Displaying Large User-Generated Virtual Worlds from the Cloud

Tahir Azim*, Ewen Cheslack-Postava†, Philip Levis‡
Stanford University

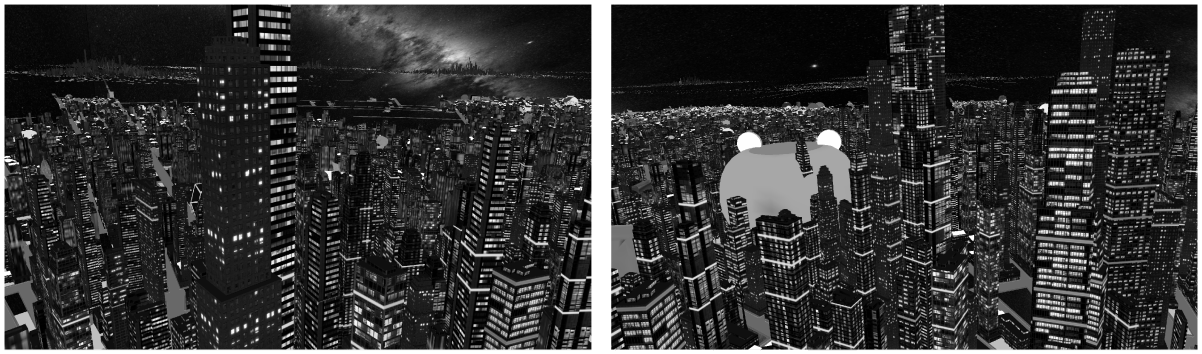


Figure 1: *Screenshots from our system of a world with over 540,000 objects composed of a total of 144 million triangles referencing ~6,000 unique textures and occupying over 24 GB on disk. This high-visibility viewpoint observes more than half the world, including five distant cities as complex as the one the user is flying over. User-generated models, such as the green piggy bank in the scene on the right, were randomly placed in the world. The client is able to render this large, complex and diverse scene between 20 and 35 frames per second, while downloading only 1.4 GB of data from the cloud, a 94% reduction.*

Abstract

Rich, large-scale user-generated virtual worlds have been imagined in the realm of fiction for decades. Such worlds, however, present significant technical challenges due to the limitations of available network and graphics resources. Since the world is user-generated, its content has to be stored in a shared, networked resource such as the cloud. Further, user-generated content is not optimized for efficient rendering, so additional processing is needed to display it efficiently in the presence of limited graphical resources.

This paper presents an approach to efficiently display a complete view of a user-generated world at scale. The key insight is that such worlds have a high degree of coherence, which enables us to deduplicate many 3D models. This greatly reduces the amount of data that needs to be transferred over the network to display the world. The deduplicated models also lend themselves to a new method of simplification, called instance-aware simplification, which efficiently simplifies 3D models consisting of many instances of the same geometry.

1 Introduction

Fictional accounts of richly detailed, large-scale, user-generated virtual worlds abound in sci-fi literature, such as Snow Crash, Ready Player One and Neuromancer. The sheer scale of such worlds mean most content is created primarily by end users: even games today such as Spore rely on more user-generated content due to rising production costs. But all of the content created by hundreds of thousands or millions of users cannot be stored locally on every disk. The content must be stored in a shared, networked resource such as the cloud and delivered to clients dynamically. Finally, since the virtual world content is not owned by a central authority, there is no assurance for object mesh and texture quality.

Serving large amounts of user-generated graphical content from the cloud presents three principal research problems to storage, networks and GPUs. First, user-generated models are not optimized for rendering in terms of their triangle count or their textures. Rendering just a few of these models can overburden the GPU. Second, a large world can have many high visibility locations where hundreds of thousands of objects may be in a client's field of view. Downloading the models for these objects can take a long time. Finally, rendering all these objects as fully-detailed textured meshes at an interactive frame rate is infeasible, especially since the GPU is limited by the number of batches the CPU can generate, generally a few thousand per frame.

Existing virtual worlds, such as Second Life and World of Warcraft, address these three problems by displaying only a small subset of objects within a small, fixed distance from the viewer. While this simplifies their system design and yields good frame rates, viewers cannot see distant, large objects such as mountains or skyscrapers. Substantial prior work has addressed lifting this limitation to render large models or scenes when all data is available locally [Funkhouser et al. 1992; Erikson et al. 2001; Crassin et al. 2009; Rusinkiewicz and Levoy 2000] and when streaming individual meshes over the network [Hoppe 1996; Rusinkiewicz and Levoy 2001]. However, streaming of large scenes has thus far relied mostly on culling and level of detail for each object [Schmalstieg and Gervautz 1996; Teler and Lischinski 2001], which still limits the display to only a few thousand objects at a time. More recent work [Cheslack-Postava et al. 2012] has proposed an approach for generating aggregates (impostors representing groups of objects) and selecting a small subset of objects and aggregates that yield a complete view of the world.

This paper, in contrast to this prior work, focuses on how how to generate and simplify aggregates in the cloud in order to overcome the limitations of client network bandwidth, GPU RAM, and GPU batch count. The approach relies on two main ideas: scene coherence and instancing. Scene coherence means that in a logically consistent world, there will often be similar or duplicate meshes clustered in close proximity. Instancing is the technique of representing many copies of the same geometry by storing one copy and applying transformations to create more copies.

The approach exploits scene coherence to maximize instancing

*email:tazim@cs.stanford.edu
†email:ewencp@cs.stanford.edu
‡email:pal@cs.stanford.edu

within aggregates, enabling efficient simplification and smaller network transfer cost. It groups graphically similar objects into aggregates, and deduplicates highly similar objects within each aggregate to reduce its size and increase the prevalence of instancing. It generates instanced meshes for these aggregates and simplifies them using a novel instance-aware simplification algorithm, which unlike existing approaches, works directly on the instanced mesh instead of the equivalent indexed triangle mesh. Texture atlasing ensures that each aggregate can be rendered with a single draw call to the GPU.

This paper makes four research contributions. First, it presents an algorithm to efficiently construct a bounding volume hierarchy (BVH) which greatly reduces the download and rendering cost of aggregates while maintaining good querying performance. Second, it presents a technique to deduplicate highly similar models within each aggregate in order to further reduce download and rendering costs. Third, it proposes a new instance-aware simplification algorithm whose resulting mesh files are up to 99.6% smaller compared to the quadric simplification algorithm. Finally, it describes an approach to reconfigure the textures referenced by aggregate meshes in order to enable a complex world to be rendered within the GPU's texture RAM and draw call budget. Combined, these techniques allows a client to stream a large, complex, user-generated scene consisting of over 500,000 objects and 144 million triangles over the network as well as display it at interactive frame rates.

