Legible Simplification of Textured Urban Models

Remco Chang* UNC Charlotte
Thomas Butkiewicz† UNC Charlotte
Caroline Ziemkiewicz‡ UNC Charlotte
Zachary Wartell§ UNC Charlotte
Nancy Pollard ¶ Carnegie Mellon University
William Ribarsky ‖ UNC Charlotte

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 applied to 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 an approach to simplifying city-sized collections of 2.5D buildings based on the principles of "urban legibility" as defined by architects and city planners. We demonstrate that our method, although similar to traditional simplification methods when compared quantitatively, better preserves the legibility and understandability of a complex urban space at all levels of simplification.

Keywords: Urban models, simplification, levels of detail.

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

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, 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 urban spaces useful for tasks such as navigation or developing 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 (Figure 2(b)). 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 for performing mesh simplification. Specifically, we examine the concept of urban legibility on which Lynch [10] has identified through user studies that the "image" of a city can be categorized into *paths*, *edges*, *districts*, *nodes*, and *landmarks*. We believe that if these elements of legibility are preserved during the simplification process, the image of the city can also be maintained, thereby creating urban models that are better understood by users. 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 (Figure 1). 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*. Polyline-based cluster merging and simplification creates logical *districts* and *nodes* while preserving *paths* and *edges*. Hierarchical texturing creates the appropriate amount of texture for each generated clusters, and finally, the LOD process enforces the preservation of significant *landmarks*.

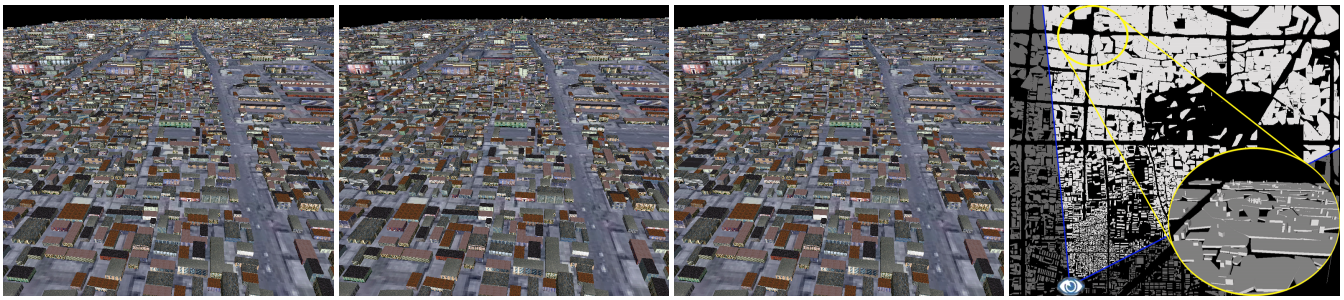
2 RELATED WORK

A tremendous amount of research has been put into mesh simplification. For a more comprehensive survey of mesh simplification techniques, see the work by Luebke [9]. Those techniques which are most relevant to our algorithm combine vertices based on their proximities and similarities to other vertices and thus are able to merge multiple meshes into one. Garland and Heckbert [4] introduce QSlim, in which "virtual" edges are added between unconnected vertices that are within a user-specified Euclidean distance τ. These virtual edges are treated in the same manner as actual edges in the mesh. We also take inspiration from Jang et al. [6], who suggest that for man-made objects, removal of entire features is typically more visually understandable than vertex removal. Our goal of urban legibility is similar in that we emphasize understandability over geometric accuracy.

In the field of cartography, the concepts of "generalization" of both building ground plans and 3D building models have been active areas of research. Kada and Luo [7] use the concept of half-space to drastically reduce the complexity of 2D building ground plans. While this approach retains the overall appearance of the original building ground plan, it sometimes creates erroneous, self-intersecting lines. Anders [1], on the other hand, shares our idea of simplifying 3D urban models by aggregating nearby buildings. His algorithm projects the building models onto three orthogonal planes (length, width, and height) and creates a simplified model based on the projections. Unfortunately, this method only works for simple symmetric models that do not self occlude during the projections. In the case of complex models whose geometry need to change in a view-dependent manner, a new algorithm is necessary.

