Hierarchical Simplification of City Models to Maintain Urban Legibility

Remco Chang† Thomas Butkiewicz† Caroline Ziemkiewicz† Zachary Wartell† Nancy Pollard‡ William Ribarsky†
University of North Carolina Charlotte† Carnegie Mellon University‡
{rchang, tjbutkie, caziemki, zwartell, ribarsky}@uncc.edu {nsp}@cs.cmu.edu

Figure 1: Left to right: (a) Original models (285,039 polygons); (b) Simplified models using our algorithm (131,409 polygons); (c) Models from (b) that have been simplified are shown in yellow; (d) Here we show view-dependent simplification of (b) from a top-down view. Notice that simplified models closer to the eye point have more detailed geometry, while the models farther away have less.

Abstract

Mesh simplification and discrete levels of detail (LOD) are well-studied areas of research in computer graphics. However, until recently, most of the developed algorithms have focused on simplification and viewing of a single object with a large number of polygons. When these algorithms are used on a large collection of simple models, many objects may be completely erased, leading to results that are misleading to the viewer. In this paper, we present a novel approach to simplifying city-sized collections of 2.5D buildings based on the principles of “urban legibility” as defined by architects and city planners. Our main contributions include a clustering algorithm tailored towards forming logical groups while respecting roads, a polyline simplification algorithm that maintains boundary facades, and a LOD process that preserves landmarks and skylines. The advantage of our approach is that the legibility and understandability of a complex urban space is preserved at all levels of simplification.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Hierarchy and geometric transformations—Curve, surface, solid, and object representations;

Keywords: clustering, levels of detail, simplification, urban legibility

1 Introduction

Traditionally, research in the areas of mesh simplification and levels of detail has focused on complex models with natural shapes. However, with the advent of 3D global visualization tools for public use such as Google Earth ([Google 2005]), the ability to render and visualize a large collection of simple models such as buildings has become increasingly important. Beyond this, there is the need to make the rendering of any urban space useful for tasks such as navigation or spatial mental maps.

Existing techniques for mesh simplification can have trouble with models of buildings, which are often nothing more than boxes with eight vertices in which polygon decimation results in a mesh that no longer retains the appearance of a building. Furthermore, when polygon decimation is performed on a collection of simple meshes given a target polygon count, the smaller objects are often completely decimated because their removal causes less overall “geometric error” (see Figure 2). For a city-sized collection of simple buildings, this could mean the disappearance of an entire residential area in which the buildings tend to be smaller than that of commercial regions. This simplified version of the city model no longer “resembles” the un-simplified one.

In this paper, we incorporate concepts from architecture and city-planning as guidelines to performing mesh simplification. Specifically, we examine the concept of “urban legibility” on which Lynch [1960] has written that the “image” of a city can be categorized into paths, edges, districts, nodes, and landmarks. We believe that by maintaining this “image,” we can produce simplified models that are better understood by viewers. Although these categories are qualitative measurements, each step of our algorithm is based on considerations of one or more of these concepts.

The key idea of our algorithm is based on merging of similar elements. Consider a row of identical houses separated by little space; when these houses are viewed from afar, we should be able to combine their geometries and render them together as one single model. To accomplish this goal, we break our algorithm down into five steps. Hierarchical clustering, cluster merging, model simplification, and hierarchical texturing are performed during pre-processing, and the runtime LOD selects the appropriate models to render. Hierarchical single-link clustering is adopted to cluster models of buildings following the principles of paths and edges. Polylne-based cluster merging creates logical districts and nodes. Model simplification preserves the paths, edges, districts, and nodes created in the previous two steps, and finally, the LOD
process allows the preservation of certain landmarks. Figure 1 provides an overview of the application in action.

Figure 2: Left: Original model. Middle: model decimated using QSLIM. Right: model simplified using our algorithm. The number of polygons used to generate the QSLIM image is the same as the one used to generate the image on the right.

