High-Performance Graphics (2023)
J. Bikker and C. Gribble (Editors)

Voxel-based Representations for Improved Filtered Appearance

Caio José Dos Santos Brito[1,2] Pierre Poulin[1] Veronica Teichrieb[2]

[1] Université de Montréal, Canada
[2] Universidade Federal de Pernambuco, Brazil

Abstract

Volumetric representations allow filtering of mesh-based complex 3D scenes to control both the efficiency and quality of rendering. Unfortunately, directional variations in the visual appearance of a volume still hinder its adoption by the real-time rendering community. To alleviate this problem, we propose two simple structures: (1) a virtual mesh to encode the directional distribution of colors and normals, and (2) a low-resolution subgrid of opacities to encode directional visibility. We precompute these structures from a mesh-based scene into a regular voxelization. During display, we use simple rendering methods on the two structures to compute the image contribution of the appearance of a visible voxel, optimizing for efficiency and/or quality. The improved visual results compared to previous work are a step forward to the integration of volumetric representations in real-time rendering.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Visibility; Volumetric models; Antialiasing;***

Keywords: Volume rendering, voxel, octree, level of detail, SGGX

1. Introduction

Highly complex scenes are very common in the film industry, involving multiple characters, environments with different textures and materials, complex light transport effects, and many extended light sources. The video game industry aims at achieving the same level of quality, but faces extremely severe rendering limitations, even with the power of today's GPUs. Levels of detail (LODs) can decrease scene complexity in order to render efficiently faraway objects affecting only few pixels or even a fraction of a single pixel (subpixel). Despite being broadly adopted by the video game industry, LOD rendering must be handled with care in order to maintain consistent appearance throughout scales over surfaces and groups of objects. This very difficult task ends up requiring regularly some form of manual intervention by artists [BAC*18].

The common LOD approach in most video games uses mesh simplification to reduce the number of triangles/quads. It assigns a different mesh to each level of a discrete LOD [LLT*20, YLH18, DSSC08]. Hoppe's *Progressive Meshes* [Hop96] encodes mesh simplification as a series of edge-collapses, based on various metrics to infer minimal geometric distortion, and made even smoother with geomorphing during an edge-collapse. Unfortunately, efficient rendering requires to use fewer LODs from mesh simplification, thus again, going back to rely on manual intervention. Moreover, it proves mostly unsuitable for distributions of small objects, such as trees' fine branches and leaves, hair, and highly detailed objects consisting of several potentially disconnected parts.

Volumetric representations, including 3D textures and sparse voxel octrees, are also used for LODs. They integrate (filter) the data within a volume to approximate its appearance when viewed from a distance. They have been applied with success to distributions of more homogeneous small objects such as fur and trees [KK89, Ney98]. Their hierarchical nature makes them suitable for extended MIP mapping filtering of their appearance [LN18, HDCD15]. However, capturing appearance changes in presence of arbitrary occlusions and complex objects, when moving around a volume (i.e., directional distribution of colors and normals), is still a challenge. Unlike MIP mapping that is widely used for surface texture filtering, volumetric representations still fall short of adoption by the real-time rendering community. We believe that this problem requires much more attention, and this is our goal here.

In this paper, we present our solution to better encode the changing appearance of a volume (voxel) when viewed at a distance from different directions. We introduce

- a simple *virtual mesh* to encode colors and normals in a voxel;
- a low-resolution subgrid to encode opacities in a voxel;
- an efficient display to sample the appearance and compute the opacity of a voxel from a given viewpoint.

The virtual mesh is a simple triangular mesh (an octahedron) approximating an ellipsoid [Ney98, HDCD15]. While requiring more space in memory, it allows for a much more detailed and flexible encoding to represent distributions of colors and normals at its faces or vertices. That virtual mesh also frees us from the ellipsoid symmetry. The shape remains simple enough to be sampled efficiently from any view direction, thus providing a more accurate appearance, with potentially smooth interpolations and sharp discontinuities. Opacities are refined in a low-resolution subgrid, also providing an efficient approximation from any view direction. In this paper, we concentrate our efforts and analysis on a single LOD from a grid of voxels. However, we expect our results to extend naturally to all LODs of a hierarchy of grids of voxels.

The rest of the paper is organized as follows. After a brief study of voxelized volumetric representations for complex scenes in Section 2, we introduce our volumetric representations in Section 3. We describe how its structures are precomputed, and how they are sampled for real-time rendering. We then discuss details of our implementation in Section 4 before analyzing our results in Section 5. Finally, we conclude by revisiting our achievements and we point out future improvements in Section 6.

2. Related Work

We build our method from a long tradition of volumetric representations to render filtered complex scenes, dating back to the voxels of Kajiya and Kay [KK89], and generalized to octrees by Neyret [Ney98]. Three problems must be addressed when dealing with such volumetric representations: (1) filtering and encoding of the volumetric data, (2) traversal of the volumetric elements, and (3) sampling of the volumetric data for efficient rendering. Understandably, much work has been devoted to efficient rendering on GPU and reducing data size.

To render high-quality massive volumes at interactive to real-time frame rates, the *GigaVoxels* of Crassin et al. [CNLE09] are based on sparse voxel octrees (SVOs). The hierarchical representation stores each tree node into a 3D texture on the GPU. It is linked to a brick pool that contains the scene's data. In addition to the geometry, it is able to render soft shadows and depth-of-field camera effects, to handle LODs, and to exploit occlusion culling. It has its limitations though. If multiple objects within a voxel occlude each other differently depending on the view direction, the approximate occlusion will not capture visibility changes. Also, the voxel occlusion being approximated to the transparency of its enclosed object, it may lead to thicker silhouettes. For instance, an opaque object that occludes only half of a voxel will approximate it to a fully occluding voxel.

To minimize the memory footprint with efficient ray casting, Laine and Karras [LK11] propose *Efficient Sparse Voxel Octrees* (ESVOs), in which voxel data are stored in conjunction with its parent voxel, allowing data compression and lower memory consumption. Instead of using a volume representation, ESVO uses a surface representation for subvoxel content that limits the voxel extent with a pair of parallel planes that approximates the subvoxel content. Ray traversal is done in depth-first order and an adaptive blur filter is applied to smooth out blockiness in the final image. The representation allows sampling in the order of millions of primary rays per second, and it reduces artifacts near the scene's silhouettes.

Christensen and Batali [CB04] propose a tiled 3D MIP map representation for volume and surface data to make global illumination more efficient and more adequate for movie productions. Data are organized into an SVO with a brick at each octree node. It has efficient caching with a least-recently-used (LRU) replacement strategy. The structure provides a hierarchical representation, and ray differential is used to determine which level of the brick map should be accessed.