We overview related work in the next section. Section 3 provides some background and states the problem in more detail. Section 4 presents a BVH construction approach which considers both proximity and mesh similarity. Algorithms to generate instanced 3D models for these aggregates and simplify the instanced models follow in sections 5 and 6. Section 7 details our texture management strategy and an evaluation of the system follows in Section 8.

2 Related Work

There is a large body of work in the area of rendering large virtual environments. However, most of this work assumes that all of the graphical data can fit locally on disk. Further, many existing techniques also assume static, pre-defined objects and little *churn* in terms of objects entering and leaving the system.

In contrast, our approach is designed for worlds that are too large to be stored locally and hence have to be stored in the cloud. Since the network is a low-bandwidth channel compared to local disk, it is crucial to have an approach that minimizes the amount of data sent over the network. Moreover, since the world is user-generated, we assume no pre-existing knowledge of objects and our approach is robust to churn in the world.

Mesh Simplification. Quadric Simplification [Ronfard and Rossignac 1996; Garland and Heckbert 1997], the most common mesh simplification algorithm, is an incremental approach that aims to minimize an error metric at every step. For every edge, it finds the optimal point at which the edge collapse would introduce the least error. Follow-up techniques extend this approach to consider other attributes, such as texture coordinates and normals [Garland and Heckbert 1998; Hoppe 1999].

Billboard clouds [Décoret et al. 2003] are an alternative technique for extreme simplification that projects a 3D model onto a set of planes with textures and transparency maps. This allows a complex model to be represented with something as simple as a textured cube. However, this only yields a good approximation when the object being viewed is distant. Furthermore, the additional textures can increase memory cost and computing the billboards is time-consuming.

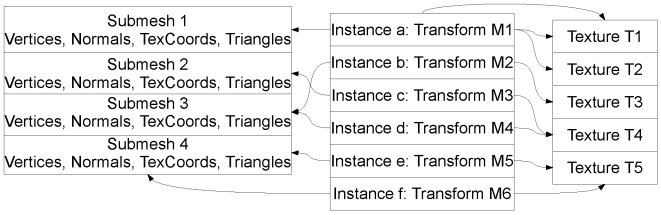


Figure 2: *The structure of an instanced mesh file.*

View-dependent simplification [Luebke and Erikson 1997] and perceptually modulated level-of-detail (LOD) [Williams et al. 2003] exploit the camera's current viewpoint to achieve higher quality rendering, but can only be run on the client side for this reason. It is important to note, however, that a client of our system could use these methods to efficiently render the aggregates and individual objects returned by our system.

A number of metrics exist to measure visual error on simplified meshes. One of the more well-established metrics is MetroMn [Cignoni et al. 1998], which has been shown in studies to correlate strongly with human perceptions of quality [Watson et al. 2001].

Instanced Meshes. These existing simplification algorithms assume that a model is an indexed triangle mesh, consisting of a set of vertices, a set of faces composed of those vertices, and other properties associated with the vertices, such as normals and texture coordinates. However, many real-world model formats, including the COLLADA standard [Arnaud and Barnes 2006], describe a 3D model as an instanced model (or equivalently, a scene graph) consisting of a set of submeshes which are instanced using different transformation matrices (Figure 2). Instances can be organized hierarchically such that their transformation is the product of all transformations from the root to the instance at the leaf.

To apply existing simplification algorithms to instanced meshes, the instanced mesh must be expanded into an indexed triangle mesh. Expanding the mesh simply means generating the overall mesh by iterating over every instance of the mesh and applying the instance's transformation matrix to its submesh. This results in geometry being duplicated for instances that reference the same submesh multiple times. In many cases, data may be duplicated tens of times as duplicate objects are grouped into aggregates (*e.g.*, trees in a forest), or even hundreds of times in an input mesh (*e.g.*, leaves on a tree).

Furthermore, expansion and subsequent simplification is a one-way transformation: once edges in one copy of a submesh are collapsed, that copy cannot be easily factored back into a single submesh along with all of the other copies. As a result, the simplified mesh may actually be larger in size than the original, albeit with lower graphical complexity. The download cost for a client increases correspondingly, sometimes by an order of magnitude.

In addition, simplifying a highly instanced mesh using existing approaches takes longer because simplification time grows with the number of edges and expanding the mesh duplicates submesh edges multiple times.

Navigating Large 3D environments. Specialized approaches exist for rendering large crowds of humans and buildings through impostors [Dobbyn et al. 2005; Cignoni et al. 2007]. In contrast, our approach is designed for arbitrary meshes.

Hierarchical LODs [Erikson et al. 2001] is perhaps the most closely related approach to our work. It groups objects into a hierarchy, applying lower LOD closer to the root to achieve high frame rates for very complex models. However, it assumes that all meshes are

available locally on disk, does not consider the multitude of textures that diverse user-generated objects may reference, and does not ensure that highly instanced meshes have small file sizes after simplification.

Chunk-based methods for massive triangle meshes such as QuickVDR [Yoon et al. 2005] and Tetrapuzzles [Cignoni et al. 2004] precompute a spatial hierarchy over the mesh so that appropriate portions of the mesh can be efficiently loaded on to the GPU and rendered. While they assume knowledge of the mesh beforehand, they could be applicable on the client side to a relatively static world.

Voxel based methods, such as Far Voxels [Gobbetti and Marton 2005] and GigaVoxels [Crassin et al. 2009], sample triangle models into voxels and introduce efficient techniques for rendering these voxels on the GPU. Despite the fact that, on a single machine, high-bandwidth channels exist to transfer these voxels to the GPU, these approaches are forced to apply novel compression techniques to load the voxel data on to the GPU and render it at interactive frame rates. Over wide area network links which can be orders of magnitude slower, it is unclear if these approaches can be equally effective.

Building on the hierarchical image caching technique [Shade et al. 1996] designed for standalone clients, a more recent approach [Chaudhuri et al. 2008] pre-computes hierarchical depth images on a cluster and sends clients a combination of nearby geometry and distant depth images to render. Since this technique displays everything beyond some distance as a depth image, users cannot interact or communicate with distant objects, irrespective of their size or importance. This is acceptable if the goal is only to display the scene, but for a system where a user may interact with objects, it is very limiting. Further, as display resolution increases, larger depth images are required to maintain the same quality, which quickly increases the network bandwidth required to display the scene.

