Hybrid Simplification: Combining Multi-resolution Polygon and Point Rendering

Jonathan D. Cohen
cohen@cs.jhu.edu
Johns Hopkins University

Daniel G. Aliaga
aliaga@bell-labs.com
Lucent Technologies Bell Labs

Weiqiang Zhang
zhangwq@cs.jhu.edu
Johns Hopkins University

Abstract

Multi-resolution hierarchies of polygons and more recently of points are familiar and useful tools for achieving interactive rendering rates. We present an algorithm for tightly integrating the two into a single hierarchical data structure. The trade-off between rendering portions of a model with points or with polygons is made automatically.

Our approach to this problem is to apply a bottom-up simplification process involving not only polygon simplification operations, but point replacement and point simplification operations as well. Given one or more surface meshes, our algorithm produces a hybrid hierarchy comprising both polygon and point primitives. This hierarchy may be optimized according to the relative performance characteristics of these primitive types on the intended rendering platform. We also provide a range of aggressiveness for performing point replacement operations. The most conservative approach produces a hierarchy that is better than a purely polygonal hierarchy in some places, and roughly equal in others. A less conservative approach can trade reduced complexity at the far viewing ranges for some increased complexity at the near viewing ranges.

We demonstrate our approach on a number of input models, achieving primitive counts that are 1.3 to 4.7 times smaller than those of triangle-only simplification.

Keywords: rendering, simplification, multi-resolution, triangles, points, hybrid.

1 INTRODUCTION

Interactive visualizations, which maintain a steady feedback loop with the application user, rely on the ability of the computer and, in particular, the graphics engine to produce images at a high frame rate. Applications with this requirement include the exploration of data through scientific visualization, enhancement of medical procedures through computer-integrated surgery, terrain visualization, production of mechanical systems through CAD visualization and rapid prototyping, and of course the pursuit of entertainment through the high-end video games which have driven the consumer graphics market in recent years.

Most such applications today employ some form of multi-resolution rendering to achieve the necessary balance between the conflicting goals of smooth, interactive performance and useful, high-quality imagery. Multi-resolution rendering uses a hierarchy of rendering primitives, allowing the application to distribute its rendering budget across a complex geometric model to produce such an optimized result.

The rendering primitives used generally depend on the application domain and the method of model design or acquisition. For example, models built from complex polygonal meshes lend themselves to the construction of polygonal hierarchies (some forms are often referred to as levels of detail), built through a process of polygonal simplification. On the other hand, models acquired as a set of points in some form, such as from a camera, laser range-finder, or other device for sampling the physical world, lend themselves naturally to the construction of a point hierarchy through the use of octree-based or other proximity-based point merging schemes. Although these points are in a pure geometric sense infinitesimal, they are usually defined with a radius of extent. Thus, they can be thought of as spheres in world space and, as a matter of rendering efficiency, are typically rasterized as circles or squares in screen space.

In some sense, these representations are interchangeable; both are capable of representing and rendering the same data given a sufficiently high representational resolution. Some applications do, in fact, choose to switch from one domain to another. Point samples may be meshed to produce polygonal models, and polygonal models may be point-sampled and these samples stored to facilitate future rendering. The process of rasterization is itself a conversion from polygons to a set of point samples, so we can clearly establish useful correspondences between triangles and their associated samples, sometimes using them interchangeably.

Both of these representations have merit, but neither is superior for all geometric models under all viewing conditions. Adaptive, view-dependent refinement schemes already employ these multi-resolution representations to adjust the number of primitives used across the model environment to suit the needs of the application's current viewing parameters. So it is natural to consider adapting not only the *number* of primitives but also the *type* of primitives rendered for a particular set of viewing parameters to produce a well-optimized balance of performance and quality. Such a hybrid approach to rendering may produce a system with improved scalability and a wider range of applicability.

1.1 Main Contribution

In this paper, we present a simplification paradigm to *tightly integrate* polygon-based and point-based rendering. Our approach begins with a polygonal model as input, which we proceed to simplify using a standard, greedy simplification procedure (our system employs edge collapse operations). The same optimization criteria that guide the polygon simplification process also trigger the substitution of one or more points for individual triangles as the situation warrants. These points are also merged to produce a complete, hybrid hierarchy. This hierarchy, built entirely as a preprocess, may then be used to perform interactive rendering using adaptive, view-dependent refinement.

Our algorithm provides the following capabilities:

- **Automatic selection:** The algorithm automatically determines where and when a subset of a model is better rendered as triangles or as points.
- **Seamless transition:** The adaptive refinement procedure transitions between triangles and points at a fine granularity.
- **Topology modification:** Multiple manifold surfaces may be merged (and thus more drastically reduced) during the point simplification phase.
- **Error management:** Polygon simplification and point merging are selected as appropriate to reduce geometric error growth within the hierarchy, and guaranteed geometric error bounds from the original surface are provided throughout.

The broader goal of this research is to explore the relative strengths and weaknesses of polygon and point representations, as well as the situations where each is most useful. We also consider how the relative capability of graphics hardware in rendering points versus polygons affects the hierarchies we build. In the long term, we aim to bridge the gap between polygon-based and image-based rendering. Images are essentially specially organized collections of points, and so this research is a stepping stone along the way, providing some useful tools and insights.

1.2 Paper Organization

We proceed by describing in Section 2 some related work in the areas of polygonal simplification and point-based rendering, followed by an overview of our integrated approach in Section 3. After that we review our central data structure, the *multi-resolution graph*, and the off-line and on-line portions of our algorithm in Sections 4, 5, and 6. We conclude with a look at our results, and a discussion of the issues they raise.

2 RELATED WORK

This research draws on previous work in the areas of polygonal simplification and point-based rendering. We now review the most relevant topics in each of these fields.

2.1 Polygonal Simplification

A number of existing polygonal simplification algorithms use priority queue driven, bottom-up decimation strategies [Guéziec 1995, Hoppe 1996, Cohen et al. 1997, Garland and Heckbert 1997]. Of these algorithms, several provide guaranteed bounds on the resulting error between all points on the original surface and all points on the simplified surface (the tightest possible measure being the Hausdorff distance) [Guéziec 1995, Klein et al. 1996, Cohen et al. 1997, Lee et al. 1998]. Our error measure happens to be based on the projection algorithm of [Cohen et al. 1997], but any of this class of guaranteed error measures would do equally well for the purpose of this research.