Crassin et al. [CNS*11] present an approach to render global illumination at interactive frame rates. They represent the scene as a prefiltered dynamic sparse voxel octree. First, incoming radiance from all light sources is stored into the leaves of the SVO, and it is filtered for the coarser levels of the octree. Light transport is then approximated by cone tracing in the SVO structure, in a similar way than GigaVoxels [CNLE09]. Voxel data are stored into six channels of directional values to capture anisotropic behaviors. To compute indirect illumination, diffuse components are estimated with larger cones, and specular components with narrower cones. The approach renders dynamic scenes at interactive frame rates with ambient occlusion. However, visibility is not accounted for in the LODs.

To improve on previous representations [CNLE09, LK11] and calculate subpixel occlusions and correlation effects, Heitz and Neyret [HN12] propose a data representation stored in an SVO that filters color variations, reduces aliasing, and works on depth-of-field images. SVO traversal is similar to the method of Crassin et al. [CNLE09], which accesses a certain level of the SVO depending on the cone radius. Each voxel stores the object's macroscopic distant field and its variance, the distribution of normals as a Gaussian lobe with a variance, and a visibility. The representation can render view-dependent effects of detailed geometry in real time. However, it is not able to reproduce objects that behave like a volume at far distances, such as grass, trees, and semi-transparent material.

Neyret [Ney98] proposes a representation to encode generic objects into a hierarchical volumetric model. Data are organized into an octree that stores opacity and distribution of normals encoded as an ellipsoid. Following this idea, Heitz et al. [HDCD15] introduce the SGGX distribution, a representation to approximate spatially-varying properties of anisotropic microflake-based participating media. It is built on the microflake theory [JAM*10]. A microflake distribution is represented with the projected area of an ellipsoid. The distribution of normals in a voxel has a compact storage (six bytes per voxel) that allows linear interpolation, analytical evaluation, and importance sampling. It can model surfaces and fiber-like materials by approximating volumetric specular and diffuse phase functions. The achieved results are visually improved when compared to previous work, but density downsampling and visibility factors are neglected, which may lead to inconsistent LODs in some cases. Unfortunately, the symmetry of the ellipsoid may generate normals that are not part of the original distribution.

As linear downsampling may lead to brightening in the final image due to the weakening of intrinsic shadowing structures, Zhao et al. [ZWDR16] propose to optimize single-scattering albedos and phase functions together, while maintaining good representation of heterogeneous and anisotropic media based on SGGX. The solution achieves good downsampling results with much lower memory consumption, but it requires a long training stage for each rendered object, and voxel clustering may lead to artifacts on spatially-varying datasets. Later, Loubet and Neyret [LN18] introduce a new microflake model that characterizes and precomputes self-shadowing, single and multiple scattering using closed-form expressions based on trigonometric lobes, and importance sampling. However, dense voxels may lead to incorrect self-shadowing, and colored multi-fiber datasets were not investigated in their work.

Complex 3D objects, such as trees, may contain both macro-surfaces (meshes) and subresolution (volumes) at a given scale, which implies that using only surface or volume LODs could lead to inaccurate rendering results. Loubet and Neyret [LN17] propose to mix both representations. An automatic macro-surface analysis looks for large connected surfaces to separate the macro-surfaces of the object. Each vertex from a macro-surface stores SGGX parameters that are prefiltered with edge-collapse [Hop96]. The volumetric part of an object is constructed by a voxelization method that casts rays from each voxel center to gather the object data (albedo and distribution of normals) and calculates a probability of occlusion; this probability is used to estimate voxel density. Finally, the volumetric part of the object is rendered in a way similar to Heitz et al. [HDCD15]. The solution generates good-quality images with low-memory consumption and fast mesh prefiltering, but the macro-surface analysis fails on disconnected or rough surfaces. Also, SGGX is limited to a single lobe, thus is unable to render complex appearances.

To represent surface-like and volumetric appearances within a single volumetric representation, Vicini et al. [VJK21] introduce a parametric non-exponential transmittance model that is able to transit between linear transmittance (opaque surfaces) and traditional exponential transmittance (volumetric scene). First, for each non-empty voxel, rays are traced to determine local appearance parameters, and gradient descent is performed to optimize representation parameters (extinction coefficient and transmittance mode). To capture the correlation effects across voxels and render surface-like objects, rays that hit an object outside the voxel being optimized are used to compute the transmittance model parameters. Unfortunately, there is no guarantee that the hits outside the voxel are correlated to the objects within it, which may lead to an overestimated transmittance.

Bako et al. [BSK22] propose a multi-scale LOD framework for prefiltering scenes with complex geometry and materials. The data are stored in an SVO and a novel transmittance function is developed. To build the representation, the phase function, albedo, and coverage mask for each voxel at every LOD scale are precomputed, and a single network is trained to compress the data into small latent feature vectors that are unpacked by a lightweight decoder. The representation is able to reproduce complex geometry and materials, local occlusions, and specularities. However, their reported network training takes approximately between half a day to two days on a 256-GPU computer.

Aiming at voxel-based real-time global illumination, Cosin and Patow [CP22] propose a clustering algorithm for voxels based on similarity of normal directions, and an efficient data structure to store the voxelization and clustering information. Their rendering can produce single bounce diffuse indirect illumination in real time. However, no treatment of voxel opacity is performed, and the distribution of normals within a voxel is approximated to a single normal, which may not adequately reproduce the voxel appearance of complex scenes.

3. Our Approach

Our volumetric representations aim to better approximate volumetric directional information in a voxel when viewed from a distance. These directional information include distributions of normals, of colors (also referred to as albedos), as well as opacities. Our representations are suitable but not specially intended for voxels with homogeneous data. Such data can be easily approximated with a single average, or with a smoothly changing value encoded with low-order spherical harmonics. Our representations are more adapted for voxels with complex content that needs filtering, and that varies at times abruptly when observed from different directions. One could interpret similar configurations as being more appropriate for higher/coarser voxels in an octree-based representation of a complex scene. While we can obviously also encode simpler content in the lower voxels, the extra data required by our representations may not be necessary at these lower levels. In fact with very fine voxels, one color and one normal could be enough, and it has been used successfully before. However, it is at a coarser, more aggregate level that a simple color and normal will not work.

To achieve this improved appearance, we introduce a first representation based on a *virtual mesh* to approximate voxel appearance. An additional representation, a low-resolution subgrid of opacities, is used to approximate the angular occlusion (or attenuation) due to a voxel. Combined, they improve the appearance of the voxel's filtered content, being surface-like or volumetric, when the voxel projection occupies a full pixel or a sub-pixel.