2 Related Work

A tremendous amount of research has been put into mesh simplification techniques, see [Luebke 2001]. We will only focus on representative work that is most relevant to our algorithm, specifically, view independent, “topologically-tolerant” vertex merging techniques. These techniques combine vertices based on their proximities and similarities to other vertices and thus are able to merge multiple meshes into one. Methods developed by Rossignac and Borrel [1993] and Low and Tan [1997] use vertex clustering and are tolerant in terms of input meshes’ connectivities and topologies, but do not guarantee that the output model is free of dangling edges and vertices, or that it visually resembles the input meshes. More recently, Lindstrom [2003] uses octrees based on vertex clustering to partition space and construct multi-resolution meshes in an out-of-core fashion. Garland and Heckbert [1997] introduce QSLIM, in which “virtual” edges are added between unconnected vertices that are within a user-specified Euclidean distance \(\tau\). These virtual edges are treated in the same manner as actual edges in the mesh. Erikson and Manocha [1999] further extends Garland’s concept and dynamically calculates the value for \(\tau\), and they apply the simplification algorithm in a hierarchical fashion [2001]. To eliminate the effect of “popping” when switching between different LODs, Borget et al. [2005] use geomorphing on a hierarchy of pre-optimized geometry patches. Lastly, we take inspiration from Jang et al. [2005] who suggest that for man-made objects, removal of entire features is typically more visually “understandable” than vertex removal. Our goal of urban legibility requires different design decisions than previous work, especially in selecting single-link clustering as an initial pre-processing step (Section 4).

Polyline (curve) simplification has a much longer history than mesh simplification. A survey by Weibel [1997] gives a good overview of the methods, and [Garland and Heckbert 1997] provides a synopsis of some of the most notable ones. Algorithms specifically relevant to this paper are the Douglas-Peucker algorithm [Douglas and Peucker 1973], based on recursive line subdivision, and quadric-based simplification [Garland and Zhou 2005]. Due to our unique approach to merging clusters (see Section 5), we found it necessary to develop a new polyline simplification algorithm that preserves vertex positions as well as the overall shape of the clusters.

Although there have been numerous applications in 3D global visualization and GIS visualization (see [VTP 2006] for a select few), the majority of the research has focused on terrain simplification and visualization (see [Losasso and Hoppe 2004] for a survey of the various algorithms). The use of simplification for displaying collections of building models has been mostly limited to discrete levels of detail in which buildings beyond a certain distance are not rendered, with the exception of [Coors 2001], where QSLIM and quad-trees are utilized to preserve buildings that are in focus. To the best of our knowledge, no existing simplification algorithm prioritizes higher levels of knowledge such as urban legibility.

Mesh simplification is not the only way to visualize an urban environment efficiently. Wand et al. [2001] use a randomized z-buffer algorithm to render complex scenes at interactive rate. Some researchers have explored the use of image-based rendering techniques for urban scenes (e.g., [Sillion et al. 1997] [Maciel and Shirley 1995]). Image-based rendering approaches produce visually compelling results. However, the problems of preserving occlusion and skyline effects in city scenes require the use of extremely large numbers of images. We also note that visibility culling is important. For city scenes in particular, when fly-throughs are near street level, occlusion culling as described in [Wonka et al. 2000], [Schaufler et al. 2000], and [Chhugani et al. 2005] is critical for achieving optimal performance.

Urban legibility was introduced by Lynch [1960], and the idea has served as inspiration for building virtual worlds (e.g., [Ingram and Bowers 1996]), wayfinding in virtual environments (see [Dalton 2002] for an overview of selected work), and navigation through abstract data (e.g., [Ingram and Benford 1995]). A number of researchers have also performed user studies to investigate the effectiveness of urban legibility in wayfinding in virtual environments (a comprehensive survey can be found in [Dalton 2002]).

3 Urban Legibility

Urban legibility is a concept that has been used for many years in the area of city-planning. In his book The Image of the City published in 1960, Kevin Lynch ([Lynch 1960]) defines legibility as:

“...the ease with which its parts may be recognized and can be organized into a coherent pattern.”

“Coherent pattern” refers to cues that people use in order to “structure and identify the environment,” and Lynch further classified them into five types of elements:

Paths: Avenues of travel, such as streets, walkways, railroads, canals, etc.

Edges: Linear elements not considered as paths, including structures or features providing boundaries. For example, shorelines, edges of development, walls.