Although there have been numerous applications in 3D global visualization and GIS visualization, the majority of the research has

*e-mail: rchang@uncc.edu
†e-mail: tjbutkie@uncc.edu
‡e-mail: caziemki@uncc.edu
§e-mail: zwartell@uncc.edu
¶e-mail: nsp@cs.cmu.edu
‖e-mail: ribarsky@uncc.edu



(a) (b) (c) (d)

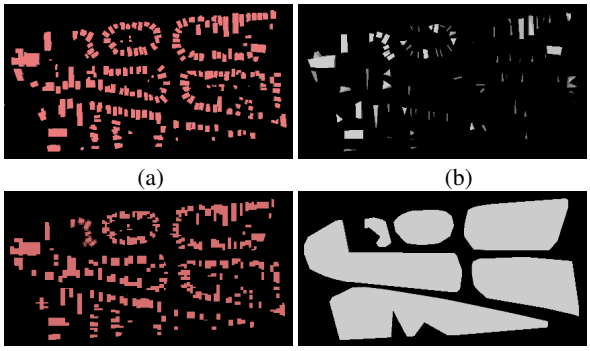
Figure 1: Levels of simplification of an urban environment. (a) original model with 285,039 polygons. (b) simplified model with 129,883 polygons, $\varepsilon = 100$ $\alpha = 2$. (c) simplified model with 85,332 polygons, $\varepsilon = 300$ $\alpha = 2$. (d) shows the same model in (c) from a top-down view. Notice that buildings far away are more aggressively aggregated than the ones near the view point. However, regardless of the amount of aggregation, all clusters obey the rules of legibility. The inset highlights the tall buildings (*landmark*) that are drawn separately to preserve the skyline.

focused on terrain simplification and visualization (see the survey by Losasso [8]). 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. To the best of our knowledge, no existing simplification algorithm prioritizes higher levels of knowledge such as urban legibility.

Urban legibility was introduced by Lynch [10], and the idea has served as inspiration for building virtual worlds, wayfinding in virtual environments, and navigation through abstract data [3]. 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's work [3]). Most relevant to our work, Shalabi [11] has used concepts from urban legibility in conjunction with impostors to visualize urban environments of limited scale.

3 Why Urban Legibility

Existing techniques for mesh simplification are often not suited for simplifying large collections of objects such as buildings because vertex, edge, or face removal often destroys geometry that is essential to the comprehension and recognizability of such objects. Figure 2 shows a comparison between the original urban model, the model simplified using a traditional simplification method, and textured and un-textured simplified models using our algorithm. The simplified models in Figures (b, c, d) contain the same number of polygons.



(a) (b) (c) (d)

Figure 2: (a) Original (textured) models of buildings; (b) Models simplified using QSlim; (c, d) Textured and un-textured building models generated with our algorithm using the same number of polygons as the QSlim model

It is obvious that these four models' appearances are very different, and arguably they convey different levels of recognizability to the viewer. However, when we use models generated from the three simplification methods in fly-through tests, we find that the amount of pixel error caused by the different simplification methods are quantitatively nearly indistinguishable (Figure 3). In other words, although the different simplification methods create qualitatively and visually different models, the quantitative measurements of the goodness of these models are almost the same.

From this simple experiment, we realize that in order to create models that are legible and understandable to the users, we cannot rely solely on quantitative measurements such as pixel differences. Instead, a higher level knowledge that encompasses our understanding of urban models must be used to guide the simplification process and generate models that are legible to the users. For such high level knowledge, we turn to the concept of urban legibility.

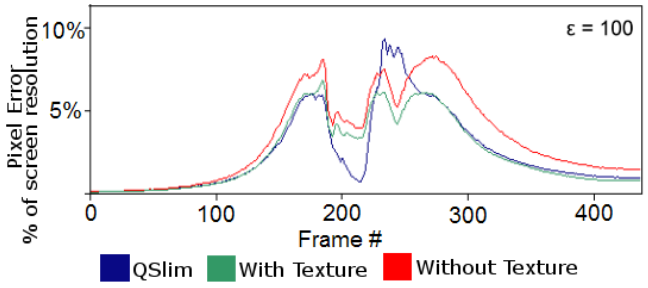


Figure 3: The amount of pixel error as percentages of the screen resolution during a fly-through using an urban model simplified by QSlim, and textured and un-textured simplified models generated using our algorithm. Pixel errors are calculated based on the number of pixel differences between images rendered using one of the simplification techniques and the original model.

4 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, Lynch [10] surveyed residents of Boston and asked them to sketch out their neighborhoods in relation to the city (Figure 4). Based on these sketches, Lynch defined these inhabitants' sense of legibility as "...the ease with which [a city's] parts may be recognized and can be organized into a coherent pattern."

