SPACE, TIME AND ACOUSTICS

by

Philip R. Z. Thompson

Bachelor of Arts
University of California at Berkeley
Berkeley, California
1984

SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARCHITECTURE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1988

Signature of Author [illegible]
Department of Architecture
February 9, 1988

Certified by [illegible]
James A. Anderson
Thesis Supervisor

Accepted by [illegible]
William Hubbard
Chairman, Departmental Committee for Graduate Students

SPACE, TIME AND ACOUSTICS

by

Philip R. Z. Thompson

Submitted to the Department of Architecture on February 9, 1988 in partial fulfillment of the requirements for the degree of Master of Architecture.

Abstract

This thesis describes the development of new concepts in acoustical analysis from their inception to implementation as a computer design tool. Research is focused on a computer program which aids the designer to visually conceive the interactions of acoustics within a geometrically defined environment by synthesizing the propagation of sound in a three dimensional space over time. Information is communicated through a unique use of images that are better suited for interfacing with the design process.

The first part of this thesis describes the concepts behind the development of a graphic acoustical rendering program to a working level. This involves the development of a computer ray tracing prototype that is sufficiently powerful to explore the issues facing this new design and analysis methodology. The second part uses this program to evaluate existing performance spaces in order to establish qualitative criteria in a new visual format. Representational issues relating to the visual perception of acoustic spaces are also explored. In the third part, the program is integrated into the design process. I apply this acoustical tool to an actual design situation by remodeling a large performance hall in Medford, Massachusetts. Chevalier Auditorium is a real project, commissioned by the city of Medford, whose program requirements closely match my intentions in scope, scale and nature of a design for exploring this new acoustical analysis and design methodology. Finally, I summarize this program's effectiveness and discuss its potential in more sophisticated future design environments.

Thesis Supervisor:	James A. Anderson
Title:	Lecturer in Architecture

Preface

I have always had a special interest in the interdisciplinary aspect of architecture and issues of visual perception. This thesis has given me the opportunity to present the analogies that I have explored in the fields of computer graphics, acoustic and design methods.

Given the interdisciplinary nature of this topic, this thesis can neither afford to cover in detail nor provide a complete lesson of all three topics. This reading is primarily intended for architects, although, acoustical engineers and graphics programmers has also been kept in mind.

Finally, I would like to thank for their support: James Anderson, Carl Rosenberg, the Computer Resource Laboratory and the Robert B. Newman Fellowship Committee.

Philip R. Thompson

Table of Contents

List of Figures

Chapter 1

Introduction

As a requisite to improving man's environment, architects have always sought to represent an understanding of their environment. The design process is, in fact, devoted to the identification and satisfaction of criteria within a particular framework of many technological and social constraints. This identification and satisfaction process is a recursive step of the design process. At the outset, only a certain number of criteria are known. As the process progresses, new issues arise. Architectural design is a heuristic process, wherein the acquisition of knowledge of what does and does not work is slow, empirical and subject to creativity (resourcefulness, inquisitiveness). The continual reinterpretation of the environment and the redefinition of its character necessitates a greater sophistication for coping with more comprehensive issues in building performance. In order to increase our insight into the environment, a new acoustical working methodology is proposed. This thesis posits a new approach in acoustical analysis and synthesis that is more responsive to the architectural context by extending our visual sensibilities, and hence understanding, to cover an otherwise intangible aspect of design.

Our understanding of the environment is primarily a visual one. The design for a lecture hall in figure 1-1, while boldly addressing the visual criteria of presentation space, is most unacceptable acoustically. The architectural rendering embodies only information that is usually visually perceived. It does not show the acoustical and thermal properties of hat space. The graphic representation of sound poses many new cognitive issues. A rendering

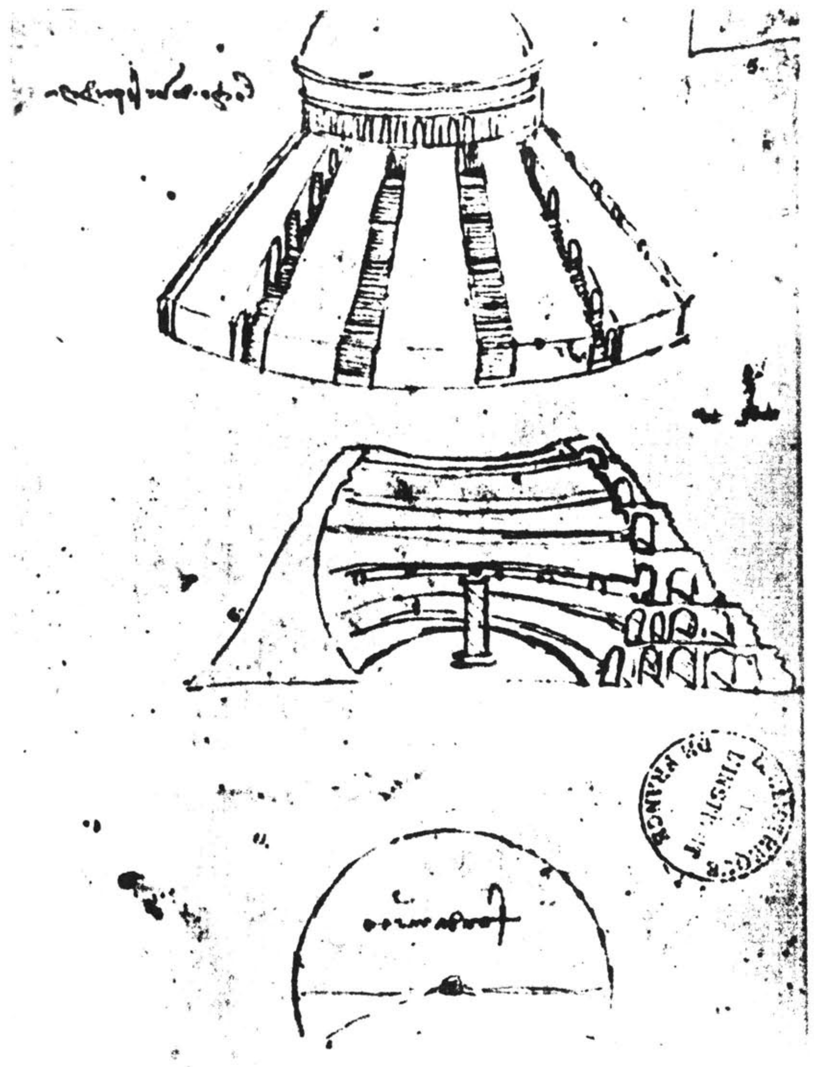


Figure 1-1: *"A place for preaching in" by Leonardo Da Vinci c. 1488-9. The speaker's platform is atop the column in the center.*

of non-visual properties provides the designer a more comprehensive understanding of the interrelationships of these properties. Realism need not be limited to the visual sense, but can extend to other perceptual modalities. It is impractical, if not impossible, to communicate certain aspects of a design directly through their respective sensory modalities (i.e. feel the temperature or hear the acoustics). While the direct perception of a space's properties may be the goal of realistic simulations, it is probably not the most effective way of presenting acoustical information for design purposes. There are often just as many reasons why a space sounds the way it does as there are critics listening to it. Acoustics, whether it is heard or interpreted from a graph, lacks an accountability that the designer can use effectively. This research presents the new concepts for presenting acoustical information in a visual format that can provide an experiential understanding as a function of analogous sensory experience. It requires a very well trained ear to decipher a spatial description from auditory perceptions only. People who are blind often have such an acute auditory sensitivity that they can perceive a room's features such as size, material, scale, etc... However, even with a finely developed sensitivity, a person cannot derive the exact acoustical features to correlate the exact effects of building elements with performance. If the architect is to shape the space that shapes the sound, then he needs to establish a precise correlation between the acoustical properties and the visual medium of drawing in which they work.

1.1 A place in the working environment

In partial response to the increased knowledge requirements of architects, computers have opened the newest frontiers in what is known as Computer Aided Design, or Drafting (CAD). I am careful to distinguish the terminology used in this rapidly growing field of computers in architecture. Few computer tools, I believe, qualify as *design* aids and should not be confused with such production tools as drafting or word processing programs that only serve to record that which has already been decided. For a program to influence a design it must produce information which guides the designer in establishing and satisfying specific criteria. I note here that a drafting system can aid the draftsman who is concerned with drawing layout and production. Better designs do not come from computer drafting systems any more than better novels are written on text formatting programs.

The differences between production and design aids is significant in terms of approach and intended clientele. Design aids are the subject of much attention in artificial intelligence research. The goals of such knowledgeable systems will be to optimize the decision making process, by assessing qualitative rather than quantitative properties. Once an acoustical response has been synthesized, it can then be evaluated for appropriateness to its intended purposes and what aspects of this response might improved to make it a "better" space. Someday, computers will be able to make qualitative assessments as to whether our treatment of a particular topic is good or bad, in architecture as in literature. However, such a complex system will not be possible without the development of low level routines through which it can operate and derive an understanding of the environment.

Before any decisions can be made by either a designer or a program, a thorough understanding of the many repercussions of decisions must be made explicit. The placement of walls is not only an arbitrary aesthetic judgment, but also one that responds to a host of usage, climate, structural, cost and social issues, to name a few. A comprehensive and responsible understanding of the interrelationships of various criteria is not possible until effective algorithms are developed to model them individually.

> Today there is a need for rule based, additive and explicit models of design knowledge, if designed objects are to mark and not to litter the ascent of man. Today concepts like analogy, typology, or memory need to be made precise enough to be computable. Only then computers will be to architecture, what instruments are to music. [Wojtowicz 86, p. 19]

In order to generate solutions, architects, like programs, need to apply production rules. When artificial intelligence is developed in the form of an expert system, production rules can either be explicitly coded into a very large program or semi-dynamically generated by a more specific sub-program. By semi-dynamically, I mean rules that are generated by a program which by definition is also a set of rules.

The comprehensive nature of the analysis presented here will, I hope, elucidate the production and construction rules as they relate to acoustics for meeting this criteria. The pictorial information produced in this program provides means for more effective exploration of memory and analogy issues in acoustics. It provides not only for sensory analogies to be made, but also for references and comparisons to other performing spaces.

The program used here provides a fundamental step in denoting and satisfying acoustical constraints in the architectural environment. The concepts of this acoustical model will serve as a platform in developing a specific design tool in architecture. The graphic media and interface developed here will, I hope, someday be applicable in modeling other non-acoustical properties of the environment.

1.2 A Short History

"Acoustics is entering a new age of precision and engineering. One hundred years ago acoustics was an art. For measuring instruments, engineers primarily used their ears. Microphones consisted of a diaphragm connected to a mechanical scratching device. About that time the names of Rayleigh, Stokes, Thomson, Lamb, Helmotz, and others appeared on important published papers. The first major publication in acoustics was Lord Rayleigh's two-volume treatise, *Theory of Sound* in 1877-78. Architectural acoustics was not advanced to the status of a science until W.C. Sabin's work and papers 1900-15. The greatest progress in the field followed the developments in electronic circuitry and radio-broadcasting in the 1920's. It became feasible to build measuring instruments that were compact and rugged. Architectural acoustics received much attention in the experiments and theories coming out of Harvard University, Massachusetts Institute of Technology, University of Southern California, and the several research centers in England and Germany. In this period, sound decay in rectangular rooms was explained in detail, along with the computation of sound attenuation of materials and ducts. The advantages of skewed walls were demonstrated and a wide variety of acoustical materials and treatments came on the market.

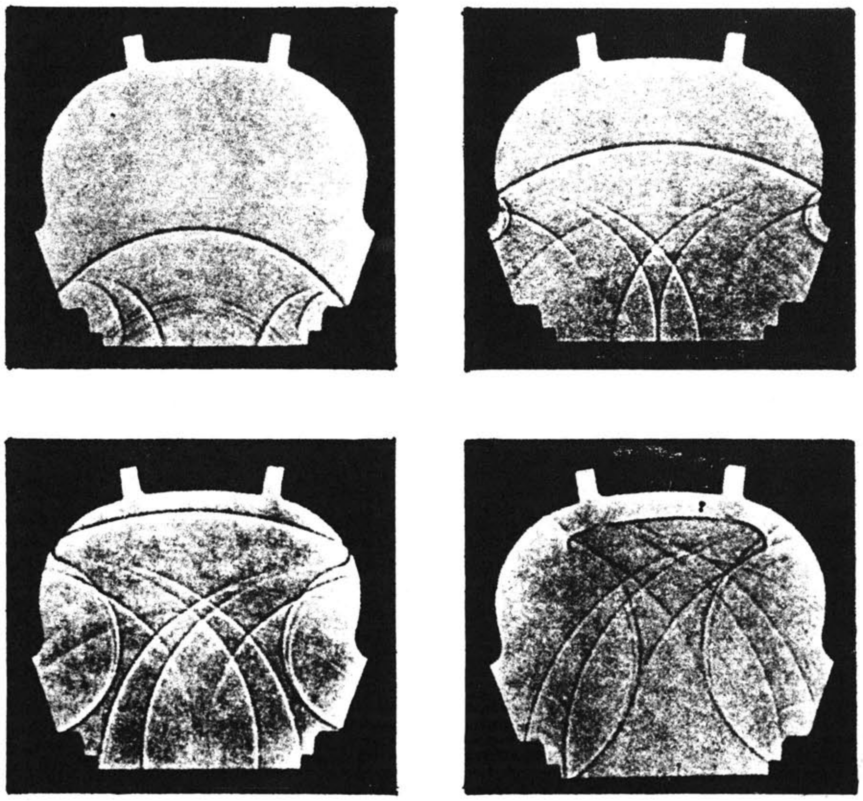


Figure 1-2: *Sabin's photographs of two dimensional sound fields.*

"Today, acoustics is passing from being a tool of the communication industry, the military, and a few enlightened architects into the concern of the daily lives of nearly every person. Workers are demanding safe and comfortable environments. The recent proliferation of performance halls, especially in small communities that cannot afford a variety of specialized spaces, is creating a demand for multi-purpose, larger, more expensive and technically demanding designs. As a result of this, architects are in rapidly increasing numbers hiring the services of acoustical engineers as a routine part of the design of buildings." [Beranek 86, p. 1-2]

1.3 The acoustical image

This program communicates its information through images that are new concepts in acoustical analysis. These images raise new issues as to their significance in visual representation. "An image is a representation of something, but what sets it aside from other representations is that an image represents something else always in virtue of having at least one quality or characteristic of shape, form or color in common with what it represents." [Dennett 81, p. 52] The information here "resembles what it represents and not merely represents it by playing a role - symbolic, conventional or functional - in some system." This visual information is the result of a significantly more powerful acoustical synthesis model.

The visual nature of plans, renderings, site visits, and photographs from which information is derived by architects is important to the way solutions are conceived. An understanding of acoustics in a format that is more analogous to our temporal and spatial perception should be more effective in producing sensitive designs. Sound has very distinct spatial qualities which, until now, had no analogous means of expression. Acoustical information has traditionally been presented in the form of numbers, which are fairly abstracted representations of sound. Many other fields of science have developed image based representations, because graphic presentation methods are more effective means of conveying spatial and temporal information. Images are now produced through ultra-sound, seismology, radar, thermal infrared photography and other non-visual physical properties. The new sound-image denotations presented here raise several new issues of representation, namely its increased effectiveness in communicating properties and the way it may be used. An image does not require a tacit knowledge of conventions because of its spatially and temporally experiential nature. For information to be communicated effectively across a range of users and architects who may have little experience in acoustics, the image relies on a more intrinsic understanding of a three dimensional depiction of space and time. The simulated three dimensional nature in which the propagation of sound is denoted provides a more analogous medium than traditional numbers and graphs, and less subject to misinterpretation.