3.1. Overview

A virtual mesh encodes the appearance of a voxel from the distribution of normals and of colors sampled from the portion of the scene contained in the volume of the voxel. It does not represent the shape of objects, but normals and colors. The virtual mesh is constructed from a set of points sampled within the voxel. Each point is visible from "outside" the voxel from at least one direction.

At the precomputation stage, for each non-empty voxel, its volumetric representation is built as follows: (1) Rays are randomly traced, starting from outside the voxel; if an intersection occurs within the voxel, its normal, color, and opacity are added to a temporary list for that voxel. (2) A virtual mesh is afterward constructed in the shape of an ellipsoid (actually an octahedron with triangular faces) fitted to the three eigenvectors from the gathered normals; we use the vertices of the octahedron faces to encode the distributions of normals and the faces for the colors. (3) The voxel opacity subgrid is computed using the rays' visibility/opacity from Step (1) and accumulated along the ray's traversed subgrid elements. An overview of the method is illustrated in Figure 1.

At rendering time for a given voxel, its virtual mesh is sampled by tracing parallel rays from the direction of interest in order to generate visible normals and colors from that direction. They are used to shade according to the voxel's content. The subgrid of opacities are projected as spheres in the same direction to compute its

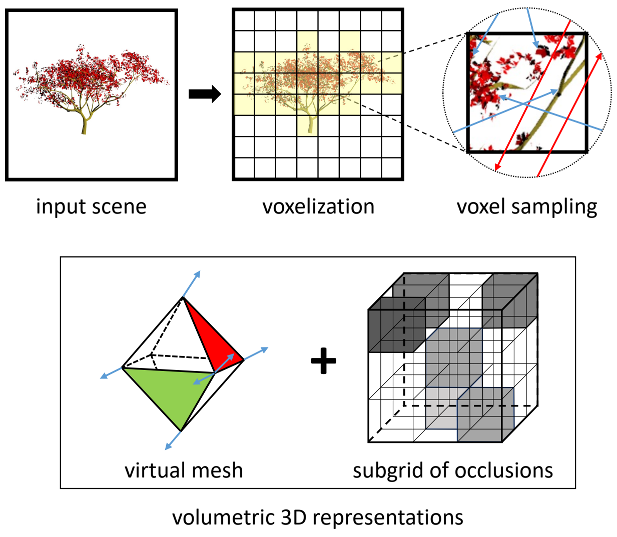

Figure 1: *Precomputations of our representations. First, the input scene is voxelized. Each non-empty voxel (marked in yellow) is sampled with random rays to gather information of normals, colors, and opacities. Then, the data from the sampling stage are used to generate a virtual mesh that approximates the distributions of normals and of colors, and a $3 \times 3 \times 3$ subgrid is generated that encodes the voxel's opacities.*

combined approximative directional opacity. It determines how the shaded voxel partly blocks farther voxels.

The following sections explain in more details all these steps.

3.2. Voxel Sampling

To build our volumetric representations, we sample data contained in a voxel by randomly and uniformly tracing rays from outside around the voxel in an unbiased way [SLH*19]. First, a direction is computed by picking a random point P on the surface of a unit sphere, properly weighted in a latitude-longitude mapping. The voxel is projected in that direction and a point within this projected area is randomly selected, to which the ray is shifted. Figure 2 illustrates this method.

Each voxel has a list of triangles intersecting its volume. From that list, the sampling ray may intersect the closest triangle (a) within the voxel, (b) outside the voxel, or (c) miss all triangles. For case (a), its intersection data (normal and color) are stored in the virtual mesh, and an occlusion is added to the subgrid elements that the ray traversed. If the closest ray intersection is on a back-facing triangle, or for case (b), only an occlusion is added to the traversed subgrid elements. For case (c), no occlusion occurs and the ray is considered as visible, but the number of rays traversing each subgrid element is updated. The accumulated opacity corresponds to a ratio of the number of intersecting rays over the total number of rays for each traversed subgrid element.

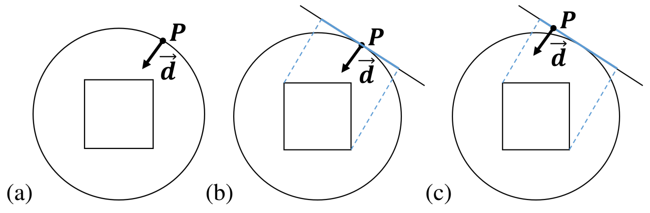

Figure 2: *Unbiased ray sampling of a voxel in 2D. (a) A point P is generated on the unit circle with direction $\vec{d}$ to its center. (b) The voxel is projected (dashed blue line) along $\vec{d}$. (c) P is randomly shifted within the projection of the voxel to P', maintaining direction $\vec{d}$.*

As a special pass after all opacities have been computed, we assign a full visibility (i.e., erase the accumulated opacity) for each subgrid element that contains no frontfacing intersection. In some sense, this special pass is inspired by Vicini et al. [VJK21] with their occlusion correlation that extends occlusion in a number of occupied voxels. However, in our case, this extension remains "local" to the current voxel.

The process is illustrated in Figure 3.

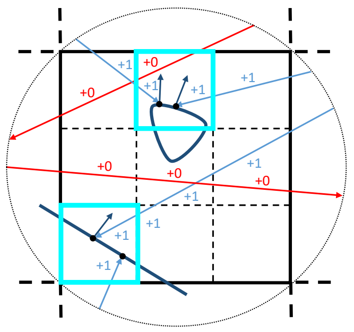

Figure 3: *Tracing rays through a 2D voxel to gather normals and colors. The opacity is accumulated in a 3^2 subgrid (3^3 in 3D) along the traversed subgrid elements. In the end of the ray tracing process, only subgrid elements with frontfacing intersections will keep their accumulated opacities, as shown in cyan; the others are marked with no opacity, i.e., full visibility.*

3.3. Virtual Mesh

Our virtual mesh is defined as an ellipsoid approximated by an octahedron. It is built from the intersection data gathered from the ray sampling step. It approximates directional distributions of visible normals and of visible colors.

In order to create the virtual octahedron triangular mesh, we leverage the ellipsoidal shape of the SGGX and its properties. We estimate the SGGX distribution parameters from the sampled normals with the method from Heitz et al. [HDCD15]. Given an SGGX distribution, a point on the unit sphere is mapped to the ellipsoidal shape by applying scaling and rotation transformations. This combined transformation is applied to the six canonical points on the unit sphere ($\pm X$, $\pm Y$, $\pm Z$) which then form the six transformed vertices of the virtual octahedron mesh.