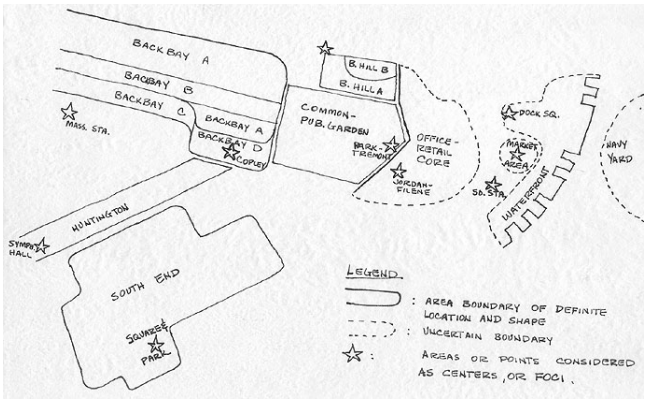


Figure 4: A hand-drawn sketch of Boston produced on November 17, 1955. Based on a series of these images, Kevin Lynch was able to identify the five elements of urban legibility that he believed were the building blocks of people's mental map of an urban environment.

"Coherent pattern" refers to cues that people use to "structure and identify the environment." 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 defined these elements as cues used by the inhabitants of a city, we believe that these elements also help people better recognize a city and maintain spatial coherence from a bird's-eye view. We base our assumption on Haken and Portugali's information theory of urban environments, which states that the amount of information per building in a cluster of similar buildings decreases as the number of buildings in the cluster increases [5]. In order to identify these clusters so as to maximize information while simplifying geometry, we can turn to Lynch's theory of meaningful urban structures. Therefore, it is with Lynch's five legibility elements in mind that we devise our algorithm for simplifying and viewing a collection of buildings. It is our belief that for as long as the simplification algorithm retains these elements of legibility when reducing geometry or texture of an urban model, the resulting urban model will remain legible to the user.

Our algorithm consists of five steps. Hierarchical clustering maintains *paths* and *edges* when grouping similar buildings together; cluster merging combines the geometries of the buildings into a single model and creates *districts* and *nodes*; simplification reduces the geometric complexity of the model while preserving *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, preserving the *landmarks* in the scene. The following sections describe the five steps in our algorithm.

5 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. Our algorithm does not require knowledge of the street layout, only building footprints, with which we maintain both *paths* and *edges* by preserving the empty spaces between buildings. To achieve this, we use single-link clustering, which creates clusters that respect boundaries of paths. In contrast, k-means and complete-link clustering both produce oval-shaped clusters that do not consider *paths* or *edges* (Figure 5).

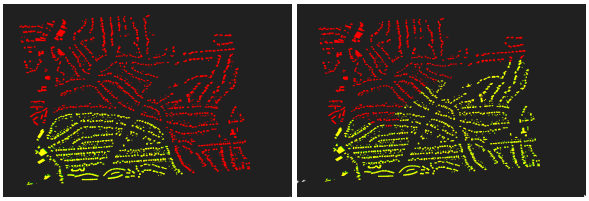


Figure 5: Left: single-link clustering. Right: complete-link clustering. There are two clusters in both images, the red buildings constitute one cluster, while the yellow ones fall into the other. Notice that single-link clustering creates clusters that follow along a path, whereas complete-link clustering, much like k-means clustering, creates oval-shaped clusters.

Single-link clustering is a greedy algorithm. At any given iteration, it finds the closest pair of elements or clusters (in Euclidean distance) and groups them together into a cluster. Therefore, for n number of elements, single-link clustering requires $n-1$ iterations, and produces a total of $n-1$ clusters. The greedy nature of the algorithm relies on distance minimization, and guarantees that a building will be clustered with other buildings on the same side of the road before being grouped with buildings from the other side of the road, as long as the distances between the buildings are not wider than the width of the road. In most urban environments in which buildings are very close to each other, this property of single-link clustering ensures that clusters obey *paths* and *edges*.

Unfortunately, single-link clustering does not guarantee the resulting cluster hierarchy (dendrogram) is balanced, meaning that the depth of the tree is often far from the ideal depth of $lg(n)$. To create a moderately balanced tree, we use a distance metric that incorporates cluster size:

$$d(C_1,C_2) = min\{\frac{size(C_1)\cdot size(C_2)}{avgClusterSize^2}\cdot d(x,y) | x\in C_1, y\in C_2\} \quad (1)$$

where $size(C_x)$ denotes the number of buildings in cluster C_x, $d(C_1,C_2)$ is the cost of merging clusters C_1 and C_2, $d(x,y)$ is the Euclidian distance between the two closest vertices in buildings x and y, and *avgClusterSize* is the average number of buildings in all the existing non-leaf nodes. With the introduction of *avgClusterSize*, we are able to reduce the depth of the tree from 726 to 170 in the Charlotte dataset, which contains 370,000 buildings.