Districts: Medium to large sections of the city which an observer mentally “enters.” For example, a historical residential area.

Nodes: Strategic spots of intense activity and/or information flow, occurring most frequently at junctions of paths. For example, Times Square in New York City.

Landmarks: Recognizable objects that are distinctive to the observers. Examples include towers, sign posts, hills, etc.

Although Lynch defines these elements as cues used by the inhabitants of a city, we believe that these elements also help people better recognize a city from a bird’s-eye view. Therefore, it is with these five elements in mind that we devise our algorithms for simplifying and viewing a collection of buildings.

Figure 3 shows the five steps in our algorithm. Our hierarchical clustering maintains paths and edges when grouping similar buildings together; cluster merging combines the geometries of the buildings into one single model and creates districts and nodes;
simplification reduces the geometric complexity of the model, but preserves paths, edges, districts and nodes; texturing adds visual fidelity to the created model; and finally, the LOD process selects the appropriate models to render at runtime, while preserving the landmarks in the scene. The next sections describe the five steps in our algorithm.

4 Hierarchical Clustering

Lynch considers paths as the predominant city elements, so it is critical that our clustering algorithm does not cluster buildings on opposite sides of a path. In our algorithm, we maintain both paths and edges by preserving empty spaces between buildings. To achieve this, we use single-link clustering, which creates clusters that “follow along” a path. In contrast, k-means and complete-link clustering both produce “oval-shaped” clusters (Figure 4). For a survey on clustering techniques, see [Berkhin 2002].

Figure 4: Left: single-link clustering. Right: complete-link clustering.

Single-link clustering is an agglomerative (bottom-up) hierarchical clustering technique which bases its clustering criteria on a distance metric. To create a moderately balanced tree, we use a distance metric that incorporates cluster size:

\[
d(C_1, C_2) = \min \left\{ \frac{\text{size}(C_1) \cdot \text{size}(C_2)}{\text{avgClusterSize}^2}, d(x,y) \mid x \in C_1, y \in C_2 \right\}
\]

(1)

where size($C_i$) denotes the number of buildings in cluster $C_i$, $d(C_1, C_2)$ is the cost of merging clusters $C_1$ and $C_2$, $d(x, y)$ is the Euclidean distance between buildings $x$ and $y$ that can be defined as the distance between the center point of the buildings, and avgClusterSize is the average number of buildings in all the existing non-leaf nodes.

Note our single-link clustering is a $O(n^2)$ algorithm because the avgClusterSize changes at every step, so all distances need to be recomputed. For efficiency, we use an approximate $O(n^2)$ algorithm that updates distance measures only for the clusters that are affected by a merge. For example, if clusters $C_a$ and $C_b$ are merged, all distance metrics referring to these two clusters are updated. Although this approximate algorithm is not exact, this step is necessary to accommodate large datasets.

5 Creating and Merging Hulls

Once the clustering process is complete, each node in the hierarchy contains a number of buildings that are geographically near each other, and are roughly bounded by paths and edges. We then merge the buildings within each cluster into a single model (Figure 12(b)), which contains and resembles the aggregate of the buildings. This merger often creates logical districts (Figure 5).

Figure 5: Creating a district by merging two clusters (left and middle) into one cluster (right).

Because all the buildings are 2.5D, we can consider merging the footprints of the buildings separately from merging the heights of the buildings. This also permits us to apply different rules to the footprints and the heights. First we find all the boundary edges of the footprint (known as footprint edges) of every mesh, and order the edges in a counter-clockwise manner (Figure 6(a)). We call these directed footprint edges the “Hull” of the mesh. Note that this Hull is not necessarily the convex hull.

To compute the Hulls of the non-leaf nodes, we recursively merge the Hulls of their two child nodes (Hull1 and Hull2). There are four different possible scenarios when merging two Hulls:

No Intersections Between Hulls: Find the two shortest, non-intersecting edges between the two Hulls (called connection edges), and select any one of the four vertices (called connection vertices) on those two edges as the pivot (Figures 6(a) and (b)). Starting from the pivot, follow the footprint edges of both Hulls. When the trace returns to the pivot point, the area forms either the merged Hull (Figure 6(c)), or the spatial error introduced by merging the two Hulls (Figure 6(d)). If the result is the latter, choose a different connection vertex as pivot and repeat the process.

