled with two sampling locations, as opposed to with one sampling location at the region of overlap. For *Grass high*, there are 10.7M visible triangles (Table 1, row 2), so there should be at least 10.7M / (1280×720) = 11.6 sampling locations per pixel; our algorithm completes 97.55% of the visibility subdivision with 36.5 sampling locations per pixel. For *Forest*, the algorithm completes 98.14% of the visibility subdivision with 24.1 sampling locations per pixel, compared to the optimal number of 7.3M / (1280×720) = 7.9.

The running times of the aggressive algorithm (row 11) are for a parallel implementation on a workstation with 24 2.27GHz X7560 Intel cores. The algorithm was parallelized by tiling the reference image with a uniform 2D grid, assigning dataset triangles to tiles based on triangle footprint, and assigning the tiles to cores in round-robin fashion. The algorithm is faster than 1M triangles per second for all datasets, and it

Table 1. Experiment datasets (rows 1-3), aggressive algorithm quality (4-8) and efficiency (9-11), and exact algorithm efficiency (12-15).

	Manhattan		Grass		Forest	Impact		Isosurface	Water	Fusion
	downtown	midtown	low	high		outside	inside			
1. Visual reference	Fig 1.1	Video	Fig 1.2	Fig 9.1	Fig 9.2	Fig 9.3	Fig 9.4	Fig 1.3	Fig 8	Fig 9.5
2. Dataset triangles	3.96M	3.96M	54.9M	54.9M	47.4M	2.08M	2.08M	497M	2.17M	500K
3. Visible set triangles	2%	10%	0.5%	19%	15%	9%	1%	3%	5%	9%
4. Vis. set completeness	89.0%	89.1%	79.2%	71.5%	55.2%	80.1%	84.8%	90.1%	97.8%	95.6%
5. Vis. subd. completeness	99.98%	99.95%	99.93%	97.55%	98.14%	99.95%	99.99%	99.66%	-	-
6. Average zoom factor	3.4x	3.4x	3.4x	3.4x	3.4x	4.8x	3.9x	3.4x	3.4x	2.4x
7. Average pixel error	0.03%	0.05%	0.11%	2.64%	2.11%	0.03%	0.01%	0.27%	0.09%	0.04%
8. Maximum pixel error	0.08%	0.17%	0.20%	3.71%	3.04%	0.12%	0.03%	0.61%	0.13%	0.07%
9. Average SL / pixel	1.60	1.45	5.72	36.5	24.1	1.93	1.41	209	2.03	3.70
10. Maximum SL / pixel	57	703	1,490	742	1,245	106	27	1,670	27	29
11. Running time [s]	2.4	2.6	12	11	12	1.6	0.50	464	3.4	1.2
12. Visible tris (i1)	96.6%	95.7%	89.1%	81.2%	73.0%	96.1%	98.6%	94.7%	-	-
13. Decided tris (i1)	99.75%	98.51%	99.76%	90.38%	86.16%	98.33%	99.87%	95.32%	-	-
14. Iterations to convergence	2	-	3	3	3	4	3	3	-	-
15. Running time [s]	5.8	-	22	294	1,078	9.7	4.1	146	-	-

is slowest for the *Isosurface*. Our particle visibility algorithm runs at about half a million particles per second.

The efficiency of the exact algorithm is summarized in rows 12 to 15 of Table 1. Row 12 gives the completeness of the visible set after the first iteration. The algorithm iterates until there are no more undecided triangles. Most work is done by the first iteration, which decides most triangles in the initial undecided set (row 13). The algorithm converges in at most four iterations (row 14). The exact algorithm was parallelized like the aggressive algorithm. The running times (row 15) exclude the running time of the aggressive algorithm that provided the initial visible set. The exact algorithm is slowest for *Forest*, which requires many small updates to the initial visibility subdivision due dataset fragmentation and high depth complexity. The nearly half a billion triangles of *Isosurface* are processed in less than 3min.

6.3 Comparison to Prior Art

We compare our algorithms to two prior art approaches.

The **first approach** aggregates the visible set by uniformly sampling the space of visualization rays with conventional images. We compare our algorithms to computing visibility with a conventional image with resolutions ranging from the resolution of the grid used by our algorithms, i.e. 1280×720 pixels, to 256×256 times the grid resolution, i.e. $327,680 \times 184,320$ pixels (Table 2). For each row, the table highlights the uniform sampling level values that bracket the aggressive algorithm value, e.g. 1 and 4 bracket 1.47 for row 1 for *Manhattan*.