Demand-driven geometry transmission [Schmalstieg and Gervautz 1996] uses distance queries and level of detail on each object. Only low LOD is used initially and each object is refined over time, resulting in a high-quality scene. However, since no aggregation is performed, the number of objects that can be displayed is limited by the number of draw calls the GPU can process.

The Paradise project uses object aggregation to reduce network traffic [Singhal and Cheriton 1996]. It creates a statistical aggregation of objects (*e.g.*, the mean and standard deviation of tanks in a battlefield), enabling clients to generate their own approximate representation. This approach only works when all objects and their meshes are pre-defined.

Sirikata [Cheslack-Postava et al. 2012], an open-source virtual world system, returns objects that occupy a minimum solid angle in the client's field of view. Using a BVH, the Sirikata server efficiently answers solid angle queries by computing a cut across the BVH that includes the set of objects satisfying a query. Leaves belonging to the cut are returned as individual objects, while internal nodes are returned as "aggregates". The server generates special aggregate meshes for them by piecing together the meshes of their children, simplifying them and uploading them to the cloud (Figure 3). Since the cut extends across the entire width of the BVH, a client can display a complete view of the world by rendering the limited number of nodes belonging to the cut.

Compression using symmetry. Recent work [Mitra et al. 2006; Pauly et al. 2008] has focused on finding symmetrical substructures within 3D models. Instancing these substructures enable the 3D model to be greatly compressed. These approaches are not directly applicable to our on-line system, since they often require many seconds or even minutes to discover these symmetries. However, they could be incorporated into the cloud storage system to compress uploaded user models.

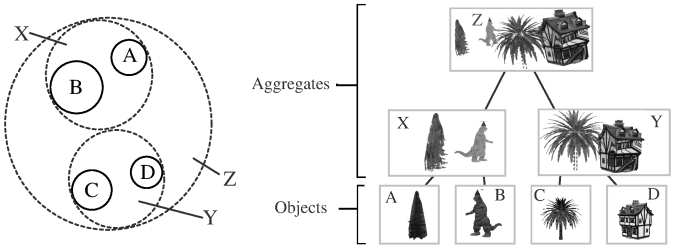


Figure 3: *Sirikata's BVH*

3 Problem Statement and Overview

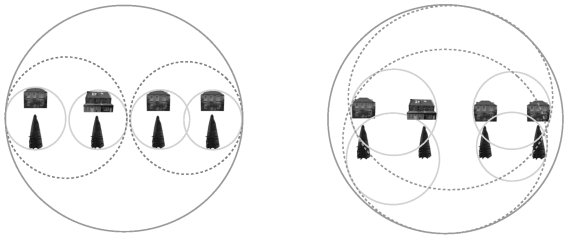
Displaying a large virtual world from the cloud requires tackling two major performance bottlenecks: the network and the GPU. User-generated worlds are often composed of large numbers of inefficient models. Without optimizations and algorithmic improvements, even a small number of objects can be too big to download and for GPUs to render. Using the open-source Sirikata system [Cheslack-Postava et al. 2012] as a starting point, this paper greatly improves download and rendering speed through three algorithmic improvements.

(A) **Grouping objects into aggregates such that their meshes have low download and rendering cost.** Existing approaches group together objects in a BVH to optimize query performance and ignore the resulting mesh complexity or size. Many scenes have collections of very similar objects: trees grouped into a forest, bricks forming a wall, streets and curbsides forming a road network, and similar houses grouped into a suburb. As Figure 4 shows, considering the graphical similarity of objects could lead to meshes that replace highly similar objects with one representative object instanced multiple times. The resulting aggregate meshes can significantly reduce file size and rendering complexity. Sections 4 and 5 describe algorithms for constructing similarity-aware BVHs and similar mesh deduplication.

(B) **Simplifying instanced meshes efficiently into small output mesh files.** User-generated virtual worlds require an open 3D format, such as COLLADA. Most such formats describe the 3D model as an instanced mesh, where a model consists of a set of submeshes which are instantiated one or more times to create the overall mesh. This reduces the download and memory cost for clients compared to an expanded version of the same mesh. It also enourages modification and reuse of meshes by users because they are easier to edit: changes to a submesh, such as a window on a house, are made across all instances. Aggregates can also increase instancing, because aggregates group similar objects together and represent them as instances of the same object.

However, because quadric simplification requires all instances in the mesh to be expanded out into a single instance, it can drastically increase the file size of the resulting mesh. Results in Section 8, for example, show that on instanced meshes, using quadric simplification to reduce the number of triangles by 80% can double the file size. Section 6 presents an instance-aware simplification algorithm that strictly reduces file size as it simplifies a mesh.

(C) **Reconfiguring aggregate meshes to better manage GPU texture resources.** Combining many meshes into aggregates can result in each aggregate having large numbers of seperate textures, which becomes a GPU bottleneck since each texture requires a separate draw call. Section 7 describes an approach towards texture man-



(a) Optimized for querying using the surface area heuristic.

(b) Optimized for small aggregate meshes using mesh similarity.

Figure 4: *Comparing BVHs optimized for querying versus aggregate mesh size.*

agement that ensures each aggregate mesh has only a single draw call for all of its textures.

4 Similarity-based BVH Construction

BVHs are generally optimized for querying performance so that renderers can quickly cull large parts of the scene. To this end, the BVH is constructed so that it minimizes the surface areas of the children of each BVH node. In a networked virtual world system, however, the actual bottleneck lies in the latency for a client to download models and render them at a reasonable frame rate. Therefore, the BVH should be optimized for download latency and rendering cost of its aggregates, instead of solely querying performance.

The BVH can optimize both download latency and rendering performance for clients by grouping together similar and duplicated objects within the BVH. Objects that are duplicated or very similar can be deduplicated and represented by multiple instances of a single unique object. This greatly compresses their representation and allows them to be displayed efficiently by the GPU.

4.1 Similarity Measurement

We quantify object similarity using Zernike shape descriptors [Novotni and Klein 2004] to compare geometry and the Color Structure Descriptor (CSD) [Manjunath et al. 2001] to compare textures. Zernike descriptors are invariant to rotation, translation and scale, while CSDs allow us to efficiently compare textures that have the same color content, but different color layout.

For every model uploaded, our content delivery network (CDN) computes and stores its Zernike descriptor by first voxelizing the model to a 128^3 grid [Min ; Nooruddin and Turk 2003], and then generating a 20-th order Zernike descriptor of length 121. This is a direct application of existing techniques.

