Using Additive Sound Synthesis to Analyze Simplicial Complexes

Ulrike Axen
Department of Computer Science
University of Illinois
Urbana, Illinois 61801
Email: axen@cs.uiuc.edu

Insook Choi
College of Fine and Applied Arts
University of Illinois
Urbana, Illinois 61801
Email: ichoi@ncsa.uiuc.edu

Abstract

We present a new technique for traversing simplicial complexes and producing sounds from the output of this traversal. The traversal algorithm was invented in order to extract temporal information from static geometric structures; this information is used as input to a sound synthesis algorithm. A systematic traversal of complexes and associating data to parameters of sound synthesis has many possible applications: the analysis of objects of dimension 4 or higher, exploration of large data sets and music composition. In this paper we limit ourselves to 3-dimensional objects and present the experimental results from the first phase of our work.

1 Introduction

When visualization techniques are applied to viewing 4-, or higher-, dimensional objects, we immediately encounter difficulties in conveying the higher-dimensional visual information to our perceptual understanding. These difficulties have little to do with the shortcomings of any particular tool. The nontrivial issue is to understand how we understand. Not only is it unnatural to bring higher-dimensional information into our visual perception, it is also a nontrivial issue to study methods for displaying 4-, or higher-, dimensional objects on a 2-dimensional computer screen. We have observed some innovative solutions made available by improved display techniques using virtual reality systems in Siggraph'94 [1].

Along with the improvements in visualization techniques, we think the conveyance of multimodal information will have a long term effects on studying higher-dimensional information [2]. The goal of our project is to create sound synthesis algorithms which will help us to understand higher-dimensional information, in particular, simplicial complexes of 4 or more dimensions.

2 Definitions

We start with some definitions from combinatorial topology; most of the definitions that follow can be found in the text by Munkres [3]. A *k-simplex* σ is the convex hull of $k+1$ affinely independent points (or vertices); dim $(\sigma) = k$ is the *dimension* of σ. So a 0-simplex is a single vertex, a 1-simplex is a line segment, a 2-simplex is a triangle, and so forth.

If V is the set of vertices of σ, then a *face* of σ is the convex hull of any subset of V. A collection K of simplices is called a *simplicial complex* if the following two conditions hold:

1. If $\sigma \in K$ and τ is a face of σ, then $\tau \in K$.

2. If $\sigma_1, \sigma_2 \in K$, then $\sigma_1 \cap \sigma_2$ is a face of both (which includes $\sigma_1 \cap \sigma_2 = \emptyset$).

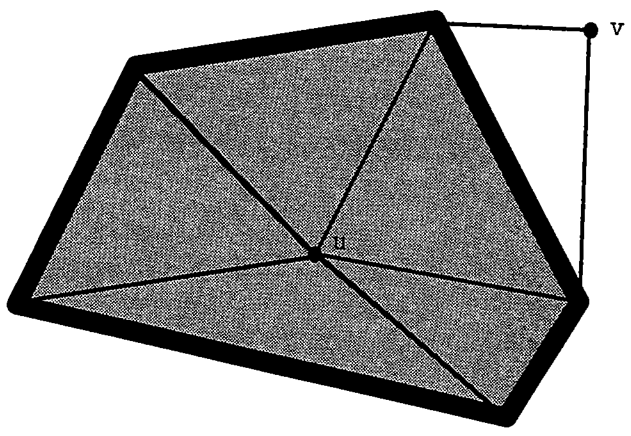

Figure 1: The closed star of u.

If the largest dimension of any $\sigma \in K$ is d, then K is a *(simplicial) d-complex*. For example, the triangulation of the surface of a torus is a 2-complex, consisting of triangles, edges and vertices. The input to many computational topology algorithms consists of simplicial complexes, since they discretize the objects under study. A *subcomplex* L of a complex K is a subset of K which is also a complex.

We define the *closure* of σ, $\mathrm{cl}(\sigma)$, as the set consisting of σ and all faces of σ. The *closed star* of a vertex v in a complex K, $\mathrm{clstar}(v)$, is the union of the closures of all simplices that contain v; i.e.,

$$\mathrm{clstar}(v) = \bigcup_{\sigma \in K, v \in \sigma} \mathrm{cl}(\sigma).$$

See Figure 1.

Similarly, the (open) *star* of v, $\mathrm{star}(v)$, is

$$\mathrm{star}(v) = \{\sigma \in K | v \in \sigma\}.$$

Now, the *link* of v can be defined as $\mathrm{link}(v) = \mathrm{clstar}(v) - \mathrm{star}(v)$, where '$-$' denotes set difference (see Figure 2). These terms can also be defined for a subcomplex L of K. The *closed star* of L, $\mathrm{clstar}(L)$ is the set

$$\mathrm{clstar}(L) = \bigcup_{v \in L} \mathrm{clstar}(v),$$

and the (open) *star* of L is

$$\mathrm{star}(L) = \bigcup_{v \in L} \mathrm{star}(v).$$

So *link* of L is defined as $\mathrm{link}(L) = \mathrm{clstar}(L) - \mathrm{star}(L)$, just as for vertices.
In a complex, we can *contract* a one- or higher-dimensional simplex σ to one of its vertices v by identifying all vertices of σ with v and removing σ and all other simplices that become redundant from this identification.