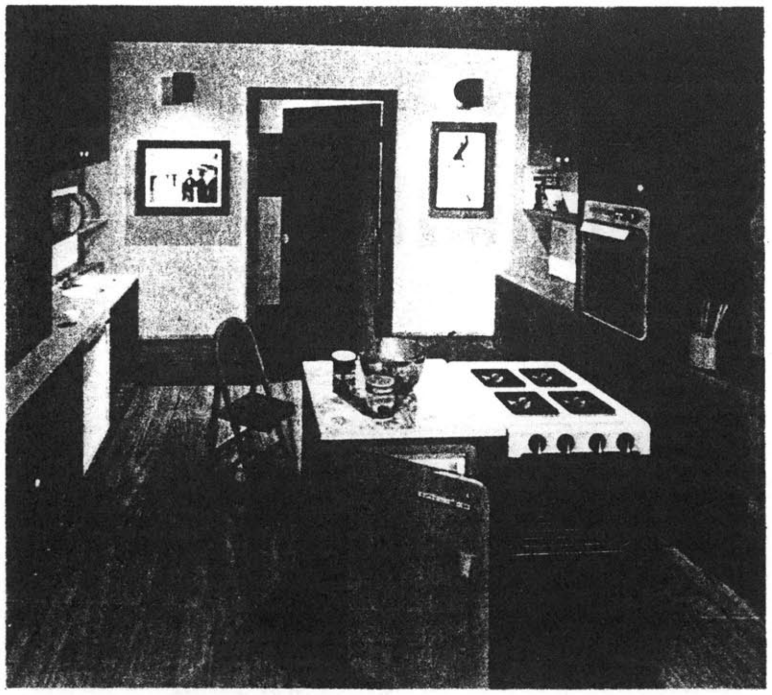


Figure 1-3: *The realism rendered using visual ray tracing techniques.*

Architects communicate primarily through buildings or pictorial representations of buildings and less through numerical or written annotations. Designs are the products of drawings and models rather than wordy descriptions of intentions. Buildings are seen through their "visible" and tactile properties such as color and texture. The exact role of perception, imagery, and graphic media used in expressing ideas is the subject of much cognitive research. I will assume though, that the visual perception of architectural properties is suited to developing and expressing ideas. The goal of these images is to extend the architect's perception of his design's properties in ways that integrate with this working process. For example, an auditorium is no longer seen only through its visual properties but also in terms of its acoustical properties.

Architects and consultants alike are not able to visualize the often complex three dimensional interactions within enclosures. Mental and manual ray tracing quickly become difficult after the first reflections. There are no physical analogies to serve as experience on which to base the propagation of sound. In order to understand this physical phenomenon, one seeks to see it. Sound cannot be marked with smoke, like the flow of air, nor are its effects directly visible like waves in water. To understand this phenomenon one then tries to build a model of it, of which ray tracing is generally a favorite. This is probably due more to its visible nature than to its accuracy.

As laymen to acoustics, most architects exhibit little control over it in practice. A space that is concerned with acoustics ought to embody these control features and not treat them on as afterthoughts, as accessory sound reflectors and diffusers. A clear understanding of acoustics in visual rather than numerical terms will allow the designer to better integrate solutions into the design product.

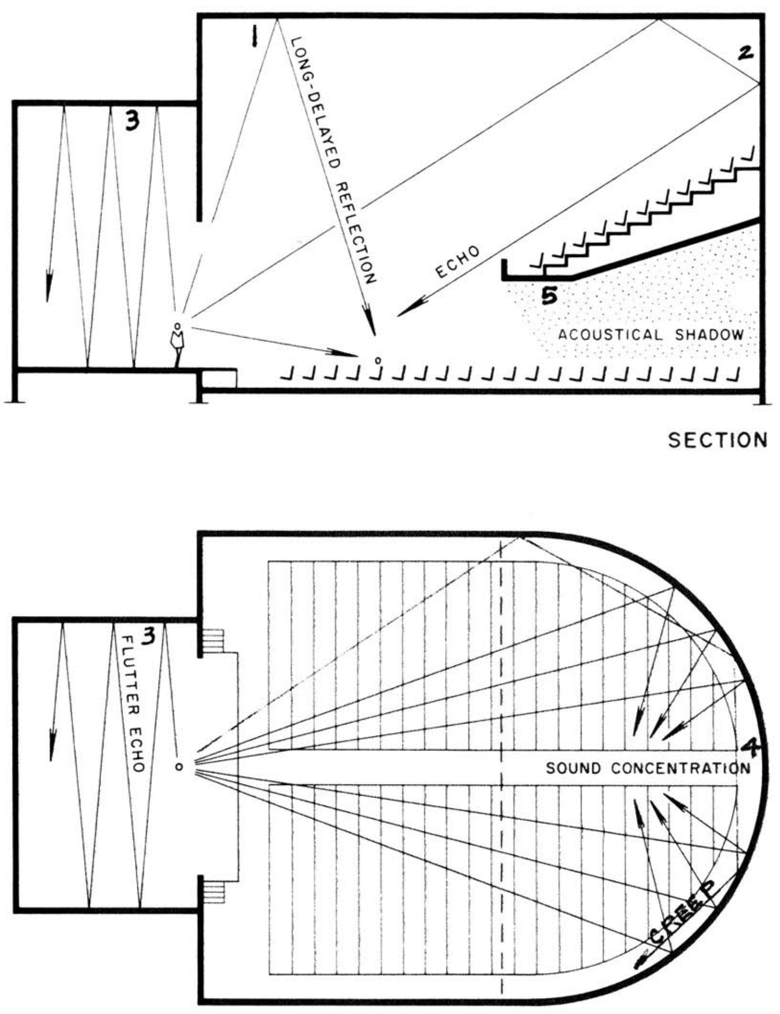


Figure 1-4: *Common acoustical design faults:* **1)** *excessive room height;* **2)** *poor sound treatment* **3)** *parallelism between surfaces at the source;* **3)** *level audience;* **4)** *curved rear wall;* **5)** *excessive balcony.*

1.4 Issues of Interpretation

Computers are finally gaining enough power to express data in very effective and graphic ways. This program communicates through renderings which are not only new in acoustics but also in a host of applications. As seen in figure 1-3, state of the art architectural renderings provide a wealth of information, leaving little to subjective interpretation.

Materials are described with an uncanny and irrefutable precision. The image establishes a direct correspondence between the many properties of that space through our principal mode of perception, that is by *seeing*. We understand a space through its properties, especially those which lend themselves to visual representation (such as materials and lighting), without having to interpret verbal or tactile descriptions. The way in which images denote the information is quite easily understood by most, and does not draw on the tacit knowledge of acoustics or the ability to make inferences from its conventions.

I believe there is a more implicit understanding of spatial and temporal change when visual analogies are used in place of numerical abstractions. One must be careful when speaking of abstractions.

> As far as I can tell there is no uniform construal in the literature of the label "abstract." It is used in a variety of ways, many of which apply to all systems or no systems (e.g. need to do not need interpretation), some of which apply equally to various pictorial and analog graphic systems (e.g. can convey information about the general and non-observable), and others which, if applicable, would make it impossible for the symbols of the system to be realized physically in psychological states of the relevant sort (e.g. the symbols are Platonistically abstract objects). [Schwartz 81, p. 127-8]

By not drawing on an abstracted representation of traditional acoustical conventions, such as decibels and decay graphs, the acoustical information can be understood by analog and direct physical references to brightness and time, respectively. A bright light represents a loud source and a quiet room is seen as a dark room. This is more effective than a number, which necessitates a scale to give it meaning (ie. nine out of ten). Some interpretation skills for this new medium must be developed, and the exact significance of these images is yet to be determined.

> Although there are various conceptions of what analog processing is, I suspect the other senses are actually derivative from the sense I am adopting. Thus, for instance, any process can be made to go through an appropriate sequence of intermediate states, and even do so in very small (quasi-continuous) steps - even a purely verbal process. ...Thus we would

count the process as analogue if its going through particular intermediate states were a necessary consequence of intrinsic properties of the mechanism or medium, rather than simply being a stipulated restriction that we arbitrarily imposed on a mechanism that could carry out the task in a quite different way. [Pylyshyn 81, p. 158]

The nature and value of images is often the subject of bitter debate in the cognitive sciences. It is important to note however, that if one wants to make predictable changes to properties displayed then he will still need some knowledge of ray tracing as an acoustical concept. One must not assume that images require a cognitive mode of information processing that involves no interpretation at all. Pictures, like many linguistic modes of communication, also require interpretation when they are serving to represent. It is unlikely that a satisfactory account of pictorial understanding can be based on unqualified notions of resemblance, similarity, or one to one correspondence between an image and what it represents. It is possible for almost anyone to look at the screen and understand that a certain sound arrives from a certain direction at a certain time. As a result it is fairly successful as a simple analysis tool for providing answers but not solutions. In the future, by coupling a constraint based system with this analysis program an understanding of ray concepts will be possible and, in turn, be able to offer solutions.

An architect does not draw lines but rather draws walls and surfaces with specific properties. Lines, however, can be subject to misinterpretation by others and even the designer himself. The realism with which these properties are depicted is a powerful aid that not only helps to communicate but also to store ideas more effectively. It also helps in determining whether a design functions properly aesthetically and functionally. The architect can look at a image and immediately see if lighting levels are adequate, materials complement each other or if the scale is correct. Acoustical information has traditionally been delivered in the form of numbers or a variety two and three dimensional graphs that are easily understood but limited in the amount of information that they can show at once. The simultaneous manner with which realism can display all of this information helps to assure the cohesiveness of a design.

1.5 A Need for a New Methodology

It has been said that a person doesn't really understand something until he teaches it to someone else. Actually a person doesn't really understand something until he can teach it to a computer, ie. express it as an algorithm... The attempt to formalize things as algorithms leads to a much deeper understanding than if we try understand things in the traditional way. [Knuth 75]

The inability to synthesize and compute acoustics precisely has in itself plagued our understanding of it. Analyses which are singularly based on mathematical formulae, graphs, plots or simple ray diagramming techniques cannot provide a sufficient understanding of the many spatial interactions of sound. The three-dimensional, temporal and invisible nature of sound is very difficult to express accurately and comprehensively. The numbers and graphs that are produced by traditional analysis methods depict only a small aspect of its nature. For example, one formula will produce reverberation time, a simple ray tracing model will locate reflections, and yet another formula may calculate the reflected energy from that reflection. The three methods alone will produce a graph, a diagram and a number. Yet for acoustics to be understood precisely a much more comprehensive or global description of the space must be made and detailed.

Recent advances in technology are allowing for more sophisticated methods of recording and analyzing existing spaces. The analysis and synthesis are two separate processes. An analysis denotes the behavior of a sound field condition. It is usually performed on data recorded from existing spaces, but can also denote the data from synthesized acoustical responses. Advanced recording technology can store a great deal of information for later recreating the listening experience, and for accurate analysis. The high performance *Monoraul* and *Single Point Quad Miking*[1] recording techniques are allowing engineers to understand and develop a vocabulary to express the properties precisely. These recording and analysis techniques, however, do not lend themselves directly to synthesis models useful for the evaluation of designs yet to be built. While many scale-modeling and mathematical modeling techniques for synthesizing design performance do exist, few are readily used by designers and consultants. Scale-modeling techniques are generally regarded as too slow and expensive while mathematical models, even those implemented on computers [Borish 84, Wayman 80, Benedetto 85], possess limited power and information, and lack usable interfaces. Those techniques presently available have not kept pace with the many acoustics applications outside of architecture.

Several methods may be combined to form a more complete description, but the disparate range in which information is delivered makes it a time consuming and difficult process. As a result, consultants often limit their analyses to only a few methods and rely a great deal on their previous design experience and on references to other halls. This is not a suitable solution for the architect who is concerned with the design of complex or unique spatial configurations. Furthermore, the comparisons against other performance spaces that are possible today by critics and users places additional demands for a thorough un-

[1]This is used by the Yamaha DSP-1 stereo component that can recreate a variety of listening environments from small jazz clubs to famous large cathedrals.

derstanding of acoustical properties in order to insure a successful design. With the present tendencies toward increasingly larger and more polyvalent performance spaces seating 3,000 or more, the designer can ill afford a hit or miss approach.

Leo Beranek summarizes a designers responsibilities in the quote:

> In performance spaces the sound emanates from the orchestra, actors, lecturers and loudspeakers; located on the stage, in the pit, and above or at the sides of the proscenium. The architect's job is to get it to the paying customers no matter where they sit without distortion or appreciable loss of intensity. A good space consists of having getting the sound energy there at a uniform intensity and time, then having it die away at a predetermined rate so as not to interfere with the next sound as it comes along.

One step in developing this new methodology was to test the implementation on a design project. Given the scope of this thesis, I have chosen a rehabilitation project, concerning myself with one of its more critical performance criteria - good acoustics. The redesign of Chevalier Auditorium is a real, concurrent project in Medford, Massachusetts. The present auditorium, a local multi-purpose hall, is to be transformed into a regional performing arts center. In order to concentrate on satisfying particular acoustic constraints of a design, and not compromise this criterion with other issues such as site, climate, use, etc..., I have chosen to design this project with one principal objective: a performing space with good acoustics.

Chapter 2

Program Development

At the heart of this research project is the implementation of this new sound synthesis model in an acoustical ray tracing program. This is the first comprehensive three-dimensional model using ray techniques that can display information about reflected and transmitted sound in a graphic manner which is readily understood by engineers, architects and laymen. Simple ray tracing (fig. 2-1) has been one of the most commonly used tools used in acoustical analysis. It has traditionally been implemented either by hand, in a simple two-dimensional fashion, or by computers to produce a three dimensional analysis limited to numerical output.

