Fast Diffraction Pathfinding for Dynamic Sound Propagation

CARL SCHISSLER, GREGOR MÜCKL, and PAUL CALAMIA, Facebook Reality Labs Research, USA

In the context of geometric acoustic simulation, one of the more perceptually important yet difficult to simulate acoustic effects is diffraction, a phenomenon that allows sound to propagate around obstructions and corners. A significant bottleneck in real-time simulation of diffraction is the enumeration of high-order diffraction propagation paths in scenes with complex geometry (e.g. highly tessellated surfaces). To this end, we present a dynamic geometric diffraction approach that consists of an extensive mesh preprocessing pipeline and complementary runtime algorithm. The preprocessing module identifies a small subset of edges that are important for diffraction using a novel silhouette edge detection heuristic. It also extends these edges with planar diffraction geometry and precomputes a graph data structure encoding the visibility between the edges. The runtime module uses bidirectional path tracing against the diffraction geometry to probabilistically explore potential paths between sources and listeners, then evaluates the intensities for these paths using the Uniform Theory of Diffraction. It uses the edge visibility graph and the A* pathfinding algorithm to robustly and efficiently find additional high-order diffraction paths. We demonstrate how this technique can simulate 10th-order diffraction up to 568 times faster than the previous state of the art, and can efficiently handle large scenes with both high geometric complexity and high numbers of sources.

CCS Concepts: • **Computing methodologies** → *Real-time simulation*; **Ray tracing**; *Mesh geometry models.*

Additional Key Words and Phrases: diffraction, geometry, acoustics, sound propagation

ACM Reference Format:
Carl Schissler, Gregor Mückl, and Paul Calamia. 2021. Fast Diffraction Pathfinding for Dynamic Sound Propagation. *ACM Trans. Graph.* 40, 4, Article 138 (August 2021), 21 pages. https://doi.org/10.1145/3450626.3459751

1 INTRODUCTION

In order to generate a convincing simulation of reality, the senses must be provided with plausible recreations of the real world. For virtual reality (VR) and augmented reality (AR) applications, high-quality audio is especially important to create immersion and a sense of presence [Hendrix and Barfield 1996]. Audio sources must be rendered in a way that seems plausible to the user, i.e. where the percept matches the user's expectation [Lindau and Weinzierl 2012]. This involves simulating how the sounds emitted by sources interact with the virtual environment through reverberation, reflections, and diffraction, among other acoustic phenomena. To create a spatial percept for virtual audio, directional filtering with the user's head-related transfer function (HRTF) must also be applied. In AR, the environment is real rather than virtual, meaning that any divergence between the simulation and the user's immediate surroundings will interfere with the plausibility of virtual sound sources [Werner et al. 2016].

One of the more difficult to simulate yet perceptually important acoustic effects in a geometric acoustics (GA) framework is diffraction [Torres et al. 2001]. Diffraction is a wave scattering phenomenon that occurs when sound interacts with a feature in the environment whose size is similar to the wavelength. With proper simulation of diffraction, sources that become occluded from view are still audible but become low-pass filtered. In a GA simulation that does not handle diffraction, occluded sources will be abruptly silenced, resulting in a jarring unnatural transition that has the potential to break the perceived plausibility of the auralization.

In this work, we present a new efficient approach for simulating diffraction within a real-time GA framework that allows for dynamic motion of rigid geometry and for high-order diffraction (i.e. diffraction over a series of several edges). In contrast to much of the previous work on diffraction, we focus on a particular subset of the diffraction problem: the simulation of *direct* diffraction only. Direct diffraction is defined as diffraction that occurs directly between a source and listener with no reflections involved. With this restriction, our approach is still able to simulate perceptually-important features, particularly the smooth transition from unoccluded to occluded state, while ignoring more complex paths which can be difficult to identify but contribute less to the overall sound field. Our approach is intended for VR and AR applications with little compute available for acoustic simulation, and as a result it puts more focus on runtime performance and perceptual quality than on objective physical accuracy. The main contributions include:

(1) A mesh preprocessing approach that extracts a reduced subset of silhouette diffraction edges and augments the mesh with diffraction flag geometry. (Section 4)
(2) A runtime approach for efficiently finding high-order diffraction paths using stochastic bidirectional path tracing and a persistent cache of paths. (Section 5)
(3) A complementary approach that uses a precomputed edge-to-edge visibility graph and the A* algorithm to quickly find high-order diffraction paths. (Section 5.4)

We have evaluated this diffraction approach in a variety of complex virtual and real environments that are typical of games, VR, and AR. Our technique is able to simulate diffraction using 0.14ms – 7.5ms per source, and the preprocessing time is less than a minute for scenes with millions of triangles. Compared to previous methods, the runtime is 2.7 to 568 times faster per source for 10th-order diffraction. Our approach is also significantly more robust to difficult geometric input and scales better to high diffraction order.

2 BACKGROUND

2.1 Sound Propagation

There exist many techniques to simulate the propagation of sound. The most accurate are based on solving the acoustic wave equation,

Authors' address: Carl Schissler, carl.schissler@fb.com; Gregor Mückl; Paul Calamia, pcalamia@fb.com, Facebook Reality Labs Research, Redmond, WA, USA.

0730-0301/2021/8-ART138
https://doi.org/10.1145/3450626.3459751

i.e. the so-called *wave-based* methods. These include the Finite Difference Time Domain (FDTD) method, Finite Element Method (FEM), and the Boundary Element Method (BEM). While accurate, these approaches are generally very computationally intensive and as a result are limited to precomputation and static scenes [Raghuvanshi and Snyder 2014; Raghuvanshi et al. 2017]. The other main class of methods are the *geometric acoustics* (GA) algorithms [Savioja and Svensson 2015]. These algorithms can simulate reflection, scattering, and reverberation efficiently, but do not handle wave phenomena like diffraction because they make the high-frequency assumption that sound travels as a ray, not a wave. Despite this drawback, GA methods are the most practical for applications where the environment may be dynamic such as with destructible geometry, user-generated content, and moving geometry (e.g. doors).

2.2 Diffraction for Geometric Acoustics

Diffraction models that are applicable to GA generally consider the case of diffraction over one or more edges of the scene geometry, where the number of edges in a path is the *diffraction order*. Given a source, listener, and sequence of diffraction edges, the Uniform Theory of Diffraction (UTD) [Kouyoumjian and Pathak 1974; Tsingos et al. 2001] can analytically compute an approximation for the diffracted sound field. UTD is attractive for real-time simulation because it is fast to evaluate, but has the drawback that it assumes every edge to be infinitely long. This can cause UTD to produce implausible results. On the other hand, the Biot-Tolstoy-Medwin (BTM) diffraction method [Svensson et al. 1999; Calamia and Svensson 2005] does not have this limitation, but requires significantly more compute to evaluate.

An alterative approach is diffraction based on the uncertainty principle (UP) [Stephenson 2010; Pohl 2014], which uses stochastic ray tracing in a Monte Carlo integrator to compute the diffracted sound field. UP is attractive because unlike UTD or BTM, it can be easily integrated into existing path tracing algorithms. However, it also has slow convergence that makes it unsuitable for real-time applications. Other object-based diffraction approaches have used a hybrid of precomputed wave simulation and ray tracing to simulate sound scattering around objects [Yeh et al. 2013; Rungta et al. 2018]. Cowan et al. [2015] proposed an ad-hoc 2D rasterization-based method for approximating occlusion that compares the diffracted path length to the straight-line distance between the source and listener. Recently, Pisha et al. [2020] described a diffraction model that uses a dense volumetric sampling of rays around existing direct and reflected propagation path segments to approximate the BTM magnitude response.

For either UTD or BTM, the main computational challenge is actually to find sequences of edges that can form valid diffraction paths.ive The number of edges in a particular path is the diffraction order. High-order diffraction (i.e. more than order 1) involves considering the interaction of every edge with every other edge recursively, and is especially important when the source is in another room than the listener. A naïve approach is to recursively consider all pairs of edges, but this has complexity $O(N^d)$, where N is the number of edges and d is the maximum diffraction order.

Frustum and beam tracing have been used to find diffraction paths more efficiently [Funkhouser et al. 2004; Chandak et al. 2008]. In these methods, frusta or beams are emitted from the source and propagated through the environment to find paths. However, these approaches have difficulty scaling to complex geometries or to high diffraction order due to the large number of child beams and time spent on intersection testing. On the other hand, Schissler et al. [2014] used ray tracing from sources to detect first-order diffraction edges, followed by a traversal of a precomputed edge-to-edge visibility graph to find all diffraction paths originating at those edges. This graph reduced the number of edges considered and allowed computation of diffraction up to order 3 or 4 in real time, but retains exponential algorithmic complexity.

2.3 Mesh Simplification for Acoustics

Compared to graphics rendering, room acoustic simulation is quite tolerant to aggressive geometry simplification, provided that properties of the environment such as volume and surface area are preserved. This is in part due to the long wavelengths of low-frequency sound, low spatial resolution of spatial audio as well as the diffuse nature of late reverberation. Simplification also tends to increase the size of planar surfaces relative to the wavelength, which may improve accuracy for GA [Savioja and Svensson 2015].

Joslin and Magnenat-Thalmann [2003] proposed a method to extract significant faces using a regular grid subdivision of the scene followed by clustering and bounding box fitting to approximate the input geometry. This method is able to drastically reduce the number of faces in a mesh but it does not preserve details, topology, or scene volume. On the other hand, Siltanen et al. [2008] used a remeshing approach to first voxelize the scene and then extract an isosurface. This isosurface extraction was followed by coplanar face merging to reduce the number of faces, and post-processing to patch cracks in the mesh surface.

Schissler et al. [2014] used a similar remeshing approach to simplify the mesh, except that the edge collapse algorithm [Garland and Heckbert 1997] was used instead of coplanar face merging. This method also generated different geometry for each simulation wavelength using proportionally-sized voxel grids, and applied a parallel edge merging step to reduce the number of edges considered for diffraction. Similarly, Pelzer and Vorländer [2010] performed frequency-dependent simplification and used meshes with varying level of detail to speed up real-time simulation. They also proposed using *time-dependent* geometry, where lower levels of detail would be used for higher-order (i.e. later) reflections. To reduce the number of edges considered for diffraction, [Taylor et al. 2009] compared the angle between adjacent faces to a threshold as a way to select significant diffraction edges.

3 OVERVIEW

Our diffraction approach can be divided into preprocessing and runtime stages. In the preprocessing stage, we apply a series of operations to first simplify input meshes, then identify important silhouette diffraction edges using a ray-based heuristic. We augment the edges with additional diffraction geometry and also precompute a visibility graph between the edges that is used to accelerate the

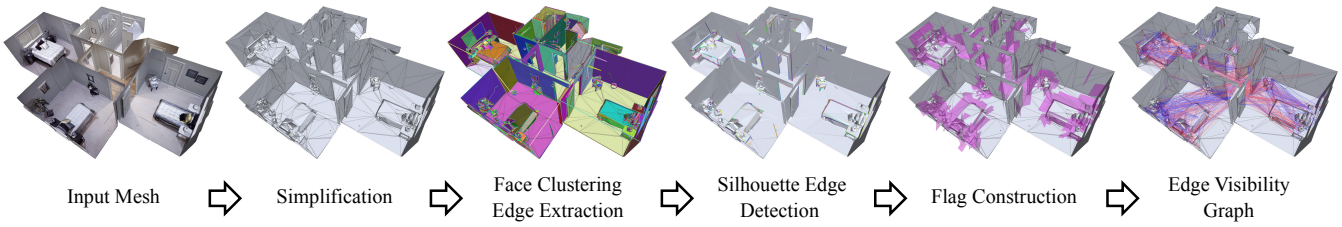

Fig. 1. The main stages of our diffraction mesh preprocessing pipeline, shown here for the top floor of the Apartment 0 scene [Straub et al. 2019]. The detailed input mesh is first simplified, then an initial set of diffraction edges is extracted by face clustering. After this, we keep only the edges that are classified as silhouettes, then construct diffraction flag geometry from the final set of edges. The final step is to precompute a visibility graph between the edges, shown here by red or blue lines between the edges.

runtime exploration of high-order diffraction paths. In the runtime stage, we use bidirectional stochastic ray tracing to explore possible diffraction paths, then validate those paths using robust visibility tests. The resulting paths are stored in a persistent cache to add temporal stability over successive simulation updates. We also utilize the precomputed edge visibility graph and the A* pathfinding algorithm [Hart et al. 1968] to efficiently and robustly find additional high-order diffraction paths. The output of the runtime simulation module is a collection of frequency-dependent room impulse response parameters (e.g. diffraction path intensities/directions, reverberation time, reverberation level, etc.). These parameters are the input to the audio rendering module which uses standard signal processing techniques to auralize the impulse response parameters. The final output audio is spatialized using the listener's head-related transfer function and then reproduced over headphones.