3 Experimental Analysis of Simplicial Complexes

Computer visualization has become an important tool for experimental mathematicians. However, for 4- and higher-dimensional objects, visualization is awkward and often unenlightening; projection into lower-dimensions invariably obscures important features. Given this, we turn to

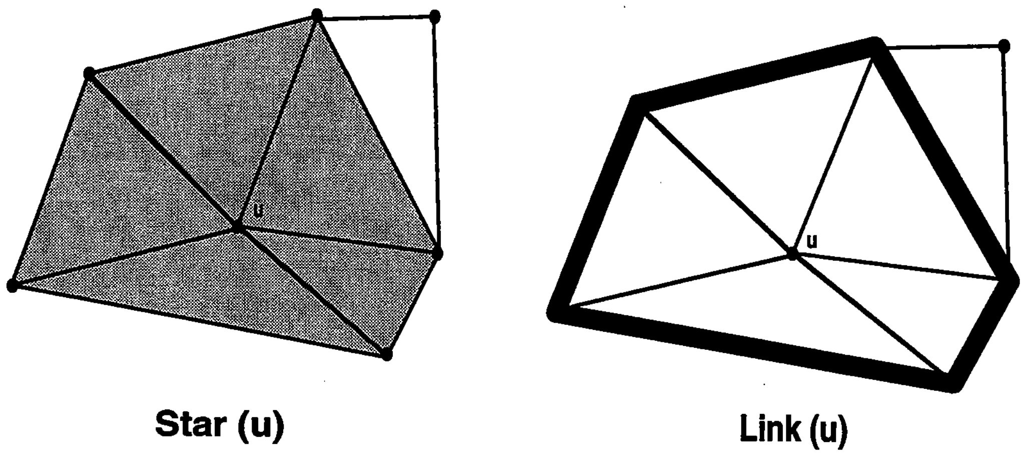

Figure 2: The open star of u and the link of u.

sound as a tool for the analysis of topological and geometrical properties of higher-dimensional spaces.

The obvious problem with using sound in this case is that sound is a temporal phenomenon, while the objects that we study are static. So our first task is to find some way to make the data dynamic, and we do so here with an algorithm that produces a sequence of disjoint subcomplexes from the complex. For lack of a better term, we call these subcomplexes *waves*. In the next section we define this traversal rigorously; here we will just sketch the output and its use for analysis. We start with a given simplicial complex K and a start vertex $u \in K$. We define u to be wave(0). The link of u is defined to be wave(1). To compute wave(2) and on we repeatedly contract the "visited" subcomplex (the closed star of u) to u, and compute the link again. We continue in this manner until the entire complex K has been covered, i.e., every vertex belongs to a wave. See Figure 3 for an example of this process on a 2-complex. If we are traversing a d-complex, we generate subcomplexes of dimension $d-1$ (or less), and so we call these $(d-1)$-waves.

How do waves help us find some topological characteristics of simplicial complexes? Let's take as an example the triangulation of the surface of the torus; this triangulation is a 2-complex. 1-waves on the torus grow larger and larger until they split into two components, which circle the "hole" of the torus until they meet again. The fact that the waves split into two components is evidence of a hole, and can be used in higher dimensions also to find such features. 1-waves generated on a 2-sphere (the surface of a ball) will not split (in most cases; they could split if the triangulation is very irregular), and 1-waves generated on an object with more holes will split into more components.
Now consider a 3-complex like the thickened torus, i.e., a torus in which the surface has been thickened so that it is 3-dimensional. We can compute waves recursively on this complex, that is, compute the 2-waves and then treat each 2-wave as a separate complex and compute the set of 1-waves on it. If the 1-waves split, then we have evidence that the inside of the torus is hollow rather than solid. See Figure 4.

More obviously, wave computation determines the number of connected components of a complex. If we've finished computing a set of waves and not all the vertices have been visited, then the complex consists of more than one component, and we compute waves on the next component starting with one of its vertices.