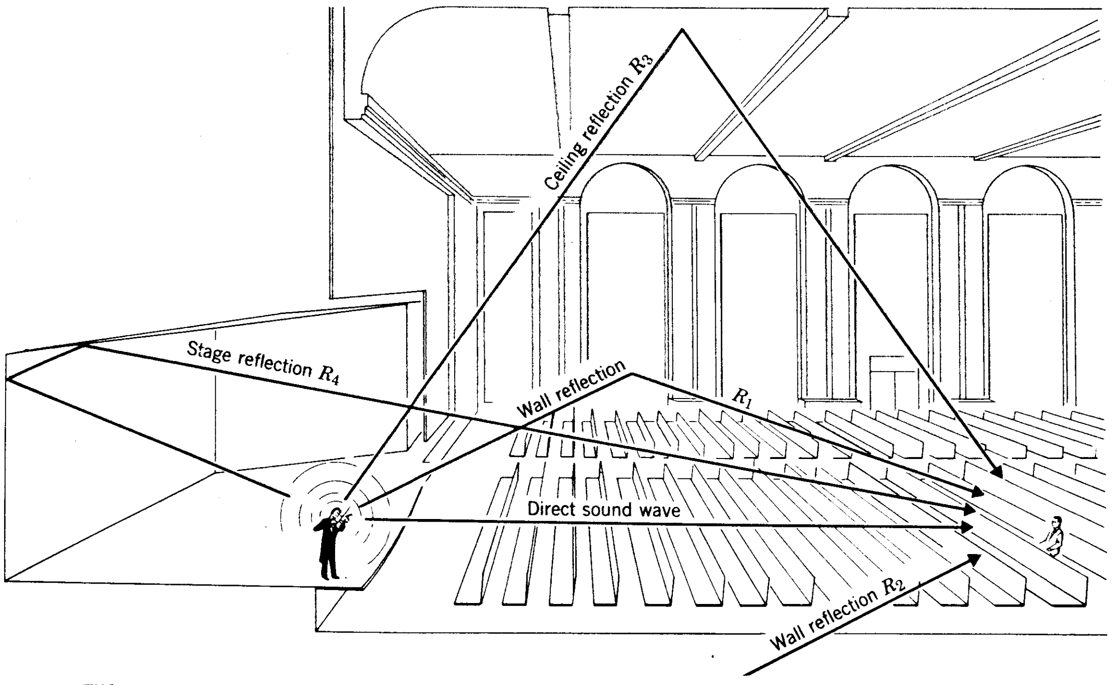


Figure 2-1: *A traditional ray diagram of the direct and several reflected sound waves in a concert hall. Reflections also occur from balcony faces, rear wall, niches, and any other reflecting surfaces.*

Recursive ray tracing is a relatively new rendering approach in computer graphics since it was originally implemented in 1979 [Whitted 80, Kay 79], it has been combined with various illumination models to produce some of the most realistic images to date (fig. 1-3). For a detailed account of computer ray tracing principles the reader is referred to the papers by J. Whitted [Whitted 80] and D. Rogers [Rogers 85]. Before ray tracing techniques were developed, the correct visualization of reflections, transparency, shade and shadows were impossible in even the most comprehensive programs. As an architectural tool, it offers immediate benefits through its ability to create renderings with accurately cast shadows and reflections with greater speed and precision than scale models.

The main similarity between optics and acoustics exploited here is that both light and sound can be modeled as vectors traveling in a line (fig. 2-3), such that the physical laws can be modeled using linear algebra. Light is, in fact, often used in place of sound to model the specular acoustical properties in scale models or to fine tune built designs. However, the respective interaction of light and sound with material properties present problems as each behaves differently under given physical circumstances. For example, a hard dark surface may reflect sound but not light, while light may easily pass through glass while sound will not. It is also not possible to use light to perform time analyses, since it travels much too rapidly for the propagation delay and reverberation to be perceptible. Hence limitations quickly appear with literal physical models that replace sound with light in other than rudimentary specular models.

A significant aspect of this ray technique is the fact that a design's properties are depicted *from the viewer's* perspective. In this ray tracing

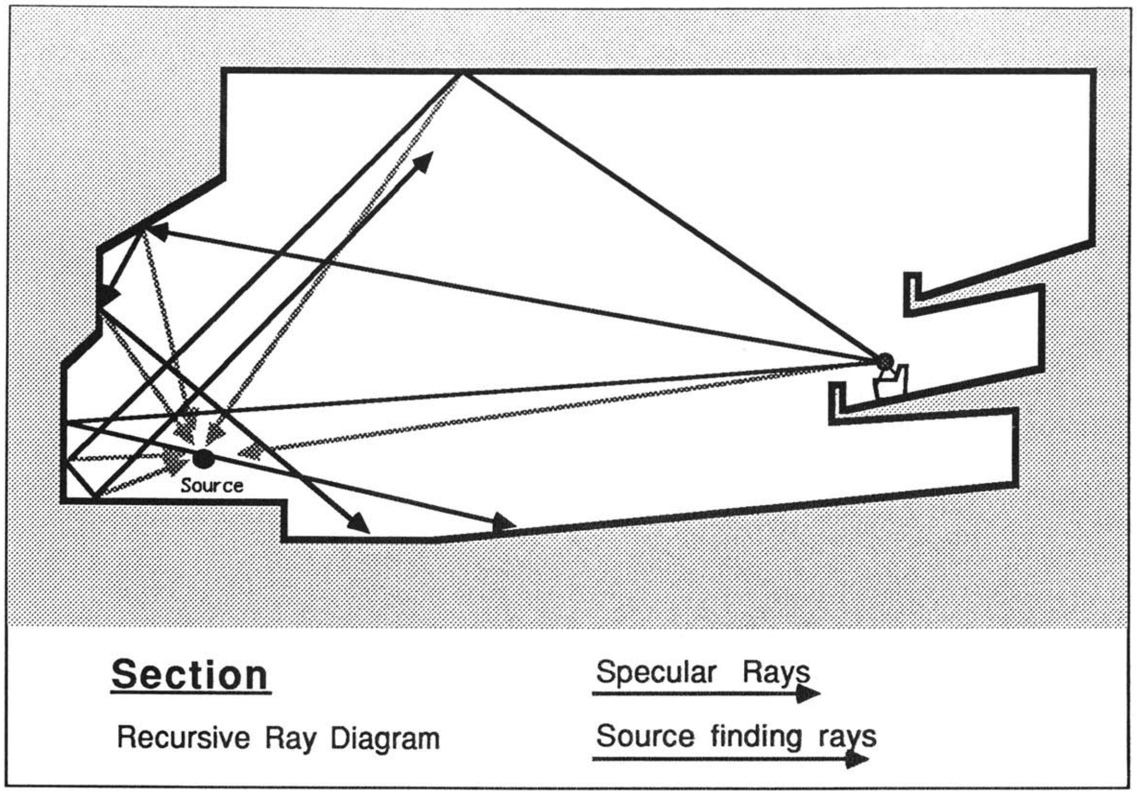


Figure 2-2: *Diagram of the rays emitted from the viewer.*

scheme, contrary to conventional acoustical methods, the receiving point emits the rays instead of the sound source (fig. 2-2). Rays are sent out from the observer's ear (or eye) and travel *back* to the source(s). This viewpoint allows renderings to produce a picture of what the listener hears instead of what the sound source "sees". It is important to realize that the propagation of sound is perceived as a function of the listener's and source's locations. The specular properties are directly related to the observer's and sources' relationship with a space. As a grammatical analogy, the sound is perceived in the first person, and removed as a the third person observer. To analyze a particular point, an observer must be situated at that point himself, not just looking at it from across the room. Each different vantage point reveals a unique performance characteristic.

The computer program has two basic parts: a ray tracing algorithm and

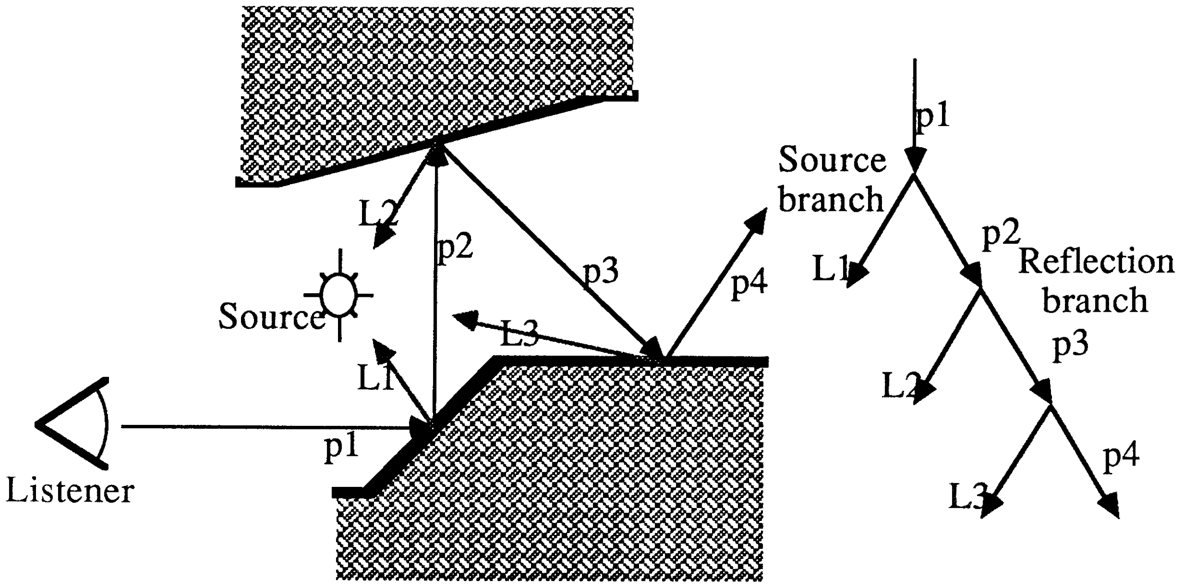


Figure 2-3: *Diagram of the reflections of a single ray in an enclosed space and its respective ray tree.*

an intensity (illumination) algorithm. In a well written program these parts operate fairly independently of each other so that different ray tracing and illumination algorithms are interchangeable in a modular fashion. The ray tracing algorithm calculates the ray paths, while the intensity model, or "shader", determines the energy incident at each ray-surface intersection, or "node" in the ray's path. The ray tracing, or visible surface, algorithm recursively creates a tree of rays for each pixel of the display (fig. 2-4) Recursive means that as each ray intersects a surface, new rays are spawned which, in turn, intersect with other surfaces. The main attribute of this tree is that it records the distance and direction of the rays to the next surface intersection. Once a ray tree is completed, it is passed to the shader for intensity calculations. Since the intensity of a node is greatly dependent on the intensity of the node preceding it, the shader traverses back up this tree accumulating the intensities at each node until the listener is reached. The ray tracing algorithm cannot only display the location and relative intensities of the real and virtual (reflected) sound sources in a steady state condition but can also produce a series of images mapping the propagation of an impulse sound field over time.

2.1 Limitations

The ray tree described above is effective in following the specular interactions in a space, but there are limitations that should be noted here. The field of computer graphics uses mathematical models which approximate but cannot completely follow the true physical interactions of light. The intensity models to date are not to be used for determining accurate pressure level readings, since they do not completely follow the diffuse second order interreflections within an enclosure. The implementation of such a complete physical illumination model is too complex and computationally expensive. The intensity formula used in this rendering tool can model the specular and first order diffuse reflections which are of primary importance in acoustical analyses.

Approximations must be made in describing the space for analysis. Attaining a high level of detail in a geometric database describing all of the elements comprising a space is prohibitively expensive to draw and, subsequently, to compute. Most professionals are interested and operate on the macroscopic aspects of the spatial geometry. The "hall of mirrors" effect, from acoustically reflective walls, becomes very confusing in a complex room with very many elements. It may be possible to provide too much information, thereby reaching a threshold where any more information becomes confusing. The interreflections create visual noise where it is impossible to attribute the causes and effects of reflection. It becomes very difficult to distinguish specific features of a space and ascertain causes of properties displayed.

Presently, a problem lies in the fact that it is not known to what degree the overall accuracy and effectiveness of the analyses are compromised by these simplifications to the ray tracer and geometric descriptions. The effects of diffuse interreflections and small elements within a room's geometry are generally not considered significant. The lack of the second order diffuse component is not significant. The viability of other ray based measurement methods [Wayman 80, Benedetto 84] and sound reproduction using ray methods [Yamaha 86], modeling only the specular components and using simplified spaces, tend to support this claim.

The diffuse interreflection calculations would require sending out a huge number of rays at each node to determine the exposure to each diffuse surface. There are alternative computer methods for accurately determining second order diffuse reflections, based on light models using *radiosity* algorithms [Cohen 85, Nishita 85]. However, these radiosity techniques can only model surfaces with perfectly diffuse (Lambertian) properties. This is not suitable for architectural acoustics, which often deals with large surfaces that, given the large wavelengths of sound, appear acoustically smooth. For example, what may appear as a visually hard dull surface may actually exhibit acoustically specular properties.

2.2 Recursive Ray Tracing

The following section describes the ray tracing or visible surface algorithm as used in this research. The algorithm as written (appendix A), is simple but slow. Its advantages are that it is clear and allows for easy modifications. For future applications it is suggested at a much faster and more sophisticated version be used [Kay 86, Fujimoto 86]. The concepts of ray tree generation are directly borrowed from lighting versions; other versions should be very portable to meet acoustic needs. The principles and functions of recursive ray tracers remain the same and are described below.

To generate an image, a ray is sent out for each of the pixels on the computer screen. Hundreds of thousands of rays are then generated that travel from a point perpendicular to the sound field so as to cover a solid angle in the direction of interest. If a ray hits a surface, reflected and/or transmitted rays are recursively generated and sent out so as to form a tree (fig. 2-4). Rays incident on specular surfaces obey the physical law whereby the reflection and incidence angle are equal. At each ray-surface intersection source-finding feelers S_j are also emitted. These source-finding rays also check for occlusion by other objects and are used for intensity calculations by the shader.

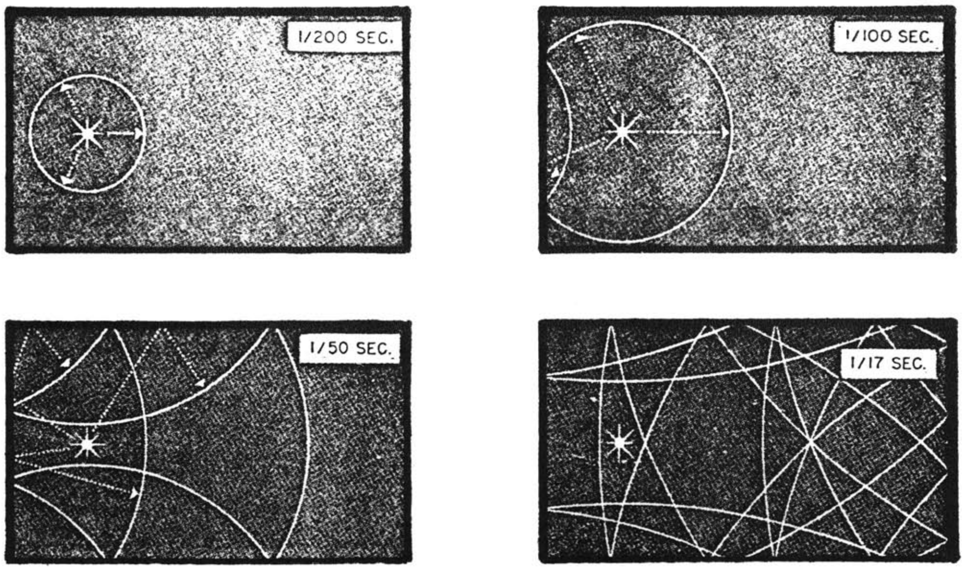


Figure 2-4: *Progressive reflection of a single sound wave in an enclosed space.*

At each surface intersection, or "node" in the tree, an attenuation may be calculated from the respective specular property of the surface intersected. Theoretically, the ray tracing tree is infinitely deep and is terminated once all rays leave a scene. The reflected sound inside of an enclosure is trapped, and the ray tree ends only once the sound is completely absorbed. In order to save on computation time, the recursive ray generation may be terminated when the accumulated impedance of the surfaces intersected reaches a specified value (such as 95 per cent), or when a prescribed maximum tree depth has been reached, usually around five to ten reflections. The recursion may may stop before a ray ever reaches a source, in which case no specular sound reaches the listener from the source.