4 DIFFRACTION MESH PREPROCESSING

The four stages of our mesh preprocessing pipeline are summarized in Figure 1. The input to the pipeline is the raw triangle mesh with acoustic materials assigned to each triangle. In the first stage, we apply standard mesh simplification algorithms to reduce the input mesh complexity to a given error tolerance (Section 4.1). Next, we identify a subset of the edges of the mesh that are relevant for diffraction using a novel silhouette edge identification method and apply additional post-processing to simplify and cluster the selected edges (Section 4.2). The output of this stage is a collection of *mesh boundaries*, i.e. sequences of edges that make up the same logical diffraction edge. An important part of our approach is to decouple the diffraction edges from the underlying surface geometry to reduce the number of edges considered for diffraction at runtime. In the third stage, we augment the simplified mesh with additional diffraction geometry (flags) that bisect the outside angle of each boundary (Section 4.3). These flags are used at runtime to detect when a ray passes nearby a diffraction edge, similar to the Uncertainty Principle method [Stephenson 2010]. The final optional stage of the pipeline involves precomputing a graph of edge-to-edge visibility that can be used to accelerate pathfinding during the runtime algorithm (Section 4.4). In the case of moving geometry, we apply this pipeline separately to each rigid part of the scene.

4.1 Mesh Simplification

In the simplification stage of the preprocess, we apply standard mesh simplification approaches to reduce the overall geometric complexity. This has many benefits downstream in the other sections of our approach. By reducing the number of triangles, the number of edges that must be considered in the other preprocessing stages is also reduced. An additional benefit is that after simplification some geometric connectivity problems may be corrected (e.g. duplicated vertices or edges). Simplification also improves the consistency of the local mesh curvature and this helps with correct identification of diffraction edges in Section 4.2.

Similar to Schissler et al. [2014], we make use of vertex welding and edge collapse operations. However, we forgo the use of voxelization and marching cubes due to the artifacts that can be introduced when using large voxels, as well as their tendency to actually *increase* the number of diffraction edges by beveling sharp corners after surface reconstruction.

Vertex welding is applied by greedily clustering each vertex with its neighbors within a certain tolerance distance $\epsilon_{weld} = 0.001$m. We use a spatial hashing approach to implement this with $O(N)$ time complexity [Hradek et al. 2003]. Our implementation of the edge collapse algorithm is similar to the standard approach [Garland and Heckbert 1997], but with a few extensions. First, we limit the maximum amount of error that can be introduced in triangle normals to $\epsilon_n = 10°$, while the original algorithm only prevents flipping of triangles ($\epsilon_n = 90°$). This helps to preserve the overall shape of the mesh better, particularly at the silhouette edges and mesh corners which are important for diffraction. We also prevent simplification across acoustic material boundaries.

4.2 Diffraction Edge Extraction

4.2.1 Initial Edge Selection. The first step in the edge extraction pipeline is to determine which edges in a mesh are relevant for diffraction. If too few edges are selected, it can cause some diffraction paths to be missed, resulting in abrupt occlusion. It is also important to use only the edges that can produce significant diffraction in order to get the best performance, but identifying those edges is a non-trivial task.

In general, edges that are between two faces with similar normals are unlikely to produce any significant diffraction. Due to this, previous GA approaches have used metrics like the dihedral angle to

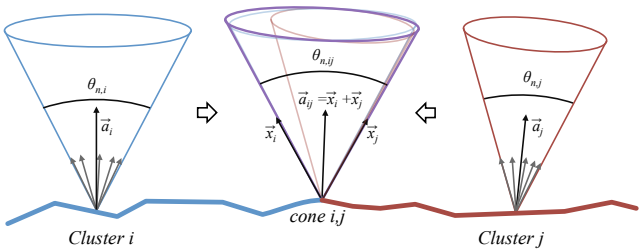

Fig. 2. The cone merging process used to determine if two clusters i and j can be combined. The opening angle, $\theta_{\vec{n},ij}$, of the merged cone is compared to a threshold to determine if i and j should be merged.

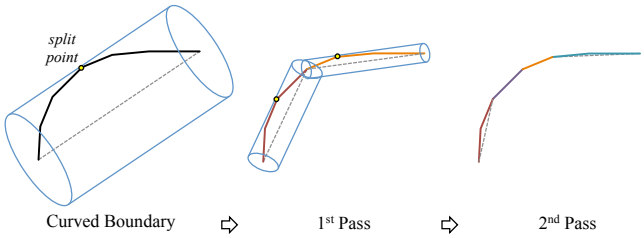

Fig. 3. An illustration of the recursive curve splitting process described in Section 4.2.2. Curved boundaries are subdivided until the aspect ratio of the bounding cylinder drops below a threshold. The final approximately-collinear boundaries are treated as individual proxy diffraction edges.

classify edges as diffracting [Taylor et al. 2009; Schissler et al. 2014]. In this approach, the angle between adjacent face normals, θ_s, is compared to a threshold angle, ϵ_θ, to determine if the shared edge is a diffraction edge. If θ_s is greater than ϵ_θ, that edge is classified as a diffraction edge. However, this can greatly overestimate the number of diffraction edges for highly-tessellated curved surfaces because it uses only local information. This is especially problematic for highly tessellated meshes or curved surfaces that have θ_s close to 0. In such meshes, the value of ϵ_θ must be close to 0 to find all relevant edges. However, this causes many extraneous edges to also be selected, resulting in poor runtime performance. In general, it is difficult to find a value of ϵ_θ that works robustly for all inputs.

To address this limitation, we propose a novel diffraction edge identification method that is based on face normal clustering. We adapt the Felzenszwalb graph segmentation algorithm [Felzenszwalb and Huttenlocher 2004] to this task, where in this case the graph is the face adjacency graph. The purpose of this algorithm is to cluster adjacent faces that have similar surface normals. The boundaries between the face clusters are then used as the initial set of diffraction edges. Compared to existing approaches based on local curvature, ours works well on meshes with any level of tessellation.

The algorithm begins by computing the weight for each edge between adjacent faces in the face adjacency graph. We propose using the cosine of the angle between each pair of adjacent face normals, $w(f_i, f_j) = \cos(\theta_s) = \vec{n}_i \cdot \vec{n}_j$. Next, these weights are sorted in decreasing order, such that face pairs that have more similar normals come first. Each face in the mesh is initially assigned to its own unique cluster, where each cluster maintains information about the distribution of surface normals for faces that belong to the cluster. We represent this normal distribution using a cone where the axial direction $\vec{a}$ approximates the average normal and the opening angle $\theta_{\vec{n}}$ approximates the spread of normals within the cluster. The algorithm proceeds by inspecting each face pair in order of decreasing weight and evaluating whether or not the clusters that the faces belong to can be merged.

To determine if merging two clusters is possible, we first compute the merged normal cone for the two clusters, i.e. the smallest cone that contains both merged cones. Given two cones i and j, this can be efficiently approximated by first computing the vector $\vec{x}_i$ on the boundary of cone i that has the greatest angle with the axis of cone j, and vice versa to yield $\vec{x}_j$. The average of the two extreme vectors is then used as the merged cone axis and the angle between them is used as the opening angle of the cone, $\theta_{\vec{n}}$. This process is shown in Figure 2. The clusters are then merged if $\theta_{\vec{n}}$ is less than the merging threshold for either cluster. The merging threshold is maintained separately for each cluster and is initially set to $\tau = \epsilon_\theta$ at the beginning of the algorithm, where ϵ_θ is the minimum dihedral angle to consider for diffraction. After two clusters are merged, the resulting cluster's threshold is increased according to the following relation:

$$\tau(F_i \cup F_j) = \theta_{\vec{n}} + \frac{k\epsilon_\theta}{|F_i| + |F_j|} \tag{1}$$

where $|F_i|$ and $|F_j|$ represent the number of faces in the constituent clusters, and where $k = 4$ is a parameter that controls the scale of the clusters. This has the effect of requiring stronger evidence for a boundary between small clusters.

The final step of the face clustering algorithm is to merge clusters smaller than a certain threshold with adjacent larger clusters. This is necessary in the case of noisy mesh data such as that from 3D reconstructions where there may be occasional small clusters that are not included in the cluster for a large flat wall. For each cluster that is considered too small (e.g. area less than 0.1m^2), we merge it into the neighboring cluster that has the most similar average surface normal.

Once the face clusters are computed, we extract the boundaries between the clusters as the initial set of diffraction edges. Each boundary is a set of edges that have adjacent faces belonging to the same 2 clusters. These boundaries are then provided to the next stage of the preprocessing pipeline.

4.2.2 Curved Boundary Splitting. Since the face clusters in the previous step can have any shape, there is no restriction on the collinearity of the edges that make up a cluster boundary. This is in conflict with our final goal of turning each boundary into a single straight diffraction edge. To handle this problem, we propose a simple approach for splitting mesh boundaries into collinear segments.

For each boundary, we first calculate a bounding cylinder of the vertices, where the axis of the cylinder represents the dominant direction of the boundary and the radius is a measure of how collinear the vertices are. The cylinder's axis is defined by the two vertices in the boundary that are farthest apart, while the radius of the cylinder is given by the maximum distance of a boundary vertex from the axis line segment. This cylinder is used to determine whether or not a boundary should be split into more than one boundary.

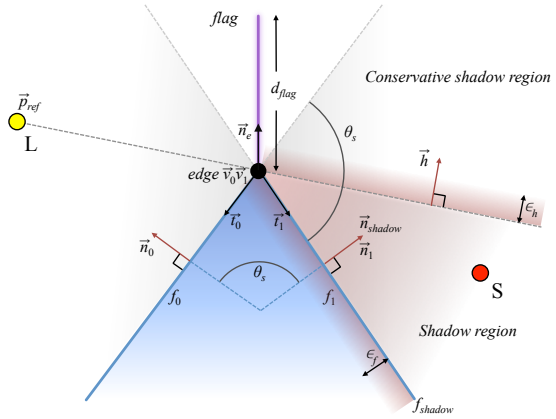

Fig. 4. A cross-section view of the geometry for a single diffraction edge. The edge axis is perpendicular to the image. The edge is shared by faces f_0 and f_1 that have face normals $\vec{n}_0$ and $\vec{n}_1$. The exterior angle between f_0 and f_1 is bisected by edge normal $\vec{n}_e$ and a planar diffraction flag that extends a distance d_{flag} from the edge. The grey-shaded areas are the *conservative shadow regions* that are defined by the planes of f_0 and f_1. The red-shaded area is the *shadow region* for a particular listener position. It is bounded by the *horizon plane* and plane of face f_{shadow}. The horizon plane, with normal vector $\vec{h}$, is defined by the edge vertices and the reference point $\vec{p}_{ref}$, which corresponds to a source, listener, or point on a previous diffraction edge. Also visible are the epsilons ϵ_h and ϵ_f that extend the shadow region on the horizon and face sides. These extra tolerances enable smoother transitions between direct and diffracted state (see Section 5.2.2).

We propose that a boundary should be split if the aspect ratio of its bounding cylinder $\left(\frac{2r}{h}\right)$ is more than a certain threshold, e.g. 0.025. The splitting point is chosen to be the boundary vertex that is farthest from the cylinder's axis, while the edges that are on either side of the split are placed into one of the two resulting boundaries. These new boundaries are then recursively split until their aspect ratio falls below the threshold. This process is illustrated in Figure 3. The output of this stage is a collection of mesh boundaries that are known to be approximately straight.

4.2.3 Silhouette Edges. Similar to previous GA diffraction methods [Tsingos et al. 2001; Schissler et al. 2014], we only consider the diffraction that occurs in the shadow region of an edge. This means that the only edges that can produce diffraction are *silhouette* edges, i.e. those edges that can cast a "shadow" when illuminated from a point in the conservative shadow region (see Figure 4).

In this section we present a novel approach to reliably identify these silhouette edges. Our approach utilizes global information about the structure of the mesh acquired through stochastic ray tracing from points on an edge to determine whether or not a given edge is a silhouette. Our approach is based on the observation that in order for an edge to contribute to diffraction, it must be able to cast a shadow, and that a source or listener with non-zero size must be able to go into the conservative shadow region on *both sides* of the edge. The main idea is that if there are other parts of the mesh that completely obstruct one or both sides of a given edge such that

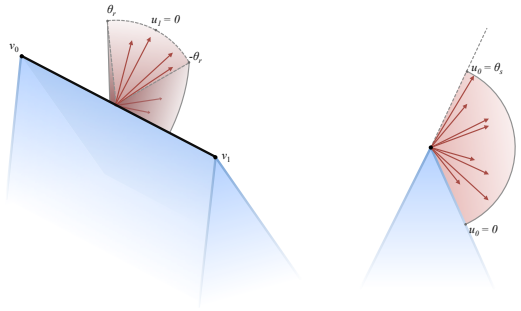

Fig. 5. The probability density function used to generate rays in the silhouette edge detection algorithm (Section 4.2.3) and diffraction visibility graph computation (Section 4.4).

no sound source or listener can form a shadowed path over the edge, that edge cannot produce any diffraction paths.

This information can be determined approximately by stochastic ray tracing in the conservative shadow regions (CSR) of each edge (see Figure 4). We emit rays that randomly sample the CSR from uniformly-sampled random points on the edge. The number of rays traced for an edge, N_{samp}, is determined by its length and the angular size of the CSR:

$$N_{samp} = \min\left(N_{samp}^{max}, \frac{\|\vec{v}_1 - \vec{v}_0\|_2}{h_d}\frac{\theta_s}{h_\theta}\right) \tag{2}$$

where $\theta_s = \cos^{-1}(\vec{n}_0 \cdot \vec{n}_1)$ is the angular size of the CSR, $h_d = 0.1$m is the distance sampling resolution, and $h_\theta = 5^\circ$ is the angular sampling resolution. In practice, N_{samp} is limited to a reasonable maximum value, e.g. 10^4, to prevent spending too much time on very long edges.