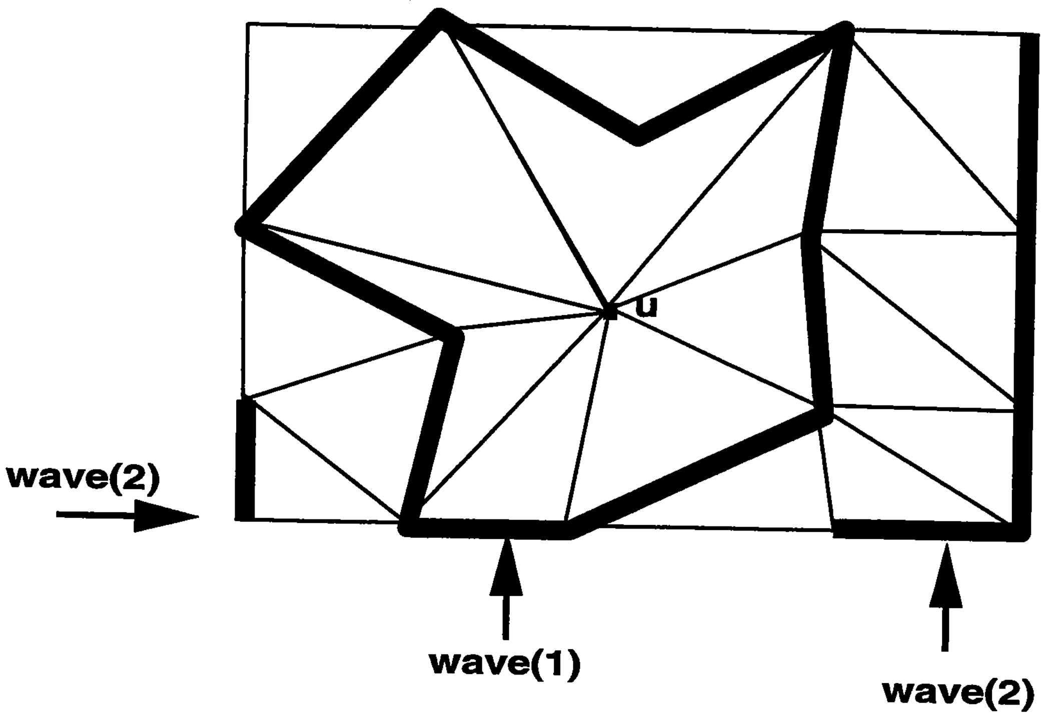

Figure 3: Waves on a 2-complex.

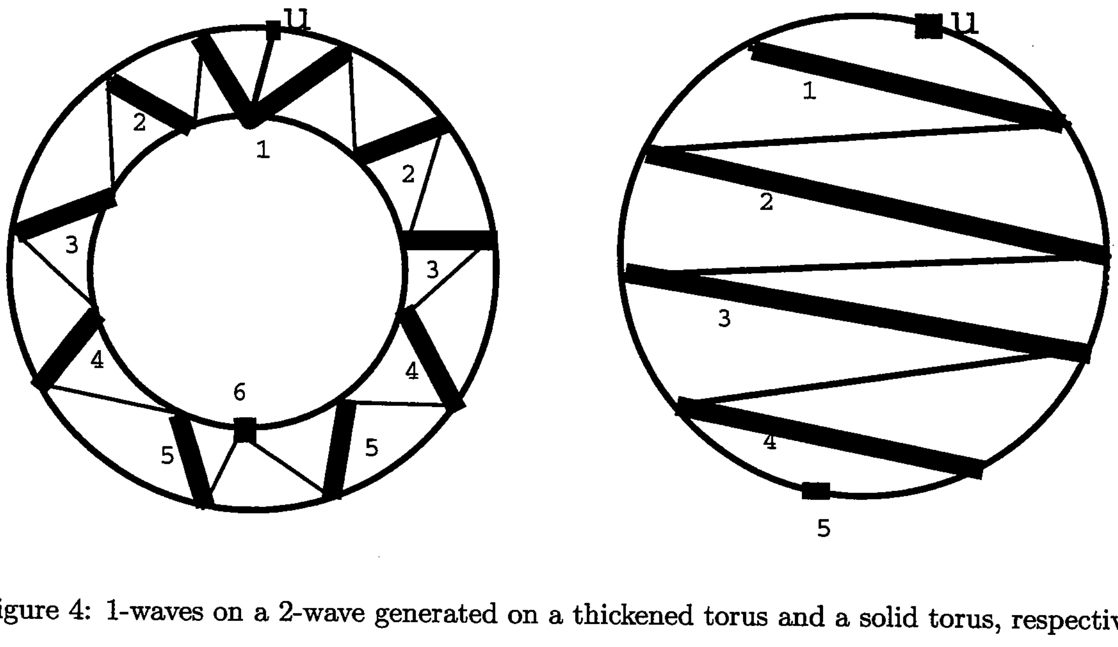

Figure 4: 1-waves on a 2-wave generated on a thickened torus and a solid torus, respectively.

```
Wave ( $K, u$ )
begin
    $L' = \text{star}(u)$;
    $L'' = \text{cl}(L')$;
    $i = 1$;
    $L_{K,u}(i) = L'' - L'$;
    while ($L_{K,u}(i) \neq \emptyset$)
    begin
        $L' = \text{star}(L'')$;
        $L'' = \text{cl}(L')$;
        $i = i + 1$;
        $L_{K,u}(i) = L'' - L'$;
    end while
end
```

Figure 5: Algorithm to define waves.