The Color Structure Descriptor algorithm, however, is only designed to describe a single, complete image. A descriptor for comparing textures in a mesh must differ in three ways. First, it must describe all textures and colors in the mesh, and it is not clear how to merge descriptors together. Second, it must ignore parts of the texture that are not referenced by the mesh. Third, it must handle texture wrapping (*i.e.*, use texture coordinates beyond the dimensions of the texture that must be wrapped), which causes multiple copies of the texture to be used in the model. This is important because a texture repeated many times has a very different appearance than the same texture applied only once.

To overcome these problems, the CDN computes a 32-bin CSD using charted versions of each object's textures [Terrace et al. 2012]. These charted textures combine all colors and textures used in a model into a single image, eliminate unused parts of the texture space, and re-map the texture coordinates to within the texture dimensions. The CDN then computes the CSD for a model using its charted textures.

4.2 Incremental Construction

When a new object is added to the world, the system inserts it into the BVH. There is a tradeoff when selecting where to insert the object. Inserting it near objects with similar bounding volumes would optimize query performance, but inserting it near objects with similar meshes would increase instancing and optimize mesh size and download cost. Ignoring one of these goals leads to bad behavior. For example, considering only object similarity can lead to extremely unbalanced trees with bad querying performance.

Instead, the system must balance mesh similarity and querying performance. To quantify similarity of a new object to other objects in a bounding volume, each bounding volume maintains the centroid of the shape and texture descriptors of its children. Then the difference between these centroids and a new object's descriptors measure the object's similarity to other objects in the bounding volume. To consider querying performance, we heuristically assume that the best place to add a new object is in the bounding volume whose volume is least increased by the addition.

To insert a new object, the BVH starts at the root and chooses the child, which minimizes the metric,

$$\begin{aligned} &\mu|(1-\omega)(S-C_s)+\omega(T-C_t)| \\ &+(1-\mu)(V_c-V_{old})/V_{max} \end{aligned} \tag{1}$$

where S and T are the new object's shape and texture descriptors, C_s and C_t are the centroid shape and texture descriptors of the child, V_c is the bounding volume of the child with the new object added, V_{old} is its original bounding volume and V_{max} is a normalizing term equal to $\max_{c \in children} V_c$. μ and ω are parameters between 0 and 1 which control the relative weight of appearance and texture similarity respectively.

Children are chosen in this manner recursively until a leaf node is reached. If the chosen node is already full, it is split to accomodate the new object. The splitting algorithm proceeds as follows: first, it finds a pair of children as "seeds" by finding the pair that, when merged, results in the most wasteful bounding volume, *i.e.*, the one with the most empty space. The algorithm takes similarity into account by choosing a pair that has the largest difference in shape descriptors while remaining within 20% of the maximum bounding volume waste. Once these seeds are chosen as the split nodes, the remaining children are grouped with one split node or the other, depending on which one minimizes the metric in expression 1.

Evaluations on a set of workloads help to find suitable values for μ and ω that can lead to a good balance of querying performance and grouping similar objects within the BVH. The first workload is composed of six unique objects instantiated into a larger 10,000-object village scene. The second workload consists of 236 unique objects arranged into a 2362-object island scene. The third workload is a city scene generated using CityEngine with 60,000 unique objects with ~3% objects replaced with random objects from our CDN. Many of these objects are very similar (*e.g.*, roads and pavements) but are nevertheless distinct. For each workload, the generated aggregates remain unsimplified so that their visual quality does not change.

Varying μ and ω, we compute the average cost of querying the BVH. For each μ and ω, we also compute the average amount of

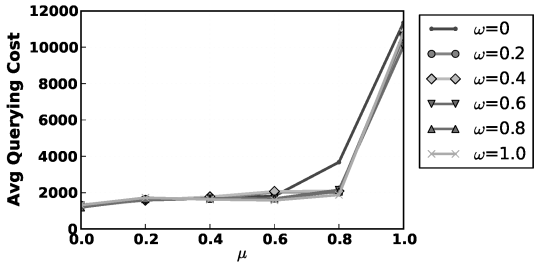


Figure 5: *Effect of μ and ω on BVH querying performance.*

data a client has to download on joining the world from ten random locations.

Figure 5 shows that the average cost of querying the BVH does not increase significantly with μ and ω, except when μ approaches 1. On the other hand, figure 6 shows that for two of the workloads, the average download size reduces by up to 25% with increasing values of μ. Each of these workloads yields a minimum download size at $\mu = 0.8$ and $\omega = 0.2$. However, different values of ω generally do not have a substantial effect on download size. This suggests that using object geometry to construct the BVH provides most of the improvements in download size and rendering performance, while object textures have a smaller role to play in this regard. We use $\mu = 0.8$ and $\omega = 0.2$ as the values of these parameters in the BVH.

Note that download size does not decrease significantly in the CityEngine workload. Increasing μ allows for more deduplication of similar meshes (as described in section 5.1) but almost all of the deduplicated meshes are very small and simple. As a result, deduplicating them enables them to be rendered more efficiently as instances of a single mesh but does not greatly reduce download size.

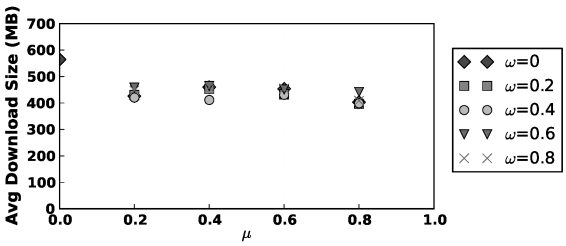
4.3 Bulk Construction

Every few hours, the system reconstructs the BVH to ensure it does not stray too far from the optimal condition. A top-down bulk construction algorithm considers a set of axis-aligned candidate split planes at each level of the tree. It computes the mutual similarity, defined as the average distance of the shape descriptors from their centroid, of the groups of objects on either side of each plane. This computation takes $O(n)$ time, where n is the number of objects in the system. The partition plane is then chosen as the plane that yields the maximum similarity among objects in the two partitions. Thus, the whole tree can be reconstructed in $O(nlog(n))$ time.