Hulls Bisect Each Other: In order to merge two bisecting Hulls (Figure 7(a)), more than two new edges are required, rendering the technique described above insufficient. A full search for the merged Hull begins with the convex hull of Hull1 and Hull2, followed by
Because this process is \( O(n^2m^2) \) where \( n \) and \( m \) are the sizes of CandidateList1 and CandidateList2, respectively, we limit the use of the heuristic search to \( n \times m \leq 12 \); otherwise, a full Hull search is used. Similarly, if no valid Hull can be found using the heuristic search (Figure 7(c)), a full hull search is also performed.

**Hulls Intersect with Intruding Vertices:** We begin the search for a merged Hull heuristically. The candidates for connection edges are all pairs of vertices on the intersecting edges from both Hulls (Figure 7(b)). In other words, for each edge in Hull1 that intersects Hull2 (including edges that are completely within Hull2), its two vertices are (uniquely) added to a list of candidate vertices called CandidateList1, and the same process is performed on Hull2.

We then choose two distinct candidate edges as connection edges and perform a trace as described in the previous scenario. “No Intersections Between Hulls.” If there is any building that falls outside the merged Hull, we reject this pair of candidate edges and choose another. This process is repeated until a valid Hull is found. Because this process is \( O(n^2m^2) \) where \( n \) and \( m \) are the sizes of CandidateList1 and CandidateList2, respectively, we limit the use of the heuristic search to \( n \times m \leq 12 \); otherwise, a full Hull search is used. Similarly, if no valid Hull can be found using the heuristic search (Figure 7(c)), a full hull search is also performed.

**Hull1 inside Hull2 or vice versa:** The merged Hull is simply a copy of the outer Hull.

### 6 Polyline Simplification

A basic polyline simplification approach is the Douglas-Peucker algorithm, in which a single line segment connecting the ends of the input polyline is recursively subdivided to approach the input polyline. The subdivision terminates when any further subdivision would create line segments shorter than a user-defined threshold. Although this algorithm produces a polyline that converges quickly to the general shape of the input polyline, the intermediary stages often resemble a star with sharp angles in which paths, edges, nodes, and districts are not preserved.

Instead, we develop a polyline simplification algorithm based on the principles of a convex hull which we believe preserves the elements of paths, edges, nodes, and districts. Using the polyline generated from the previous section (which is non-self-intersecting and watertight), we iteratively remove vertices by connecting their neighbors. At each step, we remove the vertex that adds the smallest positive area to the footprint while maintaining a non-self-intersecting constraint. The one-mouth theorem [Toussaint 1991] shows that there is always a vertex that can be removed in this way until the convex hull is found.

![Figure 7](image)

**Figure 7:** (a) Hulls that bisect each other; (b) Two intersecting Hulls. The green circles show the vertices on CandidateList1 and the pink circles show the vertices on CandidateList2; (c) An example of when the heuristic search fails. The dotted green lines represent two of the three candidate connection edges. Notice that both of them cause self-intersections; (d, e, f) The merged Hull from (a), (b), and (c) respectively.

### 7 Hierarchical Texture

As with the above urban geometry simplification, the purpose of our hierarchical texture approach is not visual quality in its strictest sense, but rather legibility of the urban environment at all scales. As such, the main goal for texturing is not necessarily to enforce small or even unnoticeable pixel errors. Instead, the goal is to create textures that maintain legibility and interactivity.

It is generally accepted that texture mapping is still one of the most resource-intensive processes in graphics rendering. For our application, the texture problem is doubled because we generate \( n - 1 \) new cluster meshes in which the geometries are often different from the original models, making it impossible to reuse their textures.

To create side textures for each cluster mesh, we iteratively generate an image for each face by placing an orthographic camera such
that its near clipping plane lies on the face. We then render an image with all the buildings scaled to the height of the cluster mesh. The combined images from all faces are set to fit into a single texture, with the resolution of each image proportional to the length of each face. The purpose of scaling the buildings to the height of the cluster mesh is to create a more pleasing and continuous texture, especially towards the seam between the faces and the roof. Although some buildings become “stretched” or “squashed,” the scaling is necessary to preserve understandability of the textured cluster.