Each triangular face of the virtual octahedron mesh can store up to three normals, one per vertex. A vertex being shared by four triangles, it may hold up to four normals in total. We use these normals in the virtual mesh to approximate a "distribution" of normals, thanks to a barycentric interpolation of its vertex data. First, each normal from the list of sampled normals is assigned to one face of the virtual mesh. To do so, we again leverage the eigenvectors of the SGGX. The orthonormal eigenvectors of the SGGX shape, which point to the vertices of the virtual mesh, divide the oriented space into eight octants, that each contains one face of the virtual mesh. For each sampled normal from the list, we determine the octant face the normal will be associated to. Note that a face may not get any normal assigned to it.

Next, for each virtual mesh triangular face f that has a distribution D of normals, we approximate its distribution by assigning three extremal normals, one per vertex of that triangle. To determine three extremal normals, we build a cone from D that is aligned with its averaged normal. The maximum angle θ_{max} between the averaged normal and the normals from D defines the opening angle of the cone. The one normal at θ_{max} is our first extremal normal. We then choose two normals on the cone, so that they enclose as many as possible of the normals in D, but as little as possible of orientations that have no normal in D. For a face that has only one-to-three normals assigned to it, we use these normals as the extremal normals and mark that face as a non-distribution approximation, because no normals other than those have been observed in the collected data.

The construction process is illustrated in Figure 4.

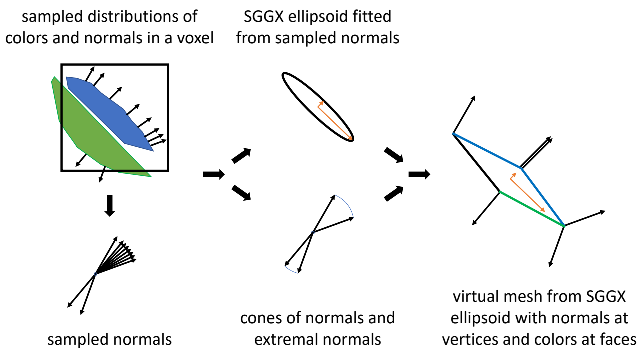
sampled distributions of colors and normals in a voxel
SGGX ellipsoid fitted from sampled normals
sampled normals
cones of normals and extremal normals
virtual mesh from SGGX ellipsoid with normals at vertices and colors at faces

Figure 4: *2D illustration of the construction of our 3D virtual mesh. Sampled normals in a voxel are used to define an SGGX ellipsoid, whose shape is used to orient the virtual mesh. A cone of normals is used to assign extremal normals to the corresponding vertices per face of the virtual mesh. Corresponding colors are also assigned to faces. A black face in the figure indicates that no color (and normal) is assigned to its face.*

When we need to extract normals from the distribution of a face, we randomly sample points in the observed direction, i.e., from the projected triangle face, and use a barycentric interpolation to compute normals from these projected points. For non-distributions, we simply pick the normal from the closest projected triangle face (when there is a single normal for that face) or vertex (when there are two-to-three normals).

We observed that in many geometric models, normals and colors are often correlated. We decided to assign one color to each virtual mesh face that corresponds to the average of the colors from the original distribution of normals for that face. This is justified because colors in shading are combined more linearly than normals.

In special configurations, interpolating from three colors of a face, like we do for normals, may prove more accurate. However, overall, we could not find enough reliable situations where the benefits outweigh the increased cost of memory of using eight colors per virtual octahedron mesh instead of up to 24 colors.

Projected Area Sampling

In order to sample the virtual mesh from a given direction, we use an approach based on ray-triangle intersection. Given a view direction $\vec{w}$ and a number of samples per face, we compute the perpendicular projection of the virtual mesh faces that hold data (three extremal normals and colors) into the view direction plane. We compute the projected area of each frontfacing triangle, and sum up the total projected area of all triangles. We generate a random point P_i in the projected triangle with a warping method [SLH*19] and create a ray with origin P_i and direction $\vec{w}$. The ray-triangle intersection provides the uv coordinates used for a barycentric interpolation of the vertices' extremal normals. The color is simply the stored average color for the corresponding triangle.

If the number of samples is lower than the number of frontfacing faces, one could use the projected areas of the faces to weigh the ray origin, and then proceed as explained just above.

3.4. Opacity Subgrid

In the method of Vicini et al. [VJK21], rendering of surface-like objects is improved by extending intersections for rays outside the voxel itself in order to capture some form of correlation effects across voxels. Unfortunately, that notion of correlation between voxels is not as intuitive to adapt to any given scene, and it may lead to over-estimation of occlusions.

To reduce such issues, we subdivide a voxel into a low-resolution subgrid (subvoxels that we call instead subgrid elements) to capture correlation effects across subgrid elements using only information local to that voxel. A non-empty voxel is subdivided into $3 \times 3 \times 3$ subgrid elements that had already stored the average opacity of each subgrid element. To compute the opacities, the intersections between the sampled rays (Section 3.2) and subgrid elements are computed. If a ray traverses a subgrid element, the occlusion value (1 for a hit and 0 for a miss) is added to the subgrid element's average opacity. At the end of the process, for each subgrid element without any intersection point, we replace its value with a null opacity, i.e., full visibility.

Directional Opacity

When rendering, to compute the occlusion due to a voxel in a view direction, the voxel is projected on a temporary view plane along that same view direction. We define a *square* window that bounds the parallel projection of the voxel. This window is subdivided into 4×4 pixels. A subgrid element (of the voxel) may contribute to the opacity of a number of these pixels. Each subgrid element is replaced by an enclosing sphere that is efficiently projected as a circle in the window. For each covered pixel, we compute analytically the overlapping area A_s of the subgrid-element circle and a circle enclosed by the pixel. The opacity of the subgrid element α_s is multiplied by the coverage A_s of the pixel to give an approximation of its contribution to the opacity of the pixel. We keep only the largest such contribution for each pixel. The final directional opacity of the entire voxel corresponds to the average of the final subgrid elements contributions for all 4×4 pixels.

The process is illustrated in Figure 5.

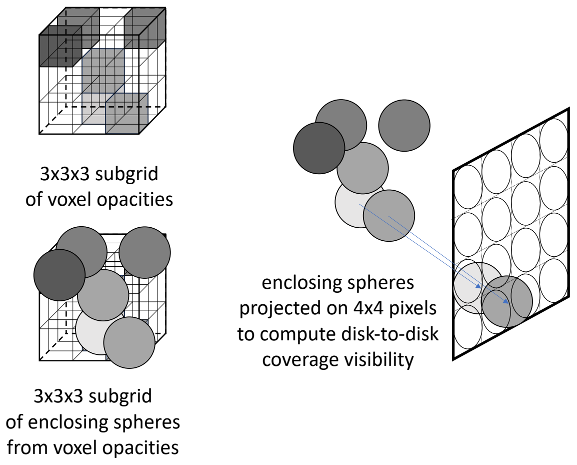

Figure 5: *The $3 \times 3 \times 3$ subgrid of opacities are converted into enclosing spheres. These spheres are projected on a 4×4 grid of pixels. The ratios of coverages of the disk-to-disk are computed to determine the directional opacity for a voxel in the view direction.*

4. Implementation

The source code of our implementation and related material are available from the `cjsb.github.io/hpg2023/` website.