Note that an optimal implementation of single-link clustering has the complexity of $O(n^2)$. However, our single-link clustering is a $O(n^3)$ algorithm. This is due to the fact that *avgClusterSize* changes at every step, causing all distances to have to be recomputed. For efficiency, we pre-compute the nearest neighbors of each building and only consider the nearest neighbors during clustering. This reduces the complexity of the algorithm to $O(k^3)$ where k is the number of neighbors. We found empirically that a value of 50 for k is sufficient for all of our models in producing clusters that follow *paths* and *edges* while drastically reducing the clustering time.

6 Merging and Simplifying Clusters

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 geometric model (called a merged hull), which contains and resembles the aggregate of the buildings. This merger often creates logical *districts* (Figure 6 top row). The merging of clusters is recursively performed until all (non-leaf) nodes in the hierarchy contain a merged hull.

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 permits us to apply different rules to the footprints and the heights. Our merging algorithm begins by finding the 2D convex hull of all the buildings' ground vertices, and iteratively subdivides a line segment to form the largest possible triangular area between the original line segment and the two newly created ones (Figure 6 bottom row). Based on the inverse of the one-mouth theorem [12], this process can continue until no line segment can be further subdivided without causing self-intersections.

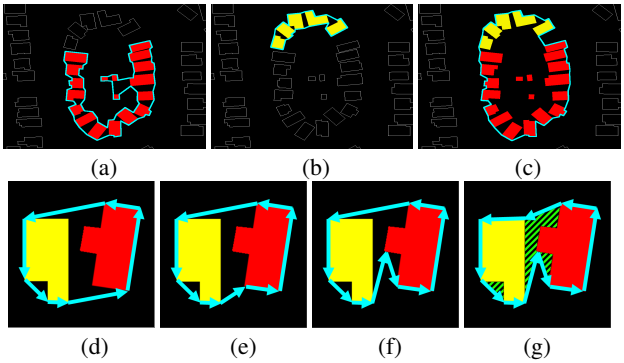


Figure 6: Top row: Creating a *district* by merging two clusters (a) and (b) into (c). Bottom row: Steps of cluster merging and simplification. (d) The convex hull of the two merging clusters; (e, f, g) three iterative steps of line segment subdivision. Notice that the merger introduces geometric error, or negative space, as highlighted by green stripes in (g).

The computation of this iterative line subdivision method has the worst case upper bound of $O(n^3)$ where n is the number of vertices. For the sake of efficiency when computing the merging of the clusters, we stop the subdivision process when the target number of edges is reached. Although the target number is a changeable parameter, it is imperative that the number of edges in a parent cluster is less than or equal to the sum of the two children's number of edges. By adhering to this rule, the cluster hierarchy remains monotonic in that traversing deeper down the hierarchy always results in more geometric detail. Empirically, we find that setting the parent cluster's number of edges to be 75% of the two children's combined edges results in a good balance between computation time and geometric detail.

Once the simplified merged hull has been calculated, we define the height of the cluster to be the weighted average height of all the buildings in the cluster, where the weight of each building is directly proportional to its area. Buildings with dramatically different heights are considered as *landmarks* and handled separately during the runtime LOD. The final polygonal model for each cluster (called a cluster mesh) is produced by protruding the merged hull towards the sky to the height of the cluster, and using OpenGL (GluTesselator) to create the triangulation of the roof.

We find the result of our merging and simplification process to produce clusters that are understandable in the form of *districts* while preserving the elements of *path*, *edges*, and *nodes* (Figure 7). Furthermore, similar to the concept of half-edge collapse, no new vertices are created during this process, making this algorithm very useful when used in conjunction with OpenGL vertex arrays for both speed and memory conservation.

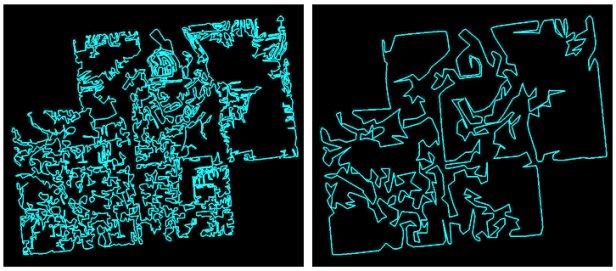


Figure 7: Polyline simplification. Left: 6000 edges; Right: 1000 edges.

7 Negative Spaces and Level of Detail

During the hull merging and simplification process, geometric errors are introduced into the final mesh. We call these geometric errors "negative spaces" because geometry is added to previously 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 (Figure 9 (a)). 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 sides of the box are projected onto screen space, and the number of pixels is compared against a user-defined tolerance (ε). If the number of pixels is greater than ε, the cluster will not be rendered, and its descendants will be checked recursively (Figure 8).