4 A Formal Definition of Waves

We can define waves algorithmically in two different ways. One algorithm makes use of the link of a vertex (as described above), and the other algorithm uses a breadth first search. We present the former here; the other algorithm and proof of their equivalence can be found in another manuscript.

Let K be a d-complex and u a vertex in K. We denote the set of waves in K starting at u by $L_{K,u}$, and wave i in this set by $L_{K,u}(i)$. Then $L_{K,u}$ can be defined using the algorithm in Figure 5.

This algorithm is a formal statement of the ideas already presented; it computes the set of waves on a single component of the complex K.

5 Implementation of the Traversal Algorithm

Our implementation does not follow the wave definition precisely; instead we contract the closed star of u to u in each iteration. Then the next wave is simply the link of u again. Figure 6 details the traversal algorithm as implemented. It takes as input a vertex u and a complex K, and outputs a sequence (array) of subsets which are the "waves" from u.

This implementation is more efficient both in terms of time and storage than would be a direct implementation of the definition. Both subroutines are implemented in such a way that the time and storage of the traversal procedure is linear in the number of simplices in K. However, we do need to show that this algorithm produces a sequence of waves as described in the preceding section.

Proof of Correctness: To prove that this implementation produces the same set of waves as our definition, we must show that wave $i+1$ is left unchanged when we contract wave i and the previous waves. Let us assign to each vertex $w \in K$ an index number i, where i is the number of edges on the shortest path from the start vertex u to w. Then all the vertices in the first wave have index number 1, all the vertices in the second wave have index number 2, and so on. One call

```
procedure wave ( VERTEX $u$, COMPLEX $K$, SUBSET waves[])
begin
    SUBSET $L =$ link ( $u, K$ ); $i = 1$;
    while ( $L \neq \emptyset$ ) do begin
            waves[i] $= L$;
            contract ( $u$, $L$, $K$ );
            $L =$ link ( $u, K$ );
            $i = i + 1$;
    end
end
```

Figure 6: The Wave procedure.

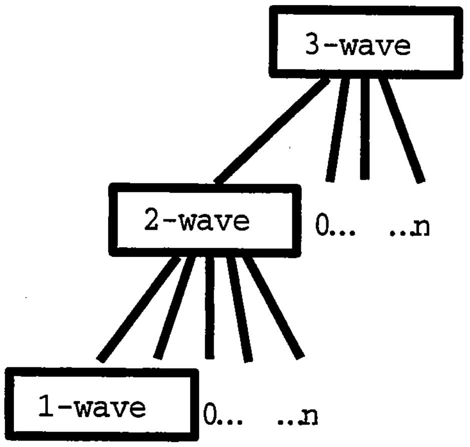

Figure 7: The "tree" of waves.

to the `contract()` routine removes exactly all index 1 vertices; they are the vertices belonging to link(u) and they are identified with u. As a consequence all indices decrease by exactly one. So the vertices belonging to wave $i+1$ are in the link of u after exactly i contraction steps. Since all the simplices in wave $i+1$ have only vertices with index number $i+1$ (this is easy to show by induction), they are unaffected by previous contractions.
For an input d-complex (which we call the d-wave), we compute a whole tree of waves, of dimension $d-1$ down to 1. We do this recursively, treating each $(d-i)$-wave as a complex and computing the set of $(d-i-1)$-waves on that, and so on, until we've created the entire tree (see Figure 7).

The tree is stored as two lists: a vertex list and a simplex list. For each vertex we store its wave number and component number in each dimension (these are unique). For each simplex, we simply record the lowest dimension in which it belongs to a wave: all simplices in the complex belong to the d-wave, and thereafter they may or may not belong to lower-dimensional waves. We then map data stored in this tree of waves to parameters of additive sound synthesis.

6 Additive Synthesis

Classical sound synthesis has shown many successful examples in analysis/resynthesis of musical signals using the additive synthesis technique [4, 5, 6]. Often a musical signal is taken for analysis to achieve a data set for resynthesis. In analysis a complex musical signal is decomposed into a number of sinusoids; the Fourier transform method allows the conversion of a time domain waveform into its frequency spectrum. Discrete amplitude measurements for each sinusoid are extracted along with their frequency locations. Thus the analysis provides a guide to resynthesis as the sum of sinusoids to simulate the original signal. The time varying amplitude of each sinusoid in resynthesis is achieved by interpolating between data from the analysis. To economize the cost of computation, the computer musician often uses perceptual guidelines to select only the most prominent partials for resynthesis [7]. Control inputs for each sine component are amplitude, frequency, and phase.