Several algorithms and hierarchical data structures allow for fine-grained, view-dependent refinement of polygonal models in an interactive setting [Rossignac and Borrel 1993, DeFloriani et al. 1997, Hoppe 1997, Luebke and Erikson 1997, Xia et al. 1997]. Of these, we have found the multi-triangulation data structure of [DeFloriani et al. 1997] to be the most compatible with our current research goals.

The benefits of this research and the properties of our hierarchical models bear some resemblance to those of simplification algorithms that provide for topological modification [Rossignac and Borrel 1993, El-Sana and Varshney 1997, Schroeder 1997, Garland and Heckbert 1998] and merging of low-resolution objects [Erikson and Manocha 1999]. However, none of the existing algorithms provides both fine-grained progressive control and guaranteed surface-to-surface error bounds (other than the most conservative approach of tracking the maximum separation between collapsed vertices).

An interesting piece of research that seems quite similar to ours is the progressive simplicial complex [Popovic and Hoppe 1997]. This data structure, like our multi-resolution graph, also allows for primitives of different types, namely simplices of arbitrary dimension. This general approach allows the simplices to collapse to progressively lower dimension. Our work, in contrast, does not require that triangle vertices be merged to become points, but rather allows this conversion to take place at an arbitrary sampling rate. This allows the hierarchy to be tuned according to a system's relative polygon and point rendering performance characteristics.

2.2 Point-based Rendering

Using points as rendering primitives has a long history in computer graphics [Levoy and Whitted 1985]. Early computer graphics systems used points to render clouds, explosions, and other fuzzy objects [Reeves 1983]. More recently, together with the advent of faster general-purpose CPUs, point rendering has been used to model and render trees, polygonal meshes, and volumetric models [Hoppe et al. 1992, Max and Ohsaki 1995].

The fundamental difficulty of using points is to create a continuous (on-screen) reconstruction of the underlying model. Algorithms leverage the simple rendering calculations of points [Grossman and Dally 1998] to cover surfaces with a sufficient number to samples. Image-Based Rendering (IBR) exploits the screen-coherence of projected points to further accelerate point rendering. By ordering points on a grid and performing incremental computations [McMillan and Bishop 1995], IBR methods can re-project a large number of points (or pixels) each frame.

In our case, we have full knowledge of the underlying model and can choose, a priori, the points to render a model at a desired error tolerance. Thus, we do not need to reconstruct the model. Furthermore, by establishing an error metric over the surface of the model, we have a criterion to sample the model and generate points for an interactive point rendering system [Pfister et al. 2000, Rusinkiewicz and Levoy 2000]. The challenge for our work is to compute, for every neighborhood of a model, when we achieve a win with polygonal rendering and when point rendering is more advantageous.

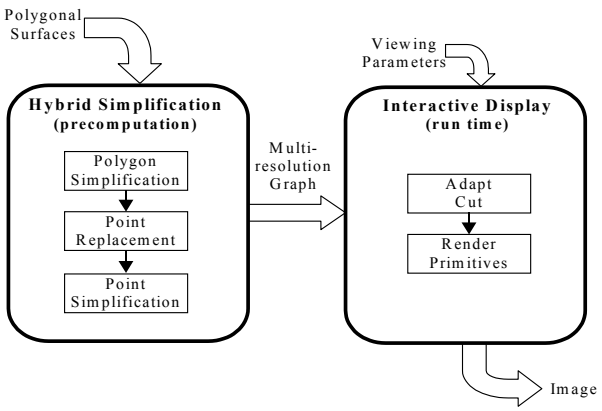


Figure 1: Components of a hybrid multi-resolution rendering system.

3 OVERVIEW

There are many approaches one could take to produce a tight integration of polygons and points in a multi-resolution framework. For example, one could construct two complete hierarchies, one for polygons and one for points, with some type of links describing the mappings between the two. Then all this information would be available at the time of rendering for the best combination of polygons and points to be selected in a view-dependent fashion.

We have opted, at the expense of some flexibility, to pursue a more practical approach of using view-independent information to construct a single hierarchy comprising both polygons and points. Thus all of the important decisions regarding the tradeoffs between the two primitive types have been made before the rendering even begins. This predetermination of the tradeoffs could have negative consequences on how well the decisions are made for any given viewing parameters, but it allows us to build a

simple run-time system based on a foundation of well-known algorithms and data structures.

Figure 1 depicts the components of our system. Our hybrid simplification process may be seen at a high level as a simplification algorithm supporting three different simplification operations: polygon simplification (e.g. edge collapse), point replacement, and point simplification. These operations are performed repeatedly in an appropriate order to ultimately produce a hierarchy. Each operation replaces some subset of the model primitives with a new set of primitives, reducing their complexity and perhaps changing their type. In particular, the point replacement operation converts a triangle into one or more points, which may then be further reduced through point merging operations. The set of operations performed, along with the affected primitives and associated error bounds are all stored in a multi-resolution graph data structure.

The interactive rendering system uses the viewing parameters for a given rendering frame to select an appropriate set of primitives from the multi-resolution graph. This set of primitives completely covers the original model (i.e. the entire model is represented by this set) and provides an appropriate resolution. Our current system allows the user to choose a screen-space error tolerance, and the primitives are chosen to be just complex enough to avoid exceeding this tolerance. Because the set of primitives selected is that which lies along a particular *cut* through the graph, and this cut may be modified incrementally from frame to frame, this selection process is referred to as "adapting the cut". Once the primitives are selected, they are rendered to produce the final image. We next present the essential details of this multi-resolution graph before proceeding to the description of the simplification algorithm.

4 MULTI-RESOLUTION GRAPH

Our *multi-resolution graph* (MRG) data structure is an extension to the elegant *multi-triangulation* (MT) data structure described in detail in [DeFloriani et al. 1997, DeFloriani et al. 1998] (Because the extension is to permit the inclusion of new primitive types, the original name is no longer appropriate). The MRG is a simplification hierarchy in the form of a directed acyclic graph. The graph is represented by a set of nodes, N, connected by a set of arcs, A. There is a unique *source* node at the root of the graph, and a unique *drain* node at the bottom. A small example is shown in Figure 2.