To sample the outgoing ray direction, we use a uniform spherical distribution that has been modified to generate rays in a wedge shape, as shown in Figure 5. The outgoing ray direction in the local tangent space is given by:

$$\vec{r}_d = \left(u_0, \sqrt{1-u_0^2}\sin u_1, \sqrt{1-u_0^2}\cos u_1\right) \tag{3}$$

where u_0 is a uniform random variable in the range $[0, \theta_s)$, and u_1 is a uniform random variable in the range $[-\sin\theta_r, \sin\theta_r]$. The parameter $\theta_r = 30^\circ$ controls the amount of spreading of the rays in the direction of the edge axis. For example, $\theta_r = 0$ would generate rays that are always perpendicular to the edge. Once the local ray direction is generated, it is rotated to mesh space by applying the orthonormal rotation matrix $\mathbf{R}_i = \left[\frac{\vec{v}_1 - \vec{v}_0}{\|\vec{v}_1 - \vec{v}_0\|_2}, \vec{n}_i, \vec{t}_i\right]$, where i is the face index.

In our approach, we classify an edge as silhouette if both sides of the CSR have at least $N_{valid} = 1$ rays that don't hit anything within a certain distance, $\tilde{d}_s$. We use this information as a proxy for whether or not a source or listener can be occluded by the edge. The distance $\tilde{d}_s$ is proportional to the diameter of a source or listener, d_s. In other words, as a source or listener grows bigger, an edge must protrude further from the nearby geometry to produce any diffraction. d_s is a parameter of our algorithm that controls how

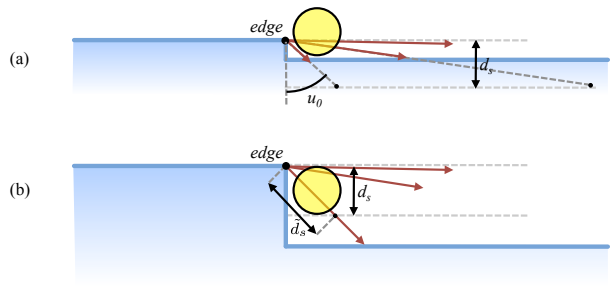

Fig. 6. An illustration of the silhouette edge heuristic for two edges. The yellow circle represents a sound source or listener with diameter d_s, and the red rays represent random rays generated according to Equation 3. Edge (a) is not a silhouette edge because the circle cannot be placed where it is completely occluded by the edge. This is because all of the rays hit another face before traveling distance $\tilde{d}_s$. Edge (b) however is classified as a silhouette edge because at least N_{valid} rays were able to travel for a least distance $\tilde{d}_s$ and because the circle can be occluded by the edge. Note that both edges pass this heuristic for the other side of the edge, but only (b) passes on both sides.

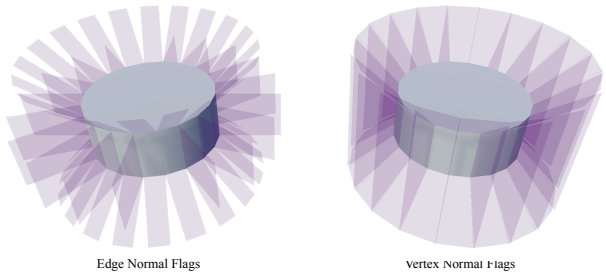

Fig. 7. The difference between diffraction flags generated using vertex normals versus flags generated using edge normals. Vertex normals produce flags without gaps, thereby increasing the probability of ray-flag intersections for locally-convex geometry.

aggressive the silhouette edge detection is. We use $d_s = 0.25$m, which roughly corresponds to the size of a human head. In Figure 6 we show an example of this heuristic for two different edges.

In order for our approach to work correctly, the value of $\tilde{d}_s$ must increase for rays that are closer to the CSR boundary. If $\tilde{d}_s$ did not increase for rays near the CSR boundary, edge (a) in Figure 6 would be erroneously classified as a silhouette edge. To address this, we calculate the value of $\tilde{d}_s$ for each ray using the following relation:

$$\tilde{d}_s = \frac{d_s}{\max(\cos(u_0), \epsilon)}. \tag{4}$$

This causes the threshold distance to increase substantially for rays that have large u_0.

To summarize, our silhouette detection algorithm inspects every input edge and traces rays to determine if that edge is a silhouette. If at least N_{valid} rays on both sides of an edge are able to travel a distance of at least $\tilde{d}_s$ before hitting other geometry, then that edge is classified as a silhouette.

4.3 Diffraction Geometry Construction

In this stage of the preprocessing pipeline, we take the final set of silhouette mesh boundaries produced in the previous stage and construct additional diffraction geometry that is used at runtime to detect when a ray passes near an edge. This idea is inspired by the Uncertainty Principle (UP) diffraction approach [Stephenson 2004, 2010]. Stephenson et al. proposed augmenting the main geometry with so-called *diffraction flags* - quadrilaterals that bisect the outside angle of diffraction edges and protrude a distance proportional to the wavelength of the lowest frequency band, e.g. $d_{flag} = 6\lambda$.

We construct a similar set of diffraction flags but do not require any particular flag length, d_{flag}, because the accuracy of our diffraction approach does not depend on the flag length due to the use of the analytical UTD diffraction model. On the other hand, changing the flag length changes the diffracted sound intensity for UP because in that model the flag is an integral domain. In our approach, the length of the diffraction flags controls how likely it is for a ray to intersect a flag and find diffraction paths over the associated edge. We use $d_{flag} = 1.0$ m as a reasonable tradeoff between finding enough diffraction paths and spending too much time on ray-versus-flag intersection tests for rays as they traverse the scene. For example, with the UP approach, a simulation with lowest frequency band of 63Hz would require flags of length 33 m. In complex scenes, this becomes problematic for performance because of numerous overlapping flags.

The first step of this stage is to convert the input mesh boundaries, each made up of one or more edges, into singular diffraction edges that act as proxies for the underlying surface geometry. In this way, we decouple the surface geometry representation from the edges used to compute diffraction effects. Each roughly collinear mesh boundary is approximated with a single straight edge. We apply the approach discussed in Section 4.2.2 a second time to compute the best-fitting proxy edge for a mesh boundary. Another issue is the calculation of the local geometric information needed for diffraction, namely the adjacent face normals of the proxy edge. To do this, we compute the area-weighted average of the face normals on each side of the boundary after assigning each adjacent face to one side or another based on the similarity of its normal vector to the faces processed so far.

The second task of this stage is to determine where to place the two far vertices for each flag. The simplest approach is to place the far vertices at distance d_{flag} from each edge vertex in the direction of the edge normal, e.g. $\vec{v}_i + \vec{n}_e d_{flag}$. This works well in many cases, but can fail with certain meshes. Consider the diffraction edges at the top ring of a tessellated cylinder, as shown in Figure 7. Placing the far vertices along the edge normal produces many gaps in the ring of flags that may reduce the effectiveness of the runtime diffraction algorithm. To remedy this, we propose using the vertex normals rather than the edge normal to determine the far vertex locations, e.g. $\vec{v}_i + \vec{n}_{v_i} d_{flag}$. The exception is that if both vertex normals point toward the center of the edge, we use the edge normal instead. This tends to use vertex normals on the convex parts of the mesh and edge normals on the concave parts.

To support intersecting rays against either the surface mesh or the flags, we put the additional flag geometry in a separate mesh and

acceleration structure with the same transformation as the surface mesh. A bitmask is then used by the ray tracer to select what type(s) of geometry each ray should intersect with. This is required because flags should not interfere with next event estimation in the path tracer or line-of-sight checks in the runtime diffraction algorithm.

4.4 Diffraction Graph

The last stage of our preprocessing pipeline is to build a separate directed edge-to-edge visibility graph between the final set of diffraction edges for each rigid mesh in the scene. It is used in the runtime graph traversal algorithm to speed up the search for diffraction paths (Section 5.4). The data structure itself is similar to the visibility graph from [Schissler et al. 2014], i.e. a flat array of edge neighbor indices, but we generate the graph in a different way that scales better to complex scenes with many edges. The graph computation algorithm from [Schissler et al. 2014] scales poorly because it considers all pairs of diffraction edges in the mesh, i.e. it is an $O(N^2)$ algorithm, while in practice most edges can only diffract with a few neighbors. In addition, their approach only traced a single ray between the midpoints of each pair of edges. This may have caused some edge pairs to be erroneously discarded because it did not consider partial visibility.

In contrast, our graph generation algorithm handles approximate partial visibility and has $O(N)$ time complexity. We utilize the diffraction flag geometry from Section 4.3 along with stochastic ray tracing to determine whether or not edges are mutually visible. For each edge e_i in the mesh, we emit random rays in the conservative shadow regions according to the same distribution used to generate silhouette rays (Section 4.2.3), but with $\theta_r = 60^\circ$. These rays are then intersected with the surface mesh to find the ray endpoint at distance d_{max}. Then, the same ray is intersected with the diffraction flags to find all hits along the ray up to distance d_{max}. For each flag intersection, we check to see if the associated edge, e_j, is in the CSR of the edge e_i that emitted the rays. This condition is met when the signed distance of an edge endpoint to the face planes of the other edge is more than $-\epsilon_f$ for one plane and less than ϵ_f for the other. This is similar to the culling test proposed in [Schissler et al. 2014], but with additional tolerance ϵ_f that prevents edge pairs that share a face plane from being discarded. If this succeeds, a directed link is added to the graph from edge e_i to edge e_j.

Another improvement we make to the graph data structure is to partition the links originating from a given edge into two sets corresponding to the two sides (i.e. CSR) of the edge that generated the connections. By partitioning the outgoing links in this way, the graph search algorithm in Section 5.4 can be sped up by about a factor of 2. For example, if the current position of the diffraction graph search is on one side of the edge, it only has to explore neighboring edges that were visible to the *far* side of the edge because the other edges would not be able to form valid diffraction paths. An example diffraction graph is shown in Figure 1, where we color the connections between edges with either red or blue to show how the outgoing connections are partitioned for each diffraction edge.

5 DIFFRACTION RUNTIME

The runtime part of our diffraction approach considers the problem of finding *direct* diffraction paths between every source and listener in the scene each time the simulation is updated. Our approach is based on the idea of intersecting rays with additional diffraction flag geometry, originally proposed for the UP diffraction method [Stephenson 2004]. However, in contrast to the UP approach, we don't rely on stochastic ray tracing to directly calculate the diffraction path intensity. Rather, we use a UP-like approach to find sequences of diffraction edges in a random ray traversal, but instead use the UTD diffraction model to analytically compute the path intensity. We also maintain a persistent cache of these paths over the course of the simulation to improve the temporal coherence (Section 5.3), and propose a graph traversal algorithm to find high-order paths more quickly (Section 5.4).

Our approach has several advantages over previous methods. First, like UP, its time complexity does not scale exponentially with the maximum diffraction order and it can be easily integrated into existing acoustic path tracers. This enables very high-order diffraction (e.g. order 10) to be calculated with good performance, even in scenes with high geometric detail. Second, unlike UP, the degree of convergence of the results does not depend on the number of rays traced. Third, our approach is able to efficiently handle diffraction between multiple dynamic objects.

5.1 Ray Tracing

At the core of our runtime system is a bidirectional path tracer (BDPT) with multiple importance sampling [Veach 1997; Georgiev 2012; Cao et al. 2016]. We use this to compute early reflections and to build an energy-decay histogram for the late reverb from which frequency-dependent reverberation times can be determined. Our diffraction approach is integrated within the path tracer and is similarly bidirectional, meaning it can find diffraction paths starting from either the listener or a source. This bidirectionality improves the likelihood of finding paths in certain geometric configurations where either source or listener is highly occluded.

In our path tracer, we consider diffraction only for subpaths originating at a source or listener that have not yet intersected any surfaces. This is consistent with our restriction to only *direct* diffraction. For these subpaths, we intersect the constituent rays with both surface geometry and diffraction flag geometry to find the nearest intersection. If the intersection is with a surface mesh, we reflect or transmit the ray according to the surface material and disable intersections with further diffraction flags. Otherwise, the ray hit a diffraction flag and remains a candidate for more diffraction events.

Since flags can stick through geometry, we trace an additional ray from the ray vs. flag intersection point toward its projection on the edge to verify that the edge is visible. If so, we try to find paths to sources or listeners in the scene that are in the *shadow region* of the edge. For each of these possible paths, we evaluate whether or not diffraction over the edge is valid, given the sequence of previous edges in the subpath. This is discussed in detail in Section 5.2. If a precomputed diffraction graph is available for the intersected mesh, we can also perform a deterministic graph search to find additional

high-order diffraction paths (Section 5.4). This is analogous to deterministic next event estimation in a path tracer. If the edge is not in a valid configuration to produce diffraction, the ray continues past the flag in its current direction without modification.

After any paths have been found for the current edge, we modify the outgoing ray direction to explore the scene further. One possible ray distribution is the diffraction probability density function (DAPDF) proposed for the UP diffraction model, however we found in practice that a simple lambertian distribution on the opposite side of the flag empirically finds more diffraction paths and is faster to sample. Once the ray is redirected, we modify the ray's frequency-dependent energy according to the DAPDF. This modification ensures that the outgoing ray has the correct diffracted energy for its direction, and also that further reflections of that subpath are influenced by the diffraction that occurred earlier in the path.