A more sophisticated technique has been developed by computing a set of spectra with the Short Time Fourier Transform (STFT) method. A complex sound is modeled as a sum of sinusoids of varying frequencies using a spectral modeling technique with a spectral peak detection algorithm [8]. In this approach, complex musical signals are resynthesized as steady-state complex waveforms; amplitudes of individual sine components are not interpolated by time-varying control, rather a method for continuation is achieved by an overlap-add technique from one steady-state spectrum to the next.

The additive synthesis technique describes sounds as complex periodic waveforms that consists of a series of sinusoidal components:

$$s(t) = \sum_{i=1}^{p} A_i(t) \sin[\Theta_i(t)]$$

where p is the number of sinusoids, $A_i(t)$ the amplitude, and $\Theta_i(t)$ the phase at given instances. The phase at given instances is defined by

$$\Theta_i(t) = \Theta_i(0) + \phi_i + \int_0^t \omega_i(t)\, dt$$

where $\omega_i(t)$ is the radian frequency, $\Theta_i(0)$ the initial phase, and ϕ_i the constant phase offset. The commonly acknowledged drawback of additive synthesis is the cost of computation. In order to acquire an interesting sound a substantial number of sinusoids are needed. For efficiency in computation we use a wave table look-up method to produce sinusoids. The wave table stores a sine function as a discrete set of 2^k values, where $k = 9$. Phase is translated to table position and frequency is specified by the increment used to select successive table positions. With the computing power in our platform we achieve up to 31 (or more) sinusoids without overriding the system's capacity for real-time synthesis. This number suits so far our need to accommodate the variety from most simplicial complexes we have examined. Another question presented when adopting additive synthesis technique has to do with the inefficiency of generating a large amount of data which varies with time. How and from where we achieve structural relevance for the time varying amplitude and frequency of each sine component adds to the complexity of the problem.

In analysis/resynthesis methods the relevance of time varying data adheres to existing musical signals in order to support the perceptual judgment. Our application of the additive synthesis technique is not based upon an analysis/resynthesis approach since our purpose is far from simulating musical sounds. We place the synthesis algorithm after our computational model to monitor the consequences reflected in sounds as the process unfolds in the model in time. By generating

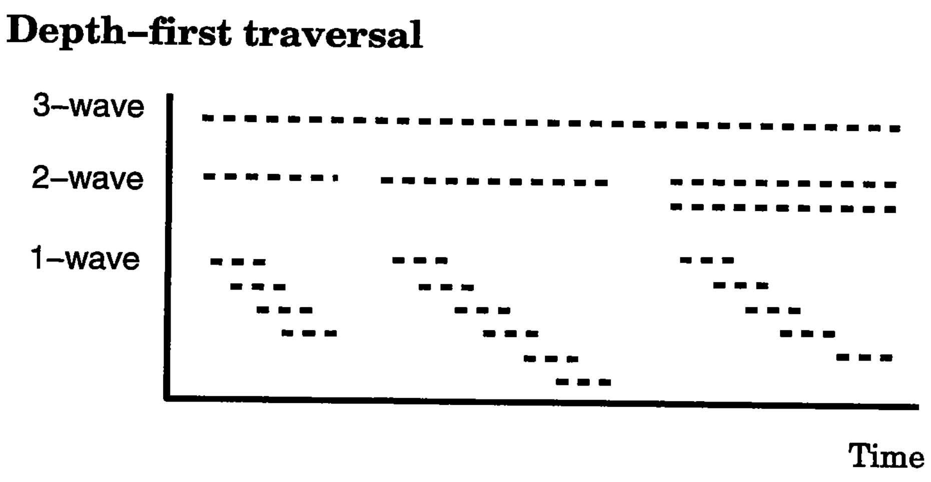

Figure 8: Sounds are generated in time as shown here.

sine components from structural features of the simplicial complex we create complex waveforms that may differ in spectral character from traditional musical signals. By traversing the complex we generate time varying spectra which are unique to the simplicial complex. The sequential traversal returns values for sine components at each time step.

To create a framework for the spectra we limit the upper and lower frequency and amplitude for sine components. The range of control values obtained from the simplicial complexes are scaled within these bounds. These complexes of varying size and structure can be compared within a uniform auditory region. We have also experimented with restricting the range of ratios between frequency values. By scaling the frequency ratios among sine components we can scale the spectral character, or harmonicity of the sound.

7 Parameter Association to Data Set

We create a playlist and a control sequence, i.e., timing information for the items in the playlist, from the output of the traversal algorithm. Each component of a wave is mapped either to a partial or to a group of partials. In our work up to date we have used exclusively 3-dimensional complexes, so we will describe the mapping to sound we used for this specific case. This program was displayed in the VROOM project at SIGGRAPH '94.

7.1 Frequency

The control of additive sound synthesis is driven by the traversal algorithm, meaning the behavior, and the birth and death of partials are also controlled by the algorithm. Sounds are generated as the time step advances from left to right in Figure 8.