One of the first extensions that had to be been made involved making of the sound source *directly* visible from the viewer. Most ray tracing programs are not concerned with this facility. The sources are well out of the field of view or are occluded by other objects so as not to be directly visible. Only the reflections of a source are visible on the objects. In a performance space, however, a listener is usually interested in looking directly toward a source[2]. The source is made visible by sending out source finding rays directly from the receiver.

2.3 The Acoustical Model

The acoustical intensity model, or shader, is used to determine the energy levels at each node of the ray tree. The intensity incident at a node is primarily composed of reflected energy, and other optional transmitted and ambient energy terms. The reflection component is comprised of specular energy and a first order diffuse energy. The specular component is further divided into

[2]References here are made to a singular source although it should be noted that they may represent several other sources.

energy incident from a previous surface and from the source. The energy from a source is then attenuated as a function of the displacement of its initial power and the properties of a surface. Properties at a node attenuate by a respective surface's impedance or Noise Reduction Coefficient (NRC) rating. If an intervening surface is transparent, the Sound Transmission Class (STC) rating characteristics of the surface is used as the attenuation factor.

When the intensity model has been completely traversed the tree back to the listener, the resulting intensity is displayed as a color at the appropriate pixel on the screen. Intensity is denoted on a grey scale, according to the display device, but usually with 256 possible shades. The shader represents little or no sound from a particular direction as zero or black on the screen.

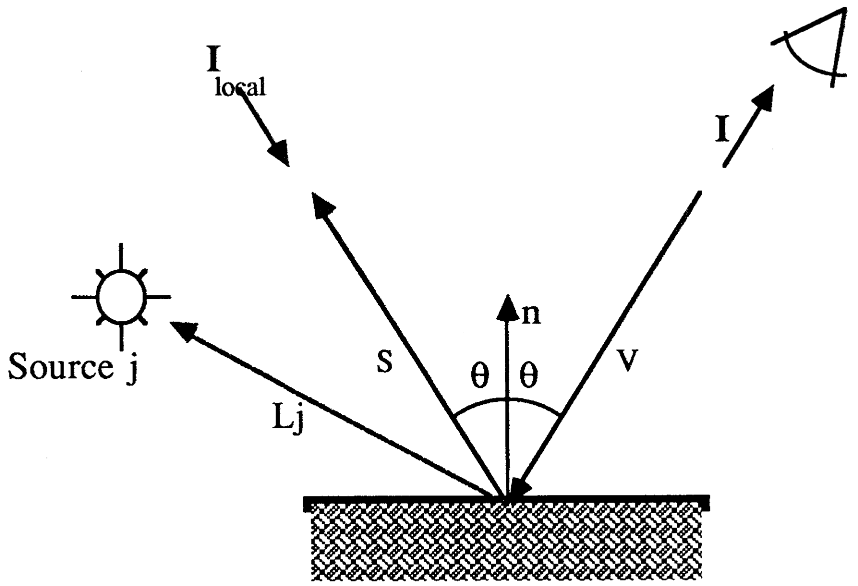


Figure 2-5: *Local Specular and diffuse reflection components for the intensity model.*

Although the exact nature of reflections from surfaces is best explained in terms of microscopic physical interactions between sound waves and the surface, most shaders depend on the aggregation of local surface variations for significant reductions in computing time. As computers become more powerful, a better understanding at this microscopic level may one day be explored to yield more accurate models. A model for the specular reflection of sound from smooth surfaces follows the formula where the incident energy is attenuated by an impedance coefficient, *k*, as in:

$$I_{reflected} = I_{incident} k_{specular}$$

As shown in figure 2-5, the sound intensity, *I*, passed to the viewer from a point on a surface, consists of the specular reflection term along the vector s. **n** is the surface normal, $\mathbf{L}_j$ is the direction of the *j*th sound source, **S** are the local sight and specular reflection directions. The global intensity[3] for each pixel is the sum of the local interactions, passed along *I*, where at each node:

k_s and k_d are the surface specular and diffuse coefficients, respectively.
I_s = energy from the source, S, direction.
I_j = energy from the specular, L, direction.

To form the formula for *j* sound sources:

$$I_{local} = k_s I_s + k_s \sum_j^{j} I_{l_j} (\mathbf{S} \cdot \mathbf{L}_j)^p + k_d \sum_j^{j} I_{l_j} (\mathbf{n} \cdot \mathbf{L}_j)$$

The first summation term is a source's specular energy contribution. It is dependent on the source being sufficiently collinear with the specular path to the viewer[4]. The second summation is the source's diffuse energy contribution at a node; this is proportional to the angle of incidence to the surface. Here, the reflection coefficients, *k*, are held constant. However, other models may be used to determine their variations with the incidence angle or wavelength. By making k_s and k_d smaller or larger, the surface can be made less or more reflective.

---

[3]This intensity model is derived from the global illumination model [Rogers 85] used for light.

[4]This is based on the "Phong" [Phong 75] spatial distribution value used to determine specular highlights.

Presently, the diffuse property is denoted by k_d and has no industry equivalent. The problem lies in that there is no standard for roughness and it may be expressed at the microscopic scale of surface particles to a macroscopic scale of furnishings and building elements. The diffuse energy is proportional to the dot product of the surface normal and the incident source energy I_{l_j}. The diffuse component is useful for demarcating shadows and energy flux on an area.

Although the shadows cast do not take into account the refraction of waves around objects, they do help demarcate those areas with direct lines of sight, which is considered an auditory and visual requirement of a performance seat. Sound waves, especially at lower frequencies, can refract *around* objects and virtually eliminate any distinct shadows. When a person is in the shadow of an object he can still hear a fair amount of sound. This is a point requiring further study.

A conceptual difference with traditional rendering application is that a source is the only energy in a space. There are none of the ambient and or external transparency terms often used in light models. The ambient energy term is a pseudo approximation of the other environmental factors which are otherwise not easily accountable, such as background audience and mechanical noises. Ambient term is a steady state value that cannot approximate fluctuating real-life conditions.

2.4 Time Analysis

Since the reverberation of light is considered instantaneous, time factors are of little interest in light models. The program as described until now is primarily derived from light-based models, and produces images showing only the integrated or steady state effects of sound. However, due to the slow speed at which sound travels, the reverberation and reflection times are very significant factors in acoustical perception. Several display methods were investigated which could communicate reverberation effectively. The most promising methods involve drawing multiple images or frames at discrete time intervals. Each frame can then be shown individually or in rapid succession, creating a stop-action or animated sequence of the sound field propagation.

Until now, general mathematical formulae had been used in determining the reverberation time of spaces. For example, Sabin's commonly used reverberation time formula[5] R_t=0.05V/a gives only a very rough notion of how a space functions temporally. It is valid only in cases of uniform room geometries, uniform distribution of absorptive material and sound propagation. In most halls however, these variables are often quite inconsistent throughout a space.

In addition to direction, each ray has a given magnitude or length which is used to deduce the time of travel. With this information, a critical assessment of sound travel over a tree's cumulative path length is possible. Reflections can then be identified precisely as to location and time of arrival.

Once a ray tree is passed to the shader, the free path lengths are then

[5]"a" is the total Sabins (absorption value) within a space

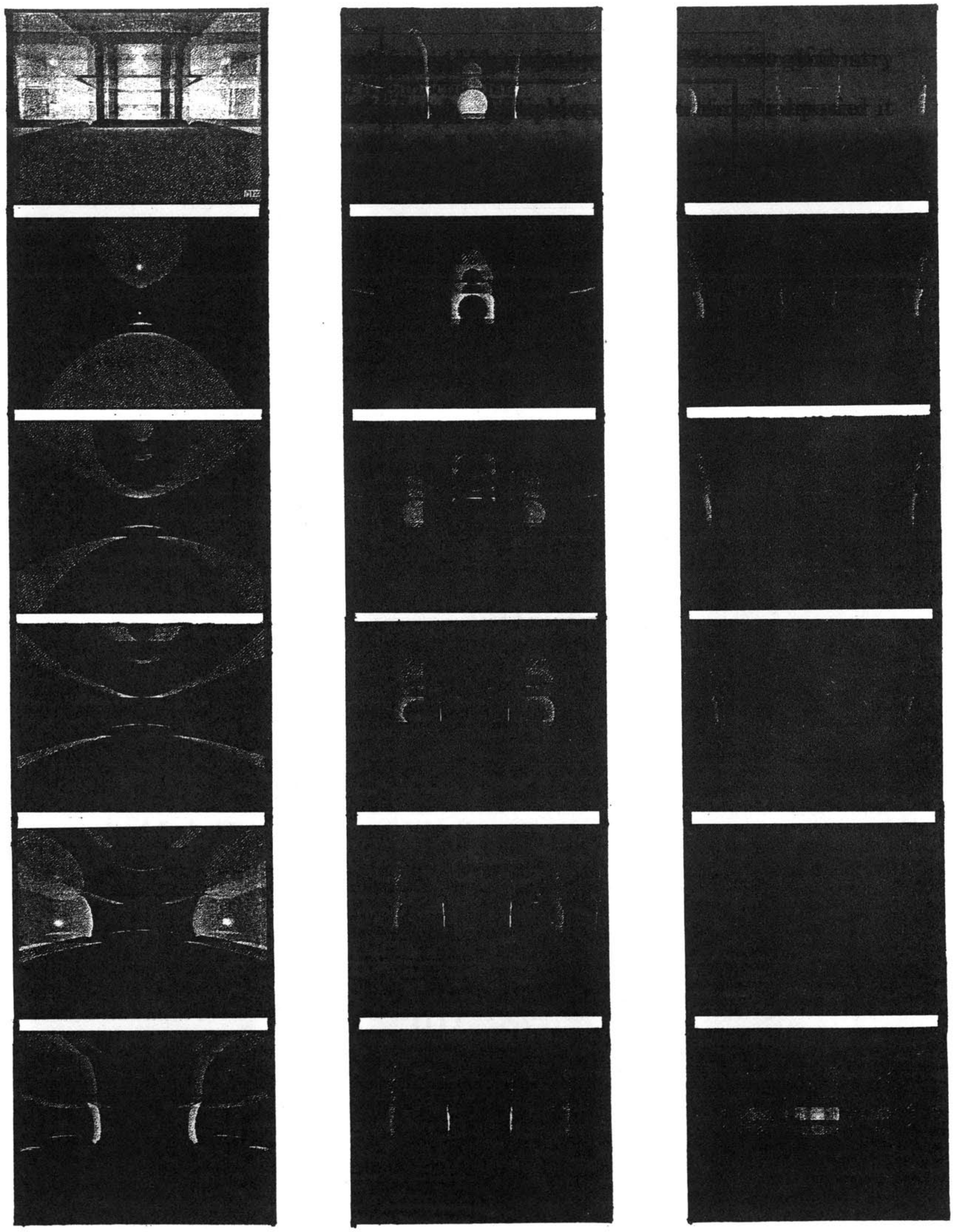


Figure 2-6: *Time frames showing all of the incident sound at 10 milliseconds intervals. View from the audience looking towards the stage.*

divided into time segments which are sorted by their respective distances from the sources. A frame is then drawn for each range of distances covered by a segment, so that in effect, the more direct reflections appear first and then the more roundabout paths appear later, as the sound field travels uniformly out from the source. This allows a person to closely follow the impulse wave front radiating out from the source towards the observer, seeing when and where various reflections take place, as in a strobe lit photographic sequence. For example, if the shader were to divide the tree at ten millisecond intervals, then the tree would be divided up into roughly ten-foot segments, with an image showing all the reflection falling within that interval (fig. 2-6).

2.5 Directions to be further explored

One display method is to superimpose the later sound fields arriving at specific intervals over the initial energy. A second time analysis method, as an extension of the second model above, would be to again divide the tree into time segments but to accumulate the intensities from the previous frame. The frames would show the total cumulative energy arriving up to a given time interval. An image would then "develop" over time. At fifty milliseconds one could see all of the sound having arrived during the reinforcement span.

Several advanced aspects of this project await further development:

- In optics, *transmitted* rays are refracted according to Snell's law; in acoustics, however, this remains to be explored.
- In modeling the effects of sound refraction, varying frequency around objects eliminates any distinct shadows. Ray tracing has been performed with cones [Amanatides 84] to simulate fuzzy shadows and distributed light sources, and an adaptability to refraction remains to be explored.
- Testing and calibrating results to real performance space measurements.

2.6 Implementation

Implementation of this program in its present state is quite simple. An outline of the parts involved in performing an analysis are a spatial database, viewing and source parameters, and the ray tracer program (fig. 2-7).

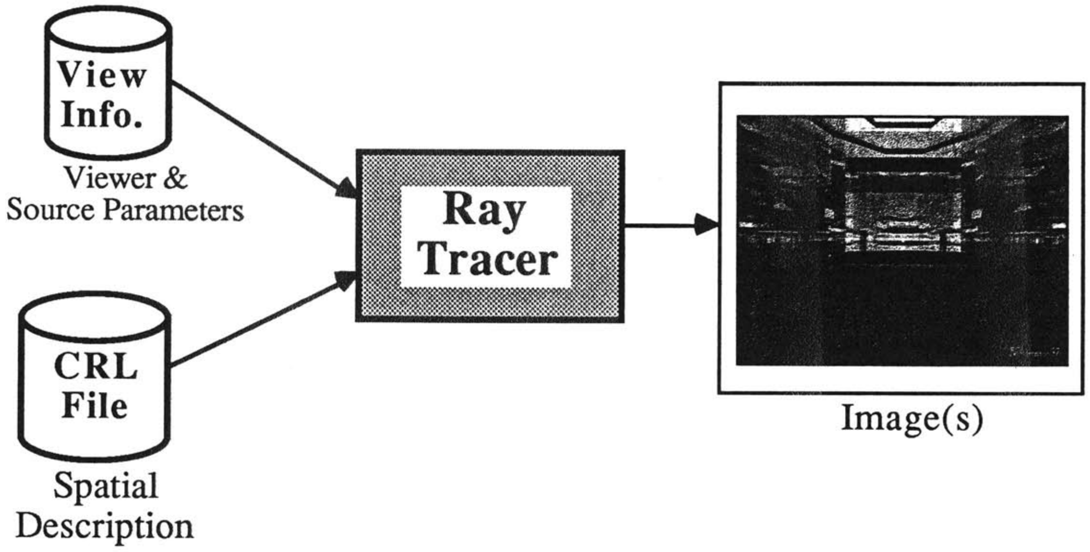


Figure 2-7: *Components needed to use the program.*

Creation of the database is presently a slow manual process that uses a text editor rather than a graphics editor. A graphics editor such as a CAD program would allow for a much quicker, easier and more complete description of spaces. The present format of the database is an early version of the CRL Schema format currently under development here at MIT. The CRL format has special facilities for recording specific knowledge about architecture and planning. It allows descriptions in terms of attributes and conceptual structure, in addition to a variety of geometric properties. The CRL schema is unique in its ability to manage general information about the world, not just an image of the world [Jurgensen 87]. The main advantage of the CRL format exploited here is its ability to store surface descriptions and a variety of architectural material and acoustical properties. The schema allows different types of application programs to access a common description of a situation, building or site. It provides a much more powerful descriptive language for sharing data amongst programs much like the *Initial Graphics Exchange Specification* (IGES)[6] and *DXF*[7] standards commonly used in the profession.