This repeats until a surface mesh is intersected or a maximum number of diffractions occur, at which point the further rays for the subpath are handled using standard BDPT.

5.2 Path Validation

5.2.1 Shadow Test. For diffraction to be possible, each diffraction edge in a subpath must intersect the *shadow region* of the previous edge, if one exists. The shadow region is defined as the intersection of the two half-spaces corresponding to the shadow face plane and the shadow horizon plane (see Figure 4). The shadow horizon plane is defined by the diffraction edge vertices $\vec{v}_0, \vec{v}_1$ and the reference point $\vec{p}_{ref}$, which for the first edge is the source or listener position. For diffraction beyond order 1, $\vec{p}_{ref}$ is the point on the previous edge that creates the largest (i.e. closest to conservative) shadow region. This can be determined by clipping the previous edge's line segment with the face planes of the current edge, so as to limit the previous edge segment to only the part in the current edge's CSR on the non-shadowed side. If the previous edge is completely outside of this region, diffraction cannot occur between the edges. Then, the clipped endpoint that creates the shadow region with greatest angle is chosen as $\vec{p}_{ref}$ and the horizon plane normal $\vec{h}$ is calculated as $\vec{h} = (\vec{v}_1 - \vec{v}_0) \times \left(\vec{p}_{ref} - \vec{v}_0\right)$. Finally, we can use a few dot products to check if the current edge intersects the shadow region for the previous edge. Please refer to the supplemental material for details. Once a potentially valid subpath is found, we can then check for connections to sources or listeners that are in the shadow region of the last edge using a similar shadow test.

5.2.2 Shadow Test Tolerances. We allow a tolerance of $\epsilon_h = 1.0m$ for the horizon plane and $\epsilon_f = 0.1m$ for the shadow face plane, as shown in Figure 4. The tolerance ϵ_f allows our approach to find diffraction paths between coplanar edges without numerical issues. It also avoids problems where the diffraction wedge geometry doesn't correspond exactly to the surface mesh. For instance, if the averaged face normals for a proxy edge (Section 4.3) are slightly wrong, the shadow test might reject otherwise valid diffraction paths. Introducing a face plane tolerance helps to avoid these geometric issues.

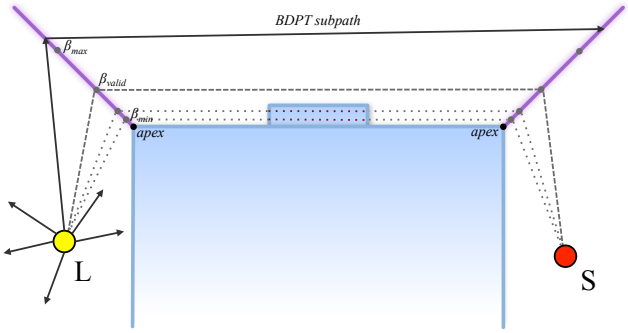

Fig. 8. An illustration of the soft visibility test for a 2nd-order diffraction path. For small values of β, the path is occluded by other nearby geometry that is not marked as diffracting due to the silhouette test. We increase β geometrically from β_{min} until either it surpasses β_{max}, or an unoccluded path is found.

The purpose of the large horizon plane tolerance ϵ_h is to *anticipate* diffraction paths before they are needed and enable smooth transitions between direct and diffracted sound. This is important when a source or listener moves from the region where direct sound is valid into the shadow region of an edge. Due to the random nature of the rays, it's possible that a ray may not immediately hit the flag for the edge, resulting in a temporary gap in the audio until the diffraction path is found. This phenomenon is more problematic with high-order diffraction because those paths are much less likely to be explored by random ray traversal. By anticipating diffraction paths that may soon become valid, those paths are more likely to be in the path cache when they are actually needed (i.e. when the direct sound becomes occluded). In the case where direct sound is unoccluded, these anticipated paths are not used for auralization.

5.2.3 Visibility Test. For each source or listener in the shadow region, we then check to see if that source or listener can form a valid path back to the listener or source that emitted the subpath. We first compute the apex points (points where diffraction occurs) on each edge using the Newton's method approach suggested by Tsingos et al. [2001]. An important detail is that we clamp the points to be on the edge's line segment. This is needed for robust diffraction around curved surfaces with many small edges. In such cases, the apex point often is not between the edges' endpoints, and rejecting these paths would make the diffraction significantly less robust.

Once the apex points are determined, we then trace a series of rays between the source, apex point(s), and listener to determine if the path is blocked by other geometry. We bias each apex point a variable distance β out from the edge along the edge normal $\vec{n}_e$ to prevent self-intersection of rays with neighboring faces, and also to implement a robust *soft* visibility test. The main idea of the soft visibility test is that if all rays in the path are unoccluded for some $\beta \in [\beta_{min}, \beta_{max}]$, then that path is considered valid. The procedure is to first set $\beta = \beta_{min}$ and then trace rays between all of the points along the path. If any rays are blocked, we then geometrically increase β by a factor of 2 and trace more rays between the new biased apex points. We repeat this until $\beta \geq \beta_{max}$. If no

β passed the visibility test, then the diffraction path is discarded. We use $\beta_{min} = 0.01m$ and $\beta_{max} = 1.0m$. In Figure 8, we show an example of where this approach helps find more diffraction paths, such as when edges that were not marked as diffracting occlude the rays between apex points.

After the visibility test is passed, we compute the frequency-dependent intensity of the diffraction path using the UTD diffraction model. We use the shadow boundary normalization scheme from [Tsingos et al. 2001] to ensure that the diffraction intensity matches the direct sound at the shadow boundary. Finally, we insert the path into the diffraction path cache.

5.3 Diffraction Path Cache

An important part of our diffraction approach is the so-called *diffraction cache*. The purpose of this cache is to reduce unnatural variation in the sound. The diffraction paths that are found on each simulation update may be different because different random rays are traced. This can lead to audible artifacts in real-time applications where the number of rays is small. To address this, we leverage the idea of a persistent cache of paths from [Schissler and Manocha 2011; Schissler 2017] and adapt it to diffraction.

The cache contains diffraction edge sequences from previous time steps that are known to be valid. The cache entries are stored in a hash map data structure accessed by an integer key that is generated from a hash of the source index, listener index, and edge indices for a path. At the beginning of each time step, the cache entries are revalidated using the approach from Section 5.2 and newly invalid entries are discarded. During ray tracing, as new valid paths are found, they are inserted into the cache. For explored paths that are known to be invalid, we also insert a special *invalid* entry in the cache to indicate that that edge sequence shouldn't be checked again this frame. We use the cache to avoid checking the same edge sequences for diffraction more than once on each frame.

In scenes with many edges, the number of paths that are in the cache can increase substantially. If the cache is too large, it can slow down revalidation of the cached paths on the next simulation update. For this reason, the cache also prioritizes the *valid* paths in the cache based on the intensity of the loudest frequency band. We use an additional min-heap data structure to dynamically rank the paths as they are found and discard all except the top N, where N is chosen to be proportional to the number of sources in the scene, e.g. $N = 20(\#sources)$. If a new path is found and it is quieter than the Nth quietest path, we don't add that path to the valid set (though we still mark that path as explored on this frame). This effectively enforces a maximum size for the set of valid paths in the cache globally for all sources/listeners. While not directly perceptually-motivated, this scheme produces an approximate kind of perceptual prioritization, where if the scene is complex, the quietest paths will be masked by the louder ones.

At the end of each time step, we inspect the contents of the cache and pick the loudest *single* diffraction path for each source/listener pair. The intensity and direction for this path is then used for the final audio rendering whenever the direct sound is occluded (i.e. the same interpolated delay line tap is used for direct sound and diffraction to ensure smoothness).

The main reason we restrict the rendered output to just one path is because it helps to overcome deficiencies with the UTD diffraction model. UTD assumes every edge is infinitely long and that the adjacent faces are also infinite. This assumption causes UTD to produce a total diffracted sound field that is much too loud in some geometric configurations, as shown by Figure 13. Since UTD considers each path to be over an infinite edge, the sum of diffraction contributions is not physically correct or plausible. By picking the single loudest (usually shortest) path, we get a diffracted sound field that is much closer to correct in these situations.

5.4 Diffraction Graph Traversal

While the approach proposed in the previous sections is viable alone, its robustness and performance can be greatly improved by the use of a precomputed edge-to-edge visibility graph similar to the one from [Schissler et al. 2014]. This graph increases the likelihood that we find valid high-order diffraction paths between sources and listeners that are separated by complex (e.g. curved) geometry, rather than relying on the random traversal of diffracted rays to find high-order paths. However, unlike [Schissler et al. 2014], we do not perform an exhaustive graph search each time a correctly-oriented diffraction flag is intersected in the path tracer. Instead, we apply the A* (A-star) algorithm from the agent navigation field [Hart et al. 1968] to find the shortest diffraction path through the graph starting from the intersected flag. While this only finds one path through the graph for each query, it tends to be a prominent path because of distance attenuation.

The graph traversal begins whenever a diffraction flag with the correct orientation (see Section 5.2.1) is intersected by a BDPT subpath. We do a separate traversal for each source or listener in the scene that was not able to form a valid diffraction path directly over the edge, i.e. we only perform the graph traversal when a lower-order path was not found using the approach in Section 5.2.3. At this point, we transform $\vec{p}_{ref}$ and the goal source or listener into the mesh's local space so that the search can operate locally to avoid transforming vertices and normals. The search starts at the graph node corresponding to the intersected flag. From there, we investigate only the neighboring nodes that are on the opposite side of the flag from $\vec{p}_{ref}$, as discussed in Section 4.4.

For each neighbor, we compute the estimated distance from the neighboring edge to the goal source or listener. This is the A* heuristic that is used to rank potential paths through the graph. The choice of heuristic influences which paths are prioritized. One possibility would be to use the Euclidean distance from the neighbor's midpoint to the goal, however we found that this does not always find the shortest path when edges are long because the midpoint may be distant from the diffraction apex point. Instead of the midpoint, we propose using the point on the neighboring edge that is closest to the line between the source and listener.

Once the distance from the closest point on the neighbor to the goal is determined, it is added to the shortest distance through the graph from the starting edge to the current edge to yield the total estimated distance for the neighbor. We discard any neighbors that are in the A* *closed set* and which have distance estimates greater than best path through the graph to that node, if the neighbor was

previously visited. If a neighbor is not in the A* *open set* or has a distance estimate lower than the best so far, we then check the edge further to see if it is in a proper geometric configuration for diffraction according to the approach from Section 5.2.1. If so, that edge is inserted into the A* node heap.

Once all neighbors are either discarded or put into the A* heap, we check the top of the heap (i.e. the node with smallest distance estimate) to see if a valid diffraction path is formed from the starting node to the top node. We first check to make sure the goal point is inside the shadow region of the final edge. If this succeeds, the edge sequence for the shortest path is reconstructed and transformed into world space for the final path validation and visibility testing. We then use the same approach from Section 5.2 to determine if the path is valid. If so, that path is inserted into the cache and the graph search terminates. Otherwise, the neighbors of that node are investigated recursively.

This process repeats until a path is found or until a maximum number of nodes has been visited. If no valid path through the graph exists, which is sometimes the case with diffraction through complex environments, A* degenerates to Djikstra's algorithm and explores the entire graph. By limiting the number of nodes that are visited, we can avoid spending a lot of time searching the extraneous parts of the graph when no path actually exists. We suggest using a limit that is $2-3$ times larger than the maximum diffraction order, e.g. we use a limit of 30 nodes for diffraction up to order 10.

6 IMPLEMENTATION

The preprocessing module of our diffraction approach is highly parallel so that it can scale to very large meshes. We parallelize the edge collapse simplification by splitting the mesh into disjoint sections using a uniform grid. The edge collapse algorithm is applied separately to each section in parallel, while avoiding simplification of the border triangles. After all sections are simplified, the borders are simplified in a final serial pass. The other parts of the preprocessing pipeline are either serial (face clustering) or embarrassingly parallel (e.g. silhouette edge detection, visibility graph computation). We are careful to reduce the total number of memory allocations in the mesh data structure, yielding performance benefits.

The runtime module simulates the propagation of sound in 4 logarithmically-distributed frequency bands: 0-176 Hz, 176-775 Hz, 775-3408 Hz, and 3408-22050 Hz. The simulation executes on a single thread of an Intel Core i7-4770K CPU. Ray intersections are accelerated by an axis-aligned bounding box hierarchy (BVH) that is partitioned into two levels: a top level that is rebuilt each frame to handle dynamic geometry, and a bottom level for the static mesh BVHs in local space. The path tracer emits 100 rays from each source and listener. Listener rays are reflected or diffracted up to 200 times before path termination, while source rays are only allowed 10 path events. The asymmetry between sources and listeners enables the path tracer to simulate multiple sources with less runtime impact. At each listener subpath event, we randomly sample connections to points on the source subpaths. The probability of making a connection can be used to adjust the balance between simulation convergence and performance.