For precomputations, we implemented a CPU-based ray tracer of triangular meshes, with textures, and an octree acceleration structure. It is used to compute our volumetric representations. To collect data in a voxel, we cast 150 rays around the voxel.

For real-time display, we implemented a GPU-based ray tracer of voxels developed with CUDA, with an octree acceleration structure. It handles primary and shadow rays. In a precomputation, we cast a shadow ray from a light source toward the center of a voxel. The ray traverses the scene's voxels between the light and the voxel to shadow, and the opacity is computed as detailed in Section 3.4. Similarly to Vicini et al. [VJK21], we avoid adding the contribution of the voxel itself to the shadow occlusion. We also avoid intersections of voxels that are closer than $1.5\times$ the size of a voxel. Their extension prevents to lose too much light during shadow computations, which would make the image darker in shadowed regions. This can be seen in Figure 6.

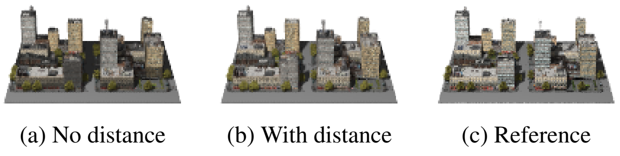

Figure 6: *Impact of the minimal distance ($1.5\times$ size of a voxel) to shadow ray intersection from Vicini et al. [VJK21], which is integrated in our method.*

On the GPU, the virtual mesh is composed by three eigenvectors (char3) and three scaling factors (uchar). The three normals (char3) of each face are stored along with one color (uchar3) per face. A boolean per triangle stores if its face encodes a distribution or a fixed set (1-to-3) of normals. One opacity value (uchar) is stored per subgrid element, thus, 27 opacities. Our representations require approximately 143 bytes of memory per voxel; the storage per test scene can be found in Table 2.

As a ray intersects a voxel, the virtual mesh is sampled to get normals and colors given the ray direction. We generate as many as 40 samples per visible face, but only samples visible to the camera are used in shading. The average shading is applied as the voxel color and the opacity is computed as discussed in Section 3.4.

We render a scene by tracing four rays per pixel through the voxels, from a set of precomputed jittered patterns. A ray traverses the voxels until it reaches an accumulated opacity threshold of 0.001. When a voxel is traversed by a ray, we compute its shading and its opacity for the direction from the center of the voxel to the camera position. Both values are cached and reused if another ray intersects that voxel. Cached values are cleared if the camera-scene configurations change.

5. Results

We compare our results with those of two different representations per voxel: (1) a naïve approach that stores one average for the normals, the colors, and the ray occlusions, and (2) the representation from Vicini et al. [VJK21] that uses a non-exponential transmittance model for opacity and SGGX [HDCD15] to encode the distribution of normals. However, we use samples coming only from triangles intersecting a given voxel. This is different from the original work from Vicini et al. [VJK21] that extends rays to intersect triangles up to a certain distance away from that voxel. Figure 11 at the last page and a video sequence as supplemental material provide some general visual comparisons with ground truth.

Our representations are able to approximate both surface-like and volumetric appearances as illustrated in Figure 11, *Island*, third column. The island as a whole and the palm tree trunks are treated as large occluders, since they entirely block visibility; the leaves (fronds) are treated as a volumetric distribution of small objects. The finest resolution of 256^3 voxels is able to approximate the fine mesh-based scene with occlusions, shading, and colors, as well as the shadows from the light above to the right. The resolution is not fine enough though, and slightly extends the size of features. Because we are using a conservative maximum occlusion for combined opacities, cast shadows also appear slightly larger.

Res.	Method	*Bunny* MSE	*Bunny* SSIM	*Trees* MSE	*Trees* SSIM	*Island* MSE	*Island* SSIM	*City* MSE	*City* SSIM	*Bistro* MSE	*Bistro* SSIM
128^3	Averages	0.0201	0.8581	0.0180	0.7888	0.0051	0.9071	0.0113	0.7613	0.0107	0.8044
	[VJK21]	0.0053	0.8383	**0.0079**	**0.8786**	0.0038	0.9162	0.0059	0.8180	0.0096	0.7977
	Ours	**0.0043**	**0.9133**	0.0104	0.8221	**0.0035**	**0.9294**	**0.0045**	**0.8548**	**0.0049**	**0.8731**
256^3	Averages	0.0205	0.8690	0.0205	0.7675	0.0065	0.8825	0.0087	0.8298	0.0115	0.7999
	[VJK21]	0.0050	0.8342	0.0100	**0.8439**	**0.0039**	0.8937	0.0050	0.8573	0.0067	0.8243
	Ours	**0.0029**	**0.9236**	**0.0093**	0.8250	0.0040	**0.9113**	**0.0036**	**0.8851**	**0.0039**	**0.8702**

Table 1: *Error metrics (MSE and SSIM) with a reference image at voxelized resolutions of* 128^3 *and* 256^3*. Best values (lowest for MSE and highest for SSIM) are in bold. Figure 8 shows the distribution of errors for two scenes, where some uniform background has been cropped.*

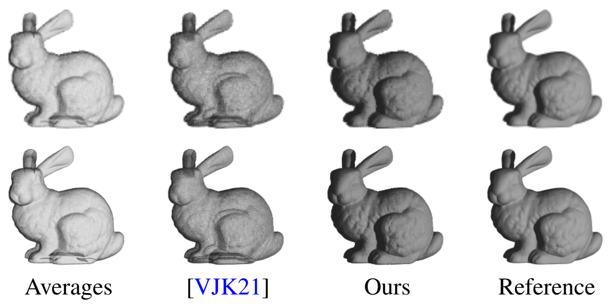

Figure 7: *The Bunny scene at two voxelized resolutions (top:* 128^3 *and bottom:* 256^3*) and rendered at the equivalent image resolutions (top:* 128^2 *and bottom:* 256^2*). The first column uses the average color, normal, and opacity per voxel. The second column uses the method from Vicini et al. [VJK21]. The third column uses our method. The fourth column is a direct ray casting (1024 rays per pixel) of the scene's mesh.*