The databases of the various spaces analyzed were procedurally defined and involved writing a program which, in turn, wrote the CRL files. The main advantages of this were that parts of the building were readily identifiable and changeable. Modifications were most easily made in this way, since CRL files are not easily readable by people (example in appendix B). A three dimensional wire frame program was also written and was essential in previewing and debugging the database in a graphic format. This program was extremely useful for correcting errors and viewing design changes.

The present ray model uses intensity values that are normalized units between zero and one. Actual values are then relative to this scale. Reflections are relatively attenuated according to surface properties, or coefficients. A source may emit an energy level of one unit which may represent one watt or 500 watts. What is important here is how much power is arriving relative to the source. This works much like reverberation time measurements, which depends on a relative 60 dB reduction in intensity, regardless of the source's power.

---

[6]IGES is produced by the National Technical Information Service for the U.S. Department of Commerce. It is a formal standard accepted by the American National Standards Institute, which is generally found on larger minicomputer- and mainframe-based programs.

[7]The DXF standard was developed by Autodesk Inc. for use in its Autocad program and is popular among PC applications.

To use the program the designer locates himself and the source(s) by specifying the points within a space[8]. Some empirical experience is useful in selecting where these points should be located. It is best to analyze only those points in suspected troublesome areas and those locations most representative of the main seating areas. For example, the sampling points should cover the edges and middle of the audience, under deep balconies, the foci of curved surfaces, and other suspect areas. Observation points need not be limited to the traditional audience locations: such as the stage itself (as if listening to an accompanying musician) or high above the stage (as a microphone in a recording configuration). The listener may even be a source himself, as long as the source and receiver points are not *exactly* coincident. This is not actually a problem in real life since a musician's ear is always at some distance from his instrument, and it is technically impossible for a source to emit and receive simultaneously.

The same constraints apply to the locations of sound sources, with one distinction being that more than one source may be defined simultaneously. Sources can be located anywhere in a space as long as they are not exactly coincident with the listening point. Additionally, an output power parameter also needs to be defined. At the present time, power output is in normalized units between zero and one.

The angle of coverage of a source is an additional property which can be defined in two different ways. The angle of coverage defines the energy distribution characteristics of a source over a specific volume of space (fig. 2-8a). Presently, this is most easily done by enclosing the source on several sides with polygonal surfaces. The angle of coverage is then modified by the geometry and opening of the bounding volume (fig. 2-8b).

---

[8]As a note, the points referred to here and in later discussions are in the three-dimensional world coordinates in which the space is defined.

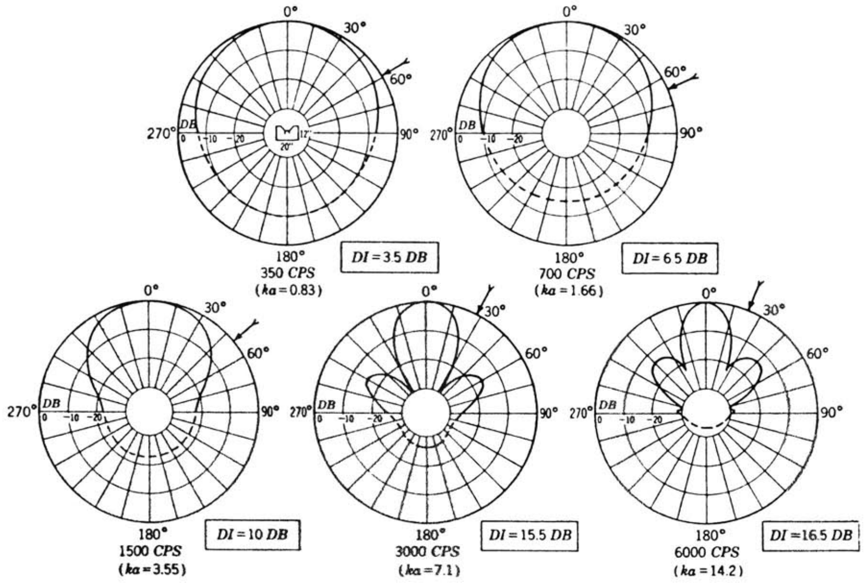


Figure 2-8: *Measured directivity patterns for a typical direct-radiator speaker coverage of a typical speaker. DI is the decibel index and CPS is the cycles per second. Note variations with frequency.*

A problem with this bounding method is that the enclosure is separately defined in the building's geometric database while the source must be properly situated as a separate viewing parameter. This simple bounding method does not provide an accurate modeling of the energy to angle relationships found in real sources.

In the future, a more accurate and computationally faster alternative will be not to use a bounding enclosure, but rather to map the energy distribution directly onto the source and to attenuate the source finding rays accordingly. Methods for entering the distribution pattern, however, need a more comprehensive and interactive interface to be developed later.

Time delay is another parameter of a sound source that can be defined to depict a common feature of modern amplification systems. It is then possible to fine tune and predict the effects of distributed sound systems commonly in use in airports, stadiums and large halls. A delay retards in milliseconds the time at which an impulse source becomes "visible". This delay feature is effective only when one or more sources is delayed relative to another source. An analysis of the synchronization of sources and the effectiveness of sound reinforcement systems is then possible. This is also particularly useful in determining the Haas effect whereby the sound of a second source must arrive within time and below a certain intensity level, so as not to confuse the location of the primary source and yet reinforce the overall intelligibility.

Chapter 3

Evaluation of Program Results

The acoustical criteria and the ability to make qualitative judgments about a space remains difficult in any medium, and is the subject of much psycho-acoustic research. This begins to answer a most important question, "What does a good sounding space *look* like?" A first step of forming visual criteria is to establish a library of visual references. For this, I attempted to evaluate Boston Symphony Hall, a space that has well-known acoustical properties. From these reference images it is then possible to extrapolate what we believe are good qualities of a space. At the outset of this project, some assumptions are made as to what constitutes a good or bad image.

The present implementation of acoustic ray tracing program is quite simple, yet it brings forward not only new data but also many associated issues with this new methodology. The program described in the previous chapter is listed in appendix A. It is sufficiently developed to substantiate and test many of the concepts behind this new analysis methodology.

3.1 Understanding an Image

Our eyes adjust to a viewing environment and the display of a computer monitor, making quantitative judgments regarding the intensity (or sound) levels difficult. This has not been a problem when using numerical data. The need for a calibrated visual reference a scale becomes apparent. Additionally, the very slight changes in grey tones make it difficult to distinguish discrete values. An intensity that may be represented by a number between 0 and 1 may have any 256 intermediate shades between black and white. Without a complete scale against which compare a value, it is very difficult to assess the energy which corresponds to the exact intensity of a color. A reference scale would be situated in much the same way as the legend on a topological map assigns ocean depths to various shades of blue.

The lack of an accurate and tested intensity model makes it difficult, if not impossible, to draw specific quantitative judgments from the images. It is very difficult to determine how a sound attenuation of 60 dB appears. While the color of a pixel may represent a new medium for representing acoustical data, the image on a much larger scale presents a completely new format for storing and communicating that information. The color values of a pixel on a computer screen are only an alternate form of denoting an intensity usually described by a number. An image is nothing more than a two-dimensional matrix of intensities. Using this principle it would be easy to represent these intensities directly with numbers so as to create a matrix of numbers rather than color values. A problem with this is that image coherence would be lost since the area numbers cover does not become lighter or darker proportionally to its value; the display would then loose its analogous pictorial quality.

Similar problems of determining the value of a single pixel extend to regions and the entire image itself. The difficulty of comparing images is in assessing whether more or less energy arrives between them. To solve this problem, the intensities are summed for regions of the display. The summation of the pixel values produces a more quantifiable number depicting the total or average value for that region. This number can then be compared, referenced and passed to other programs more easily.

As a preliminary step towards a more interactive design and analysis interface, techniques for using a mouse to interact with the image is an another important aspect of this environment. It is possible to use a mouse to define areas of interest and point to particular locations in order to extract more information. This is discussed in the conclusion.

It is also useful to examine the interactions account for the value of a particular pixel or a region. The ray program is able to deliver the ray tree description of a particular pixel that, in turn, allows for the careful analysis of the effects of objects and properties. In the future, this description of the ray tree will allow the program to interact with other high level applications such as expert systems and constraint managers.

It is important to realize that the propagation of sound is perceived as a function of the listener's and source's locations. The specular properties are directly related to the observer's and source's relationship with a space. As a grammatical analogy, the sound is perceived in the first person, and not removed as a the third person observer. To analyze a particular point, an observer must be situated at that point himself, not just be looking at it from across the room. Each different vantage point will reveal a unique performance characteristic.

A particular feature of the images which requires some getting used to is the three dimensional perspective of the space in the images. When a person looks at the images, especially for the first time, there is an inherent tendency to compensate the foreshortening of the perspective views. This tendency appears to be more pronounced when looking at the time sequence of an impulse response than in the steady-state conditions where time is not a factor. Reflections that *appear* to be further away tend to naturally be interpreted as arriving later. This, of course, is unnecessary since the time sequence displays the energy sorted according to distance of the listener, regardless of its apparent distance.

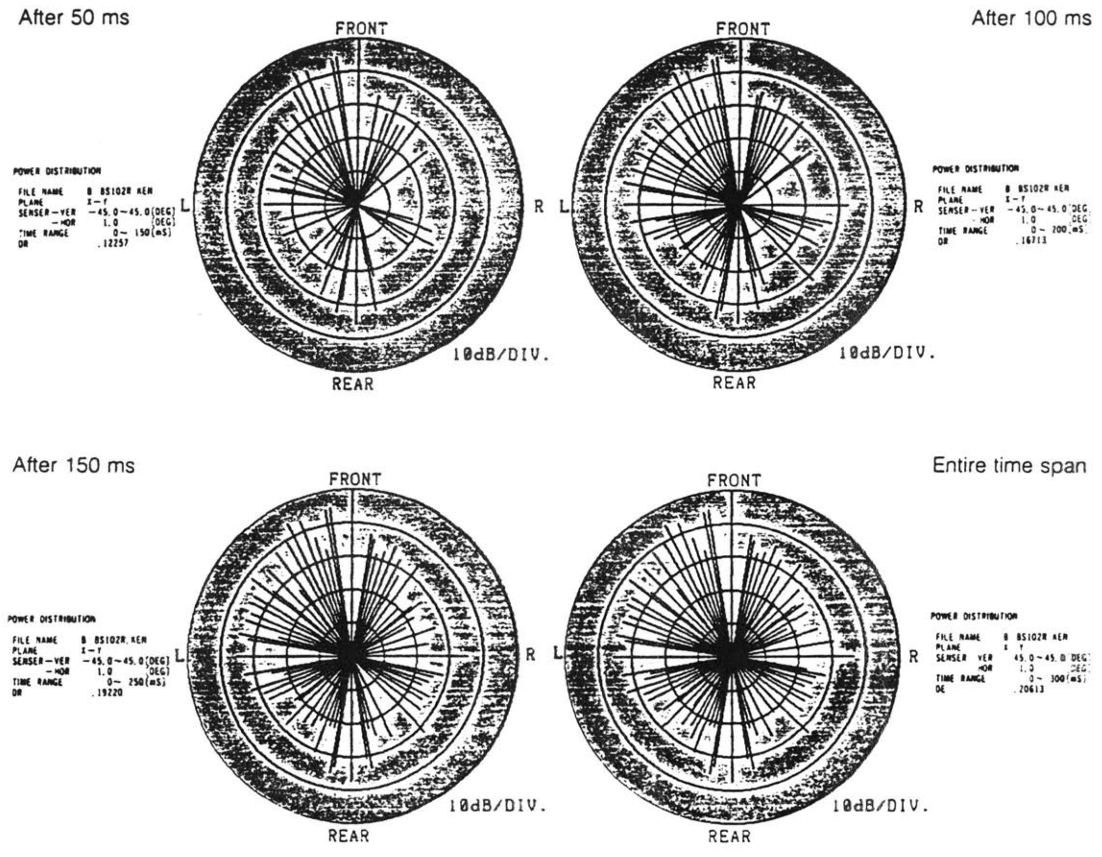


Figure 3-1: *Directivity response patterns showing the sound energy incident on a listener.*

Recent technological advances can now provide some precise analyses of the the performance of concert halls. The use of *quad miking* technology has promoted the use of radial graphs depicting the *directivity response pattern* of performance spaces. Radial directivity graphs plot the specular responses of a space from a particular listener location, much like the images produced here. I note here that there have been other computer acoustical models that can also generate directivity graph techniques, but these are accurate only in rooms with simple geometries [Borish 84]. An obvious shortcoming of these graphs is that they do not show the cause of a particular response.

The visual representation of the directivity response pattern gives an easier understanding of how the directions of the reflections coming in a certain length of time are distributed, and of the time-dependent changes of the distribution. [Yamaha 86]

Recently, much attention has been paid to the perception of spatial extension. The perception of reverberation depends on the volume of the room and the average absorption ratio, and the sense of extension depends on the magnitude of lateral reflections, and thus on the shape of the room [Yamaha 86, p. 9]. The perception of reverberation is related to the the magnitude of the lateral reflections and the delay of these reflections. There are many other parameters: the perception of extension is affected by such factors as the reflection delay time. The delay time between the arrival of the direct sound and the early reflections is referred to as the "initial delay gap" or IDG (see fig. 3-2).

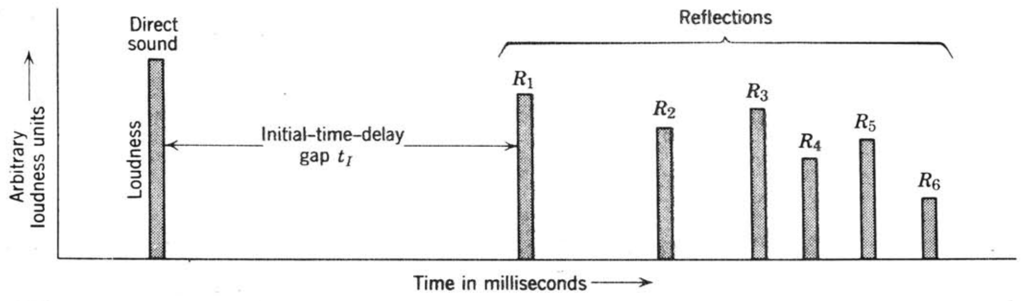


Figure 3-2: *Reflection patterns along the time axis. Sound arrives from the performer first, and after a gap, reflections from the walls, ceiling and other reflective surfaces arrive in rapid succession.*

The perception of extension will be heightened by higher lateral reflection volumes and larger IDG's. A particularly interesting fact is that the perception of extension bears almost no relation to reverberation or its length. This distinction between the reverberation time and lateral extension is an indication of our growing understanding of specific room properties.

The time sequence images provide a powerful means for analyzing the notion of extension and a variety of acoustical and visual representational issues. In tests involving a very simple lecture hall and more complex spaces described later in detail, the ray techniques used in synthesizing the sound field prove to be successful and function very much as originally planned. The accuracy and behavior of the reflections appear to be consistent and accountable in all of the test cases.