The number of partials at a given instance is determined by the number of waves present at given time steps from an entire path on the tree, from root to leaf. Thus simultaneous presentation of 3-dimensional information is available at every time step. We have also implemented optional switches for monitoring information from a selected combination of the dimensions; e.g., one can locally monitor only 1-waves when one so chooses. The entire 3-complex is assigned to a single partial with a frequency of 100 Hz (cycles per second). This partial remains through all time steps until the traversal is completed. 100 Hz and 1600 Hz were chosen as the minimum and maximum boundary to assign frequency to any partial or base frequency for a group of partials. Then we apply the relative combinatorial size of each wave component to the frequency of the associated

partial with an exponential formula. The relative combinatorial size of each wave component will be determined by the number of vertices in each wave component compared to the total number of vertices in the complex. The exponential formula is to counteract our perceptual responses to the frequency information; our perceptual response to frequency is called *pitch* and the exponential increase of frequency induces the linear increase in pitch perception. We also found that we had to relate the sizes of the waves logarithmically (see formula below) to the frequency assignments, rather than linearly, because in general 2-waves are much smaller than the 3-wave, and 1-waves even smaller. The linear association would result in frequency distribution concentrated in the higher end of the resulting frequency range. We studied work in progress sounds and came up with the following implementation of frequency distribution of sound output in respect to the data. We associate the number of points to frequency so that the size can be perceived by the frequency and the components of unlike size are not fused. We have arrived at the following formula for our data.
Let f be the frequency, v the number of vertices in the wave, and n the total number of vertices in the complex. Then,

$$f = 100 + r * 1500, \text{where}$$
$$r = \frac{2^{4*(1-\frac{\log v}{\log n})} - 1}{15}.$$

The 2-waves are actually mapped to a group of partials, so the frequency determined is the base frequency of the group of partials.

7.2 Waveform

The geometric shape of the 2-waves vary as they advance through the complex. In an attempt to reflect the geometric quality in the acoustic quality at the waveform composition level we have designed an algorithm to calculate a parameter we associate with smoothness. The parameter's values range from 0 to 1, interpolating between a pure sine wave to a simulation of a sawtooth wave. The difference in the perceptual effects between a sine wave and sawtooth wave can be described in terms of buzziness or brightness. The closer the parameter is to 1, the more sawtooth characteristics are added. Thus when more angles or irregularities are imbedded in the geometric shape we will perceive buzzier or brighter sounds.

7.3 Duration

We play an entire path of the wave tree at once — 3-, 2-, and 1-waves, so the complexity of the sound at each time step informs us of the complexity of the object at each instance of the traversal. The algorithm for computing duration for each time step is designed in a bottom to top approach. The partial associated with the 3-wave is always present during the entire traversal. In order to avoid the drone effect we attenuate the amplitude of this particular partial after initial activation. The option to turn it off is available. The partial from a 2-wave plays as long as all components of its 1-waves play. This way the duration of the timestep of a 1-wave depends on the completion of its components, the duration of the timestep of 2-waves depends on the completion of 1-waves, and 3-wave remains as along as 2-waves complete the traversal. Thus once the shortest constant for timestep for the lowest component of 1-wave is assigned as an *explicit time* [9], all other durations are determined from lower dimension to higher dimension. The 2-waves with simpler structure have shorter duration than the 2-waves with more complicated structure since it takes longer to render the more detailed geometric shapes. While all components of the 2-wave simultaneously appear in the sequence, we offset the onset play time with small duration for the number of components of the 1-wave. When we hear the number of successive onsets of events we know the components are split and we know how many components are born. This brings an acoustic cue for split components which often escapes from visual observation when the split is hidden from the angle of the view.

7.4 Amplitude

The amplitude of the partials is not associated with any data considerations at this stage of our project. Amplitude is assigned as constant and balanced based upon acoustic/listening considerations. Since amplitude increases as the number of partials increases according to the complexity of the complex, we want to constrain the growth of amplitude within the limit of perceptual threshold as well as the system limit. When partials gather up, the relative amplitude assigned to each partial is attenuated for balanced distribution of acoustic energy. In order to avoid unintended override of one-dimensional information over another, the maximum amplitude for each dimension is attributed in advance. The assigned maximum value is then divided by the number of components within the dimension.

7.5 Alternative Sound Synthesis Technique

We also use granular synthesis in the CAVE program, although somewhat artificially, both in terms of data mapping and in terms of synthesis. The CAVE implementation includes an explosion of the complex as a graphic option. This allows the observer a better view of the inside of the object. There is a parameter called the explosion factor (which has nothing to do with the data itself, but only with the graphics display) which we map to the parameters of the granular synthesis algorithm. The greater the explosion factor, i.e., the further apart the simplices of the complex go, the greater the time interval between audio grain onsets and the shorter the audio grain durations. The sounds being granulated are the sounds that are currently generated by the data set.