The aggressive algorithm generates a more complete visible set and a more complete visibility subdivision than uniform sampling, for the same number of sampling locations per pixel (rows 1). For example, for *Manhattan*, the uniform sampling levels that bracket the aggressive algorithm in terms of sampling locations per pixel are 1×1 and 2×2, which yield a less complete visible set and visibility subdivision. For the same completeness of the visible set, uniform sampling has to generate more sampling locations per pixel, see highlighted cells in rows 2. For example, for *Manhattan*, uniform sampling has to generate between 256 and 1024 sampling locations per pixel to match the visible set completeness achieved by our aggressive algorithm with only 1.47 sampling locations per pixel. Similarly, for the same completeness of the visibility subdivision as our aggressive algorithm, uniform sampling has to generate more sampling locations per pixel (rows 3).

Rows 4 give the running times, where the uniform sampling running times were measured on a GPU that worked on tiles of the ultra-high resolution image. As expected, the GPU beats the 24 core parallelization of the aggressive algorithm for the smaller datasets, computing a visibility subdivision that is more complete than the aggressive algorithm. However, the advantage diminishes as dataset size increases. For *Isosurface*, given the same amount of time, the GPU computes a visibility solution that is less complete than that computed by the aggressive algorithm on the CPU cores. As shown by the 1×1 uniform sampling running times, it is of course faster to render an output frame from the entire dataset than to compute visibility from a viewpoint. However, the 1×1 running times are too slow for interactive visualization. On the other hand, computing visibility brings a substantial reduction in the number of primitives, see lines 2 and 3 in Table 1. A visualization application can render from the same visible set hundreds of frames, with short per frame times that allow the user to change view direction and to zoom in interactively (see video accompanying our paper).

No matter how high the resolution of uniform sampling, the visibility solution is not complete. Visible triangles are missed even when sampling with over 65,000 samples per pixel. For example, for, *Isosurface*, the exact algorithm computes in fewer than 20min the exact set that eludes uniform sampling even after 10h of processing.

The **second prior art approach** to which we compare our algorithms is *Guided Visibility Sampling* (GVS), an aggressive from region visibility algorithm that samples heuristically the space of visualization rays that originate from a rectangle or a box [32]. Early rays generated stochastically guide the generation of subsequent rays based on two heuristics: *adaptive border sampling*, which looks for new visible triangles adjacent to visible triangles that were already found, and *reverse sampling*, which looks for visible triangles in unsampled but accessible space defined by depth discontinuities. The iterative search for visible triangles is terminated heuristically based on a predetermined ray budget or on a lower threshold for the rate of visible triangle discovery. GVS was recently updated to GVS++, which has improved heuristics and performance optimizations, and which takes advantage of the Vulkan graphics API and of RTX ray tracing [23].

For a first set of experiments we restricted GVS++ to from-point visibility by reducing the view region to a view point. For the midtown viewpoint of the *Manhattan* dataset, GVS++ casts 983 million rays to find 399,401 of the 406,036 triangles in the exact visible set, whereas our aggressive algorithm finds 374,172 triangles with only 8.7 million sampling locations. More importantly, although it samples visibility with *two orders of magnitude* fewer rays than GVS++, our algorithm nearly completes the visibility subdivision at 99.9860%, which is slightly better than the 98.9857% completeness for GVS++. Leveraging graphics hardware (NVIDIA GeForce RTX 3060 Laptop GPU - 6 GB VRAM), GVS++ runs in 0.8s, whereas the CPU implementation of our aggressive algorithm takes 2.6s. Compared to our exact algorithm, GVS++ fails to find all visible triangles and to complete the visibility subdivision even after casting nearly one billion rays. Furthermore, the GVS++ implementation is not robust: the visible set it finds also contains 3,232 triangles that are *not* visible. Finally, GVS++ does not provide the visibility subdivision, as needed for applications of visibility such as antialiasing, and the visibility subdivision has to be computed from the visible triangles in an additional step.