The concept of *landmarks* is perhaps the most subjective of

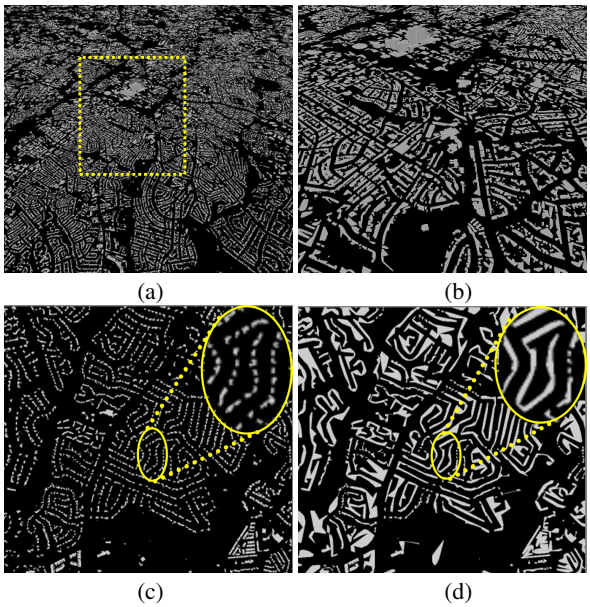


Figure 8: (a) View of downtown Charlotte and its surrounding regions from afar; (b) what is actually being rendered when the selected yellow box region in (a) is enlarged. (c) using a pixel tolerance ε of 50; (d) setting ε to 500: notice that as ε increases, so does the amount of simplification. Nonetheless, both models obey the principles of legibility.

Lynch's categories. However, it is reasonable to assume that taller buildings have higher visual importance than shorter ones because of their roles in defining the skyline. A user-defined threshold (α) in numbers of pixels is used to determine the acceptable error in height. During runtime, α is projected onto each cluster mesh and converted to a height value in world coordinate (called α_{height}), shown in Figure 9(b). If any building is taller than its cluster's α_{height}, the original building mesh is rendered along with the cluster mesh (Figure 9(c)). Figures 9(d, e, f) show the affects of changing α on a city's skyline.

8 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 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 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. The combined images from all faces are set to fit into one single texture, with the resolution of each image proportional to the length of each face.

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: a top-down view and 45-degree views from north, east, south, and west. Buildings are 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.

Due to the limited texture resources, it is important that we constrain the amount of texture generated to a minimum. Analytically,

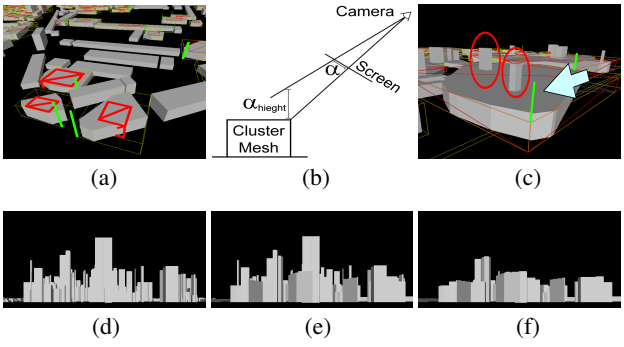


Figure 9: Top row: (a) The negative spaces are approximated as 3D boxes, shown in red. The green lines represent α_{height}, or acceptable height tolerance for each cluster; (b) Finding α_{height}: the user-defined height tolerance α is projected onto a cluster mesh and converted to height tolerance (α_{height}) in world coordinate. (c) Buildings in a cluster mesh (circled in red) that are taller than (α_{height}) (shown as a green line), are considered to be *landmarks* and are drawn separately over the cluster mesh. Bottom row: (d) Original skyline (243,381 polygons) (e) Simplified skyline that resembles the original skyline ($\alpha = 2$, $\varepsilon = 100$, 15,826 polygons) (f) Skyline that loses its resemblance to the original skyline by not maintaining *landmarks* ($\alpha = 10000$, $\varepsilon = 100$, 13,712 polygons).

we can find the upper bound of the maximum texture resolution required for any cluster mesh given the user-defined pixel tolerance ε. Since no cluster mesh should have a texture resolution that exceeds its maximum size on screen, the cluster mesh's maximum size defines the upper bound of the texture resolution. We know the footprint area of the cluster mesh, and the area of the negative space. So given ε, we can trivially compute the maximum size that a cluster mesh would ever appear on screen, and hence the maximum number of texture pixels for the roof t_{roof} and the sides of the cluster mesh t_{side}:

$$t_{roof} = (area_{clusterMesh}/area_{negativeSpace}) * \varepsilon \quad (2)$$

$$t_{side} = (\sqrt{area_{clusterMesh}}/\sqrt{area_{negativeSpace}}) * \varepsilon * 4 \quad (3)$$