Texturing the roof is more difficult because the negative spaces between buildings are more visible from a top-down view. If the camera angle changes slightly from such a view, the viewer expects to see parts of the facades of the buildings. We take images of the roof from five different camera angles – top-down view, and 45-degree views from north, east, south, and west (Figure 9). Buildings are again scaled to the height of the cluster mesh to avoid “shifting” between the camera angles. During runtime, the system chooses the texture that is closest to the look vector.

We sort the clusters in ascending order of their cluster meshes’ negative space area, then divide them into $\log(n)$ bins, with the first level bin having the first $n/2$ clusters, the next level $n/4$, etc. Six texture files are generated for each bin (one side texture and roof textures from the five different angles), with all texture files from all bins having the same resolution (defaulted to $1024 \times 1024$). For example, the cluster meshes in the first bin would each have $\frac{1024 \times 1024}{n/2}$ pixels for each of their six textures, while the single cluster mesh in the last bin would have all $1024 \times 1024$ pixels. This results in the overview of the city model getting the most texture. Our hierarchical approach is based on the observation that, at any given time, the LOD process on our balanced hierarchy tends not to render cluster meshes of drastically different depths in the tree.

To ensure that there is enough memory for the textures, we implemented a simple priority queue similar to the one described by Erikson et al. [2001] in which the least recently used texture is swapped out when memory becomes a constraint.

8 Negative Spaces and Level of Detail

During both the Hull Merging process (Section 5) and the Hull Simplification process (Section 6), geometric errors are introduced into the final mesh. We call these geometric errors “negative spaces” because geometry is added to areas where there used to be empty spaces.

The area of the negative space of a cluster mesh is the difference in area between its footprint and the sum of the buildings’ footprints. Our LOD algorithm will not render a cluster if the visual effect of this area is too large. In our implementation, we approximate this negative-space area as a rectangle with the same ratio in dimensions as the axis-aligned bounding box of the merged Hull. During the LOD process, this negative-space area is converted into a 3D box with the same height as the cluster mesh. The camera-facing faces of the box are projected onto screen space, and the number of pixels is compared against a user-defined tolerance ($\varepsilon$). If the number of pixels is greater than $\varepsilon$, the cluster will not be rendered, but its descendants will be checked recursively. Figure 15(a) (Section 9) shows the effect of $\varepsilon$ on actual screen pixel error in a fly-through sequence. The concept of landmarks is perhaps the most subjective of Lynch’s categories. However, it seems reasonable that taller buildings have higher visual importance than shorter ones because of their roles in defining the skyline. A user-defined threshold ($\alpha$) in numbers of pixels is used to determine the acceptable error in height. During runtime, $\alpha$ is projected onto each cluster mesh and converted to a height value (called $\alpha_{\text{height}}$), shown in Figure 10(c). If any building is taller than its cluster’s $\alpha_{\text{height}}$, the original building mesh is rendered along with the cluster mesh (Figure 10(b)).

9 Results and Analysis

In this section, we show the results from various steps of our algorithm and perform analyses using a dataset of downtown Xinxiang (a city in the Henan province in China), which contains 26,016 buildings and 285,039 polygons, and a dataset containing parts of Atlanta, Georgia, which has 11,536 buildings and 243,381 polygons. All tests are performed on a Pentium 4 3.0GHz desktop computer with 2.0GB of memory, using an nVidia 6800 graphics card with 256MB of memory. In all the analyses, no display lists are used.

First, we show some results of our clustering. Figure 11(a) shows the original layout of the buildings, and 11(b) shows a few clusters in that area. Notice that our clustering algorithm follows the paths and edges around the buildings, and creates “natural-looking” clusters.