Each node of the MRG represents a change to an underlying geometric model – a refinement if we are traversing downward, or a simplification if we are traversing upward. Thus, as we build this graph (from drain to source, in bottom-up simplification), each of our simplification operations is stored along with its associated error bound as a node.

The primitives of the model are stored with the arcs. The primitives removed from the model by an operation are associated with the child arcs of the operation's node, and those inserted by the operation are associated with its parent arcs (one or more primitives may be stored with an arc). From the arc's perspective, the node beneath it (its *end* node) produces its primitives, and the node above it (its *start* node) consumes them (assuming we are traversing upward).

The arcs represent the dependencies of one mesh operation on another. So, for example, if we wish to perform the refinement indicated by a node, we must first perform the refinement indicated by all of the node's parents. Performing the node's operation amounts to replacing the primitives of a node's parent arcs with those of its child arcs, or vice versa. The model coverage of these two sets of primitives are generally the same to avoid local cracks (i.e. missing surface coverage) or multiple coverage (which may be inefficient, but not necessarily incorrect) across the model.

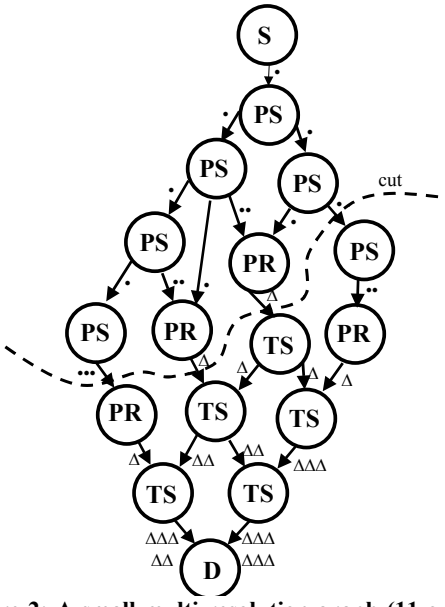


Figure 2: A small multi-resolution graph (11 original triangles). The letters indicate the various node types: D (drain), S (source), TS (triangle simplification), PR (point replacement), and PS (point simplification). The cut contains 2 triangles and 4 points.

To extract a connected, consistent representation of the surface, we generate a *cut* of the graph. A cut is a set of arcs that partitions the nodes of the MRG, leaving the source node above the cut, and the drain node below it. In addition, if the cut contains arc a, then it must not contain any ancestor or descendent of a. The triangles of such a cut represent our input surface at some resolution. In general, we find such a cut by performing a graph traversal starting from the source, testing the error of each visited node against a particular error threshold to decide whether to continue. We can also begin the traversal with an existing cut and move portions of the nodes upward or downward across the cut to modify the local resolution of the surface.

We choose to use this MRG as our simplification graph representation because it has a couple of desirable properties which it inherits directly from the MT.

First, the graph fully specifies all the primitives to be used for all surface resolutions as well as the dependencies between all changes in resolution. Because all the primitives that may be used as part of the rendered model are known in advance, we can provide rigorous bounds on their quality. Not all simplification hierarchies have this property. For example, the well-known simplification hierarchies of [Hoppe 1997] and [Luebke and Erikson 1997] do not have these properties. Looser dependencies may give these hierarchies greater flexibility, because the ordering of vertex merges is not quite so fixed. But the particular triangles that can be extracted from these hierarchies are not known in advance, and vary depending on the order of vertex merges.

Second, the MRG allows for explicit representation of its primitives and a single, general replacement operator. This is incredibly convenient for research purposes when the goal is to explore different primitive types. This very general operation specifies to replace the primitives in set A with the primitives in set B, without any specific knowledge of the primitive types involved. Operators that allow a more implicit representation of

this primitive conversion may produce a more compact data structure, but they are not so convenient for exploration.

5 HYBRID SIMPLIFICATION

As mentioned in Section 3, our simplification process comprises three simplification operations: polygon simplification, point replacement, and point simplification. Although the simplification optimization process could be implemented directly using a single priority queue or queue for each type of operation, we actually separate the simplification process into three distinct components, performing them one after the other in their entirety.

The polygon simplification process explicitly maintains a priority queue of edge collapses that can be used to replace a set of triangles with a smaller set of triangles. A point replacement queue is maintained implicitly in the following way. After computing the optimization value of an edge collapse operation, we evaluate the optimization value of the point replacement operations associated with each of its triangles. If any of these point replacements takes precedence over the edge collapse, we remove the edge collapse from the queue. When the polygon simplification process is finished, point replacement operations are performed on all the remaining triangles. This produces the same result as would an explicit point replacement queue.

Once all the point replacements have been performed, the point simplification process begins. We apply an octree-guided point merging process to simplify the points produced by the replacement operations. The result may differ from a priority-driven point simplification process, but it is efficient and works well in practice.

As the entire simplification process proceeds, we build an MRG from the bottom up (as shown in Figure 2). Each operation we perform adds a node to the graph, and the geometric error bound for the operation is stored with that node. The operation also enables us to create and connect the node's child arcs. The creation of parent arcs is delayed, however, until we know which nodes will consume the primitives created by this operation. If multiple nodes consume these new primitives, then the node gets multiple parents. Notice that each point replacement node has a single child arc containing one triangle, and produces one or more points. Each point simplification node, on the other hand, has at least two child points and produces only a single point. Thus the point simplification nodes can have only one parent arc, and the top portion of our graph is actually a tree.

Now we shall discuss the optimization function used to determine the order of triangle simplification and point replacement operations. We follow this with a more detailed description of each of the three simplification operations.

5.1 Queue Optimization Function

For a given simplification operation, we compute its optimization value as its cost divided by its benefit. The cost is the increase in error, $\Delta\varepsilon$, and the benefit is the decrease in number of primitives, $-\Delta p$, that would occur as a result of performing the operation in question. In fact, we can plot the number of primitives versus the error for the entire simplification process (as in Figure 7), and this optimization function is just the slope of that curve. Thus we attempt to produce a curve in which the error grows as slowly as possible by choosing the operation with the smallest optimization value.

For simplification algorithms that perform only one type of operation, the benefit factor is often unnecessary because it is a constant for all the operations in the process (although for models with borders, operations taking place on the borders generally provide less benefit than those on the interior do).

By ordering both triangle simplification and point replacement operations according to this optimization function, we generally produce error curves that stay entirely below that of a triangle-only simplification process. This outcome relies on the fact that the slope of the point simplification portion of the curve is generally the lowest of all.