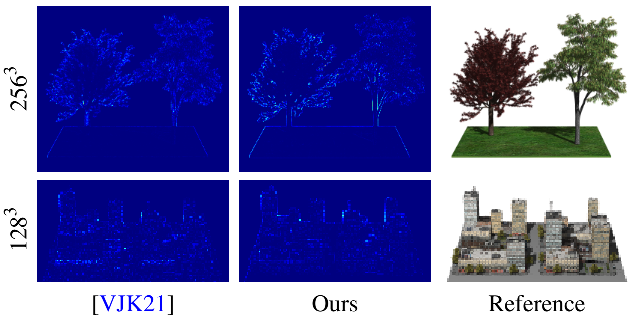

Figure 8: *Distribution of MSE errors in two scenes.*

Figure 7 and Figure 11 at the last page show comparison details for three volumetric methods on four scenes voxelized at two resolutions and displayed at the equivalent image resolutions (empty space at top/bottom was cropped to better fit in the table). The method using averages (color, normal, opacity) suffers from too much transparency and faded colors and shading. The method of Vicini et al. [VJK21] improves on the opacity and shading, but also suffers at some locations from the same problems. Figure 9 zooms in two portions of the *Island* scene. The two methods, Averages and Vicini et al. [VJK21], are unable to properly handle different occlusions, illustrated by the blue from the sea leaking through the large occluder formed by the island (red inset), and the continuity of the palm tree trunks against the white background (blue inset). To be fair with the method of Vicini et al. [VJK21], extending the opacity test outside a voxel does help in some situations, but one needs to be careful to set the distance properly for a given scene. Our method improves on the colors, normals, and opacities, even though it has a tendency of extending opacities due to the maximum opacity test, which shows up as larger or denser voxels and shadows.

To quantitatively compare the different representations, we computed the Mean Squared Error (MSE) and the Structural Similarity Index (SSIM) [WBSS04]against a ground-truth image computed as a ray-traced version of the polygonal scenes, with 1024 rays per pixel, averaged with a simple box filter. As shown in Table 1, our representations have a better score in both metrics in most of the cases. However, in the *Trees* scene, since we use a conservative maximum occlusion that makes the small occluders larger, Vicini et al. [VJK21] was able to better reproduce the appearance of the trees. Rendering at a higher resolution reduced this difference. Also, the method of Vicini et al. [VJK21] gets a better MSE score for the *Island* scene with a 256^3 resolution, but with a minor difference compared to ours, while our method gets a better SSIM score. Figure 8 shows the distribution of MSE errors in two typical images. In the top row, our method is slightly better (MSE = 0.0093 vs ours = 0.0100), but we observe how worse errors are distributed along contours. In the bottow row, our method is better overall (MSE = 0.0045 vs ours = 0.0059).

A finer resolution of voxels reduces thicker silhouettes, but at an increase (about $8\times$ for an octree subdivision) of memory and computations. As an experiment, we rendered with Vicini et al. [VJK21] the *Island* scene at 256^3 in a 256×256 image, and then averaged 2×2 pixels into one pixel. When comparing the resulting 128×128 image with the ground truth, their MSE = 0.0038 remained the same (ours at 128^3 is at 0.0035, cf. Table 1), but their SSIM = 0.9085 got worse (ours at 0.9294). This illustrates that using a previous method at a finer scale does not necessarily improves results. Our method remained superior in this example.

Our representations can reproduce diffuse and specular shadings, as shown in Figure 10 (top). Since we encode the distribution of normals using only three extremal normals, specularity may lose some of its sharpness when the distribution of normals is spread

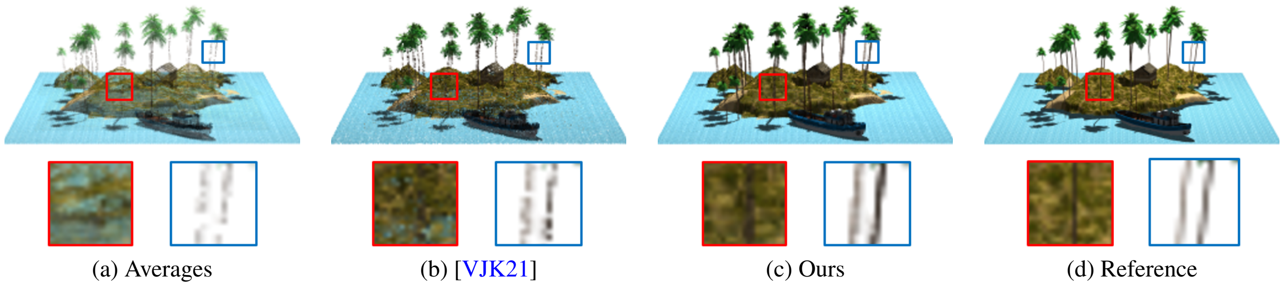

Figure 9: *Comparison of different parts of the Island for a voxelization of* 256^3. *Both methods of Averages and Vicini et al. [VJK21] are unable to fully represent different occlusions, as they show leakage from the voxels containing the island (red inset) and in the palm tree trunks (blue inset). Our method can better reproduce both of these occlusions, as it reduces such leakage. It also shows more details, for instance, palm tree trunks in front of the island can be distinguished more clearly (red inset).*

out a little more by the cone of normals. However, compared to previous work, while it is known that SGGX is able to reproduce well both diffuse and specular shadings, the opacity computation suffers from a stronger leakage, which contributes to reduce much the appearance of the shading.

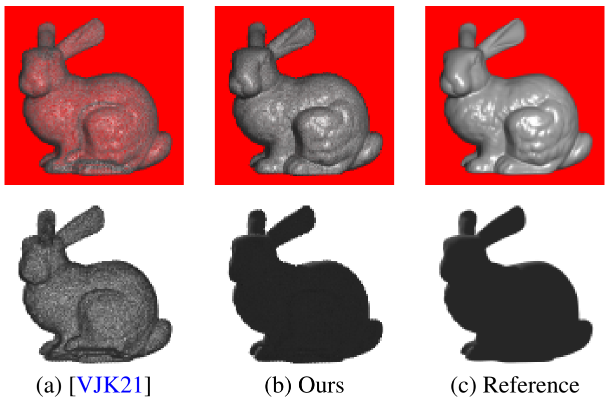

Figure 10: *Comparison of diffuse and specular shading results for a voxelization at* 128^3 *and rendering at* 128^2. *Top: With a light in front of the Bunny, frontfacing normals are properly handled, but leaking of opacities by Vicini et al. [VJK21] reduces their contributions. Note: A red background has been added to better differentiate transparency vs. specularity. Bottom: With a light behind the Bunny, combined SGGX symmetry and opacity leaking result in improper shading from backfacing normals lit by the light source.*

In cases where the visible normals of the distribution of normals are not visible from the light source, our representations are able to better reproduce the shading of the object with much less light leakage, as shown in Figure 10 (bottom) where the light source is located behind the Bunny. It is noteworthy that the SGGX symmetry is forced to generate normals also in the opposite direction than the sampled normals. These shaded normals, in addition to weaker opacity, result in such shading appearing in Figure 10(a) (bottom).