In Figure 12, we show the result of applying merging and simplification to the clusters in Figure 11(b). The models are drawn as un-textured cluster meshes (Figure 12(b)), textured cluster meshes (Figure 12(c)), and placed next to the original (un-simplified) models (Figure 12(a)). It can be seen that our merging and simplification
process produces cluster meshes of simple geometries while maintaining districts and nodes. With the addition of textures, the textured cluster meshes resemble the original models both in appearance and in legibility. Figure 12(d) shows using QSLIM to decimate the same set of clusters. Although QSLIM is one of the best algorithms for simplifying natural objects, comparing Figures 12(c) and (d) demonstrates that QSLIM is not well suited in retaining legible urban features.

To maintain a legible and consistent skyline, we allow the user to modify the user tolerance \( \alpha \) (see Section 8). In Figure 13, we see the effect of changing the value of \( \alpha \). As can be seen, by effectively controlling the \( \alpha \) value, we can maintain a recognizable skyline similar to the original un-simplified model, while incurring a low cost in terms of polygon count. For example, the scene in Figure 13(c) contains 13,712 polygons (5.6% of the polygons in the original scene), while the scene in Figure 13(b), with landmark buildings rendered separately, contains only 2114 additional polygons. By setting \( \alpha \) to 10000 in 13(c), we effectively remove consideration of landmarks and render all clusters at their average heights.

Figure 14(a) shows the frame rates using our algorithm, and Figure 14(b) shows the number of rendered polygons. The camera used in these tests starts far away from the city and slowly zooms in. It then rotates around the city and flies away. The occasional abrupt drops in frame rates using our algorithm are caused by runtime loading of texture. In Figure 14(a), where the frame rates using our algorithm fall below that of the un-simplified meshes is when the overhead of the LOD process becomes more time consuming than rendering of the polygons. This typically occurs when the camera is closest to the city model. Not surprisingly, the inverse relationship between frame rate and polygon count is reflected in Figure 14(b). Notice that we can achieve drastic simplifications for view points that are far away from the city model (frames 1 to 100).

Figure 15(a) shows the number of pixel errors in images rendered using our algorithm compared against images rendered using the un-simplified meshes. There are two sets of tests – textured and un-textured. The un-textured tests compare the rendered grey-scale images between our algorithm and the un-simplified meshes and count the number of pixels that have different intensity values. The total number of pixel differences (pixel errors) are divided by the resolution of the images. For the textured test, the rendered color images are compared. However, instead of considering pixels that have different RGB values as pixel errors, we define two pixels to be the same if their RGB values are within 40 in each of the color channels (see Figure 15(c) for two colors that are considered to be the same). It can been seen that adding texture to the cluster meshes decreases the overall pixel errors. Figure 15(b) shows the pixel errors generated using our algorithm and QSLIM. In each frame, QSLIM uses the same number of polygons as our algorithm. It is shown here that both algorithms produce approximately the same amount of pixel errors (see Section 10 for further discussion).

Table 1 shows the amount of time used to pre-process our data in...
various stages. The most time consuming process is the polyline simplification process where intersection tests need to be performed for every candidate contraction.

<table>
<thead>
<tr>
<th></th>
<th>Atlanta</th>
<th>Xinxiang</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering</td>
<td>10.9</td>
<td>73.4</td>
</tr>
<tr>
<td>Merging</td>
<td>13.9</td>
<td>72.2</td>
</tr>
<tr>
<td>Simplification</td>
<td>23.1</td>
<td>243.0</td>
</tr>
<tr>
<td>Texture Generation</td>
<td>21.7</td>
<td>39.2</td>
</tr>
</tbody>
</table>

Table 1: Pre-processing time for the Atlanta dataset and the Xinxiang dataset. All measurements are in minutes.

Lastly, we vary the value of $\epsilon$ in Figure 16 to see the visual effects of simplifications. It is clear that paths, edges, districts, nodes and landmarks are well preserved with all values of $\epsilon$ in this sequence of images. Unless the images are enlarged greatly, it is difficult to see the artifacts of the simplifications. The only directly notable visual difference is the increasing “darkness” of the images as $\epsilon$ increases. This is due to the approximations we use when generating textures, which generally lead to less of the facades being visible than in the original image. The three approximations we use are the limited view angles from which we generate our textures, the use of orthographic camera, and the limited texture resolution for each cluster. The first two approximations can be alleviated by generating textures from more view angles, while the last can be alleviated by increasing the texture resolution. However, increasing either the view angles or the resolution would cause the load time and memory requirement to increase as well.