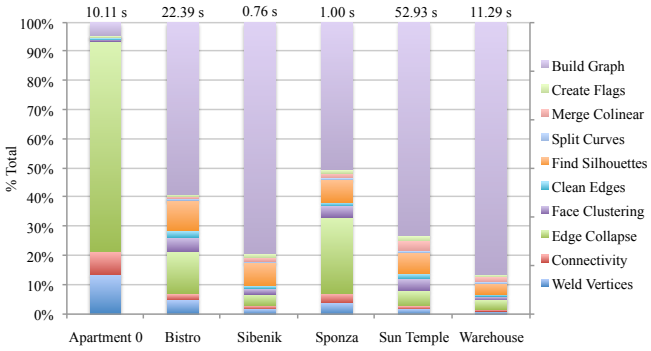

Fig. 9. The time taken by each section of our preprocessing pipeline. For most scenes, the construction of the diffraction graph takes the majority of the time. The Apartment 0 scene requires more time for edge-collapse simplification due to the highly tessellated input mesh.

Frequency-dependent audio rendering is implemented by first passing all input monaural source audio through a 4-way crossover filter bank and then applying separate gains to each band in parallel using SIMD instructions. We use interpolated delay lines to implement the rendering of direct sound, diffraction, and early reflections up to order 2. Late reverberation is rendered using artificial reverberators whose rates of decay and gain in each band are derived from energy-time histograms computed by the path tracer. Reverb and early reflections are spatialized as 1st-order ambisonic signals, while direct sound and diffraction are 3rd-order. The audio for all sources is summed as world-space ambisonics and a single convolution with an ambisonic HRTF [Zaunschirm et al. 2018] is performed to convert ambisonics to binaural. The listener's head rotation is applied by rotating the ambisonic HRTF by the inverse head rotation before convolution.

7 RESULTS

7.1 Scenes

To evaluate our diffraction method, we selected six scenes of varying size and geometric complexity. The attributes and results of these scenes are summarized in Table 1. The *Apartment 0* scene was chosen as an example of dense geometry derived from 3D reconstruction. The *Bistro* scene [Lumberyard 2017] interior has highly detailed geometry with many very small diffraction edges. The *Sibenik* and *Sponza* scenes [McGuire 2017] have lower complexity but some challenging features (cylindrical columns, curtains). The largest scene, *Sun Temple*, encompasses a very large outdoor environment (about 1km^2), and has many difficult geometries such as wooden walls with many small holes through which diffraction can occur. The *Warehouse* scene is particularly challenging for the diffraction graph traversal because its numerous mutually-visible diffraction edges result in a graph with a high average degree. Please refer to the accompanying video to hear the audio results of our method on these scenes.

Table 1. The main results of our diffraction approach. All scenes simulate diffraction at runtime up to order 10 on a single Intel Core i7-4770K 3.7GHz CPU thread. Preprocessing results use 8 CPU threads. Speedup is per source relative to [Schissler et al. 2014].

				Preprocessing					Runtime			
Scene	#Triangles	#Edges	#Flags	Graph Size	Time	[Schissler 2014]	Speedup	#Sources	Total	Per Source	[Schissler 2014]	Speedup
Apartment 0	9.1M	147.7K	2,446	145 KB	10.11 s	145.2 s	**14.4**×	6	2.1 ms	0.35 ms	0.95 ms	**2.7**×
Bistro	3.8M	2.8M	52,364	8.40 MB	22.39 s	206.3 s	**9.2**×	7	36.5 ms	5.2 ms	107.8 ms	**20.7**×
Sibenik	75.2K	66.7K	4,837	671 KB	0.76 s	8.07 s	**10.7**×	2	0.28 ms	0.14 ms	5.80 ms	**42.2**×
Sponza	282.1K	103.7K	5,075	489 KB	1.00 s	8.73 s	**8.7**×	6	4.2 ms	0.70 ms	3.04 ms	**4.4**×
Sun Temple	4.7M	3.8M	229,512	54.5 MB	52.93 s	16,175 s	**305.6**×	16	120.7 ms	7.5 ms	4.29 s	**568**×
Warehouse	602K	377.3K	34,336	8.69 MB	11.29 s	83.6 s	**7.4**×	20	72.0 ms	3.6 ms	390.6 ms	**109**×

7.2 Preprocessing

We applied our preprocessing pipeline and measured the time taken by each section. These results are shown in Figure 9. Additional results and images of the scenes for each pipeline stage are located in the supplemental material. For most scenes, the preprocessing time is less than 20 *s*. For the simpler *Sibenik* and *Sponza* scenes, it takes only about 1 second for the entire process. The largest scene, *Sun Temple*, still only takes 52.93 *s*. Aside from the *Apartment 0* scene where edge-collapse dominates performance due to dense tessellation, most of the time is spent on the construction of the edge visibility graph. Our preprocessing pipeline is able to reduce the number of diffraction edges considered for simulation by at least an order of magnitude, and in some cases by up to 60 times (*Apartment 0*). The total memory required for the diffraction graph is less than 10MB for all but the *Sun Temple* scene.

For comparison, we tried the same scenes using the implementation from [Schissler et al. 2014]. Our approach is about an order of magnitude faster on most of the scenes, and up to 305.6 times faster for the *Sun Temple* scene, which took almost 5 hours to compute using the old method. The main reason for this large speedup is the lack of a voxelization step, and our new method for computing the edge visibility graph that has much better time complexity ($O(N)$ instead of $O(N^2)$). For all scenes, our approach also produces fewer final diffraction edges than [Schissler et al. 2014].

7.3 Runtime

To evaluate the runtime method, we placed various sound sources throughout the scenes and recorded the time taken to calculate diffraction for the demos shown in the accompanying video. For the simpler scenes (*Apartment 0, Sibenik*, *Sponza*), our method is able to simulate 10th-order diffraction in less than 1ms per source. The other scenes have many more diffraction edges and larger diffraction graphs, and as a result take 3.6ms – 7.5ms per source. Generally, the time required increases with the number of diffraction flags as well as the size and interconnectedness of the graph.

In Figure 10, we present results for how the runtime performance of our method varies with respect to the maximum diffraction order. Compared to previous approaches like [Schissler et al. 2014] which have exponential time complexity, our approach exhibits a roughly linear relationship. This enables our approach to scale to much larger diffraction order within real-time constraints and enables it to handle more difficult curved geometry.

In Figure 11 we show how our runtime method performs with different numbers of rays as well as with and without the A* graph search and path cache on the *Warehouse* scene. The path cache improves the smoothness of the curve (i.e. temporal coherence) compared to without the cache, as well as increases the number of paths found by about 10 times, for a small overhead of about 0.2ms. The A* graph search also increases the number of paths found by about 10 times relative to without the graph search, though it has a greater runtime overhead. When both cache and A* are enabled, our approach is able to find nearly as many paths with 100 rays as

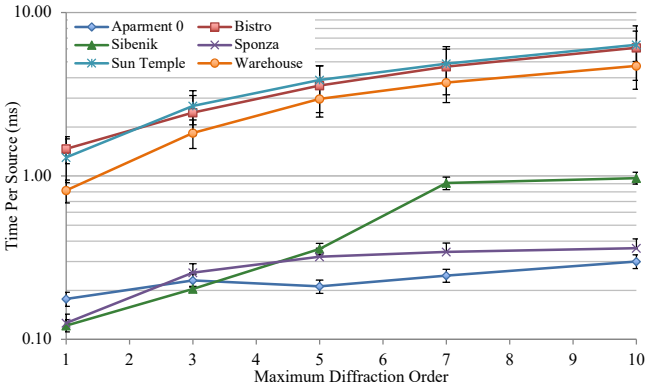

Fig. 10. A graph showing how the runtime performance of our approach changes with respect to the maximum diffraction order. Please note the logarithmic vertical axis. Error bars correspond to one standard deviation.

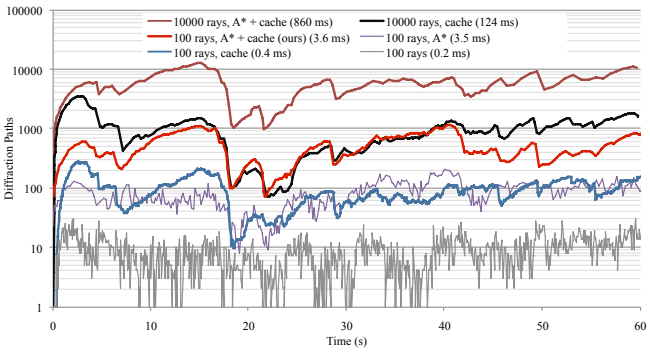

Fig. 11. A graph showing how the number of paths found in the *Warehouse* scene over time in the demo video is affected by the number of rays traced as well as presence or absence of the A* graph search and path cache. Times for each condition are per-source.

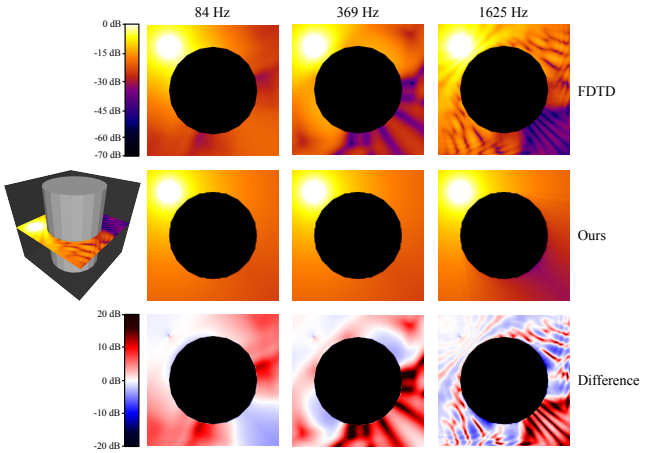

Fig. 12. A comparison between our diffraction approach and an offline FDTD simulation on a cylinder scene where all surfaces are fully absorptive. This illustrates the accuracy of UTD diffraction up to order 8. For most positions in the scene, the difference between the methods is less than 5 decibels. In general, UTD overestimates the level of the diffracted sound, particularly for areas with high-order diffraction.

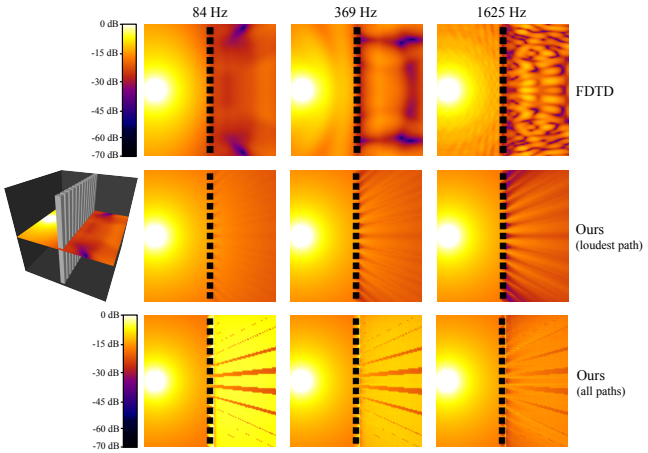

Fig. 13. A comparison between FDTD and two different configurations of our approach on a scene that resembles a wooden fence. When configured to render all paths, there is significant overestimation of the diffracted sound field for the 84Hz and 369Hz bands, whereas using the loudest path produces results that more closely match the FDTD simulation. The hard shadows in the *all-paths* case are the result of rendering either direct sound or diffraction (but not both).

compared to 10,000 rays without the A* graph search, yet is 34 times faster. In the video, we also show the audio impact of the cache and A* search.

Compared to [Schissler et al. 2014], our runtime approach is between 2.7 and 568 times faster per source for 10th-order diffraction. The performance of their method scales exponentially with the maximum order, making diffraction beyond order $3-5$ infeasible in real-time except for smaller scenes. On the other hand, our approach's use of the A* algorithm to search the graph avoids this problem, and enables better results with 100 rays than [Schissler et al. 2014] with 1000 rays. In the accompanying video we compare audio results in the *Sibenik* and *Sponza* scenes to show how our method is more robust with difficult curved geometry.

The recent diffraction approach of Pisha et al. [2020] reports performance that is comparable to ours. However, their approach was tested on low complexity scenes ($\leq$ 1024 triangles), and uses a powerful GPU for ray tracing. In contrast, our technique is applicable to scenes that are more than 1000× larger, and runs on a single thread of a modest CPU. Despite this, its good agreement with BTM makes [Pisha et al. 2020] an attractive replacement for UTD evaluation of the diffracted sound pressure in our approach.

7.4 Validation

To evaluate the accuracy of our approach, we compared it to an offline FDTD simulation on a few simple scenes. We set all surfaces and boundary conditions to be fully absorptive to isolate only the direct and diffracted sound. In Figure 12, we present the results for a scene containing a 20-sided cylinder, where the maximum diffraction order is at least 8. Overall, there is a relatively good match between the methods, with a difference of less than 5dB for most of the scene. The greatest errors occur in the areas with higher-order diffraction, where energy-based UTD overestimates the diffracted sound field by $10-15$dB, and doesn't exhibit the interference patterns visible in the wave simulation. Please refer to the supplemental material for more comparisons of 1st and 2nd-order diffraction.

We also performed an experiment to justify the choice of rendering only the loudest diffraction path as opposed to all paths. Figure 13 shows this in a scene that resembles a wooden fence, where diffraction paths are produced around each of the columns. In the case where all diffraction paths are rendered, the sound field is overestimated by about 20dB for the 84Hz frequency band. This may be caused by fundamental limitations of UTD such as the assumption of an infinite diffraction wedge. In comparison, using only the loudest path results in a diffracted sound field that much more closely matches the FDTD results at low frequencies.