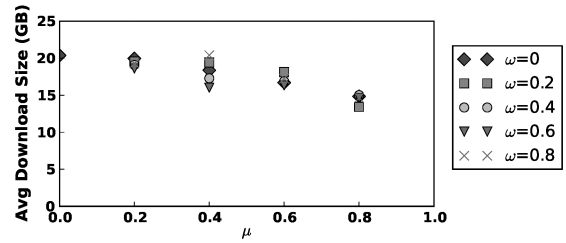
5 Aggregate Generation

The server generates aggregates for each BVH node by working bottom-up, merging the meshes of each node's children. The aggregate generation module downloads, triangulates, centers and caches the model for each child. It deduplicates similar models within each aggregate, replacing them with multiple instances of a single model. The deduplication approach is described in Section 5.1

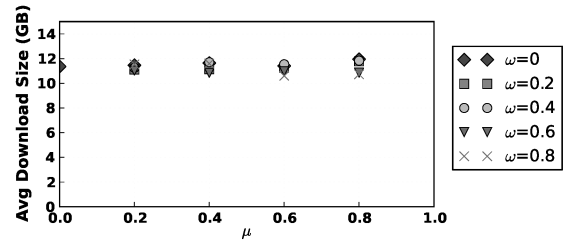
Next, it creates a new aggregate model with the submeshes and instances from this set of deduplicated models. However, the transformation matrix for each instance is modified so that the children's meshes are positioned and oriented to match the corresponding objects in the scene. Finally, the aggregate model is simplified using a novel instance-aware simplification algorithm (Section 6) and reconfigured so that the whole scene can be rendered with a limited



(a) Suburb Scene



(b) Island Scene



(c) CityEngine Scene

Figure 6: *Effect of μ and ω on average download size for a client.*

number of GPU draw calls (Section 7).

5.1 Mesh Deduplication

As described in Section 4, the BVH attempts to group together similar and proximate objects. This allows near-duplicate objects within the aggregate to be replaced by a single, instanced object.

While deduplicating meshes, it is critical to ensure that only objects that appear very similar are deduplicated. This requires finding suitable thresholds for differences in shape and texture descriptors, below which a pair of models can be considered nearly identical.

We discover these thresholds by doing an offline analysis of a dataset of 748 models in our CDN uploaded by users over the course of one summer. We render each model as images from three mutually orthogonal directions and also store smaller versions of these images that are downsampled by powers of two. We compute the Zernike and texture descriptors for each model as described in Section 4.1. For each pair of models, the difference in their Zernike descriptors, z_{diff}, is defined as the L2-distance between them, and the difference in texture descriptors, t_{diff} as the L1-distance. For each pair of models, at each image size, we use a perceptual difference metric [Yee and Newman 2004] to determine if their images are indistinguishable from all three directions.

Table 1 shows, for each image size, the minimum values of z_{diff} and t_{diff} below which every pair of models is indistinguishable. This information allows us to efficiently identify candidates for

Image size (pixels)	Solid Angle	z_{diff}	t_{diff}
< 48	< 0.00002	0.5	500
< 192	< 0.00007	0.01	350
< 768	< 0.0003	0.003	50
< 3072	< 0.001	0.001	10
< 12288	< 0.004	0.001	10
< 49152	< 0.018	0	10
< 196608	< 0.073	0	0

Table 1: *Deduplication thresholds for objects of various sizes. The solid angle is computed assuming a 1920x1080 pixel, 34 inch display viewed from 24 inches away [Deering 1998]. Every pixel is assumed to subtend the same solid angle at the viewer.*

deduplication. Under the assumption that the maximum solid angle a client is allowed to query for is $\Omega = 1.0$, this implies that a client will only receive aggregates that subtend a solid angle of at least 1.0. This allows us to compute the minimum distance d at which a given aggregate can be returned to a client, using:

$$d = \frac{R}{\sqrt{(1-(1-\frac{\Omega}{2\pi})^2)}} \tag{2}$$

where R is the radius of the aggregate. The solid angle S subtended by a child of the aggregate having radius r can then be computed as:

$$S = 2\pi(1-\sqrt{1-\frac{r^2}{d^2}}) \tag{3}$$

Then, for a pair of individual objects A and B within an aggregate, the system replaces A with B if, based on the solid angle S computed for A, z_{diff} and t_{diff} between A and B lie within the thresholds given in table 1.

One final detail is that once we choose to replace a model with a similar one, we align the two models by using a transformation to line up their major and minor axes.

6 Instance-aware Simplification

This section presents an instance-aware variant of quadric simplification that simplifies an instanced mesh without expanding it into its equivalent triangle soup or indexed triangle mesh, either in-memory or on disk. First we provide an overview of the quadric simplification algorithm and then describe how we extend it to operate on instanced meshes without expanding the submeshes or allowing them to diverge during simplification.

6.1 Basic Quadric Simplification

Quadric mesh simplification [Garland and Heckbert 1997] executes in two phases. During the initialization phase, the algorithm assigns an error quadric, $\mathbf{Q}$, to each vertex, $\mathbf{v}$. $\mathbf{Q}$ is computed on the basis of the planes (triangles) neighboring $\mathbf{v}$, and is given by $\mathbf{Q} = \sum_{p \in planes(\mathbf{v})} \mathbf{Q_p}$, where $\mathbf{Q_p}$ is the quadric for plane p and is computed as:

$$\mathbf{Q_p} = area(p).\begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{4}$$

Here a, b, c and d are the normalized coefficients of the equation $ax+by+cz+d=0$, which defines the plane p, while $area(p)$ is the area of the triangle corresponding to plane p. With this formulation, given a vertex $\mathbf{w}$, $\mathbf{w}^T\mathbf{Q}\mathbf{w}$ is a measure of the distance of vertex $\mathbf{w}$ from the set of planes in planes($\mathbf{v}$).

For every edge ($\mathbf{v_1}$,$\mathbf{v_2}$), assuming that quadrics $\mathbf{Q_1}$ and $\mathbf{Q_2}$ are associated with $\mathbf{v_1}$ and $\mathbf{v_2}$, it then computes an optimal contraction target $\bar{\mathbf{v}}$ for which the cost is given by

$$cost(\bar{\mathbf{v}}) = \bar{\mathbf{v}}^T\ (\mathbf{Q_1}+\mathbf{Q_2})\ \bar{\mathbf{v}}. \tag{5}$$

Assuming $\mathbf{K} = (\mathbf{Q_1}+\mathbf{Q_2})$, $\bar{\mathbf{v}}$ is computed using the formula:

$$\bar{\mathbf{v}} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{12} & k_{22} & k_{23} & k_{24} \\ k_{13} & k_{23} & k_{33} & k_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

In the edge collapse phase, edges are collapsed iteratively in increasing order of these error values, with the cost of vertices neighboring $\mathbf{v_1}$ and $\mathbf{v_2}$ updated after each collapse.

6.2 Instance-Aware Simplification (IAS)

As described in section 2, an instanced mesh takes a set of submeshes and creates an overall mesh by instantiating and transforming each submesh one or more times. Instead of operating on the hierarchical representation of an instanced mesh M, we convert it into a flat representation where M consists of a list of instances, and the transform associated with each instance is the product of all transforms from the root of the hierarchy down to the instance itself. Each instance indexes into a list of a submeshes, which contain geometry information in indexed triangle mesh format.