### 10 Conclusion and Discussion

In this paper, we introduce a novel way of simplifying multiple 2.5D meshes. Instead of the traditional vertex/polygon/edge decimation or merging, we aggregate meshes into “logical” clusters and simplify and texture each cluster separately. We contribute a clustering approach, a polyline simplification algorithm, and an algorithm for landmark preservation designed with the goal of maintaining urban legibility. In particular, we demonstrate that using single-link hierarchical clustering results in clusters that adhere to paths and edges. Polyline merging and simplification create cluster meshes that maintain districts and nodes. Height discrimination during the LOD process ensures that landmark buildings are preserved. Finally, hierarchical textures are applied to strengthen visual fidelity while minimizing storage and memory requirements. Our system as a whole maintains legibility and understandability of a complex urban space using simplified models that allow for an interactive environment.

It is our belief that many applications can benefit from our algorithm. Google Earth (and other 3D geographical information systems) as well as any spatial data visualization applications (including scatter plots) can all use logical, simplified clusters to represent large amounts of spatial information. Currently, our implementation of both the clustering and simplification processes have complexities that approach $O(n^3)$, making our algorithm inefficient in processing an arbitrarily large amount of data. However, the most time consuming aspect of these two processes lies in intersection detection (which is $O(n^2)$). Incorporating a fast intersection detection algorithm into our system should dramatically improve scalability. In addition, it is likely that very large urban landscapes can be successfully partitioned along legible boundaries prior to clustering (perhaps with the aid of additional geographical information), which will reduce the overall clustering and simplification complexity to much less than $O(n^3)$.

Evaluation of simplification techniques for city models remains an open problem. Figure 15(b) illustrates that pixel error is not a good way to compare algorithms for this approach. Although our technique provides similar pixel errors to a traditional decimation algorithm, the visual difference is significant (Figures 12(c) and (d)). Unfortunately, quantifying urban legibility is a difficult problem. As noted in Section 2, some researchers have tried to understand legibility via user studies [Dalton 2002], while others attempted to quantify it mathematically [Osmond 2005], but so far there has not been a generalized rule that can be applied to evaluate urban legibility. One promising direction is to develop benchmark localization and navigation tasks to evaluate user performance through user studies similar to those presented in [Dalton 2002].

There are a number of opportunities for improving our algorithm. For one thing, our LOD process only considers negative space errors, while visual errors created through texture simplification should also be considered. Similarly, the clustering process merges clusters based on (weighted) Euclidean distances, but we could also include other differences such as colors, textures, sizes, and shapes in the distance function. Furthermore, it would greatly generalize our algorithm to integrate mesh decimation techniques (such as QS-lim) in the pre-processing step so that we can accept input meshes of detailed geometric complexity. This extension along with improving our clustering and simplification for very large meshes will enable us to address a challenging problem in urban rendering, namely, flying freely over very large urban scenes at bird-eye view and then diving in at any time for a detailed close-up of any building, with everything unfolding smoothly and naturally.

Lastly, additional user interaction techniques to facilitate clustering could be extremely valuable. The user might interactively add roads prior to the clustering process to prohibit clusters from two sides of the road from being merged together. However, this kind of interaction only “prohibits” but does not “facilitate” clustering. It is very difficult for an expert user to interactively modify clustering results based on higher level of understanding of the environment – e.g., the knowledge that a particular district in the city includes an interstate highway running through the center of it (perhaps because the interstate is “raised” above ground, and therefore not in the mental maps of most inhabitants). Without such knowledge, our algorithm would always separate the two sides of such a district around the interstate, when in fact, the city might be more “legible” to an observer if that segment of the interstate is not put into consideration during the clustering process. We will explore incorporating such user input into our algorithms.

### References


Figure 16: Levels of simplification. Left to right: original image (285,039 polygons); $\varepsilon = 100 \alpha = 2$ (129,883 polygons); $\varepsilon = 1000 \alpha = 2$ (53,020 polygons).