8 CONCLUSIONS

In this work, we presented a complete approach for simulating approximate acoustic diffraction for real-time AR and VR applications. Our diffraction approach uses a novel mesh preprocessing pipeline to identify a reduced set of diffraction edges as well as construct diffraction flag geometry and edge visibility graphs. The runtime component of our approach traces rays against the diffraction flags to probabilistically explore possible diffraction paths, then computes the path intensities using the UTD diffraction model. We also utilize a precomputed edge visibility graph and the A* algorithm to greatly speed up the exploration of high-order diffraction paths. This diffraction technique is between 2.7 and 586 times faster than the previous state of the art in real-time high order diffraction, depending on the scene, and is able to scale efficiently and robustly to large scenes with high geometric detail. We have also evaluated its objective accuracy by comparing to an offline FDTD wave simulation.

However, there are some limitations. First, we only consider *direct* diffraction between a source or listener, i.e. diffraction paths consisting of only edge diffraction with no reflections. In theory, combinations of reflection and diffraction are compatible with our approach, but would require changes to how paths are stored and accessed in the diffraction path cache, and also changes to how the path intensity is evaluated. Additionally, the generation of unique cache identifiers for diffuse reflections may prove more difficult than for diffraction edges or specular reflections. Since we use the UTD diffraction model to calculate diffraction path intensities, our approach has all of the limitations of UTD such as inaccuracy with small edges. However, other more-accurate diffraction models like BTM or [Pisha et al. 2020] could be used in place of UTD to ameliorate some of these issues. Another limitation is that the diffraction graph traversal algorithm only finds a single path per edge, though this nevertheless produces plausible results. Since the graphs for each mesh in the scene are disjoint, the graph search can only find diffraction paths around individual objects, though this limitation does not apply to the rest of the runtime algorithm.

In the future we hope to apply this diffraction method to mobile-class devices where compute is extremely limited. We would also like to explore possibilites for leveraging more precomputation to further reduce the runtime overhead of diffraction.

REFERENCES

Paul T Calamia and U Peter Svensson. 2005. Edge subdivision for fast diffraction calculations. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.* IEEE, 187–190.

Chunxiao Cao, Zhong Ren, Carl Schissler, Dinesh Manocha, and Kun Zhou. 2016. Interactive sound propagation with bidirectional path tracing. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–11.

Anish Chandak, Christian Lauterbach, Micah Taylor, Zhimin Ren, and Dinesh Manocha. 2008. Ad-frustum: Adaptive frustum tracing for interactive sound propagation. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1707–1722.

Brent Cowan and Bill Kapralos. 2015. Interactive rate acoustical occlusion/diffraction modeling for 2D virtual environments & games. In *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA).* IEEE, 1–6.

Pedro F Felzenszwalb and Daniel P Huttenlocher. 2004. Efficient graph-based image segmentation. *International journal of computer vision* 59, 2 (2004), 167–181.

Thomas Funkhouser, Nicolas Tsingos, Ingrid Carlbom, Gary Elko, Mohan Sondhi, James E West, Gopal Pingali, Patrick Min, and Addy Ngan. 2004. A beam tracing method for interactive architectural acoustics. *The Journal of the acoustical society of America* 115, 2 (2004), 739–756.

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques.* 209–216.

Iliyan Georgiev. 2012. Implementing vertex connection and merging. *Technical Report. Saarland University* (2012).

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

Claudia Hendrix and Woodrow Barfield. 1996. The sense of presence within auditory virtual environments. *Presence: Teleoperators & Virtual Environments* 5, 3 (1996), 290–301.

Jan Hradek, Martin Kuchař, and Vaclav Skala. 2003. Hash functions and triangular mesh reconstruction. *Computers & geosciences* 29, 6 (2003), 741–751.

Chris Joslin and Nadia Magnenat-Thalmann. 2003. Significant facet retrieval for real-time 3d sound rendering in complex virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology.* 15–21.

Robert G Kouyoumjian and Prabhakar H Pathak. 1974. A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface. *Proc. IEEE* 62, 11 (1974), 1448–1461.

Alexander Lindau and Stefan Weinzierl. 2012. Assessing the Plausibility of Virtual Acoustic Environments. *Acta Acustica united with Acustica* 98, 5 (2012), 804–810.

Amazon Lumberyard. 2017. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). (July 2017). http://developer.nvidia.com/orca/amazon-lumberyard-bistro

Morgan McGuire. 2017. Computer Graphics Archive. (July 2017). https://casual-effects.com/data

Sönke Pelzer and Michael Vorländer. 2010. Frequency-and time-dependent geometry for real-time auralizations. In *Proceedings of 20th International Congress on Acoustics, ICA.* 1–7.

Louis Pisha, Siddharth Atre, John Burnett, and Shahrokh Yadegari. 2020. Approximate diffraction modeling for real-time sound propagation simulation. *The Journal of the Acoustical Society of America* 148, 4 (2020), 1922–1933.

Alexander Pohl. 2014. Simulation of diffraction based on the uncertainty relation. (2014).

Nikunj Raghuvanshi and John Snyder. 2014. Parametric wave field coding for pre-computed sound propagation. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

Nikunj Raghuvanshi, John Tennant, and John Snyder. 2017. Triton: Practical pre-computed sound propagation for games and virtual reality. *The Journal of the Acoustical Society of America* 141, 5 (2017), 3455–3455.

Atul Rungta, Carl Schissler, Nicholas Rewkowski, Ravish Mehra, and Dinesh Manocha. 2018. Diffraction kernels for interactive sound propagation in dynamic environments. *IEEE transactions on visualization and computer graphics* 24, 4 (2018), 1613–1622.

Lauri Savioja and U Peter Svensson. 2015. Overview of geometrical room acoustic modeling techniques. *The Journal of the Acoustical Society of America* 138, 2 (2015), 708–730.

Carl Schissler. 2017. *Efficient Interactive Sound Propagation in Dynamic Environments.* PhD thesis, UNC Chapel Hill.

Carl Schissler and Dinesh Manocha. 2011. GSound: Interactive sound propagation for games. In *AES 41st International Conference: Audio for Games.*

Carl Schissler, Ravish Mehra, and Dinesh Manocha. 2014. High-order diffraction and diffuse reflections for interactive sound propagation in large environments. *ACM Transactions on Graphics (SIGGRAPH 2014)* 33, 4 (2014), 39.

Samuel Siltanen, Tapio Lokki, Lauri Savioja, and Claus Lynge Christensen. 2008. Geometry reduction in room acoustics modeling. *Acta Acustica united with Acustica* 94, 3 (2008), 410–418.

Uwe M Stephenson. 2004. *Beugungssimulation ohne Rechenzeitexplosion: die Methode der quantisierten Pyramidenstrahlen.* PhD thesis, RWTH Aachen.

Uwe M Stephenson. 2010. An energetic approach for the simulation of diffraction within ray tracing based on the uncertainty relation. *Acta Acustica united with Acustica* 96, 3 (2010), 516–535.

Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. 2019. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797* (2019).

U Peter Svensson, Roger I Fred, and John Vanderkooy. 1999. An analytic secondary source model of edge diffraction impulse responses. *The Journal of the Acoustical Society of America* 106, 5 (1999), 2331–2344.

Micah Taylor, Anish Chandak, Zhimin Ren, Christian Lauterbach, and Dinesh Manocha. 2009. Fast edge-diffraction for sound propagation in complex virtual environments. In *EAA auralization symposium.* Citeseer, 15–17.

Rendell R Torres, U Peter Svensson, and Mendel Kleiner. 2001. Computation of edge diffraction for more accurate room acoustics auralization. *The Journal of the Acoustical Society of America* 109, 2 (2001), 600–610.

Nicolas Tsingos, Thomas Funkhouser, Addy Ngan, and Ingrid Carlbom. 2001. Modeling acoustics in virtual environments using the Uniform Theory of Diffraction. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* ACM, 545–552.

Eric Veach. 1997. *Robust Monte Carlo methods for light transport simulation.* Vol. 1610. Stanford University PhD thesis.

Stephan Werner, Florian Klein, Thomas Mayenfels, and Karlheinz Brandenburg. 2016. A summary on acoustic room divergence and its effect on externalization of auditory events. In *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX).* IEEE, 1–6.

Hengchin Yeh, Ravish Mehra, Zhimin Ren, Lakulish Antani, Dinesh Manocha, and Ming Lin. 2013. Wave-ray coupling for interactive sound propagation in large complex scenes. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–11.

Markus Zaunschirm, Christian Schörkhuber, and Robert Höldrich. 2018. Binaural rendering of ambisonic signals by head-related impulse response time alignment and a diffuseness constraint. *The Journal of the Acoustical Society of America* 143, 6 (2018), 3616–3627.

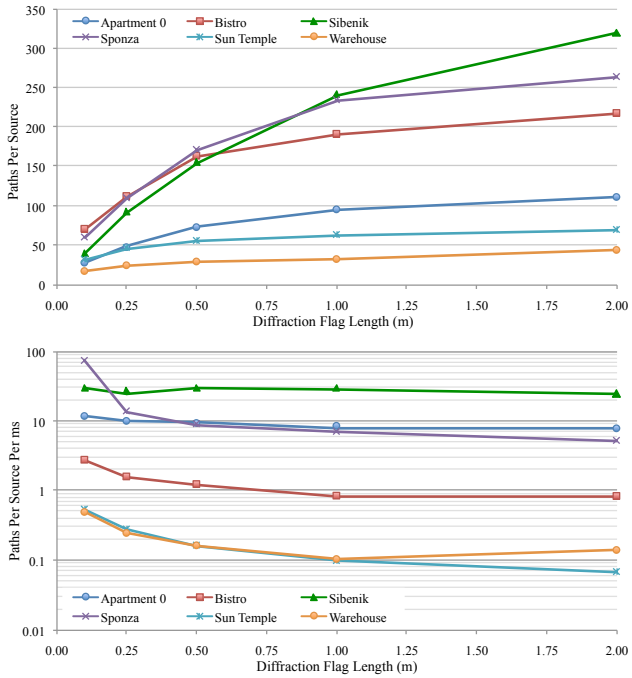

Fig. 14. An evaluation of the runtime performance and number of paths found for varying diffraction flag lengths from 0.1 m to 2.0 m.

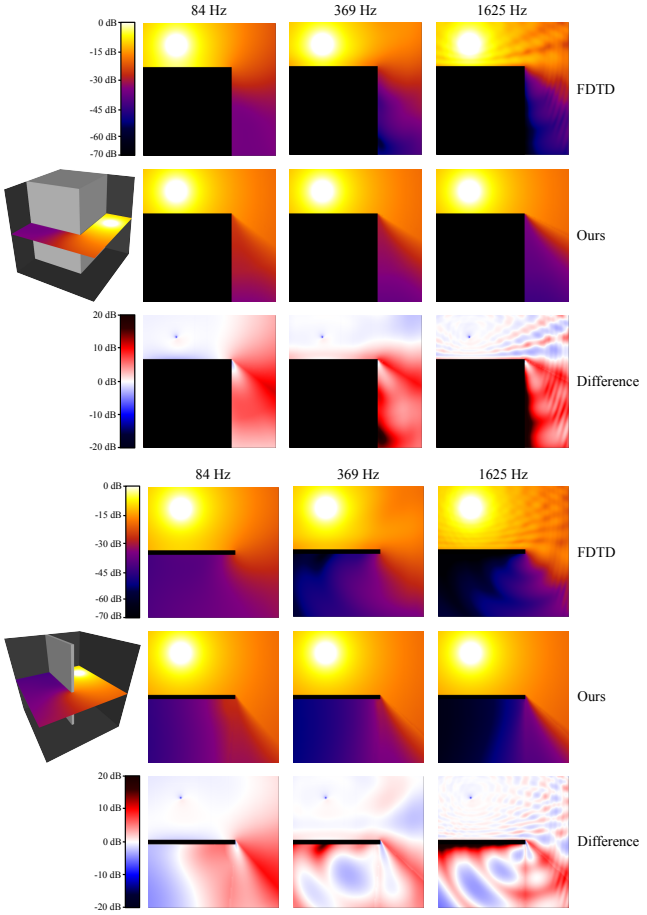

Fig. 15. A comparison between our diffraction approach and an offline FDTD simulation on two scenes showing 1st and 2nd-order diffraction. For most positions in the scene, the difference between the methods is less than a few decibels. The largest differences occur near the shadow region boundary, where the UTD interpolation from [Tsingos et al. 2001] increases the level of diffracted sound.

A PATH VALIDATION DETAILS

In Section 5.2.1, we presented the *shadow test* for path validation. Here, we provide additional details about how each edge is evaluated to check if it is within the shadow region of the previous edge. An edge is considered to be in shadow if it crosses the shadow face or horizon planes of the previous edge, or if it is completely contained inside the shadow region. This can be determined by evaluating the dot products $\delta_i^f = (\vec{v}_i - \vec{v}') \cdot \vec{n}_{shadow}$ and $\delta_i^h = (\vec{v}_i - \vec{v}') \cdot \vec{h}$ for $i = \{0, 1\}$, where $\vec{v}'$ is a point on the previous edge. The edge crosses the face or horizon planes when $\text{sign}(\delta_0^f) \neq \text{sign}(\delta_1^f)$ or $\text{sign}(\delta_0^h) \neq \text{sign}(\delta_1^h)$, respectively. The edge is completely inside the shadow region if $\delta_i^f > -\epsilon_f$ and $\delta_i^h < \epsilon_h$ for $i = \{0, 1\}$, where ϵ_h and ϵ_f are the shadow test tolerances for the respective planes. We apply this test for each edge intersected along a subpath.