3.2 Boston Symphony Hall

I decided to synthesize the performance of Boston Symphony Hall and compare its results with what is known about it from other independent empirical observations. If the "good" qualities can be associated with the images, then the visual characteristics derived from this hall could then be taken as desired features to look for in images of other halls. Boston Symphony hall is very similar in size and scale to Chevalier auditorium. This makes it possible to correlate the behavior of the two halls based upon their images. Symphony hall, because of its relatively simple shape, is well suited to the the present ray tracer's capacity for geometric descriptions.

Leo Beranek in his book *Music, Acoustics and Architecture* gives a description Symphony Hall: "Symphony Hall, built in 1900, is known as the first hall designed on scientifically derived principles of acoustics. In some ways it is reminiscent of the Leipzig neues Gewandhaus; nevertheless it is quite different, primarily because it seats 2631 compared to 1560 in the Gewandhaus. [Beranek 62]" Beranek goes on to provide other acoustical evaluations of the hall by noted conductors and musicians as follows:

> The sound from Symphony Hall is clear, live, warm, brilliant and loud, without being overly loud. The hall responds immediately to an orchestra's efforts. The orchestral tone is balanced, and the ensemble is excellent.
>
> Bruno Walter said, "This is a fine hall, a very good hall. ... It seems very live. It is the most noble of American concert halls."

With one exception, the conductors who were polled rated this hall as the best in America and one of the three best in the world. Sir Adrian Boult wrote, "The ideal concert hall is obviously that into which you make a not very pleasant sound and the audience receives something that is quite beautiful...." Ten of the thirteen American and Canadian music critics rate it among the best in the world: "The sound is excellent; the hall has full reverberance; the orchestra is in good balance unless one is too near the stage on the main floor." "This is wonderful; it has the right loudness; music played in it is clear and clean."

There are a few negative features in Symphony Hall, as there are in every hall. The seats in rear corners under the overhangs of the balconies are shielded from the reverberant sound of the upper hall, and the reflected sound that reaches these seats from the soffits of the side balconies is somewhat unnatural. In the centers of the side balconies, echoes from the corners of the rear wall can be heard when staccato trumpet notes are played. Both blemishes involve very few seats. [Beranek 62]

The relatively simplistic evaluations above demonstrate the limited abilities of auditory assessments from well trained ears. The need for more scientific and precise evaluation becomes obvious if progress is be made in establishing cause and effect relationships.

A most striking feature of the Boston Symphony Hall images are the clear and strong arrival of specular reflections and continuity of the sound field as denoted by the diffuse reflections. Strong specular reflections arrive at very much the same time (fig. 3-6), which may account for the often cited special clarity of the hall. Most of the specular reflections also arrive within 60 to 70 milliseconds of the direct sound, providing good sound reinforcement for the classical music most commonly played there. The dispersion of the sound field without any marked discontinuities, as is visible by the diffuse properties, also contributes to the clarity by providing a smooth and consistent decay curve.

From the higher balcony locations, the proximity of the ceiling provides vertical reflections similar to that of the stage enclosure. The side balconies are shallow and high enough that they do not block the sound radiation to balcony locations. Had these side balconies been extended another three feet to

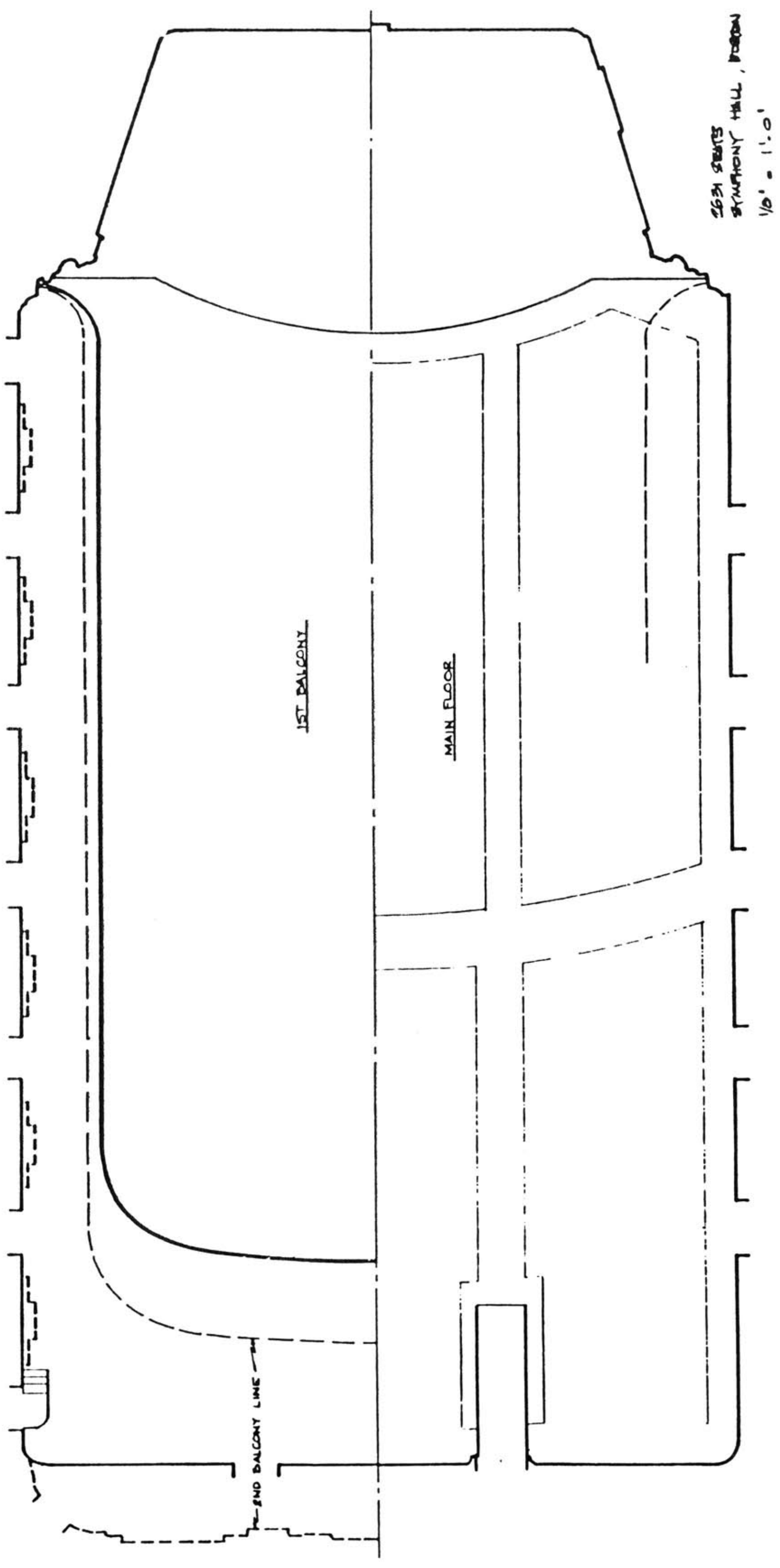


Figure 3-3: *Symphony Hall - First and Second floor half-plans.*

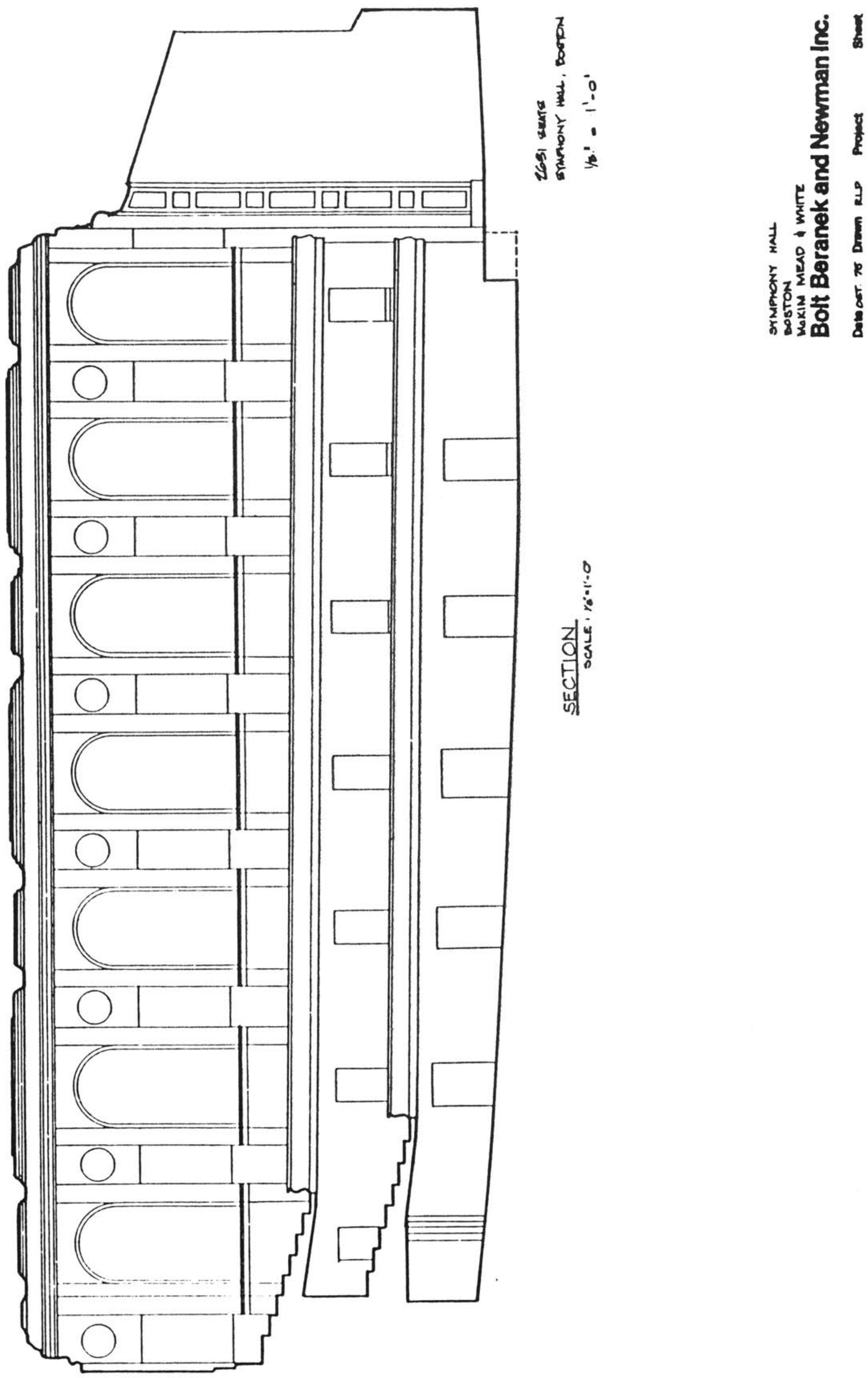


Figure 3-4: *Symphony Hall - Section.*

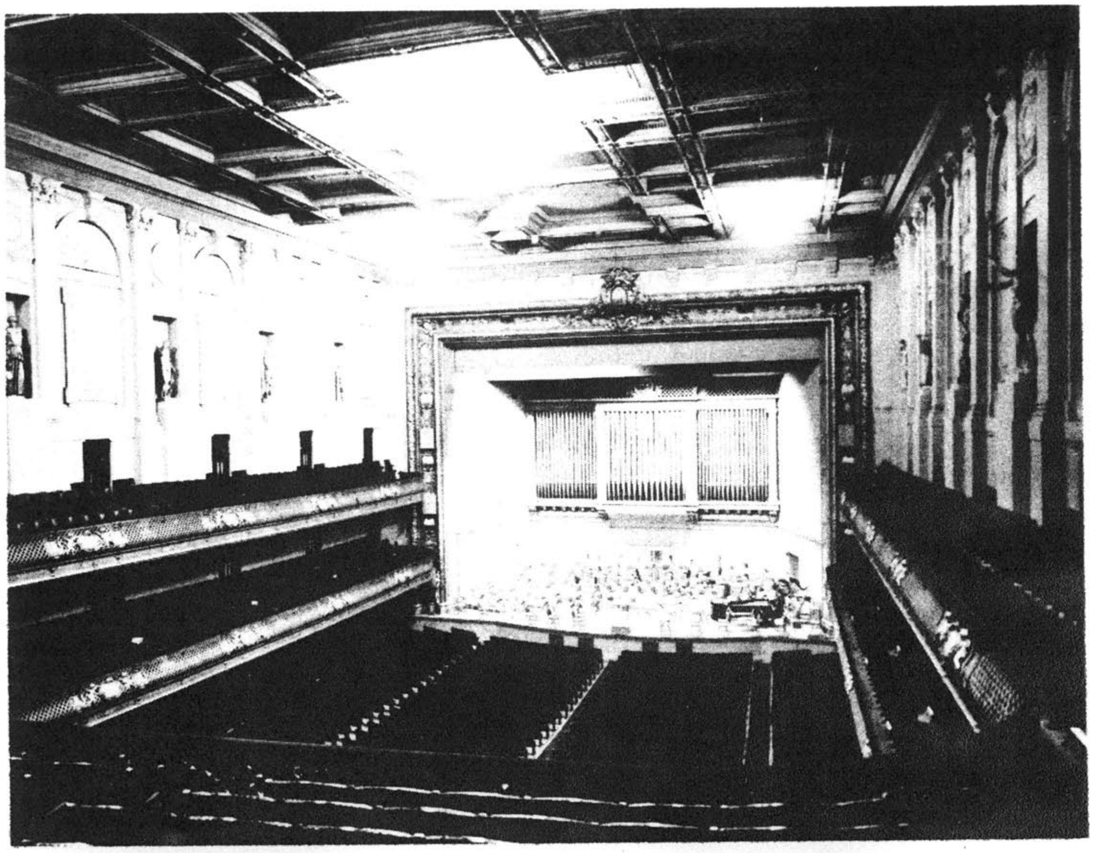
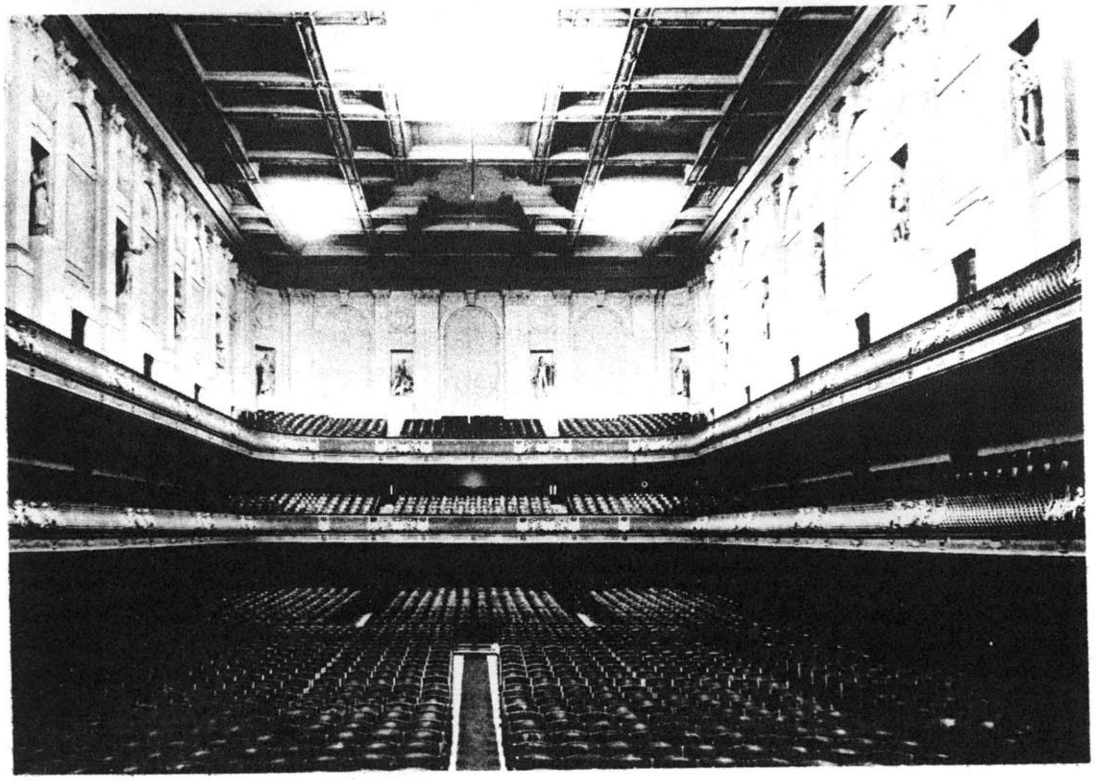