For a second set of experiments we compared our approach to GVS++ on from-region visibility. The region is modeled as a box, which GVS++ takes directly as input. To compute from-region visibility with our approach we ran our aggressive from-point algorithm

Table 2. Comparison to uniform supersampling.

Manhattan	**Aggr.**	**1 × 1**	**2 × 2**	**4 × 4**	**8× 8**	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**Exact**
1. SL / pixel	1.47	1	4	16	64	256	1024	4,096	16,384	65,536	1.53
2. Visible triangles [%]	91.2	35.2	49.6	64.5	77.9	87.8	93.6	96.6	98.1	98.9	100
3. Vis. sub. compl. [%]	99.9	97.4	99.6	99.9	99.9	99.9	99.9	∼100	∼100	∼100	100
4. Time [s]	2.36	0.34	0.34	0.35	0.35	1.4	5.6	22.4	89.6	358.4	7.16
Grass	**Aggr.**	**1 × 1**	**2 × 2**	**4 × 4**	**8× 8**	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**Exact**
1. SL / pixel	5.2	1	4	16	64	256	1024	4,096	16,384	65,536	5.7
2. Visible triangles [%]	83.5	31.0	47.5	63.6	77.2	87.0	93.0	96.3	98.1	99.0	100
3. Vis. sub. compl. [%]	∼100	96.3	99.0	99.7	99.9	99.9	99.9	99.9	99.9	∼100	100
4. Time [s]	15.96	3.2	3.2	3.3	3.3	13.2	52.8	211.2	844.8	3,379	37.96
Forest	**Aggr.**	**1 × 1**	**2 × 2**	**4 × 4**	**8× 8**	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**Exact**
1. SL / pixel	22	1	4	16	64	256	1024	4,096	16,384	65,536	45
2. Visible triangles [%]	69.1	6.90	20.1	42.2	64.3	80.1	89.4	94.6	97.2	98.6	100
3. Vis. sub. compl. [%]	98.2	60.3	80.3	94.1	97.8	98.8	99.3	99.9	99.9	∼100	100
4. Time [s]	12.43	3.3	3.3	3.4	3.4	13.6	54.4	217.6	870.4	3,481	1,020
Isosurface	**Aggr.**	**1 × 1**	**2 × 2**	**4 × 4**	**8× 8**	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**Exact**
1. SL / pixel	193	1	4	16	64	256	1024	4,096	16,384	65,536	213
2. Visible triangles [%]	91.2	2.46	9.84	33.5	62.9	80.6	89.3	93.2	94.8	96.2	100
3. Vis. sub. compl. [%]	99.5	5.23	20.75	64.0	93.1	98.7	99.3	99.5	99.7	99.9	100
4. Time [s]	464.4	33.4	34.6	35.8	36.4	145.6	582.4	2,329	9,318	37,273	610.4

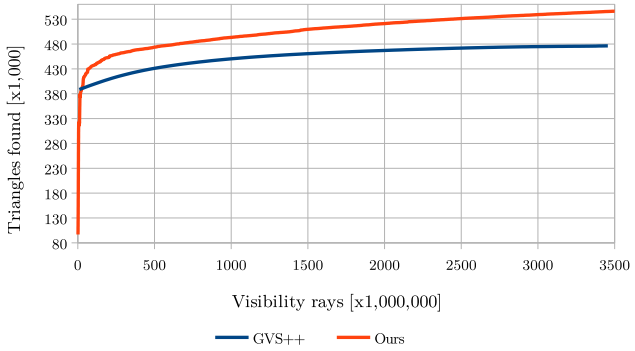

Fig. 12. Comparison between our approach and the prior art approach GVS++ on from-region visibility. Our approach converges more efficiently.

on viewpoints selected randomly inside the view box. The example in Figure 12 uses the *Manhattan* dataset and a 30m view cube centered at the midtown viewpoint. Our approach sampled the box with 2,666 viewpoints, accumulating 3.5 billion sampling locations to find 546,000 visible triangles. GVS++ handles the box in a single run, which gives them a substantial running time advantage, but the visible set converges more slowly, with GVS++ finding only 476,000 visible triangles after evaluating a comparable number of visibility rays. GVS++ does not run on our bigger datasets due to insufficient GPU memory.