However, this conservative ordering delays the introduction of points into the hierarchy, leaving less time to benefit from the small slope of the point simplification. Thus in the interest of producing the best overall curve rather than one which is everywhere beneath the triangle-only simplification curve, we may wish to allow a more aggressive schedule for initiating point replacement operations. To achieve this, we introduce a user-specifiable *transition factor*, τ, which scales the optimization values of all the triangle simplification operations. Setting τ to 1 achieves the same result as the cost/benefit optimization we have already described, whereas setting it to a value greater than 1 will introduce points sooner.

The parameter τ is used to trade an increased primitive count in the lower error ranges for a decreased primitive count in the higher error ranges (seen as a hump in the curve in Figure 8). This is desirable for models of large environments where efficiency for distant portions of the model may be almost as important or more important than efficiency for near portions of the model (because respecting a constant screen-space error tolerance across the model allows greater world-space error for distant portions of the model).

5.2 Triangle Simplification

Our triangle simplification operation is an edge collapse, which merges the two existing vertices of an edge into a single, new vertex. Our implementation is based on the algorithm described in [Cohen et al. 1997], which measures the error of an edge collapse using planar projections. This error is a bound on the Hausdorff distance (a max of min distances) between the original triangles and the simplified triangles. This particular algorithm operates only on manifold surfaces and preserves topology, but this is not a requirement for our hybrid simplification. The only properties we require of a triangle simplification operation is that it provide a guaranteed error bound. (Unfortunately, most operations that allow topology modification or non-manifold inputs do not provide guaranteed error bounds). Thus, a number of other existing algorithms are applicable and may compare favorably [Guéziec 1995, Klein et al. 1996, Lee et al. 1998].

As described earlier, for each edge collapse operation, we also compute the optimization cost of the point replacement of each of its triangles. If any of these point replacements has priority over the edge collapse, the edge collapse is removed from consideration.

5.3 Point Replacement

The point replacement operation provides a means of transitioning from triangles to points in our multi-resolution hierarchy. An important question to consider is how many points we should use to replace a triangle. In a correct MRG, the error always increases as we move upward in the hierarchy. As a matter of principle, we wish to guarantee that the rendering complexity always decreases at the same time. Thus one can always move down the hierarchy to increase quality or up the hierarchy to increase performance. With this in mind, we should never replace a triangle with so many points that the performance is decreased.

To help our system meet this performance constraint, we introduce a system-specific performance ratio, κ, which is the ratio of point-rendering performance to triangle-rendering performance on a particular system. Specifying this ratio correctly should make it

possible to generate a hierarchy that is well tuned for the system in question. Now it is clear that we never wish to replace a triangle with more than $\lfloor \kappa \rfloor$ points, because this would actually decrease performance. Notice that if $\kappa \leq 1$, then a point replacement can never directly increase performance. Using this definition of κ, we can now refer to a number of primitives in this system, as:

$$primitives = triangles + \frac{points}{\kappa}$$

In practice, we want to perform the point replacement such that it minimizes the optimization value for the operation. Using κ we find that the benefit for a point replacement operation is:

$$benefit = 1 - \frac{replacement\ points}{\kappa}$$

This benefit is just the resulting change in primitive count as we replace a single triangle primitive with a number of points equivalent to a fraction of a triangle primitive. The cost of the point replacement is the associated increase in error due to replacing a triangle with points. We can show that if we completely cover the triangle, this cost is just the point radius.

The main tool we have to work with to solve this discrete optimization problem is a procedure for generating a set of replacement samples, given a specified sampling distance. We next describe this sampling procedure and then a method for optimizing the choice of sampling distance for a triangle.

5.3.1 Sampling Procedure

The sampling procedure takes as input a sampling distance and generates a set of point samples that entirely covers a triangle. For a given sampling distance, we wish to generate as few points as possible to optimize performance. These are the main considerations for the sampling algorithm to work well.

As mentioned above, each sampling point can be treated as a sphere, specified by a center (the sample location) and a radius (the sample's range of coverage). It is intuitively clear that a set of spheres (or circles in the plane of the triangle) must overlap somewhat to completely cover the triangle. Squares, on the other hand, tile the plane quite nicely without overlap. Thus we can tile the plane with circular discs that circumscribe these square tiles.

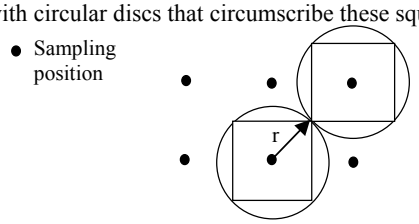


Figure 3: Squares vs. circles to cover a region.

Figure 3 depicts the case in which squares (or circles) just cover the region (i.e. any further separation would lead to a hole). In this sense, it's quite reasonable to use squares to address the sampling problem.

To make the algorithm easy to implement as well as to minimize the number of points, we proceed to sample the triangle one row at a time, as shown in Figure 4. We begin by choosing a coordinate frame such that the largest edge is considered to be horizontal at the bottom of the triangle.

We start the sampling from the left side of edge **AB**, and make the first sampling point cover as much of the triangle as possible but without introducing any gap or hole at the bottom and left side. The following points will be sampled in the same row, which is parallel to edge **AB**, until all the triangle space in that row is covered. This procedure is repeated until the whole triangle is covered. Notice that each row may shift left-to-right with respect to the previous rows, so our sampling does not exactly follow a 2D grid.

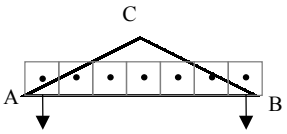


Figure 4: The first row of a triangle is sampled.

In addition to computing the sample's center, we may also wish to sample other attributes, such as color. To make it possible to interpolate rather than extrapolate, we ensure that all the sample centers are located inside or on the edge of the triangle. In Figure 4, if the sampling square has a center outside of the triangle (above edge **AC**), we will push down the square to make the center just located on the edge **AC**. The similar case will occur when the sampling point is close to edge **BC**. To the right side sampling point, we will first move it as left as possible (just cover vertex **B**), then we will adjust it up or down to make it inside the triangle. It is important to get these special boundary conditions right, because when we sample a triangle with a small number of points, all the points may exhibit some boundary condition.