Figure 3-5: *Symphony Hall - Interior photo views.*

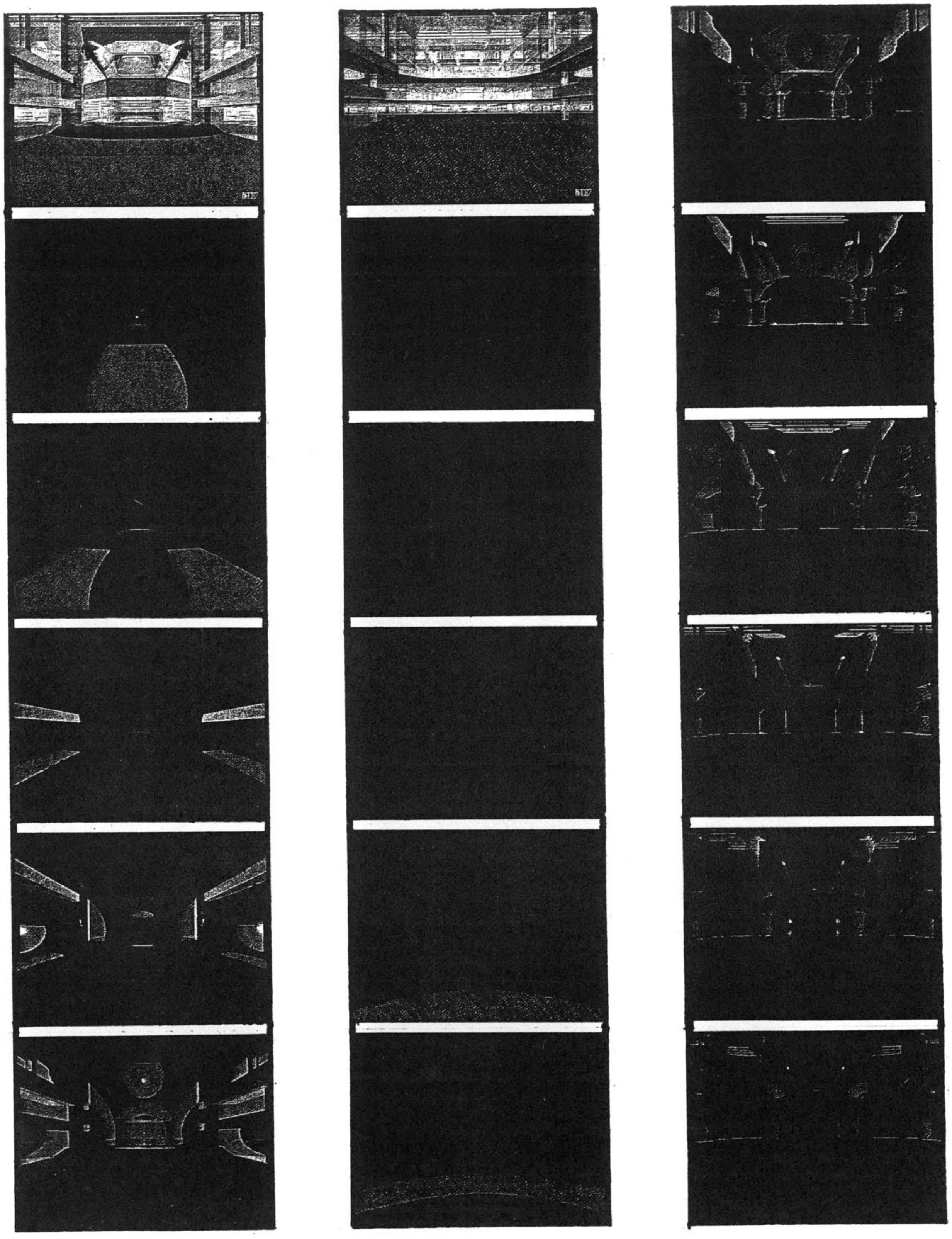


Figure 3-6: *Symphony Hall - a time sequence at 10 millisecond intervals with views in the front and back directions.*

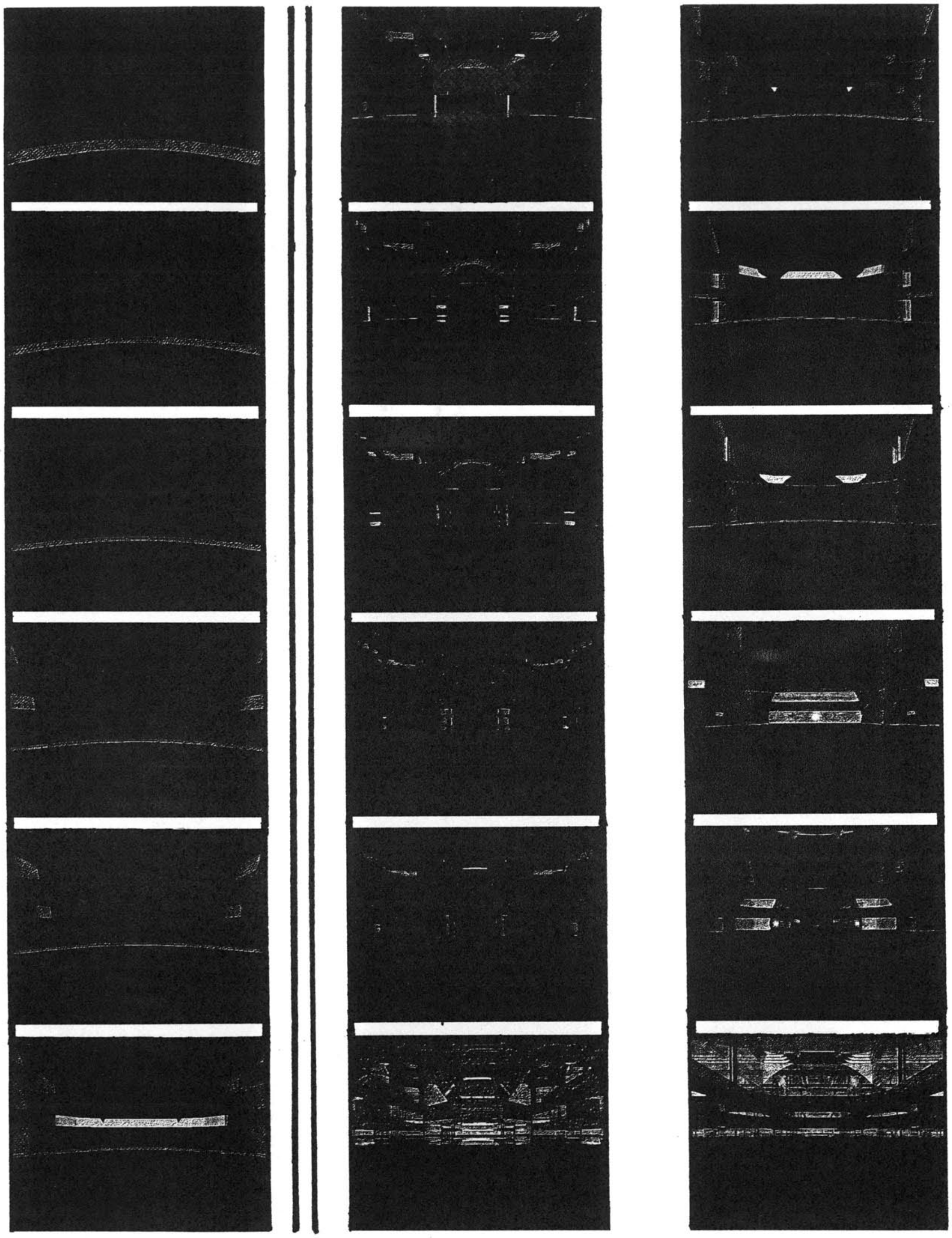


Figure 3-6, continued.

accommodate another row of seating for example, these lateral reflections would have been blocked (fig. 3-7).

From the less desirable seats in the audience far back under the balcony, the results can be seen as less desirable. The overhead balconies and the relatively low angle of incidence of the reflections on the undersides of the balconies tend to make overhead reflections non-existent. The recent low importance that is ascribed to vertical reflections in the perception of extension does not explain the lack of extension commonly ascribed to these seats. One explanation is in the simplification of room geometry and the modeling of a very empty hall, which does not contain people or seats. These lateral reflections should be greatly absorbed by the neighboring audience, creating a much more "dead" space that is consistent with the actual perceptions from those locations.

In the rear side corner of the audience (fig. 3-8) there is a very noticeable alternation between the right and left wall reflections without overhead reflections to moderate them. The result, I assume, would be a noticeable reduction in the intelligibility as these reflections arrive over a relatively long period and without any reinforcing grouping.

In directivity plots, the specular highlights of the images are depicted by the radial lines on the graphs. While there are no directivity plots available for Symphony Hall, the arrival times of the specular reflections can be seen to be consistent with directivity plots of other concert halls of similar size (fig. 3-1). Work is currently being done in plotting directivity graphs which will allow for accurate comparisons of the actual measured responses, in addition to the perspective images of the ray tracing program. This additional information will most easily be generated where additional information can be extracted from the ray trees still in memory. This information of the direction and lengths of the specular reflections can be quickly and accurately plotted.