B ADDITIONAL RESULTS

In this section we present additional preprocessing and runtime results for our diffraction approach.

B.1 Preprocessing

Figures 16 - 21 show the results of our preprocessing approach on each of the six scenes. For each scene, we provide two viewpoints and show the state of the mesh at each stage of the preprocessing pipeline. We also report the number of diffraction edges at each step to show the reduction in diffraction edges achieved by each pipeline stage.

In most scenes, the silhouette edge detection stage is the most successful at eliminating unnecessary edges, providing a reduction of $2.7\times - 14.5\times$ versus the output of the previous face clustering stage.

B.2 Runtime

We also evaluated how the length of the diffraction flags impacts the performance and number of paths found by our approach, as shown in Figure 14. As expected, the total number of paths generally increases with the flag length, though the increase is not very large beyond 1.0 m. However, when the paths found per unit time are plotted instead, we can see that very short flags are actually more efficient at finding diffraction flags in most scenes, including the large outdoor *Sun Temple* scene. This suggests that flags shorter than our chosen $d_{flag} = 1.0$ m may be more optimal.

B.3 FDTD Comparison

We performed further comparisons with FDTD to evaluate the accuracy of our approach on simple scenes with 1st and 2nd-order diffraction. These results are presented in Figure 15, where we show the magnitude at 3 different frequencies for a horizontal slice of the scene. Overall, our approach produces subjectively similar results to the FDTD simulation, with a difference of less than 10dB for most points in the scenes. Surprisingly, there is a closer match for second-order diffraction than for first-order diffraction. The greatest discrepancies occur near the shadow region boundary, where our approach overestimates the diffracted sound pressure. This is caused by the UTD interpolation proposed by [Tsingos et al. 2001] which artificially increases the diffracted sound pressure near the shadow region boundary in order to produce a smooth transition from direct to diffracted sound. This interpolation is required to produce a smooth sound field when diffraction outside the shadow region is ignored.

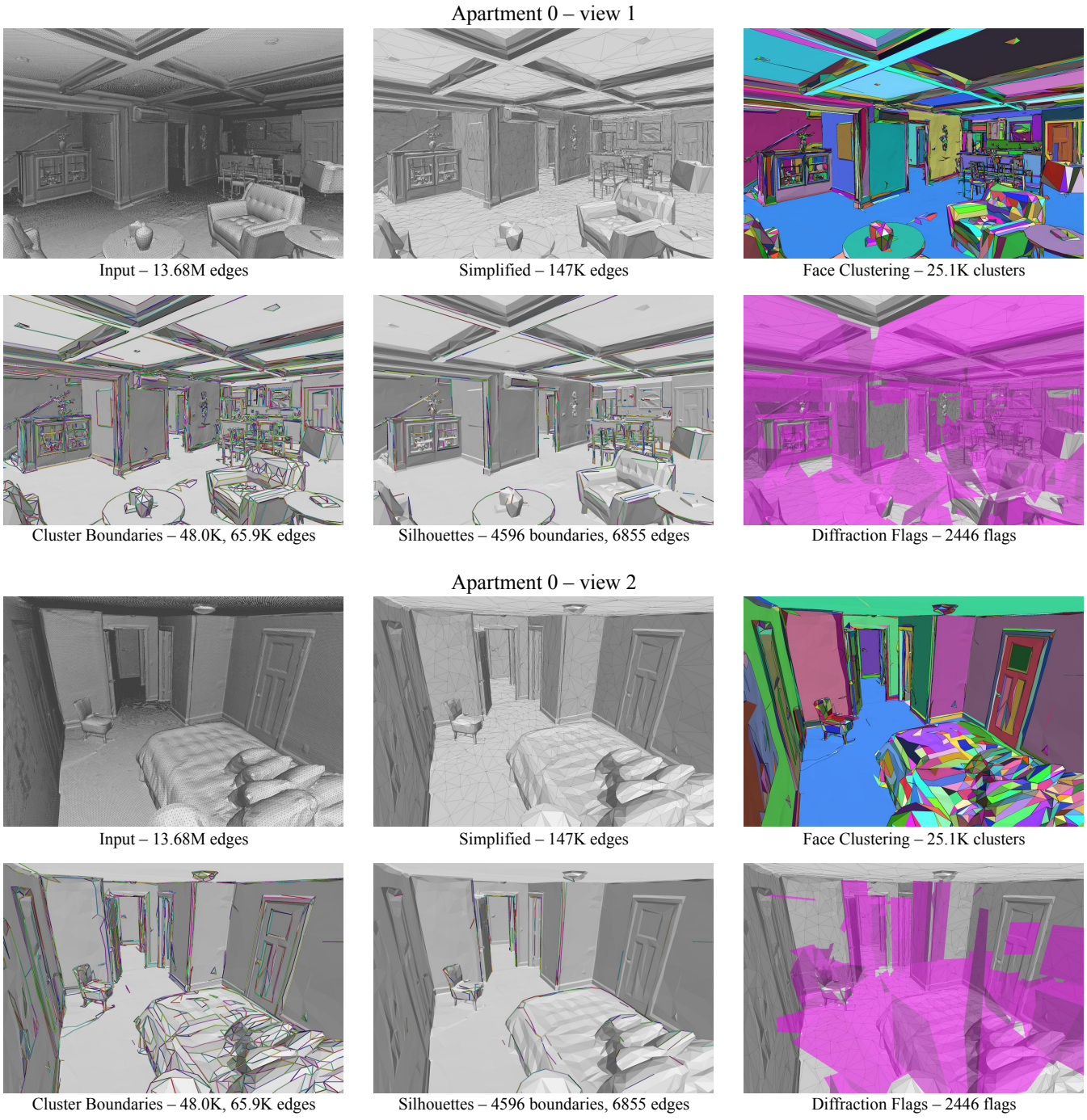

Fig. 16. The preprocessing results for the Apartment 0 scene for two different viewpoints at each stage in the pipeline. Our silhouette edge detection approach reduces the number of significant edges by 9.6×. Note how the silhouette edge detection removes the unnecessary edges on bumpy but otherwise flat surfaces (e.g. bed, chair). It also removes the edges from small protrusions (e.g. baseboard moulding).

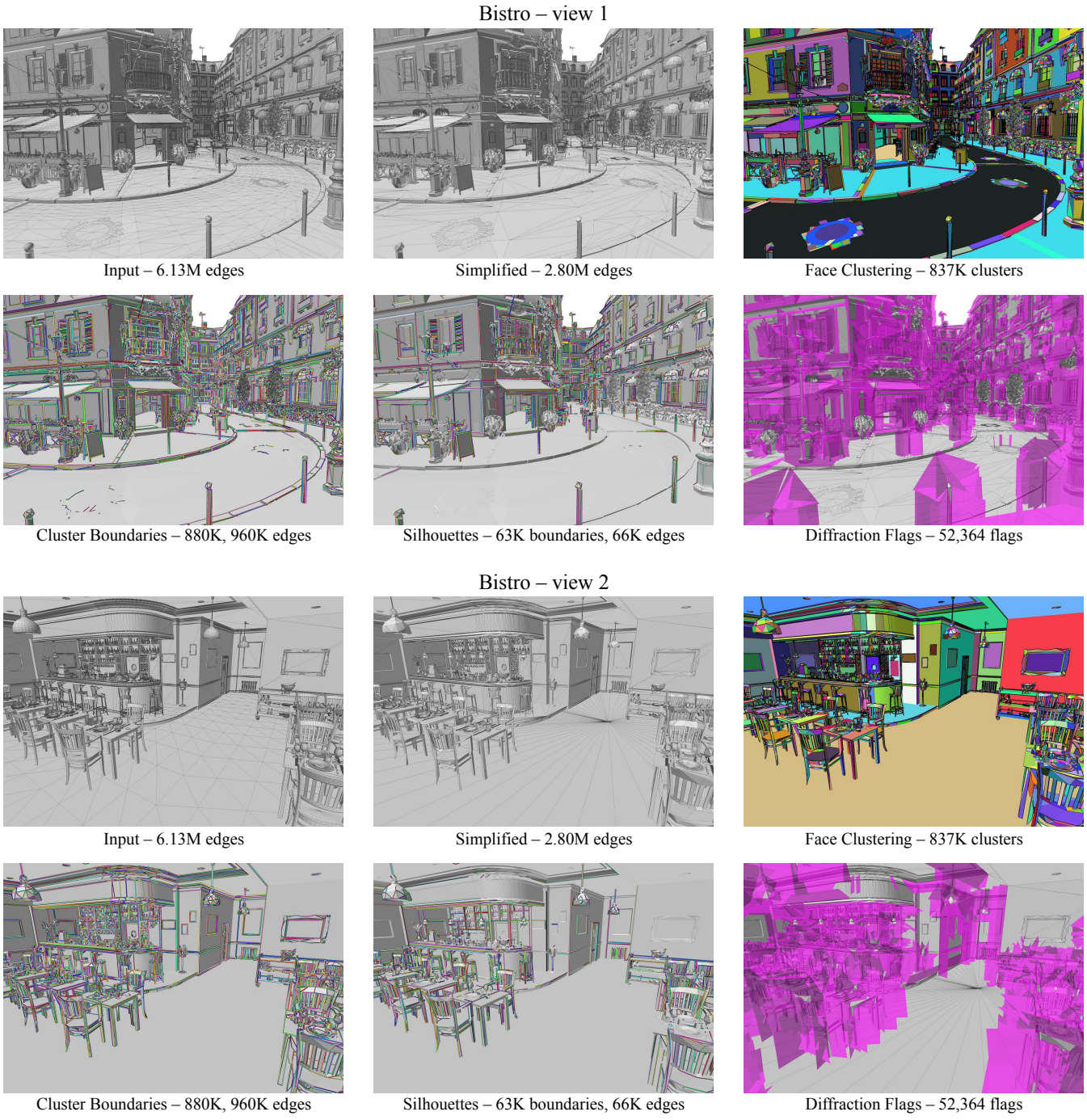

Fig. 17. The preprocessing results for the Bistro scene for two different viewpoints at each stage in the pipeline. Our silhouette edge detection approach reduces the number of significant edges by 14.5×. Silhouette edge detection removes edges such as the curb that cannot produce diffraction for a human-sized listener.

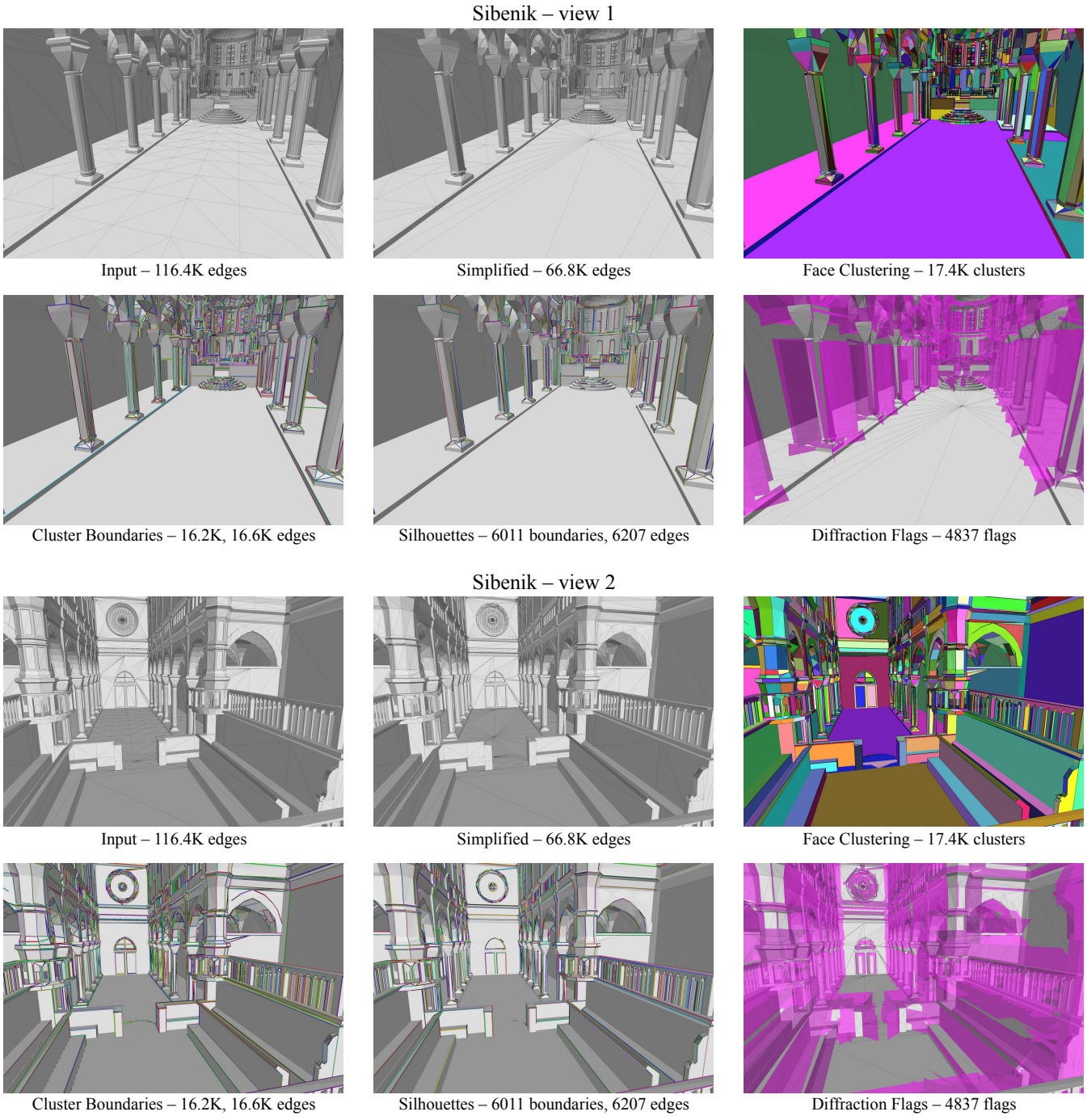

Fig. 18. The preprocessing results for the Sibenik scene for two different viewpoints at each stage in the pipeline. Our face clustering and silhouette edge detection techniques together reduce the number of significant edges by 9.3× versus the simplified mesh.