In terms of performance, precomputations (software implementation, single thread, no parallelism) for our larger scenes take on average 22.6 sec. at 128^3, and 101.6 sec. at 256^3, on an Intel i7-8700K with 32 GB of memory. A frame on the same machine with an NVIDIA RTX2080, with shading and shadows from one light source, is rendered on average at 59 fps at 128^3, and 27 fps at 256^3. The shadows are precomputed and caching is used in order to achieve more interactive frame rates. More detailed statistics for precomputations and rendering can be found in Table 2.

		Memory		Precomput.		Rendering	
Scene	Tris	128^3	256^3	128^3	256^3	128^3	256^3
Bunny	144k	10.6	52.4	15	66	53	27
Trees	540k	18.7	106.2	33	150	48	19
Island	307k	4.7	22.4	8	34	76	40
Bistro	3420k	19.9	135.5	29	156	46	18
City	635k	8.9	48.0	28	102	72	30

Table 2: *Number of triangles, memory (MB), precomputation time (seconds), and rendering perfomance (fps) with four rays per pixel for our representations per test scene.*

Our representations are about 13× larger than Vicini et al. [VJK21], which requires 11 bytes (6 bytes for the SGGX distribution of normals, 2 bytes for their occlusion model, and 3 bytes for the color average). Our representations are about 20× larger than the method of Averages, which requires 7 bytes (3 bytes for the color, 3 bytes for the normal, and 1 byte for opacity). As an example, the *City* scene at resolution 128^3 requires 8.9 MB with our representations, and 0.6 MB for Vicini et al. [VJK21], and 0.4 MB for Averages. The same scene is rendered at 72 fps with our representations and 96 fps with Vicini et al. [VJK21] and Averages. However, image quality is not the same for all of the cases as our representations can better reproduce the different appearances as illustrated throughout this paper.

6. Conclusions

Volumetric representations offer many benefits to render efficiently complex scenes, simulate light transports, and filter appearances. They are also very well suited to hierarchical computations. This makes them a great solution for highly constrained real-time rendering in video games. Major contributions have demonstrated their potential for real-time display (e.g., [CNLE09,HDCD15]) and their suitability for light transports (e.g., [CNLE09, CNS*11, CP22]). However, in order to gain popularity, key limitations must be overcome, a major one being their visual quality.

In this paper, we studied occlusion and distributions of colors and normals as they vary under changing view directions. We introduced a virtual mesh and an opacity subgrid to capture variations of appearance of a volume. They build on the idea of fast rendering steps of simple structures to approximate directional changes. While the two structures increase memory requirements and computations, they allow for better and more flexible approximations that are key for improving visual quality of volumetric representations. We showed that for moderately complex scenes, our rendering prototype offers a reasonable compromise that is a step toward better approximations.

Our voxel-based data have been designed to be suitable for hierarchical treatment. However, at this moment, precomputations are performed on the entire polygonal scene for each resolution of voxels, i.e., for each level of an octree. In a near future, we will investigate solutions to merge distributions of normals, colors, and opacities directly from the structures at a finer resolution, rather than precomputing them from the polygonal scene itself. A form of interpolations between voxel data at adjacent LODs would help to generalize the structures for MIPmapping rendering strategies.

Because our structures are general, we should also be able to integrate them in light transport simulations in order to provide a more complete volumetric real-time rendering system, as well as for high-quality offline rendering. In particular, the many-light methods could adapt well in the context of global illumination for volumetric rendering.

Finally, a much more challenging scenario involves animated scenes. While we could always differentiate between static and dynamic scene elements, their impact on precomputations still remains dominant in volumetric representations.

Acknowledgements

This work was financially supported in Canada by the NSERC Discovery of Pierre Poulin.

References

[BAC*18] Burley B., Adler D., Chiang M. J.-Y., Driskill H., Habel R., Kelly P., Kutz P., Li Y. K., Teece D.: The design and evolution of disney's hyperion renderer. *ACM Trans. Graph. 37*, 3 (July 2018). doi:10.1145/3182159. 1

[BSK22] Bako S., Sen P., Kaplanyan A.: Deep appearance prefiltering. *ACM Trans. Graph. 42*, 2 (Apr. 2022). doi:10.1145/3570327. 3

[CB04] Christensen P. H., Batali D.: An irradiance atlas for global illumination in complex production scenes. In *Proc. Eurographics Symp. on Rendering* (2004), EGSR'04, p. 133–141. 2

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. Symp. on Int. 3D Graph. and Games* (2009), I3D '09, p. 15–22. doi:10.1145/1507149.1507152. 2, 9

[CNS*11] Crassin C., Neyret F., Sainz M., Green S., Eisemann E.: Interactive indirect illumination using voxel cone tracing. *Comput. Graph. Forum 30*, 7 (2011), 1921–1930. doi:https://doi.org/10.1111/j.1467-8659.2011.02063.x. 2, 9

[CP22] Cosin Ayerbe A., Patow G.: Clustered voxel real-time global illumination. *Computers & Graphics 103* (2022), 75–89. doi:https://doi.org/10.1016/j.cag.2022.01.005. 3, 9

[DSSC08] Daniels J., Silva C. T., Shepherd J., Cohen E.: Quadrilateral mesh simplification. *ACM Trans. Graph. 27*, 5 (Dec. 2008). doi:10.1145/1409060.1409101. 1

[HDCD15] Heitz E., Dupuy J., Crassin C., Dachsbacher C.: The sggx microflake distribution. *ACM Trans. Graph. 34*, 4 (July 2015). doi:10.1145/2766988. 1, 2, 3, 4, 6, 9

[HN12] Heitz E., Neyret F.: Representing appearance and prefiltering subpixel data in sparse voxel octrees. In *Proc. ACM SIGGRAPH / Eurographics Conf. on High-Performance Graphics* (2012), EGGH-HPG'12, p. 125–134. 2

[Hop96] Hoppe H.: Progressive meshes. In *Proc. Conf. on Comput. Graph. and Int. Tech.* (1996), SIGGRAPH '96, p. 99–108. doi:10.1145/237170.237216. 1, 3

[JAM*10] Jakob W., Arbree A., Moon J. T., Bala K., Marschner S.: A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph. 29*, 4 (July 2010). doi:10.1145/1778765.1778790. 2

[KK89] Kajiya J. T., Kay T. L.: Rendering fur with three dimensional textures. In *Proc. Conf. on Comput. Graph. and Int. Tech.* (1989), SIGGRAPH '89, p. 271–280. doi:10.1145/74333.74361. 1, 2