In the case of t_{side}, we multiply the number of calculated pixels by 4 because there are four sides to each 3D negative space. Although the fact that t_{side} does not depend on the height of the cluster mesh is counter intuitive, it indicates that the geometry simplification and texturing are inter-dependent. During runtime, given ε, the LOD process chooses the appropriate geometry to render, and it guarantees that the associated texture for that geometry is optimal.

Due to the large size of the urban datasets, texture storage and retrieval is also a critical aspect in rendering. Conceptually speaking, we separate clusters into groups that are geographically close to each other and combine their textures into texture atlases. In practice, we consider three types of clusters - those in the cut, those above the cut, and those below the cut, where the cut is found by using the LOD process looking down at the city model. The clusters in the cut are clusters visible in the LOD and approximately equal in size. Together, they define groupings of buildings that are geographically close to each other.

Choosing an appropriate texture atlas size is also a challenge. Large atlases require fewer disk I/O operations and can make more efficient use of space, but we found the lower runtime texture memory needed for small atlases to be worth the tradeoff. In the Charlotte dataset, we use texture atlases of 256x256, and create a cut of approximately 12,220 groups.

During runtime, the geometries of the model are left in-core. However, to ensure that there is enough memory for the textures during runtime, we implement a simple priority queue in which the least recently used texture is swapped out when memory becomes a constraint.

9 Results and Analysis

All graphs shown in this section are generated using a fly-through of the Charlotte model which contains 369,929 buildings and 4,088,254 polygons (with the parameters $\varepsilon = 100$ and $\alpha = 2$). To showcase our system's speed and efficiency from all levels of detail, the camera's fly path starts zoomed-in locally on a number of buildings at the ground level, and then zooms out and up slowly until the entire city is in view from above. Note that the Charlotte model does not contain accurate height information or textures.

The computer used in generating the following statistics is a Pentium 4 3.0Ghz desktop with 2GB of RAM and an nVidia 7950 graphics card with 512MB of memory running Windows XP Professional.

Polygon Count We demonstrate that our algorithm drastically reduces the number of polygons rendered during a fly-through in our Charlotte model. Figure 10 shows that the number of polygons rendered ranges between 1% to 8% of the total polygons.

Frame Rate Figure 10 shows the frame rate of the fly-through of Charlotte using our algorithm (shown in blue) compared to the rendering of each building individually (shown in magenta). Both implementations use OpenGL Vertex Buffer Objects (VBO). However, our algorithm takes advantage of hierarchical frustum culling in removing clusters outside of the view frustum, whereas the implementation of rendering individual buildings does not cull buildings outside of view.

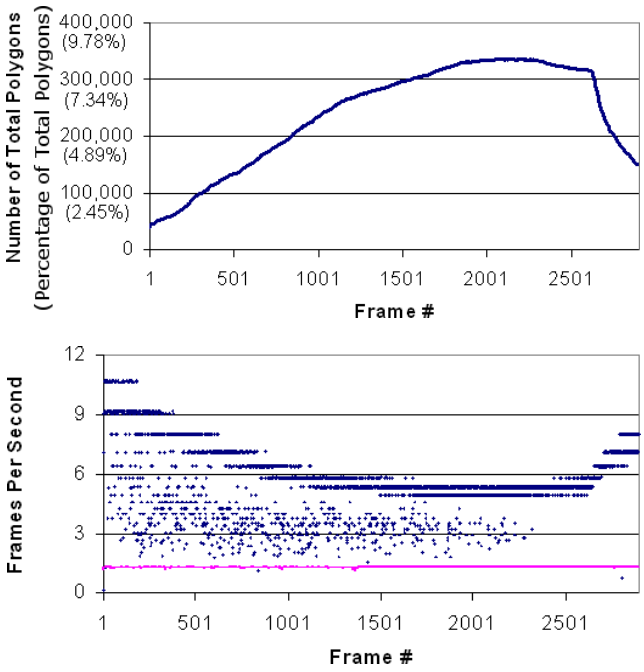


Figure 10: Top: The polygon count and the percentages of the total number of polygons of the original Charlotte model during a fly-through. Compared to the unsimplified Charlotte model (4 million polygons), our algorithm drastically reduces the polygon count to less than 9% from all view distances and angles during the fly-through. Bottom: The frame rate of the fly-through of Charlotte. The blue dots show the frame rate using our algorithm, and the magenta dots show the frame rate if every building is drawn individually in every frame. Note that the fluctuations in our frame rate correspond to the frames in which textures are loaded from file.