8 Sound Synthesis Environment

We implemented these ideas on Silicon Graphics computers, using GL to program the graphics (and additional libraries for display in the CAVE [10]) and a new sound server developed by the Audio Development Group at NCSA. The architecture and the applications of the Sound Server is presented in [11, 12, 13]. Since many sound projects overlap with graphic applications in our group we support the working model in Figure 9 (from [11]) integrating graphics and sounds as parallel processes. The sound server (vss) runs as a separate application and waits for messages from the client to produce notes with correct note IDs. vss is an object-oriented system that allows the client to set up and communicate with objects that control the lower level synthesis algorithms, and with higher level objects. For example, the client can set up an additive synthesis object, and then send messages telling it to begin a note, change frequency or amplitude, set up the number of partials, and so on. Figure 10 summarizes the sound synthesis algorithm in relation to the process of traversing simplicial complexes.

In our program we use two synthesis objects: the spectral object, and granular object. The spectral object sets up a group of partials for each requested tone; this is useful when one wants to embed preset complexity in the micro-structure of the tones, for example we use this to simulate a sawtooth wave consisting of 6 sine waves. The granular object granulates the sound output,

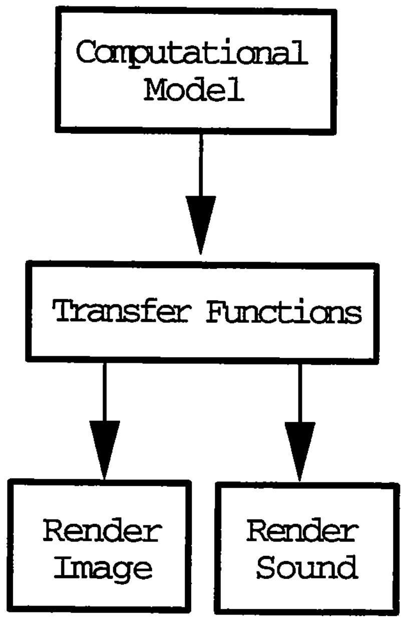

Figure 9: Parallel rendering engines driven from a single computational model.

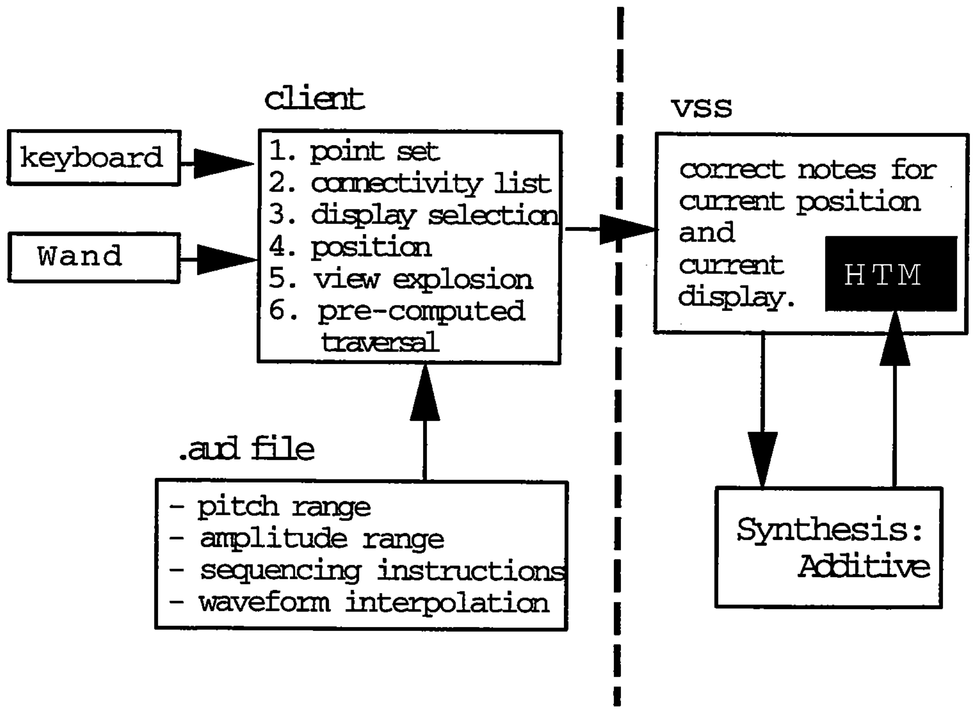

Figure 10: A client-server implementation of the synthesis pipeline.

superimposing overall amplitude shapes to the specified duration of grains. We also use a number of higher level objects, in particular a sequence and a state machine object [13]. A sequencer object handles a list of time-stamped events, and a state machine can be in any N states, numbered 0 to $N-1$; it starts playing a sequencer with transition from the old state to the new state for each state transition. These objects give us the capability to free the temporal control of the sound from the graphics. For example, since we do not display the 1-waves in the CAVE, there is no reason that the graphics program should control the timing for these notes. So we set up a state machine object with one state for each 2-wave. When we display a 2-wave, we send the server a message containing information about the current state, and it plays a sequence (the sequence of notes produced by the 1-waves on that 2-wave) independent of any control messages from the client.