Finally, we need to compute the error bound for a point. In terms of the two-sided Hausdorff distance measure, we know that every point on the triangle is zero distance from a sample sphere, and every point on a sample sphere is within the sphere's radius from a point on the triangle. Thus the incremental error due to sampling is the sphere radius, *r*. If the sampled triangle already has some error bound ε_Δ, then the total error bound of the point is:

$$\varepsilon_p = r + \varepsilon_\Delta$$

Given the neighboring sample distance, *d*, the radius of the circumscribing circle is just:

$$r = d / \sqrt{2}$$

5.3.2 Computing Optimum Sample Distance

Now that we can generate a set of samples or, using the same procedure, *count* the number of samples generated for a given triangle and sampling distance, it is possible to effectively optimize the sample distance using a fairly simplistic approach.

First, we would like the ability to find the smallest sampling distance, *d*, which produces no more than a given number of samples, *s*. We can initialize *d* to a value that makes the total area of *s* square samples equal to the triangle area:

$$Area_\Delta = sd^2 \rightarrow d = \sqrt{\frac{Area_\Delta}{s}}$$

This is the theoretical minimum sampling distance to generate no more than *s* samples. Then we double *d* until it produces more than *s* samples and finally binary search within the resulting interval to find the best sample distance to within some relative tolerance.

Given this discrete function for *d* as a function of *s*, we can now optimize for the *d* that produces the smallest cost/benefit value. The cost value is just the radius, which we have seen in Section 5.3.1 is just a constant multiple of *d*. The benefit value varies with the number of replacement points, *s*. Because *s* as a function of *d* is a step function, this optimum *d* will occur just at the top of one of these steps. Thus the most straightforward way to compute the optimum sampling distance is to find *d* for each integer step with $s<\kappa$ (which we have just described above), and choose the one resulting in the smallest cost/benefit. This works in practice for small values of κ. For larger values, we may wish to estimate a derivative of this step function to provide a faster optimization.

5.4 Point Simplification

After the priority queue has emptied and all remaining point replacements have been applied, we begin the point simplification process by inserting all the original sampling points into the cells of an octree according to the position of the sample center. We then use the octree cells to indicate which sets of points to merge. Each merge can combine up to eight children, and produces exactly one parent point. The center and radius of the new point are chosen such that the parent sphere contains all the child spheres, as shown in Figure 5 (we do not currently use the optimal algorithm, but a simple heuristic). The color of the parent point is a weighted average of those of its children with weights assigned according to surface area.

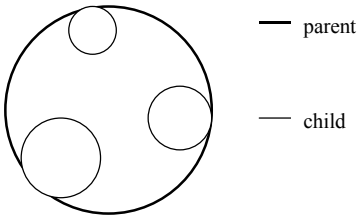


Figure 5: 3 children points are merged into 1 parent point.

Each node of the octree corresponds to one node of the MRG with a single parent arc. As in the case of the original point samples, the radius is only a part of the error bound for the merged point and its generating node; it must be combined with any existing triangle error to compute the total error bound. The total error bound for a point is just its radius plus the maximum triangle error bound component associated with its children (this component is just the child point's error minus its radius). The creation of the root node completes the MRG data structure and the hybrid simplification process.

6 INTERACTIVE DISPLAY

Our interactive display system allows the user to navigate through a 3D environment described by a multi-resolution graph. The model is statically pre-lit with diffuse illumination so that no normals are required for the point primitives (this is still common practice in the image-based rendering community, although current research is gradually reducing this limitation).

The user can select either a screen-space error tolerance, in terms of pixels of deviation, or an object-space error tolerance in terms of a percentage of the length of the environment's bounding box diagonal. Choosing a screen-space tolerance invokes view-dependent refinement every frame as the user navigates the environment. Alternatively, selecting an object-space tolerance causes a one-time refinement to the specified tolerance. This is useful for looking at the various model resolutions up close, and allows navigation without any changes in the model primitives.

As described in Section 4, the set of primitives to be rendered is determined by finding a cut through the graph; the primitives associated with the arcs on the cut are rendered. The cut is adapted by evaluating an error criterion to determine if a node is above or below the current error threshold. For an object-space error tolerance, the stored error is divided by the length of the MRG's bounding box diagonal for comparison with the threshold. For a screen-space threshold, the arc's screen-space depth is computed using a conservative bounding sphere approximation (a bounding sphere is stored with each arc). From this depth, a scaling factor is computed to convert the error length from object-space coordinates to a screen-space pixel distance, which is now comparable with the specified pixel threshold. The same scaling factor is used to convert a point primitive's radius to a screen space radius for rendering as a circle.

Our current implementation renders OpenGL points on a SGI InfiniteReality II platform. Other more efficient point rendering systems have been developed [Grossman and Dally 1998, Pfister et. al 2000, Rusinkiewicz and Levoy 2000], some of which include texture filtering. Any of these can readily be used in the context of our hybrid simplification framework. We hope to take more of a 3D image warping approach in the future, incorporating some form of LDI tree [Chang et al. 1999] or a derivative of it, in order to achieve greater point performance (and thus a larger κ).

7 RESULTS

We have implemented both the hierarchical simplification and interactive display algorithms described in Sections 5 and 6, and tested them on several models. These models are listed in Figure 6. Most of the preprocessing time is spent in the evaluation and prioritization of potential edge collapses. This time is increased somewhat by the optimization of sampling distances for potential point replacement operations, but not excessively for small values of the input parameter κ. The time required for actually generating the samples and performing the octree-based point simplification is typically quite small compared to the rest of the algorithm.

Model	Input Tris	MRG Tris	MRG Points	Simp Time
Armadillo	1,999,404	8,962.427	154,651	111:29
Bunny	69,451	308,738	10,825	1:56
Bronco	74,308	257,694	70,841	2:29
Horse	96,966	432,878	10,429	2:45

Figure 6: Test Models (data reported for κ=3, τ=1; simplification time in minutes:seconds).

It is informative to observe the behavior of the curves produced by plotting the number of primitives versus the object-space error for various choices of the κ and τ parameters. Figure 7 shows such a plot for the Bronco model, with a fixed value of κ=3 and a several different values of the transition factor, τ. Notice that setting τ=1 achieves a curve that is everywhere less than or equal to the curve of triangle simplification alone (this is not actually guaranteed by this non-optimal greedy process). However, if we relax this constraint by increasing τ, we reduce the error values for the lower primitive counts at the expense of increasing it for some of the higher primitive range.