[LK11] Laine S., Karras T.: Efficient sparse voxel octrees. *IEEE Trans. Vis. and Comput. Graph. 17*, 8 (2011), 1048–1059. doi:10.1109/TVCG.2010.240. 2

[LLT*20] Lescoat T., Liu H.-T. D., Thiery J.-M., Jacobson A., Boubekeur T., Ovsjanikov M.: Spectral mesh simplification. *Comput. Graph. Forum 39*, 2 (2020), 315–324. doi:https://doi.org/10.1111/cgf.13932. 1

[LN17] Loubet G., Neyret F.: Hybrid mesh-volume lods for all-scale pre-filtering of complex 3d assets. *Comput. Graph. Forum 36*, 2 (May 2017), 431–442. doi:10.1111/cgf.13138. 3

[LN18] Loubet G., Neyret F.: A new microflake model with microscopic self-shadowing for accurate volume downsampling. *Comput. Graph. Forum 37*, 2 (2018), 111–121. doi:https://doi.org/10.1111/cgf.13346. 1, 3

[Ney98] Neyret F.: Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Trans. Vis. and Comput. Graph. 4*, 1 (1998), 55–70. doi:10.1109/2945.675652. 1, 2

[SLH*19] Shirley P., Laine S., Hart D., Pharr M., Clarberg P., Haines E., Raab M., Cline D.: Sampling transformations zoo. In *Ray Tracing Gems*. Springer, 2019, pp. 223–246. 4, 5

[VJK21] Vicini D., Jakob W., Kaplanyan A.: A non-exponential transmittance model for volumetric scene representations. *ACM Trans. Graph. 40*, 4 (July 2021). doi:10.1145/3450626.3459815. 3, 4, 5, 6, 7, 8, 10

[WBSS04] Wang Z., Bovik A., Sheikh H., Simoncelli E.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612. doi:10.1109/TIP.2003.819861. 7

[YLH18] Yi R., Liu Y.-J., He Y.: Delaunay mesh simplification with differential evolution. *ACM Trans. Graph. 37*, 6 (Dec. 2018). doi:10.1145/3272127.3275068. 1

[ZWDR16] Zhao S., Wu L., Durand F., Ramamoorthi R.: Downsampling scattering parameters for rendering anisotropic media. *ACM Trans. Graph. 35*, 6 (Nov. 2016). doi:10.1145/2980179.2980228. 2

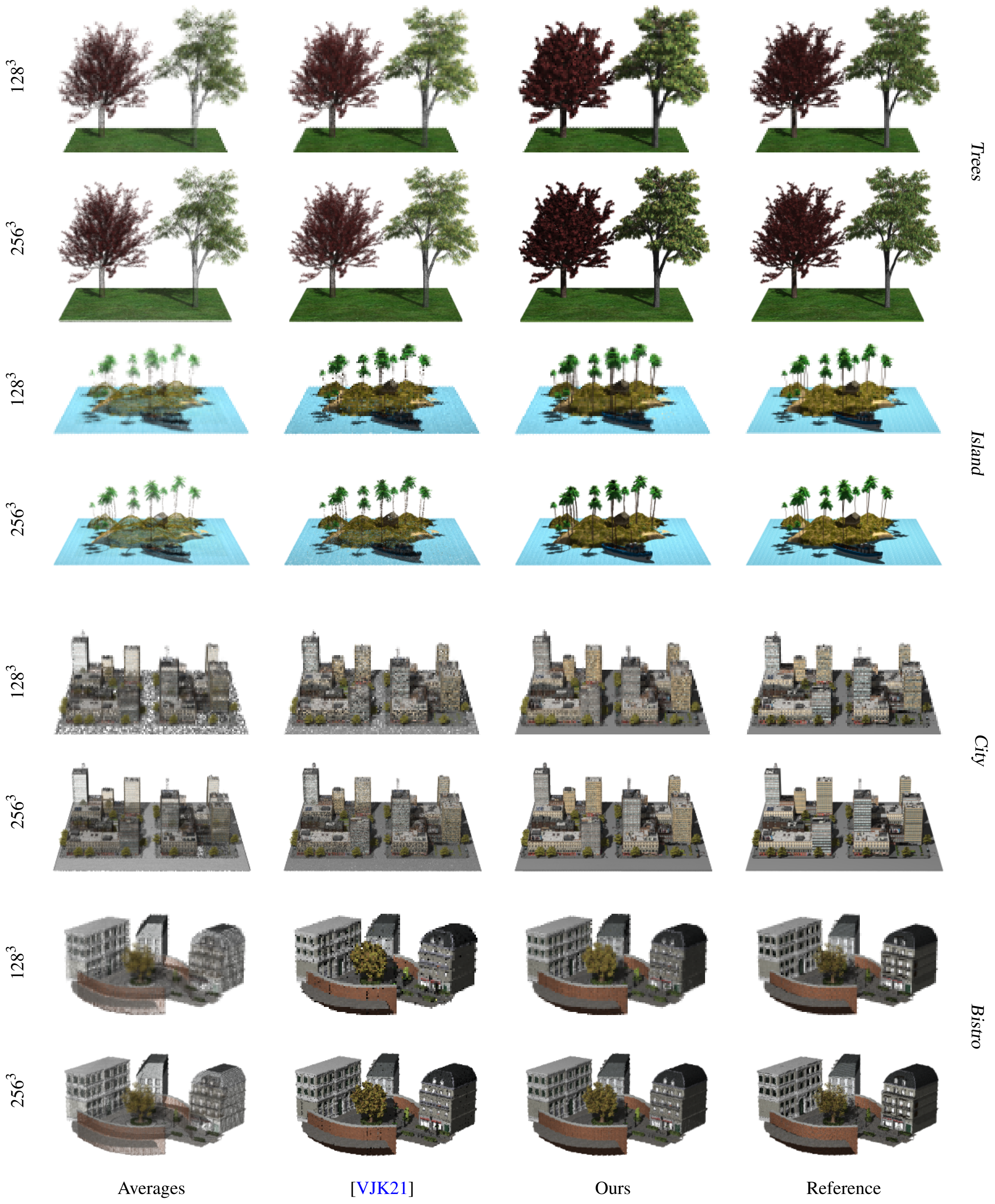

Figure 11: *Four test scenes at two voxelized resolutions (128^3 and 256^3) and rendered at the equivalent image resolutions (128^2 and 256^2). Note that empty spaces at the bottom and top of some images have been cropped to better fit in the table. The first column uses the average color, normal, and opacity per voxel. The second column uses the method of Vicini et al. [VJK21]. The third column uses our method. The fourth column is a direct ray casting (1024 rays per pixel,box filtering) of the scene's mesh.*