Formally, $M = \{I_1, I_2, I_3, ..., I_n\}$, where n is the number of instances constituting the mesh. Each instance $I_j =< T_j, S_k >$, where T_j is the transformation matrix associated with I_j and S_k is the submesh referenced by I_j. The submesh S_k is chosen from a list of submeshes $\{S_1, S_2, ..., S_m\}$, such that $1 \leq k \leq m$ and $m \leq n$. Each submesh S_k consists of a set of vertices and a set of triangles referencing those vertices.

A naive approach to simplify such an instanced mesh would be to run quadric simplification on each submesh independently and maintain all the intermediate levels of detail for each submesh. Then to simplify the overall mesh to a target triangle count, choose an appropriate level of detail for each sub- mesh. However, this leaves open the question of how to choose these levels of detail, which would be especially complicated if the submesh is instanced using wildly varying transformations.

To solve this problem, instance-aware simplification applies quadric simplification to the underlying submeshes but accounts for the fact that each submesh may be instantiated multiple times and transformed in different ways to create the overall mesh.

One way to do this intuitively is to iterate through the list of mesh instances, compute the cost of each edge in the overall mesh, and add it back to the total cost of the underlying submesh edge. Then, order each submesh edge by its total cost and collapse edges in increasing order of their total cost. However, this approach does not help us find an optimal contraction target for a given submesh edge.

Instead, we use the observation that the error quadric $\mathbf{Q}$ associated with a vertex $\mathbf{v}$ is derived from the set of planes neighboring $\mathbf{v}$. In an instanced mesh, therefore, $\mathbf{Q}$ for a submesh vertex $\mathbf{v}$ can be computed by accounting for all the neighboring planes that exist in all instances of the submesh. Suppose $\mathbf{M}$ is the transformation matrix for a given instance of a submesh, and $\mathbf{v}$ is a submesh vertex which maps to $\mathbf{x}$ in that instance. Since $\mathbf{x}=\mathbf{M}\mathbf{v}$, we

can write the distance of $\mathbf{x}$ from its set of neighboring planes as $\mathbf{x}^T\mathbf{Q}\mathbf{x}=(\mathbf{M}\mathbf{v})^T\mathbf{Q}(\mathbf{M}\mathbf{v})=\mathbf{v}^T\mathbf{M}^T\mathbf{Q}\mathbf{M}\mathbf{v}$, where $\mathbf{Q}$ is computed from the neighboring planes in that instance.

Then, $(\mathbf{M}^T\mathbf{Q}\mathbf{M})$ is the error quadric giving the distance of the submesh vertices from the neighboring planes in the instance. Summing it up over all instances, the quadric for a submesh vertex is given by $\sum_{i \in instances(submesh)} \mathbf{M}_i^T\mathbf{Q}_i\mathbf{M}_i$, where $\mathbf{M}_i$ is the transform associated with instance i and $\mathbf{Q}_i$ is the quadric computed for the instantiated vertex.

Using this new quadric, we can find the optimal contraction target for a submesh edge similarly as basic quadric simplification. The final algorithm, then works as follows:

1. For each instance i applying transform $\mathbf{M}_i$ to submesh $\mathbf{S}_i$:
   (a) For each triangle t in submesh $\mathbf{S}_i$:
      i. Transform t by applying $\mathbf{M}_i$ to each of its vertices.
      ii. Compute $\mathbf{Q_p}$, the error quadric for the transformed triangle, using Equation 4.
      iii. Compute the error quadric for the untransformed triangle t as $\mathbf{M}_i^T\mathbf{Q_p}\mathbf{M}_i$ and add it to the error quadrics for each of t's untransformed vertices.
2. In each submesh $\mathbf{S}_i$, compute the optimal contraction target $\bar{\mathbf{v}}$ and its cost for each submesh edge $(\mathbf{v}_1,\mathbf{v}_2)$ using Equations 6 and 5, where $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are the submesh error quadrics associated with $\mathbf{v}_1$ and $\mathbf{v}_1$ respectively.
3. Collapse submesh edges in increasing order of their cost. Compute how many triangles become degenerate after each collapse and decrement the number of triangles in the model by that times the number of instances of the submesh. At each step, since only a submesh edge is collapsed, the cost has to be updated only for neighboring vertices in that submesh.

6.3 Preserving boundaries

A boundary edge is an edge that exists in only one triangle. IAS considers an edge to be a boundary edge as long as it is a boundary edge within its submesh. IAS uses the same basic approach as quadric simplification for boundary edges. For each edge in the overall mesh, if it is a boundary edge, IAS generates a perpendicular constraint plane running through the edge. It then computes the quadric for this constraint plane, weights it by the length of the edge and adds it to the quadrics for the endpoints of the edge. In our experience, this results in much better results than simply marking such edges as incollapsible since it still allows small boundary edges to be collapsed, instead of forcing other longer edges to be collapsed.

6.4 Discussion

Since the edge collapse step of IAS operates only on submesh edges, highly instanced meshes can be simplified faster than quadric simplification. Not only are there fewer submesh edges than edges in the overall mesh, but collapsing a single submesh edge effectively collapses multiple edges in the overall mesh, allowing simplification to proceed faster towards the target triangle count.

On the other hand, IAS does not compute quadrics across submeshes, so it has less information about the overall mesh than quadric simplification. This can, in theory, result in lower quality outputs. However, our evaluations in section 8 show that, in practice, IAS often results in even higher quality meshes, or introduces very little additional error otherwise.

It is useful to note that expansion of an instanced mesh is not a concern for instanced meshes that only instance each submesh once, since there is no submesh duplication. Also, if every instance of a submesh scales or modifies the submesh in the same way, then existing simplification approaches can still be trivially applied: just multiply the edge collapse cost from one instance by the number of instances. However, the problem becomes much more complicated when multiple instances of the same submesh transform it in completely different ways.

The current implementation of IAS does not optimize other properties associated with vertices, such as texture coordinates and normals. Similar to the approach followed in [Garland and Heckbert 1998], extending the quadric to include the values of these properties may be a possible solution.

7 Texture Management

GPUs are also constrained by the number and size of textures needed to render each frame. Current GPUs cannot render more than one to ten thousand unique textures per frame at interactive frame rates. The total size occupied by textures is also limited by GPU texture RAM, with current cards generally restricted to at most a gigabyte.