Looking at Figure 8 shows us that the bunny model does not benefit as much from point replacement as much as the Bronco. This is not surprising, because the bunny is a single, highly tessellated manifold surface, while the Bronco comprises 339 individual manifolds, which are not so highly tessellated to begin with. These manifolds are not merged in the triangle domain by our polygon simplification algorithm, so for this reason as well, they benefit from the transition to points.

Figure 9 examines changing the system performance ratio, κ, for a fixed value of τ (τ=1.0). This plot demonstrates, not surprisingly, that if we develop systems with increased levels of point-rendering performance, our hierarchies will directly benefit by switching to points sooner in the simplification process.

In Figures 10 and 11, we report the simplification results for a fixed screen-space error of 3 pixels. We fly-through an environment consisting of 73 Bronco models and record the number of primitives for hybrid simplification with κ=3, τ=1, for hybrid simplification with κ=3, τ=5, and for triangle-simplification only. We are able to reduce the primitive count by an average factor of 1.6 (minimum 1.3, maximum 4.7). As expected, the simplification for τ=5 is slightly better when most of the environment is in the distance (e.g. beginning of the path) and slightly worse when the

viewer is mostly surrounded by the environment (e.g. middle of the path).

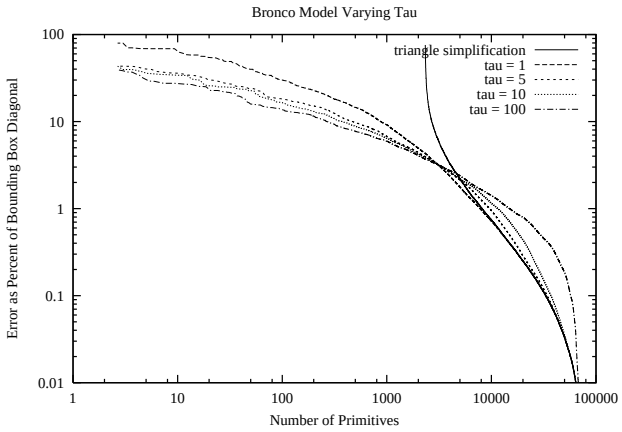


Figure 7: Varying τ for the bronco with κ=3.

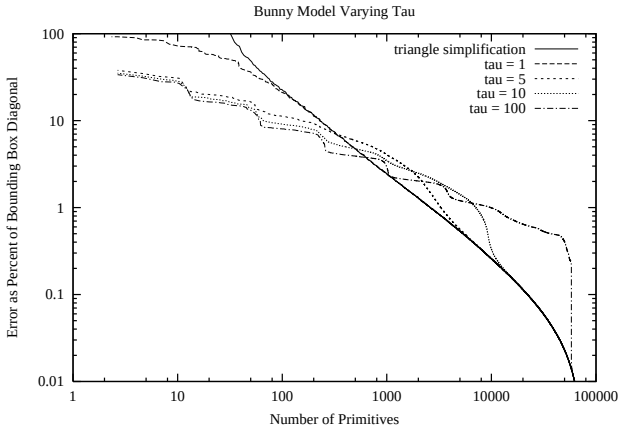


Figure 8: Varying τ for the bunny model with κ=3.

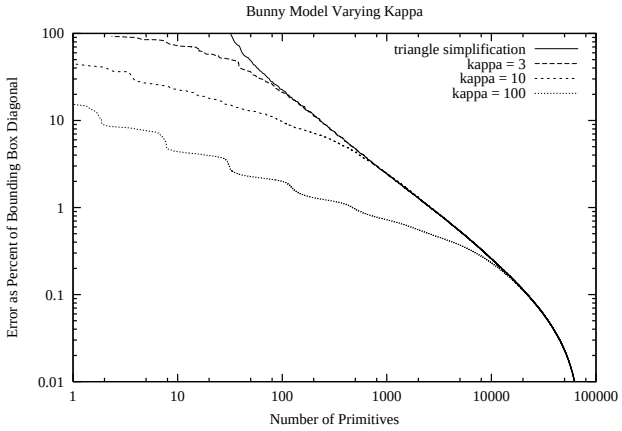


Figure 9: Varying κ for the bunny model, with τ=1.

These graphs show us the nature of error growth in the hierarchies, but they cannot portray the localization of point replacements or the geometric configurations where point replacement is called for. We get a much better intuition for these characteristics from rendered images of the test models.

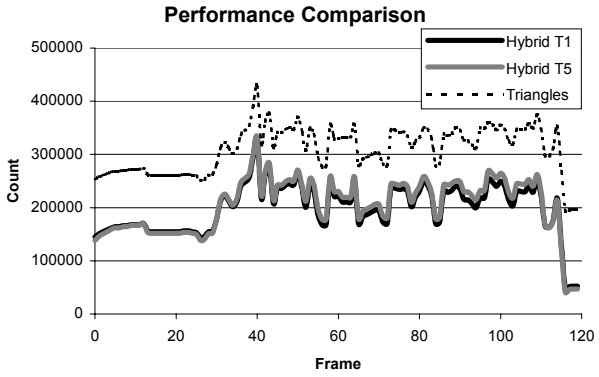


Figure 10: Total primitive counts for a fly-through of the Bronco environment. Hybrid T1/T5 refers to a hybrid simplification using κ=3, τ=1 and κ=3, τ=5, respectively. Triangles refer to a pure triangle simplification.

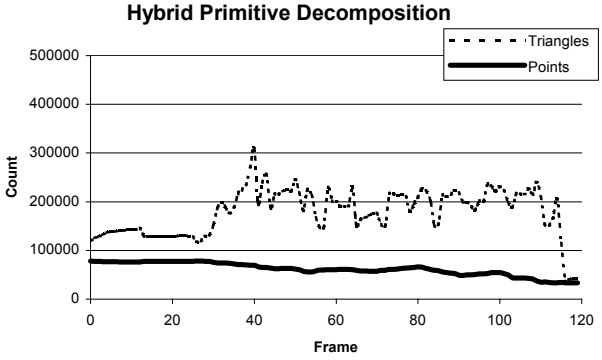


Figure 11: We show the decomposition of the hybrid primitives during the example fly-through of the Bronco model.