6.4 Limitations

The implementation of our aggressive visibility algorithm has an asymptotic running time quadratic in the number of sampling locations per pixel, as a new sampling location is created only after checking that no current location is inside the fragment. We have not observed a quadratic slowdown in our experiments, which means that the linear term dominates for the relatively short lists of sampling locations at the pixels of the grid. The running time per one million triangles varies from 0.2s to 0.9s and is uncorrelated with the average or the maximum number of sampling locations per pixel. In particular, the *Isosurface* has average 209 and maximum 1,670, yet is only 15% slower than *Impact outside* with average 1.9 and maximum 106. Should the quadratic slowdown become a problem, it could be avoided by using a grid resolution commensurate with the average triangle footprint, thereby bounding the average number of sampling locations per pixel. Another option is to subdivide pixels hierarchically, e.g. with a quadtree.

Another limitation of the current work is that exact visibility for particle datasets can only be computed by tessellating the particles, to be able to run our exact triangle visibility algorithm. Since tessellation increases the number of primitives substantially, future work could explore extending the exact algorithm to particles. The challenge is that the projection of a particle has curved edges, so the visibility subdivision is more expensive to build and update.

7 Conclusions and Future Work

We have described a novel approach to from-point visibility based on image generalization. The regular sampling grid of a conventional image is enhanced as needed with sampling locations defined by dataset geometry. A small number of additional sampling locations per pixel are sufficient to reveal most visible triangles. We provide a visibility approximation that guarantees that no completely visible triangle is omitted, no matter how small its footprint, which guarantees front surface continuity. We couple a sample-based and a continuous visibility analysis of the image plane to complete the visible set efficiently, in a remarkably small number of iterations. Our exact visibility algorithm is implemented robustly. The exact visible set allows rendering *accurate* images, by taking into account all triangles visible at a pixel, overcoming the aliasing artifacts that pervade visualizations of complex datasets even when high levels of conventional antialiasing is used.

Our visualization algorithms are currently implemented on CPU cores. One direction of future work is to increase performance by leveraging the programmability of current graphics hardware. The challenge is the variable number of sampling locations per pixel. Another option is to devise hardware extensions that bring native support to rendering over irregular framebuffers.

A second direction of future work is to use the image generalization paradigm to develop more general visibility algorithms. Our from-point algorithms compute visibility for a 2D visualization ray space. We have already shown that our algorithms can provide a good approximation of visibility for the 4D ray space of from-rectangle visibility, by aggregating visibility over a set of viewpoints sampling the view rectangle. However, sampling the view rectangle is both redundant, as the visible sets of nearby viewpoints have substantial overlap, and approximate, as no view rectangle sampling resolution can guarantee finding all visible triangles. We envision generalizing the 0D image plane sampling locations used in from-point visibility to 1D and 2D sampling locations that can capture the triangles visible from a segment and a rectangle efficiently, accurately, and non-redundantly. The exact visibility algorithm can be generalized to an interval of viewpoints by constructing the initial subdivision and updating it at viewpoints where structural changes occur, for example a triangle becomes visible or vanishes. The ultimate goal is to develop 3D visibility sampling locations, suitable for the 5D visualization ray space of from-rectangle visibility of dynamic datasets.

REFERENCES

[1] O. Apostu, F. Mora, D. Ghazanfarpour, and L. Aveneau. Analytic ambient occlusion using exact from-polygon visibility. *Computers & Graphics*, 36(6):727–739, 2012.

[2] T. Auzinger, M. Wimmer, and S. Jescke. Analytic visibility on the gpu. *Computer Graphics Forum*, 32(2pt4):409–418, 2013.

[3] J. Bittner, O. Mattausch, P. Wonka, V. Havran, and M. Wimmer. Adaptive global visibility sampling. *ACM Transactions on Graphics (TOG)*, 28(3):94, 2009.

[4] J. Bittner, M. Wimmer, H. Piringer, and W. Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum*, 23(3):615–624, 2004.

[5] L. Carpenter. The a-buffer, an antialiased hidden surface method. *ACM SIGGRAPH Computer Graphics*, 18(3):103–108, 1984.

[6] E. Catmull. A hidden-surface algorithm with anti-aliasing. *ACM SIGGRAPH Computer Graphics*, 12(3):6–11, 1978.

[7] A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha. Ad-frustum: Adaptive frustum tracing for interactive sound propagation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1707–1722, 2008.

[8] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):412–431, 2003.