The fluctuations in the frame rate correspond to the frames in which the system fetches the needed textures from the file system. Our current implementation of texture-loading does not include pre-fetching of textures, so textures are loaded from file on an as-needed basis. While pre-fetching is a well-studied technique, we note that adding pre-fetching into our system is not a trivial task due to the already large memory requirement when viewing a textured, city-sized urban model. Because of the memory constraints, correctly identifying the needed textures for pre-fetching is a difficult problem that is outside the scope of this paper and requires further investigation.

Preprocessing Time Table 1 shows the amount of time required for each stage of pre-processing. Although the computation can be performed in parallel, for consistency's sake, all preprocessing is done on a single computer. Note that the large difference in clustering times between the two datasets is due to the fact that the distance metric of the clustering algorithm is based on distances between vertices. Therefore, the more detailed the buildings are, the longer it takes to calculate the minimum distance between them.

	Xinxiang	Charlotte
Number of Buildings	95,042	369,929
Number of Polygons	814,592	4,088,254
Number of Ground Vertices	502,336	2,414,056
Clustering	8 mins	102 mins
Merging and Simplification	3 mins	15 mins

Table 1: Statistics and preprocessing times for the Xinxiang dataset and the Charlotte dataset.

10 Discussion

Evaluation of the legibility of a simplified city model remains an open problem, as it is difficult to quantify a person's sense of spatial awareness in an urban environment. While spatial cognition is an active field, it has not often been related to the analysis of urban legibility, as defined by Lynch, in the context of urban visualization. As noted in Section 2, some researchers have tried to understand legibility via user studies [3], while others attempted to quantify it mathematically, 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 [3].

We have performed an informal expert evaluation on an exploratory system built on our simplification method called *UrbanVis* [2], which examines Census information within an urban environment. The system was evaluated by surveying 14 experts with disparate backgrounds including geographic information system (GIS) experts, city planners in local government, school district planners, population experts in academia, and commercial real estate developers. The experts were given some time to familiarize themselves with the system, then asked for subjective feedback on the potential usefulness of the system in their daily tasks as well as any suggestions on future improvements. The results of the survey indicate that for the purposes of navigation and spatial awareness and understanding, urban models created based on our simplification algorithm remain legible at all levels of simplification, even in the most extreme cases.

This finding is significant in two ways. First of all, it suggests that the legibility elements introduced by Kevin Lynch do indeed form a basis for most people's mental understanding of an environment. This is interesting because an individual's concept of a "legible" environment is inherently subjective; how a person orients himself often depends on his familiarity with the surroundings. A long-time resident of a city could use a local restaurant as a landmark, whereas tourists might rely on skyscrapers and major roads for orientation. However, based on our survey results, Lynch's legibility elements may serve as a common ground between each person's mental image of the urban environment. All participants of our survey, with their disparate backgrounds, understood and oriented themselves successfully using the simplified urban model generated with our algorithm.

More importantly, the results of the survey strengthen the argument presented in Section 3 that pixel-level accuracy in simplification is often not the most important measurement in evaluating urban simplification algorithms. As Figure 3 shows, qualitatively different urban models can produce similar quantitative pixel errors. Our evaluation reinforces this point by demonstrating that even with an aggressive amount of simplification, the user retains spatial awareness of the environment. This indicates that if the simplified model resembles the user's mental image of the urban environment, the user can successfully orient himself and navigate the environment effectively.

11 Conclusion

In this paper, we introduce a method for simplifying large urban models while retaining their legibility. We show that simplification of urban models should not be done solely using traditional vertex, edge, or polygon removal strategies. Employing such strategies may produce quantitatively acceptable models, but not models that are understandable or legible.

Instead, we contribute the concept of simplification through merger of 3D urban models in which geometries and textures of buildings are merged together to create new models. We argue that the resulting simplified models, although not quantitatively better than models created using traditional simplification methods, are more understandable and legible to the user.

This is particularly true when the merger is guided by the concept of urban legibility as described by architects and urban planners. Each step of our simplification process seeks to retain or create elements of legibility. Single-link clustering groups buildings into clusters that adhere to the boundaries of *paths* and *edges*; cluster merging creates logical *districts*; polyline simplification maintains *paths*, *edges*, *nodes*, and *districts* while reducing the model's complexity; and the runtime LOD process renders only visually appropriate models while preserving the skyline by identifying *landmarks*. Finally, analytically determining the necessary amount of texture and applying it hierarchically to the models strengthens the visual fidelity of the urban scene.