To deal with these constraints, the system generates aggregates that can each be rendered with a single draw call, *i.e.*, reference only one texture. Further, it limits each texture to a maximum size of 128KB. Clients specify a solid angle query whose response contains less than four thousand objects. The client can then render these in less than three thousand draw calls using at most 512 MB of texture RAM.

We use a simple approach, texture atlasing, to ensure that an aggregate references only a single texture. If a scene uses many unique textures, the system combines the textures of each child of an aggregate into a single atlas, and remaps the texture coordinates in the aggregate mesh to point to the appropriate coordinates in the atlas.

Atlasing involves some complicated details, however. Some models use wrapping texture coordinates which can result in the same texture being used repeatedly. Others reference only a small part of the actual texture image. As described in section 4.1, the CDN addresses these issues by asynchronously computing a charted, combined version of each model's textures. Given these modified textures as input, atlasing becomes a relatively simple process.

Some large scenes do not use many unique textures. This is often true for procedurally generated scenes, where textures from a small set are used repeatedly to create the scene. To handle such scenes without the overhead of atlasing, we only atlas textures in aggregate meshes if the scene has more than 100 textures or the textures take up more than 64 MB.

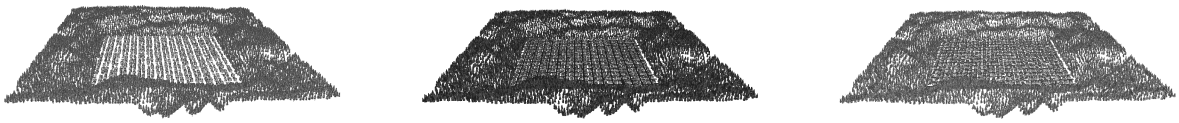
This approach leaves open the question of texture space waste caused when a frequently used texture is repeated many times as a part of different atlases. We leave this question to future work.

8 Evaluation

This section evaluates our results from using these techniques on individual models and on a large multi-server virtual world.

Model	Time (Quadric)	Time (IAS)	Time Reduction	Size (Quadric)	Size (IAS)	Size Reduction	Error (Quadric)	Error (IAS)	Error Reduction
Maple Tree	13 s	5 s	58%	28 MB	110 KB	99.6%	0.158	0.103	34.8%
Village	30 s	17 s	44%	31 MB	6 MB	79.6%	5.601	5.504	1.8%
Patio Chair	40 ms	43 ms	-8%	34 KB	31 KB	8.8%	0.251	0.055	78.0%
Stonehenge	140 ms	608 ms	-334%	379 KB	373 KB	1.6%	0.516	0.520	-0.8%
Bunny	390 ms	1890 ms	-384%	235 KB	235 KB	0%	0.200	0.200	0%

Table 3: *Performance comparison of IAS and Quadric simplification, demonstrating that instance-aware simplification consistently outperforms quadric simplification in simplified mesh file size while introducing comparable error. Each model is simplified to 20% of its original triangle count. Error is the Hausdorff distance between the original and simplified mesh computed using the Metro tool.*



(a) Quadric (20% triangles, 31 MB) (b) Original, 16 MB (c) IAS (20% triangles, 6 MB)

Figure 7: *Village mesh from Table 3 simplified to 20% of the original triangle count. The visual quality of the two simplified versions of the mesh is not very different, but the output file size is almost 80% smaller using IAS.*

(a) Quadric (20% triangles, 28 MB) (b) Original, 546 KB (c) IAS (20% triangles, 110 KB)

Figure 8: *Maple tree with highly instanced leaves from Table 3 simplified to 20% of the original triangle count. IAS results in better quality than quadric simplification, while also achieving much smaller file size.*

(a) Quadric (20% triangles, 34 KB) (b) Original, 92 KB (c) IAS (20% triangles, 31 KB)

Figure 9: *Patio chair with highly instanced slats from Table 3 simplified to 20% of the original triangle count. IAS results in surprisingly better quality than the reference implementation of quadric simplification.*

Model	Submeshes	Instances	Instances per submesh	Triangles
Maple Tree	18	9324	518	1818074
Village	79	13523	171.18	1254696
Patio Chair	6	68	11.33	2240
Stonehenge	59	80	1.36	10061
Bunny	1	1	1	20000

Table 2: *Properties of models used to evaluate IAS.*

8.1 Instance-aware Simplification

Table 3 shows the benefits and trade-offs of instance-aware simplification (IAS) on a set of models described in Table 2. The village and dense maple tree meshes (figures 7 and 8) are highly instanced with over a million triangles each. The houses in the middle of the village scene are instantiated from a set of six unique house submeshes. The surrounding trees are instances of a single unique submesh. The leaves on the maple tree mesh are instantiated from two distinct leaf submeshes. Using IAS, simplification of these heavily instanced models proceeds faster than quadric simplification, results in a small output mesh and approximately the same or even less error as simple quadric simplification.

On the other hand, the Patio chair (figure 9) and Stonehenge meshes have a number of submeshes but they are not instanced many times. IAS is slower on these lightly instanced meshes but the output meshes are almost 10% smaller in size and have approximately the same or even less simplification error. The larger simplification time is expected since IAS has to do additional matrix multiplications to compute the transformed submeshes and their quadrics.

The bunny mesh has only a single submesh instanced exactly once. It takes significantly longer to simplify it using IAS, but the visual error remains almost the same. For such non-instanced meshes, it is more efficient to simply use quadric simplification for LOD generation.

8.2 Evaluating a Large Scene

Finally, we demonstrate the benefits of our approach on a large-scale virtual world running on multiple Amazon EC2 servers. The world runs on nine c1.xlarge EC2 servers, with each server running a tiled copy of a 60,000 object city scene, created using CityEngine [ESRI]. To diversify the world with user-generated models and textures, we replace 3% of the models in the world with randomly chosen models from the CDN. This results in the world having a total of 540,000 objects composed of over 144 million triangles referencing more than 6000 distinct textures. In total, the models and textures in the scene occupy over 24 GB on disk.

Figure 1 shows this world rendered by our client at a resolution of 1920x1080. The client runs on a machine with an AMD FX-8150 eight-core CPU, 16 GB of RAM and an NVIDIA GeForce GTX 650Ti graphics card. The techniques outlined in this paper allow this complex scene to be rendered at frame rates hovering between 20 and 35 FPS. Without these approaches, the client is overwhelmed by the sheer size and number of models and textures to be downloaded and displayed, rendering the scene at less than 1 FPS. Using the optimizations described in this paper, a client connecting to the world for the first time downloads approximately 1.4 GB of data from the CDN, a 94% reduction. This allows it to load a high quality view of the complete world in 6 minutes. Using unoptimized aggregates, the client downloads 2.2 GB from the CDN before the GPU is unable to render the scene due to texture load.