Figure 12 shows the bunny being gradually covered by points as it recedes into the distance. The first points appear around the sharp tips of the ears and the curves of the toes, then clusters of points appear in the ridges of the neck and hindquarters, and finally the rest of the triangles are consumed by this phenomenon. The last places to remain triangles are the back and rear of the bunny, which are the flattest. It is pretty clear that the points benefit the most in the regions of high curvature, where simplification is most limited. This occurs primarily when the dihedral angles between faces become small and the mesh is the coarsest. Figure 15 shows the simplification by increasing an object-space error tolerance; thus the rendered points nearer to the eye position are larger, with larger screen-space error.

Figures 13, 14, and 17 show similar transitions for the Bronco, horse, and armadillo models. The Bronco in particular suffers from self-penetration artifacts due to many surfaces with different colors and tight tolerances. We have shown it for a system with κ=10 rather than κ=3 because such a system can reduce these artifacts somewhat by using finer point samples. Triangle simplification alone also suffers from such artifacts on a model like this, but they may be less pronounced than those of the fatter point primitives are.

Figure 16 shows captured frames from the Bronco environment fly-through and also a comparison to a frame with a screen-space error of one pixel rendered using triangle-only simplification.

8 DISCUSSION AND FUTURE WORK

To our knowledge, the hybrid simplification framework presented is the first system that tightly integrates polygon and point rendering into a single multi-resolution hierarchy. This hierarchy is optimized according to the relative performance characteristics of the primitive types on a particular architecture. For a given error bound, it achieves a greater reduction in the overall primitive count as compared to a single-representation hierarchy. As part of this research, we have explored two parameters that influence the characteristics of the hierarchy we build, and we have investigated how the replacement of triangles with points manifests itself in the context of several models.

As future work, we plan to use our hierarchy with a data structure such as an LDI tree. The exact arrangement of the images we warp may differ from other 3D image warping applications because our points are being dynamically added to and removed from inclusion in the warp. One challenge will be to keep our images dense enough with points for warping that we still benefit from the additional efficiency it provides over the transformation of individual points.

Another interesting avenue for exploration is the construction of such a hybrid system in conjunction with a topology-modifying simplification operator. Allowing topological modifications in the polygon domain may "level the playing field" due to their ability to continue simplifying longer. It will be interesting to see how the comparison plays out when both operators can merge objects.

And finally, the system we have presented here only takes into account the geometric deviation of the simplification. It does not account for color error, texture error, nor does it provide for illumination of points. Although measuring the error of attributes is somewhat well understood in their own domains, it is much less clear how to combine them into a useful screen-space metric. Thus attribute errors may be used to guide the off-line optimization, but their use in the interactive display system is a more open-ended problem.

9 ACKNOWLEDGMENTS

We would like to thank Subodh Kumar for help with the MT library and viewer applications and the Bell Labs Multimedia Lab for their support of this research. We also extend our gratitude to the Stanford Computer Graphics Lab for the bunny model, Viewpoint Labs for the Bronco model, Venkat Krishnamurthy, Marc Levoy, and Peter Schröder for the armadillo model, and Cyberware for the horse model.

REFERENCES

Chang, Chun-Fa, Gary Bishop, and Anselmo Lastra. LDI Tree: A Hierarchical Representation for Image-Based Rendering. *Proceedings of SIGGRAPH '99.* pp. 291-298.

Cohen, Jonathan, Dinesh Manocha, and Marc Olano. Simplifying Polygonal Models using Successive Mappings. *Proceedings of IEEE Visualization '97.* pp. 395-402.

Curless, B and M Levoy. A Volumetric Method for Building Complex Models from Range Images. *Proceedings of SIGGRAPH '96.* pp. 303-312.

DeFloriani, Leila, Paola Magillo, and Enrico Puppo. Building and Traversing a Surface at Variable Resolution. *Proceedings of IEEE Visualization '97.* pp. 103-110.

DeFloriani, Leila, Paola Magillo, and Enrico Puppo. Efficient Implementation of Multi-Triangulations. *Proceedings of IEEE Visualization '98.* pp. 43-50.

El-Sana, Jihad and Amitabh Varshney. Controlled Simplification of Genus for Polygonal Models. *Proceedings of IEEE Visualization'97.* pp. 403-410.

Erikson, Carl and Dinesh Manocha. GAPS: General and Automatic Polygonal Simplification. *ACM Symposium on Interactive 3D Graphic s '99.* pp. 79-88.

Garland, Michael and Paul Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. *Proceedings of IEEE Visualization '98.* pp. 263-270.

Garland, Michael and Paul Heckbert. Surface Simplification using Quadric Error Bounds. *Proceedings of SIGGRAPH '97.* pp. 209-216.

Grossman, J P and W Dally. Point Sample Rendering. *9th Eurographics Workshop on Rendering '98*, pp. 181-192.

Guéziec, André. Surface Simplification with Variable Tolerance. *Proceedings of Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery (MRCAS '95).* pp. 132-139.

Hoppe, Hugues. Progressive Meshes. *Proceedings of SIGGRAPH '96.* pp. 99-108.

Hoppe, Hugues. View-Dependent Refinement of Progressive Meshes. *Proceedings of SIGGRAPH '97.* pp. 189-198.

Hoppe, H, T DeRose, T Duchamp, J McDonald, and W Stuetzle. Surface Reconstruction from Unorganized Points. *Proceedings of SIGGRAPH '92.* pp. 71-78.

Klein, Reinhard, Gunther Liebich, and Wolfgang Straßer. Mesh Reduction with Error Control. *Proceedings of IEEE Visualization '96.*

Lee, Aaron W. F., Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Proceedings of SIGGRAPH '98.* pp. 95-104.

Levoy, Marc and Turner Whitted. The Use of Points as a Display Primitive. Technical Report TR 85-022. University of North Carolina at Chapel Hill. 1985.

Luebke, David and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. *Proceedings of SIGGRAPH '97.* pp. 199-208.

Max, N and K Ohsaki. Rendering Trees from Precomputed Z-Buffer Views. *Proceedings of Rendering Techniques '95.* pp. 45-54.

McMillan, L and G Bishop. Plenoptic Modeling: An Image-Based Rendering System. *Proceedings of SIGGRAPH '95.* pp. 39-46.

Pfister, H, M Zwicker, J van Baar, and M Gross. Surfels: Surface Elements as Rendering Primitives. *Proceedings of SIGGRAPH 2000.* pp. 335-342.