[9] J. Cui, P. Rosen, V. Popescu, and C. Hoffmann. A curved ray camera for handling occlusions through continuous multiperspective visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1235–1242, 2010.

[10] X. Décoret, G. Debunne, and F. Sillion. Erosion based visibility preprocessing. In *Proceedings of the 14th Eurographics workshop on Rendering*, pp. 281–288. Eurographics Association, 2003.

[11] F. Durand. *3D Visibility: analytical study and applications*. PhD thesis, Université Joseph Fourier, 2010.

[12] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 239–248. ACM Press/Addison-Wesley Publishing Co., 2000.

[13] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33:13, 2007.

[14] G. W. Furnas. Generalized fisheye views. *SIGCHI Bull.*, 17(4):16–23, 1986.

[15] M. T. Goodrich. A polygonal approach to hidden-line and hidden-surface elimination. *CVGIP: Graphical Models and Image Processing*, 54(1):1–12, 1992.

[16] C. J. Gribel, R. Barringer, and T. Akenine-Möller. High-quality spatio-temporal rendering using semi-analytical visibility. *ACM Transactions on Graphics (TOG)*, 30(4):54, 2011.

[17] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *ACM SIGGRAPH Computer Graphics*, 18(3):119–127, 1984.

[18] J. Hladky, H.-P. Seidel, and M. Steinberger. The camera offset space: Real-time potentially visible set computations for streaming rendering. *ACM Trans. Graph.*, 38(6), nov 2019. doi: 10.1145/3355089.3356530

[19] W. Hunt, M. Mara, and A. Nankervis. Hierarchical visibility for virtual reality. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(1), jul 2018. doi: 10.1145/3203191

[20] G. S. Johnson, J. Lee, C. A. Burns, and W. R. Mark. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics (TOG)*, 24(4):1462–1482, 2005.

[21] T. R. Jones and R. N. Perry. Antialiasing with line samples. In *Rendering Techniques 2000*, pp. 197–205. Springer, 2000.

[22] M. J. Katz, M. H. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. *Computational Geometry*, 2(4):223–234, 1992.

[23] T. Koch and M. Wimmer. Guided visibility sampling++. *Proc. ACM Comput. Graph. Interact. Tech.*, 4(1), apr 2021. doi: 10.1145/3451266

[24] N. Max and K. Ohsaki. Rendering trees from precomputed z-buffer views. In *Rendering Techniques' 95*, pp. 74–81. Springer, 1995.

[25] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 39–46. ACM, 1995.

[26] S. Nirenstein and E. H. Blake. Hardware accelerated visibility preprocessing using adaptive sampling. *Rendering Techniques*, 2004:15th, 2004.

[27] R. Overbeck, R. Ramamoorthi, and W. R. Mark. A real-time beam tracer with application to exact soft shadows. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pp. 85–98. Eurographics Association, 2007.

[28] V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, and L. Nyland. The warpengine: An architecture for the post-polygonal age. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 433–442. ACM Press/Addison-Wesley Publishing Co., 2000.

[29] J. Shade, S. Gortler, L.-w. He, and R. Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 231–242. ACM, 1998.

[30] M. Sharir and M. H. Overmars. A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics (TOG)*, 11(1):1–11, 1992.

[31] K. Weiler and P. Atherton. Hidden surface removal using polygon area sorting. *ACM SIGGRAPH Computer Graphics*, 11(2):214–222, 1977.

[32] P. Wonka, M. Wimmer, K. Zhou, S. Maierhofer, G. Hesina, and A. Reshetov. Guided visibility sampling. *ACM Transactions on Graphics (TOG)*, 25(3):494–502, 2006.

[33] C. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, eds., *Handbook of discrete and computational geometry*, chap. 41, pp. 927–952. CRC Press, Boca Raton, FL, second ed., 2004.

[34] J. Yu and L. McMillan. General linear cameras. In *Computer Vision-ECCV 2004*, pp. 14–27. Springer, 2004.

[35] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 77–88. ACM Press/Addison-Wesley Publishing Co., 1997.

[36] Y. Zhou, L. Wu, R. Ramamoorthi, and L.-Q. Yan. Vectorization for fast, analytic, and differentiable visibility. *ACM Trans. Graph.*, 40(3), jul 2021. doi: 10.1145/3452097