Finally, our quantitative and qualitative evaluations indicate that our method not only reduces the geometry of complex urban models, but successfully preserves the legibility of the environment on all scales. The resulting simplified models are faster to render and still effective in a user's navigation and orientation of the urban environment.

12 Future Work

There are a number of opportunities for improving our algorithm. Currently, the clustering process merges clusters based on Euclidean distances between vertices of buildings, but we could also include other attributes such as buildings' colors, textures, sizes, and shapes in the distance function. Furthermore, it would greatly generalize our algorithm to integrate mesh decimation techniques in the pre-processing step so that we can accept true 3D building models of arbitrary geometric complexity. This extension along with improving the texture loading process to incorporate concepts such as pre-fetching will enable us to address a challenging problem in urban rendering, namely, flying freely over very large urban scenes at a birds-eye view and then diving in at any time for detailed close-ups of any building, with everything unfolding smoothly and naturally.

In the rendering aspect, we acknowledge that our system does not utilize OpenGL's Vertex Buffer Object (VBO) to its full potential, which limits the frame rates shown in the Results and Analysis section (Section 9). Two limitations prohibit the rendering engine from reaching ideal frame rates; first, OpenGL's current implementation of VBO limits each vertex to one surface normal. Unfortunately, in the case of 2.5D models, each vertex needs to represent two surfaces with different surface normals that cannot be blended. In order to render each face of a building with a different surface normal, we render each face independently after setting its surface normal (using GL_QUADS instead of GL_QUAD_STRIP). The second limitation of the rendering engine is due to updating the VBO on every frame. In our implementation, ground vertices and roof vertices share the same (x, y) coordinates and are stored in the same VBO. During runtime, however, the z value (height value) of the roof vertices need to be updated every frame based on their corresponding cluster meshes. Although no new vertices are introduced into the VBO, the z coordinate updates still slow down the rendering speed. Overall, our evaluation suggests that the bottleneck in our rendering system is not in the graphics card, but is instead in the data transfer between the CPU and the GPU. We believe that if these two issues can be resolved, the rendering speed should increase significantly.

Lastly, we are very interested in furthering the evaluation of how individuals understand their surroundings in an urban environment. We would like to examine through user studies how the first three images in Figure 1 differ perceptually and cognitively. If we can understand better the way mental images of urban surroundings are formed, we believe we can apply those concepts and create more simplified, yet more legible urban models.

Acknowledgements

The authors wish to thank Eric Sauda and Jose Gamez of the UNC Charlotte College of Architecture for sharing their expertise on urban legibility; to Sonia Leach for her introduction on single-link clustering; to Hunter Hale for preparing the Charlotte dataset; and to the group at Google Earth for their encouragement and feedback.

References

[1] K.-H. Anders. Level of detail generation of 3d building groups by aggregation and typification. In *International Cartographic Conference*, 2005.

[2] R. Chang, G. Wessel, R. Kosara, E. Sauda, and W. Ribarsky. Legible cities: Focus-dependent multi-resolution visualization of urban relationships. In *Transactions on Visualization and Computer Graphics*, volume 13, pages 1169–1175. IEEE Computer Society, 2007.

[3] R. C. Dalton. Is spatial intelligibility critical to the design of large-scale virtual environments? In *Journal of Design Computing 4. Special Issue on Designing Virtual Worlds*, 2002.

[4] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 1997*, pages 209–216. ACM, 1997.

[5] H. Haken and J. Portugali. Legiblethe face of the city is its information. In *Journal of Environmental Psychology*, volume 23, pages 385–408. Elsevier, 2003.

[6] J. Jang, P. Wonka, W. Ribarsky, and C. Shaw. Punctuated simplification of man-made objects. In *The Visual Computer*, volume 22, pages 136–145, 2006.

[7] M. Kada and F. Luo. Generalisation of building ground plans using half-spaces. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume 36, 2006.

[8] F. Losasso and H. Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. In *Proceedings of SIGGRAPH 2004*, pages 769–776. ACM, 2004.

[9] D. Luebke. A developer's survey of polygonal simplification algorithms. In *IEEE Computer Graphics and Applications (May/June)*, pages 24–35, May 2001.

[10] K. Lynch. *The Image of the City*. The MIT Press, 1960.

[11] S. M. Shalabi. Analysis of urban morphology for real time visualization of urban scenes. Master's thesis, Massachusetts Institute of Technology, May 1998.

[12] G. Toussaint. Anthropomorphic polygons. In *American Mathematical Monthly*, pages 31–35, 1991.