9 Future Projects and Conclusion

While sound is informative at all time steps, the continuous presence of sounds can be tiring for listeners. In addition to implementing attenuation of amplitudes and switches to turn on and off the sound options for user's interaction, we would like to investigate the use of silences applied in the synthesis algorithm itself. The effective use of silences will bring the rests and pauses to articulate the overall *temporal grammar* of the sounds generated from the topological structure. For musical application we want to implement *polyphony* so that users can activate a number of initial vertices for simultaneous multiple traversals. We speculate this might also be useful for scientific purposes since this capability will add the ability to compare the length of traversal paths by listening. The audio rendering of the complex can be also completely independent from graphical rendering processes.

The implication is that we can achieve much more flexible temporal adjustment if we do not limit audio rendering speed to the speed of graphic rendering. Then sounds can be rendered and presented at any specified tempo, possibly with interactive control of the tempo. This suggests implementing a new controlling time scaling factor in our algorithm to vary the tempo of the traversal. The detailed timbral composition should be worked out based upon the desired tempo. At lower speeds *one-to-many* parameter association to data will be appropriate, and at higher speeds *one-to-one* or even *many-to-one* associations will be appropriate [9].
The number of simultaneous partials represent dimension and complexity of the simplicial complex. The changes in sound from one time step to the next inform us of the detailed structure at given instances. We anticipate the eventual extension of this algorithm to complexes of greater than 3 dimensions. Our design goals include an algorithm that is robust enough to reproduce meaningful results for d-dimensions, without encountering the need to alter the synthesis algorithm to accommodate higher dimensions.

Acknowledgments

We thank Herbert Edelsbrunner of the CS department, who directed the research contained in the first part of this paper, Ping Fu of the Software Development Group, NCSA, who worked on the CAVE implementation with us, and Robin Bargar, of the Audio Development Group, NCSA, who provided extensive suggestions and guidelines for synthesis implementations. Not only have they made available research positions for us, they have shared their knowledge of their respective disciplines through many discussions and seminars, encouraging creative collaborations. We also thank our fellow members of the Audio Development Group for their productive contributions to the NCSA Sound Server.

References

[1] DeFanti, T., C. Cruz-Neira, D. Sandin, R. Kenyon, J. Hart. "The CAVE." *Comm. of the ACM* **35(6)** (1992).

[2] Bargar, R. "Pattern and Reference in Auditory Display." In *Auditory Display*, edited by G. Kramer, 151–165. Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVIII. Redwood City, CA: Addison-Wesley, 1994.

[3] Munkres, J. R. *Elements of Algebraic Topology*, Addison-Wesley, 1984.

[4] Risset, J.-C., and M. V. Mathews. "Analysis of instrument tones." *Physics Today* **22(2)** (1969): 23–30.

[5] Moore, F. R. "Table Lookup Noise for Sinusoidal Digital Oscillators." In *Foundations of Computer Music*, edited by C. Roads and J. Strawn, 326–334. Cambridge, MA: MIT Press, 1985.

[6] Dodge, C., and T. Jerse. *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1985.

[7] Risset, J.-C. "Timbre Analysis by Synthesis: Representations, Imitations, and Variants for Musical Composition." In *Representations of Musical Signals*, edited by G. De Poli, A. Piccialli, and C. Roads, 7–43. Cambridge, MA: MIT Press, 1991.

[8] Serra, X. "A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic Plus Stochastic Decomposition." Ph.D. Thesis, Department of Music, Stanford University, Stanford, CA, 1989. Department of Music Technical Report STAN-M-58.

[9] Choi, I. "Sound Synthesis and Composition Applying Timescaling Techniques to Observing Chaotic Systems." In this volume.

[10] Fu, P., H. Edelsbrunner, U. Axen, R. Bargar, and I. Choi. "Stepping into Alpha Shapes." VROOM (Virtual Reality Room) Presentation, SIGGRAPH 94 Conference, Orange County Convention Center, Orlando, FL, July, 25–28, 1994.

[11] Bargar, R., I. Choi, S. Das, and C. Goudeseune. "Model-Based Interactive Sound for an Immersive Virtual Environment." *Proceedings of the International Computer Music Conference*, Aarhus, Denmark, September, 1994.

[12] Das, S. "An Organization for High-Level Interactive Control of Sound." In this volume.

[13] *VSS 2.0 Sound Server Reference Manual*, internal document, Audio Development Group, NCSA. Urbana: University of Illinois, October, 1994.