9 Conclusion

Displaying large, user-generated virtual worlds is challenging due to the limitations of the network and GPU. This paper presented an approach to tackle this problem by exploiting the similarity of proximate objects in a coherent scene, using instancing to reduce the file size of models a client has to download. Our approach constructs a BVH out of objects in the world trading off querying performance for higher object similarity within a bounding volume (or "aggregate"). We then deduplicate highly similar meshes within each aggregate, replacing them with many instances of a single unique mesh. This increases the amount of instancing in the aggregates, potentially reducing their file size. An instance-aware simplification algorithm simplifies the aggregate meshes, resulting in a smaller file size for the aggregate mesh than existing approaches. Finally, texture atlasing enables efficient rendering of aggregate meshes. Evaluations show that this approach allows large user-generated virtual worlds to load quickly, require less download bandwidth and render at interactive frame rates.

References

ARNAUD, R., AND BARNES, M. C. 2006. *Collada: Sailing the Gulf of 3d Digital Content Creation*. AK Peters Ltd.

CHAUDHURI, S., HORN, D., HANRAHAN, P., AND KOLTUN, V. 2008. Distributed rendering of virtual worlds. Tech. Rep. 2008-02, Stanford University, Computer Science.

CHESLACK-POSTAVA, E., AZIM, T., MISTREE, B. F. T., HORN, D. R., TERRACE, J., LEVIS, P., AND FREEDMAN, M. J. 2012. A scalable server for 3d metaverses. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX Association, Berkeley, CA, USA, USENIX ATC'12.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2, 167–174.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. In *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, 796–803.

CIGNONI, P., DI BENEDETTO, M., GANOVELLI, F., GOBBETTI, E., MARTON, F., AND SCOPIGNO, R. 2007. Ray-casted blockmaps for large urban models visualization. *Computer Graphics Forum 26*, 3, 405–413.

CRASSIN, C., NEYRET, F., LEFEBVRE, S., AND EISEMANN, E. 2009. Gigavoxels: ray-guided streaming for efficient and detailed voxel rendering. In *Proc. Symp. Interactive 3D Graphics*.

DÉCORET, X., DURAND, F., SILLION, F. X., AND DORSEY, J. 2003. Billboard clouds for extreme model simplification. In *ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, SIGGRAPH '03, 689–696.

DEERING, M. F. 1998. The limits of human vision. In *2nd International Immersive Projection Technology Workshop*, vol. 2.

DOBBYN, S., HAMILL, J., O'CONOR, K., AND O'SULLIVAN, C. 2005. Geopostors: a real-time geometry / impostor crowd rendering system. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '05, 95–102.

ERIKSON, C., MANOCHA, D., AND BAXTER, III, W. V. 2001. Hlods for faster display of large static and dynamic environ-

ments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, I3D '01.

ESRI. http://www.esri.com/software/cityengine.

FUNKHOUSER, T., SEQUIN, C., AND TELLER, S. 1992. Management of large amounts of data in interactive building walkthroughs. In *Proc. Symp. Interactive 3D Graphics*.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error metrics. In *Proc. ACM SIGGRAPH*.

GARLAND, M., AND HECKBERT, P. S. 1998. Simplifying surfaces with color and texture using quadric error metrics. *Visualization Conference, IEEE 0*, 264.

GOBBETTI, E., AND MARTON, F. 2005. Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, 878–885.

HOPPE, H. 1996. Progressive meshes. *Computer Graphics 30*, Annual Conference Series.

HOPPE, H. 1999. New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the conference on Visualization '99: celebrating ten years*, IEEE Computer Society Press, Los Alamitos, CA, USA, VIS '99.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 199–208.

MANJUNATH, B., OHM, J.-R., VASUDEVAN, V., AND YAMADA, A. 2001. Color and texture descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on 11*, 6 (jun), 703 –715.

MIN, P. http://www.cs.princeton.edu/~min/binvox.

MITRA, N. J., GUIBAS, L. J., AND PAULY, M. 2006. Partial and approximate symmetry detection for 3d geometry. *ACM Transactions on Graphics (TOG) 25*, 3, 560–568.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. on Visualization and Computer Graphics 9*, 2 (Apr.).

NOVOTNI, M., AND KLEIN, R. 2004. Shape retrieval using 3d zernike descriptors. *Computer-Aided Design 36*, 11, 1047 – 1062. Solid Modeling Theory and Applications.

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. J. 2008. Discovering structural regularity in 3d geometry. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 43.

RONFARD, R., AND ROSSIGNAC, J. 1996. Full-range approximation of triangulated polyhedra. In *Computer Graphics Forum*, vol. 15, Wiley Online Library, 67–76.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 343–352.

RUSINKIEWICZ, S., AND LEVOY, M. 2001. Streaming qsplat: a viewer for networked visualization of large, dense models. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, I3D '01, 63–68.

SCHMALSTIEG, D., AND GERVAUTZ, M. 1996. Demand-driven geometry transmission for distributed virtual environments. In *Computer Graphics Forum*, 421–433.

SHADE, J., LISCHINSKI, D., SALESIN, D. H., DEROSE, T., AND SNYDER, J. 1996. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proc. ACM SIGGRAPH*.

SINGHAL, S., AND CHERITON, D. 1996. Using projection aggregations to support scalability in distributed simulation. *International Conference on Distributed Computing Systems*.

TELER, E., AND LISCHINSKI, D. 2001. Streaming of complex 3d scenes for remote walkthroughs. In *Computer Graphics Forum*.

TERRACE, J., CHESLACK-POSTAVA, E., LEVIS, P., AND FREEDMAN, M. J. 2012. Unsupervised conversion of 3d models for interactive metaverses. In *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, 902 –907.

WATSON, B., FRIEDMAN, A., AND MCGAFFEY, A. 2001. Measuring and predicting visual fidelity. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01.

WILLIAMS, N., LUEBKE, D., COHEN, J. D., KELLEY, M., AND SCHUBERT, B. 2003. Perceptually guided simplification of lit, textured meshes. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM, New York, NY, USA.

YEE, Y. H., AND NEWMAN, A. 2004. A perceptual metric for production testing. In *ACM SIGGRAPH 2004 Sketches*, ACM, New York, NY, USA, SIGGRAPH '04, 121–.

YOON, S.-E., SALOMON, B., GAYLE, R., AND MANOCHA, D. 2005. Quick-vdr: Out-of-core view-dependent rendering of gigantic models. *Visualization and Computer Graphics, IEEE Transactions on 11*, 4, 369–382.