Popovic, Jovan and Hugues Hoppe. Progressive Simplicial Complexes. *Proceedings of SIGGRAPH '97.* pp. 217-224.

Reeves, W T. Particle Systems: A Technique for Modeling a Class of Fuzzy Objects. *Proceedings of SIGGRAPH '83.* pp. 359-376.

Rossignac, Jarek and Paul Borrel. Multi-Resolution 3D Approximations for Rendering. *Modeling in Computer Graphics.* Springer-Verlag 1993. pp. 455-465.

Rusinkiewicz, S and M Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. *Proceedings of SIGGRAPH 2000.* pp. 336-352.

Schroeder, W. A Topology-Modifying Progressive Decimation Algorithm. *Proceedings of IEEE Visualization'97.* pp. 205-212.

Xia, Julie C., Jihad El-Sana, and Amitabh Varshney. Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models. *IEEE Transactions on Visualization and Computer Graphics.* vol. 3(2). 1997. pp. 171-183.

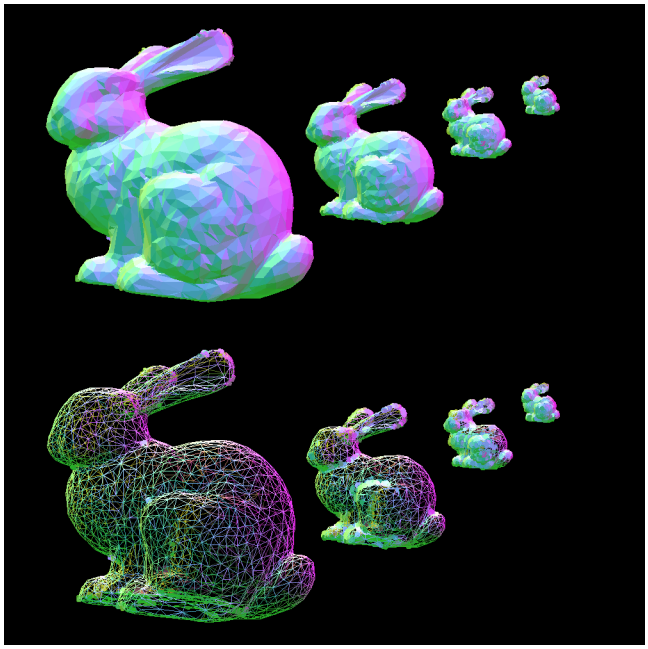


Figure 12: Bunny model with screen-space error of 5 pixels (κ=3, τ=5). (wireframe indicates triangles)

Figure 13: Bronco model with 5 pixels of deviation and zoomed in at 20 pixels of deviation (κ=10, τ=1).

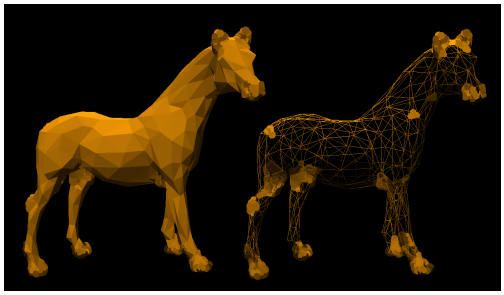


Figure 14: View of the horse model with 20 pixels of deviation (κ=10, τ=1).

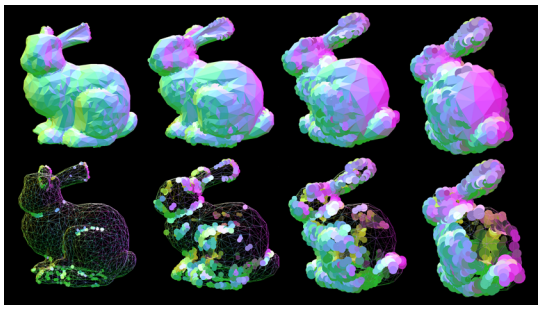
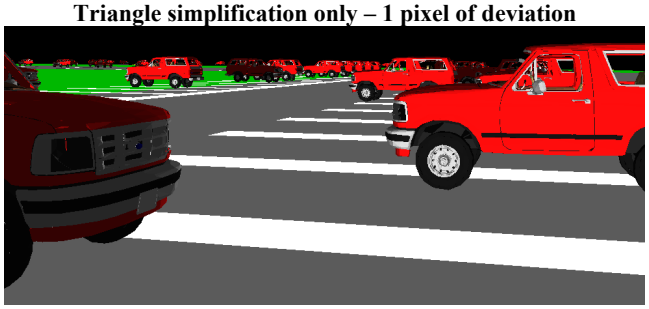
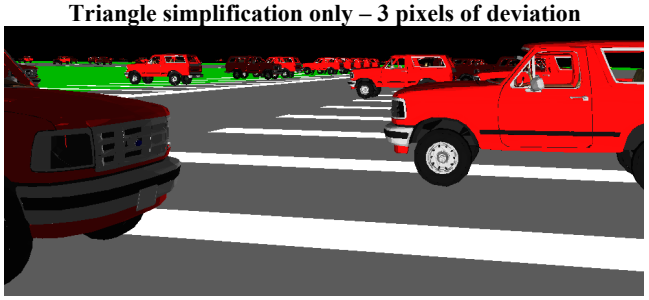


Figure 15: Bunny model with object space deviation of 1%, 2%, 3%, and 4% of its bounding box diagonal (κ=3, τ=5).

Triangle simplification only – 1 pixel of deviation



Triangle simplification only – 3 pixels of deviation



Hybrid simplification – 3 pixels of deviation (κ=3, τ=1)

Figure 16: One frame from the Bronco environment fly-through.

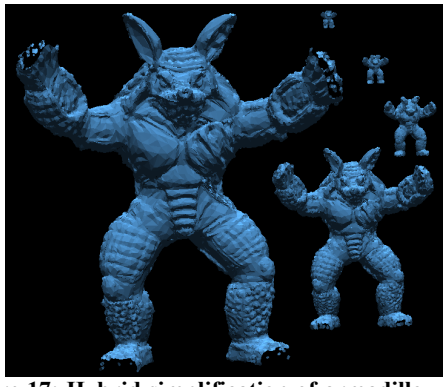


Figure 17: Hybrid simplification of armadillo with 3 pixels of deviation (κ=3, τ=5).