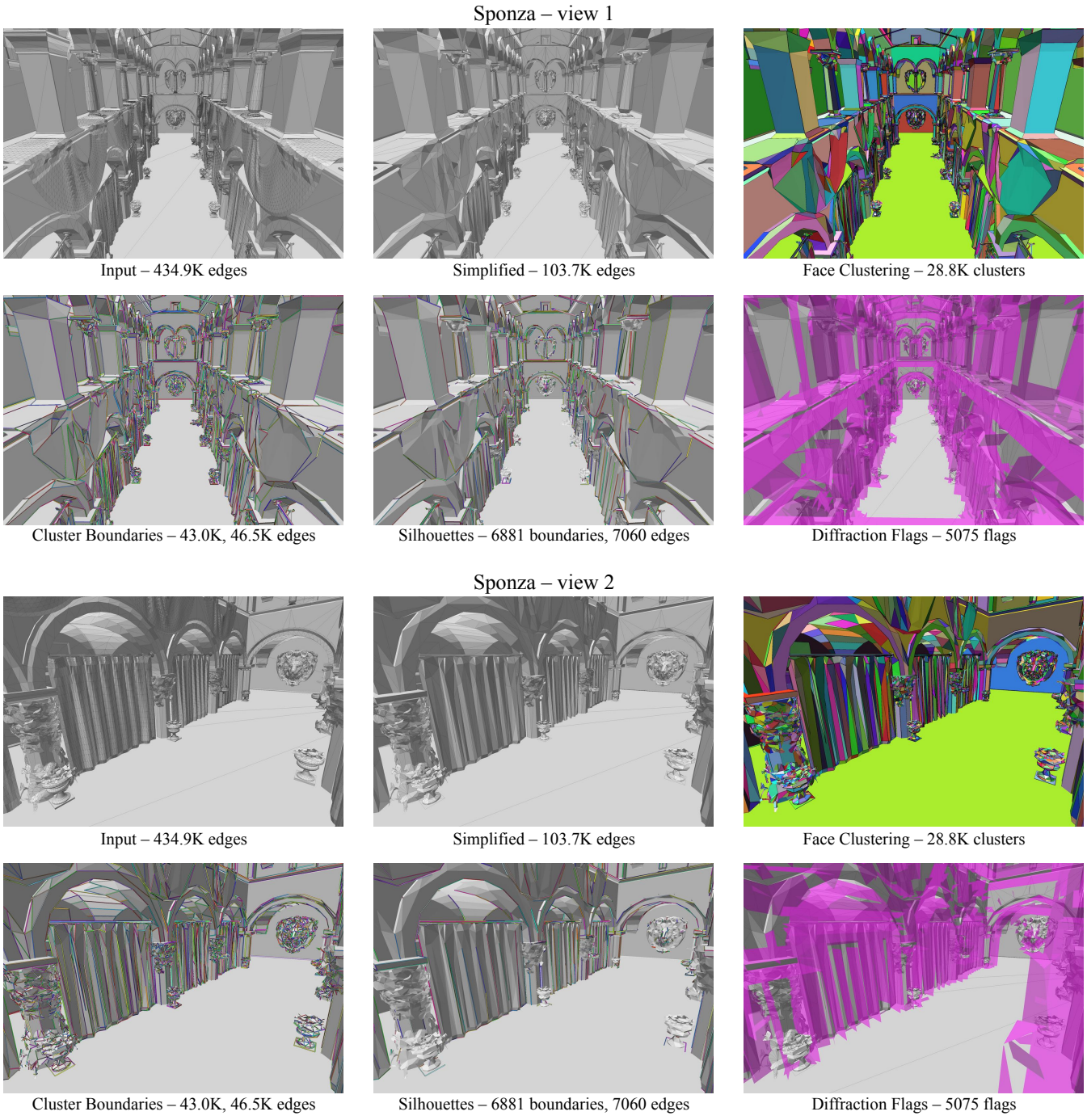

Fig. 19. The preprocessing results for the Sponza scene for two different viewpoints at each stage in the pipeline. Our face clustering and silhouette edge detection techniques together reduce the number of significant edges by 14.7× versus the simplified mesh.

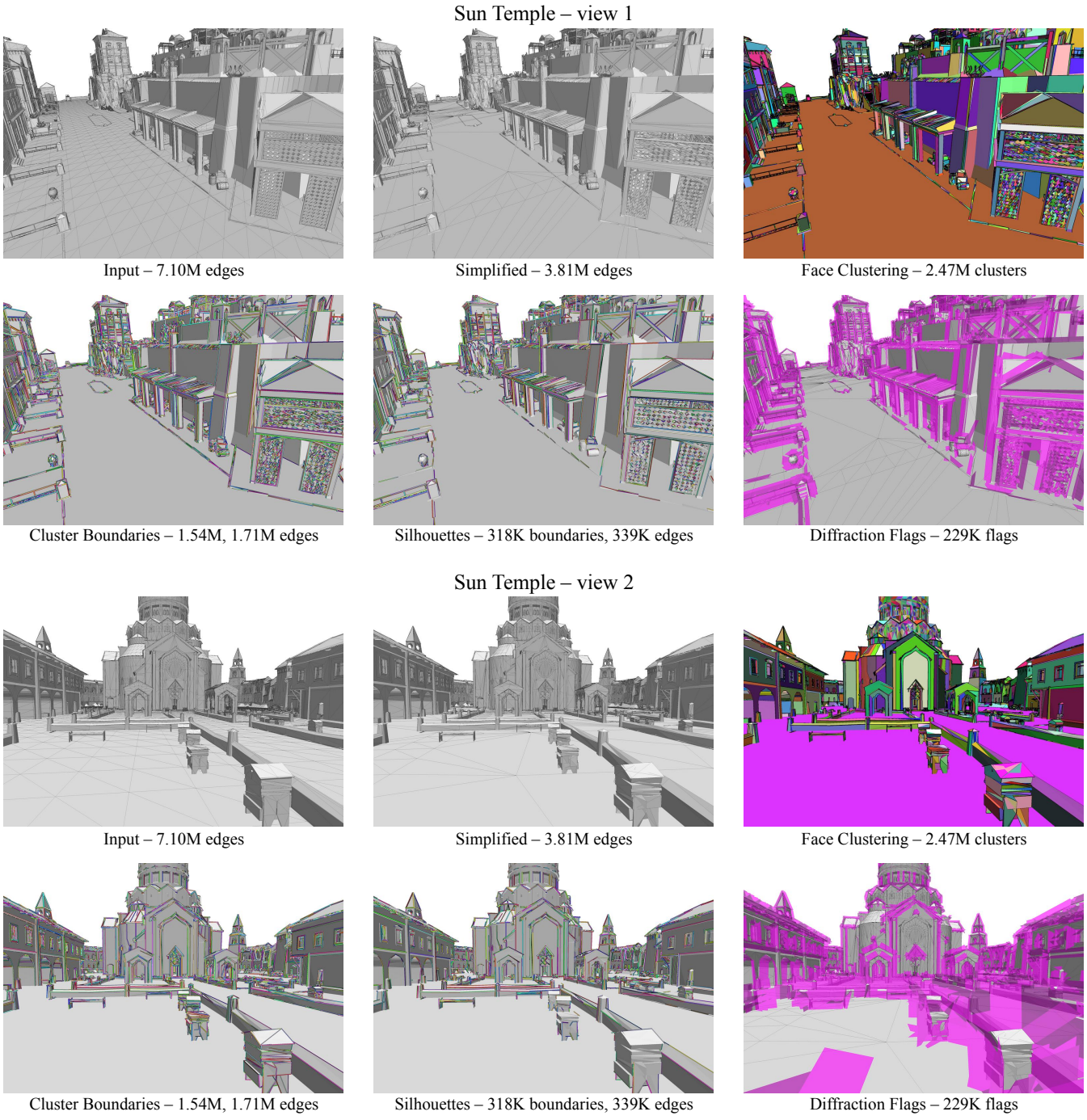

Fig. 20. The preprocessing results for the Sun Temple scene for two different viewpoints at each stage in the pipeline. Our face clustering and silhouette edge detection techniques together reduce the number of significant edges by 11.2× versus the simplified mesh.

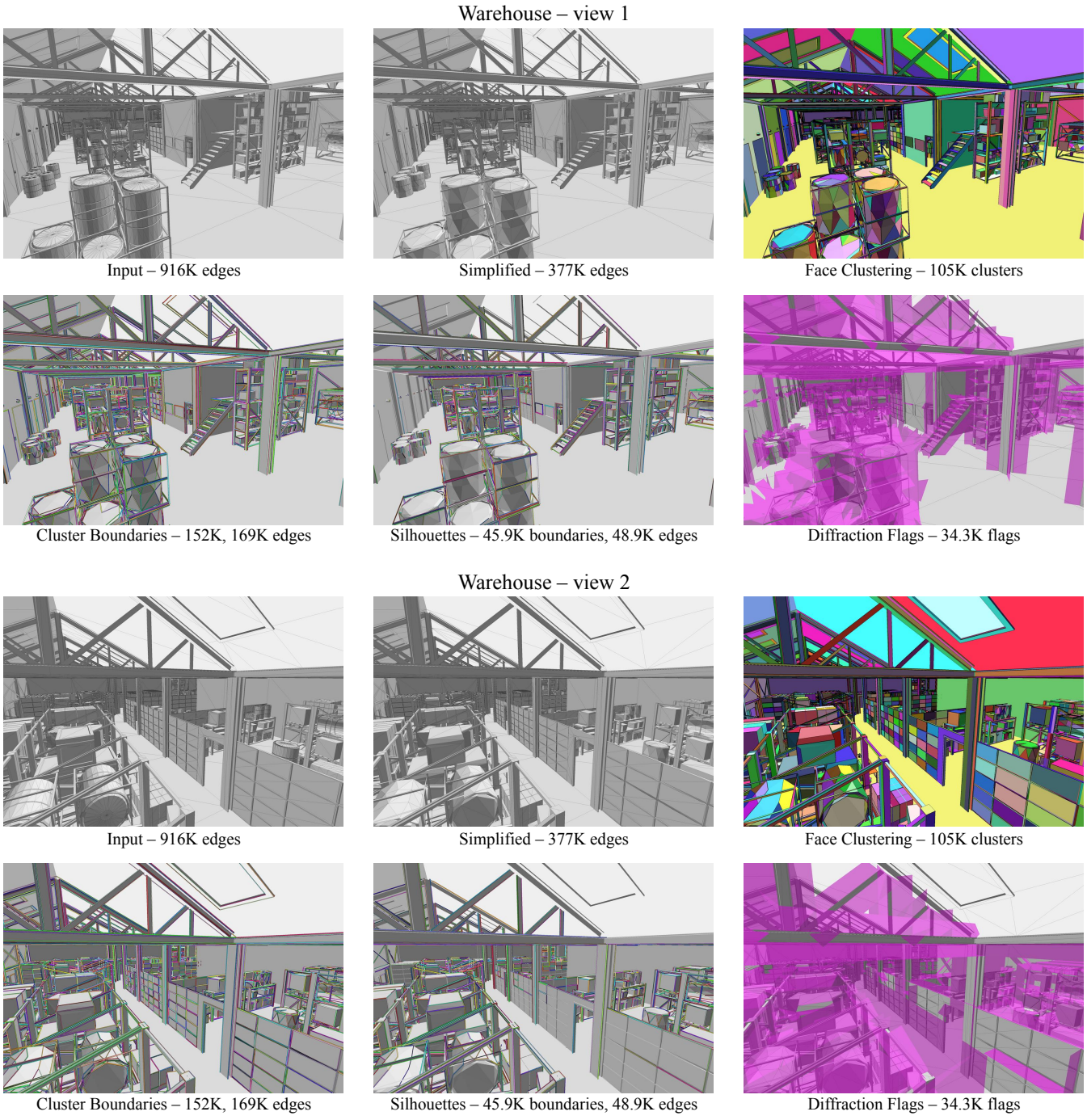

Fig. 21. The preprocessing results for the Warehouse scene for two different viewpoints at each stage in the pipeline. Our face clustering and silhouette edge detection techniques together reduce the number of significant edges by 7.7× versus the simplified mesh. Silhouette edge detection removes most of the edges along flat surfaces like the segmented wall in view 2 while retaining the important edges.