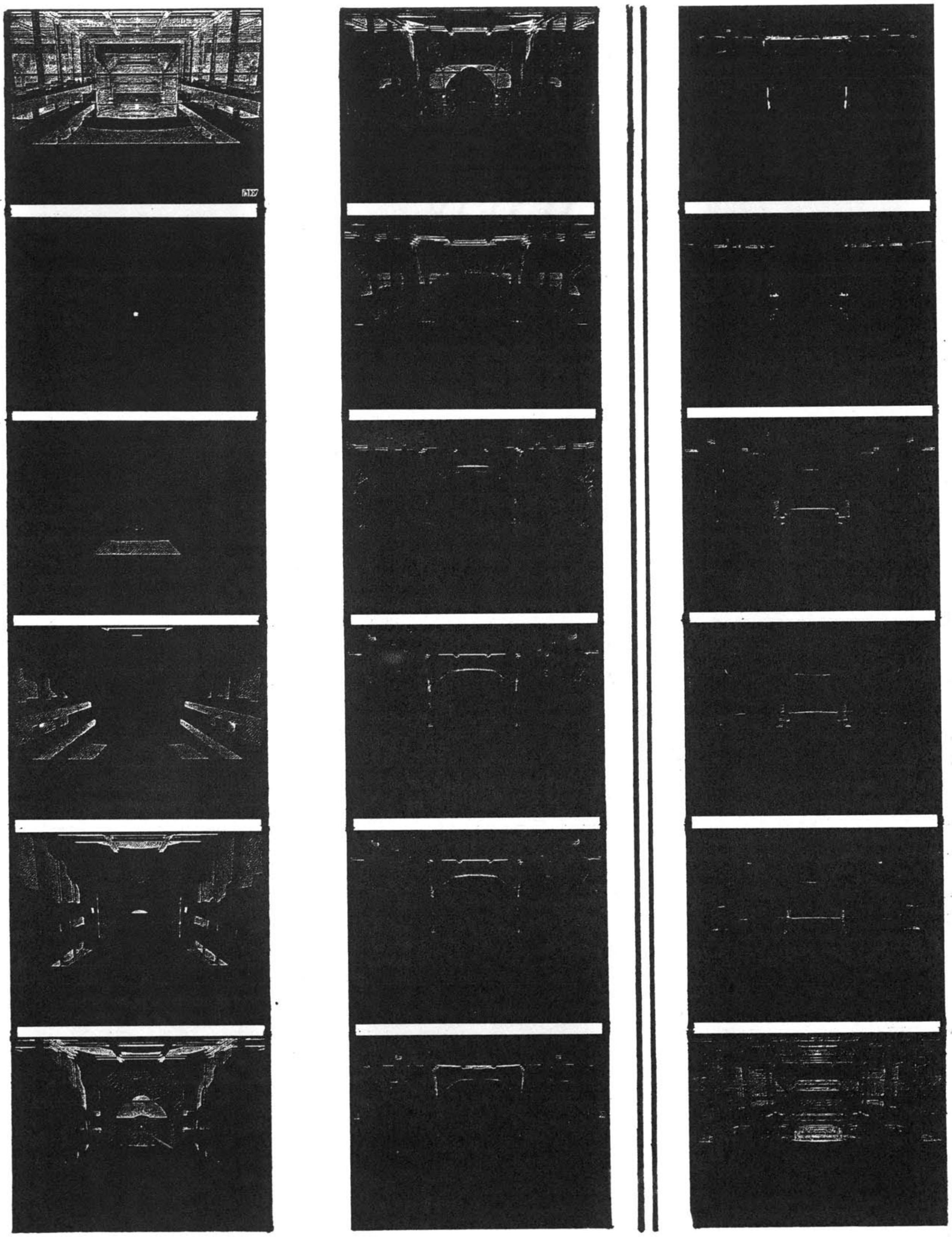


Figure 3-7: *Symphony Hall - Time sequence from balcony.*

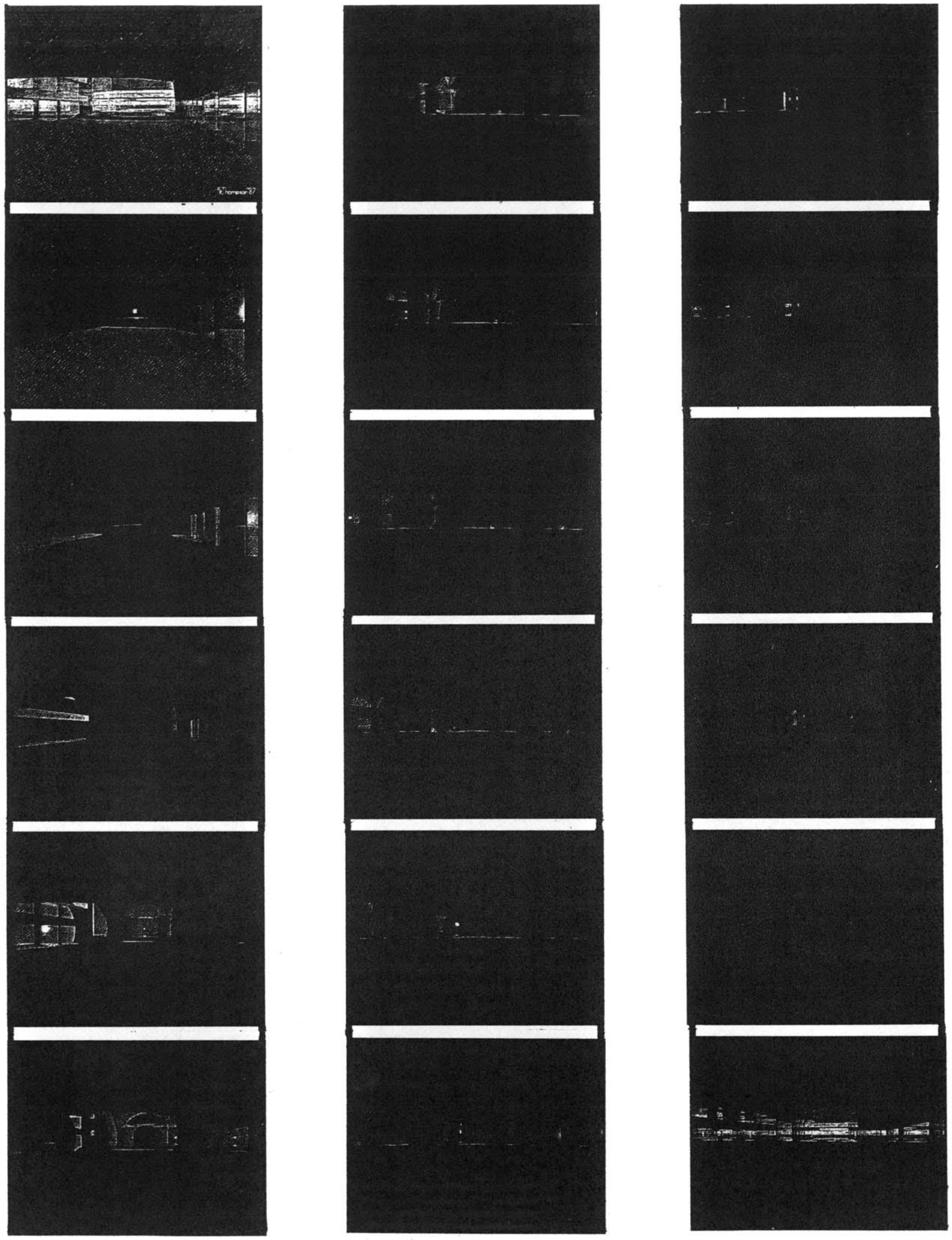


Figure 3-8: *Symphony Hall - Time intervals from the rear side corner.*

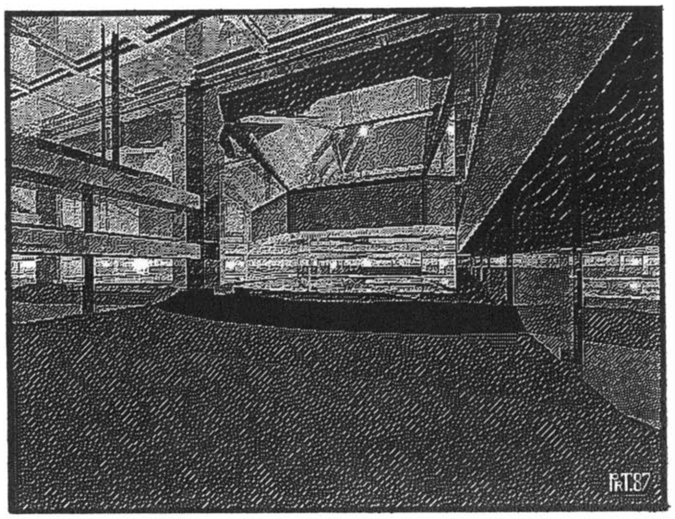


Figure 3-9: *Symphony Hall - Side from under Balcony.*

3.3 Variations on Symphony Hall

As a first test this program's ability to visualize the qualitative aspects of the acoustic behavior of the space, changes were made to Symphony Hall's design. Alterations to the designs were used to explore the changes in images and attempts were made to correlate them to presumed degradations and improvements. This provided a first notion of the magnitude or scale with which degradations or improvements in performance appeared correspondingly worse or better in the images. A particularly feature of Symphony Hall is its liveliness and clear sound which I assume is due in a large part to the clearly visible, temporally coincident reflections from the stage enclosure. To determine the effectiveness of the present configuration, changes were made to this one aspect of the hall's design.

The new configuration (fig. 3-10), was produced by moving out the sides, ceiling and the back of the enclosure to form a square. One notable result was the decrease in how much sound energy came from the enclosure area. Although this result is very much as expected, it was difficult to predict exactly how this would be represented in the images. The stage opening was noticeably darker, indicating that the sound was not being directed out to the listener, and as I would suspect, was being contained by the parallel walls.

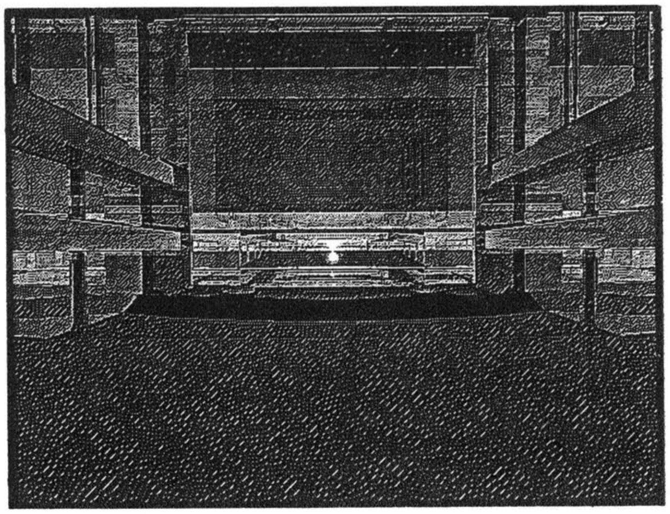


Figure 3-10: *Square stage enclosure modifications.*

The coffered ceiling and pillasters do not appear to have the pronounced effect which I had initially expected. It appears that these elements, because they are parallel to the walls and of the same materials, do not constitute significant reliefs. These protruding elements were visible only as the small shadows which they produced. They do not protrude enough from the walls to create a noticeable difference in reflections in terms of directions changes and attenuation. Figure 3-10 shows that there is little difference between images with the coffers and the pillasters removed.

A significant temporal displacement of sound is created by the balcony, which extends out approximately ten feet from the walls. The small frontal

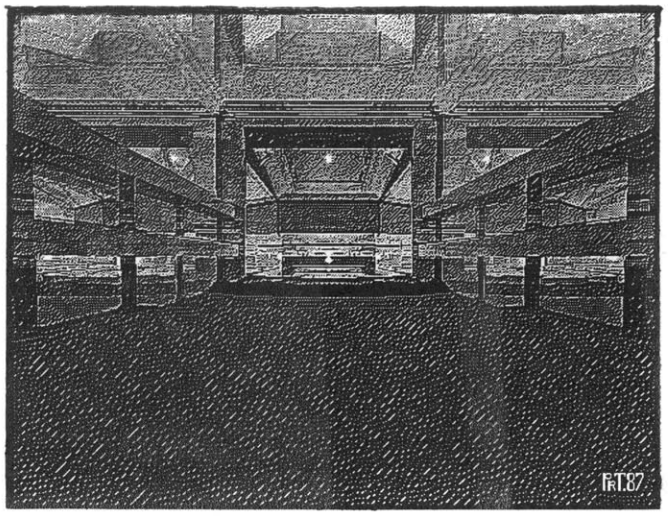


Figure 3-11: *A variation with smooth upper walls and ceiling.*

area and non-reflective latticework of the balcony appear to minimize this effect and do not produce significant reflections. This is especially true in the case of high frequency sound. In the case of low frequency sound, where the length of the wave is larger than the reflective element's dimension, the reflective property would be noticeably changed and is the subject of further research.

3.4 Conclusion

From what is known in psycho-acoustics and the consistent results of the program, I believe that the some of Boston Symphony Hall's best acoustical properties were visible. The most notable features are probably its clear and bright specular reflections from the stage enclosure and the side walls. The importance, for example, of the sloped stage enclosure becomes readily clear. The enclosure ceiling is quite effective in directing overhead sound to the middle audience area on the floor. This direct overhead reflection along with the reflections off of the sides walls are able to provide strong sound reinforcement with short IDG's for such a large space. Overhead reflections occurring at almost the same moment as the lateral stage reflections provide clarity and liveliness. The coffering on the ceiling seems to have less effect, especially at the high frequencies, than I would have expected.

Another observation gathered from the analysis of Symphony Hall and a differently scaled lecture room was that in this visual format the sense of extension was inversely related to the visual sense of direction. A long and narrow space has reflections that arrive more forward of the listener yet with smaller IDG's, increasing intelligibility even though the listener is more distant from the source. A wide space has lateral reflections arriving more at right angles to the listener and greater IDG's, reducing intelligibility although the source is closer. This dichotomy of the visual and auditory perceptions of the space are more noticeable in this time analysis and image format.

It would appear that as a visual criterion, a temporal grouping of the sound is a desired feature of clarity. In addition, the distribution of specular reflections provides a sense of spatial extension, and an overall image brightness indicates of a fairly live space.

Chapter 4

Design Contexts for the Program

Chevalier auditorium, named after a World War One hero from Medford, is a building of much interest to the community. The auditorium and gymnasium underneath, and after ten years of disuse, are the subject of a $2.5 million rehabilitation project.

The scope of the services and actual redesign as outlined by the city of Medford are:

> ... Said rehabilitation includes, but is not limited to, roof reconstruction, construction of air conditioning system, provision of handicap access facilities, lobby and auditorium restoration, gymnasium restoration, construction of a stage gallery, stage manager's gallery, and dressing rooms, relocation of stairway, and window replacement.[9]

The requirements do not specify the extent of any major structural changes which may alter the shape of the space. These original requests I believe are flexible enough to allow some significant changes. I have applied this acoustical methodology to evaluate Chevalier's existing design as an intermediate step towards a new design. This allows me to make the necessary change in the final redesign but stay within its original intentions.

Chevalier auditorium and Symphony Hall are very similar in scale and capacity, but differ significantly in plan. The most notable features of Chevalier auditorium are its curved walls and its very deep balcony. Symphony hall distributes its audience over three levels while Chevalier accommodates them all with just two levels. Oval plans as used in the auditorium

[9]Taken from the "*Request for Proposals:* Design services for the phase II rehabilitation of the Chevalier Auditorium/Gene Mack Gymnasium Facility."

are generally regarded as being acoustically problematic, being prone to poor energy distribution and a whispering gallery effect along the sides of the hall.

4.1 Chevalier Hall

The following description of Chevalier auditorium is taken from a 1985 appraiser's report of the building.[10]

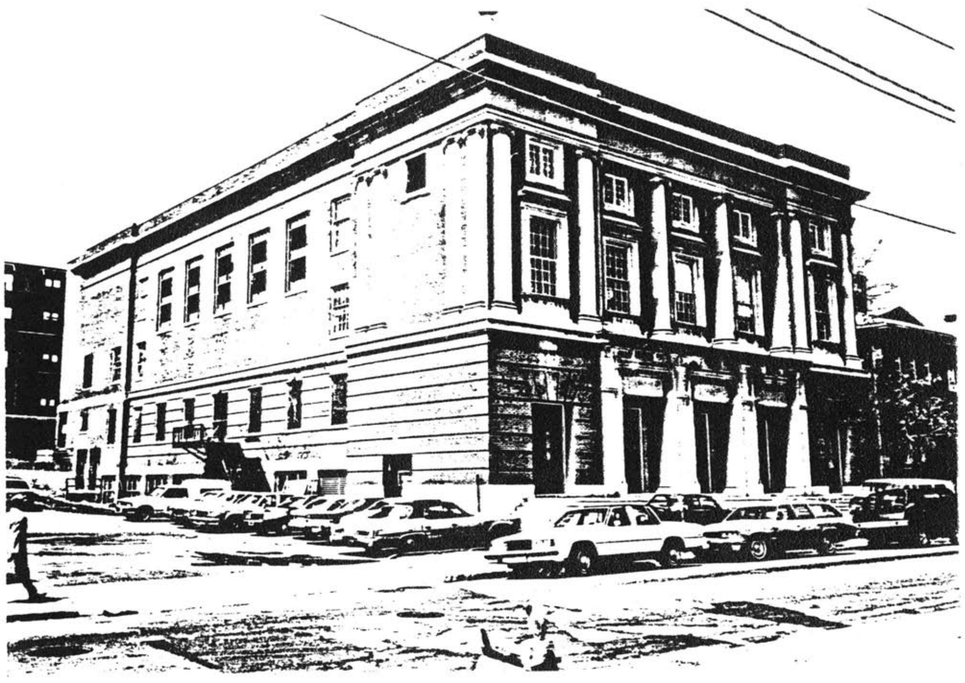


Figure 4-1: *Exterior front and side elevation photograph.*

"The structure is composed of solid brick, with steel and wood framing, rests upon a poured concrete foundation. The side and rear elevations are simple, with solid brick walls broken only by double-hung windows, pedestrian doors and one tail-gate loading door. The street elevation, however, presents a formal ordering of marble and sandstone column designs, three main and two

[10]The appraisers were T.H. Reenstierna & Sons of Arlignton, Mass., for the purpose of estimating the market value of the simple fee title as of May 1, 1985.

side access doors, and a stone portico with steps leading to the Forest Street sidewalk. The north side actually remains in contact with the main portion of the three-story High School building.

"Some of the connecting doors are still operable, but this building has been converted to condominiums, and the doors will be blocked after secondary egress arrangements for the dwelling units have been secured.

"Inside, the main floor of the theater contains approximately 1,489 solid veneered wood folding seats. The balcony contains an additional 700 seats for a total maximum seating capacity of approximately 2,189. The volume of the space is 375,000 cubic feet. The height of the theater averages approximately 30 feet. An appraisal analysis of the auditorium states that it could possibly be reproduced for $4,378,000 or roughly $2,000 per seat. Its market value as of May 1985 was $1,500,000. The total project costs including architectural/design/engineering, constructions costs cannot exceed two million dollars to restore and update the facility as a performing arts center."

Once the database with the surface descriptions had been entered, the evaluation of the preliminary design were made. Again, simplifications in the description of the hall were made, primarily in approximating its curved surfaces with polygons. The hall is very simply decorated and the approximations of the diffuse component are not as severe as in Symphony hall. From personal observations in visiting the hall, the curved wall did produce some of the problems associated with curved walls, namely that of discrete echoing and creep. A discreet echoing problem was noticeable when speaking at one side of the room on the balcony level. A whispering gallery effect was noticeable when a person would speak to another person at the other end of a side wall at the floor level. Also, a brief intelligibility test with a person speaking from the

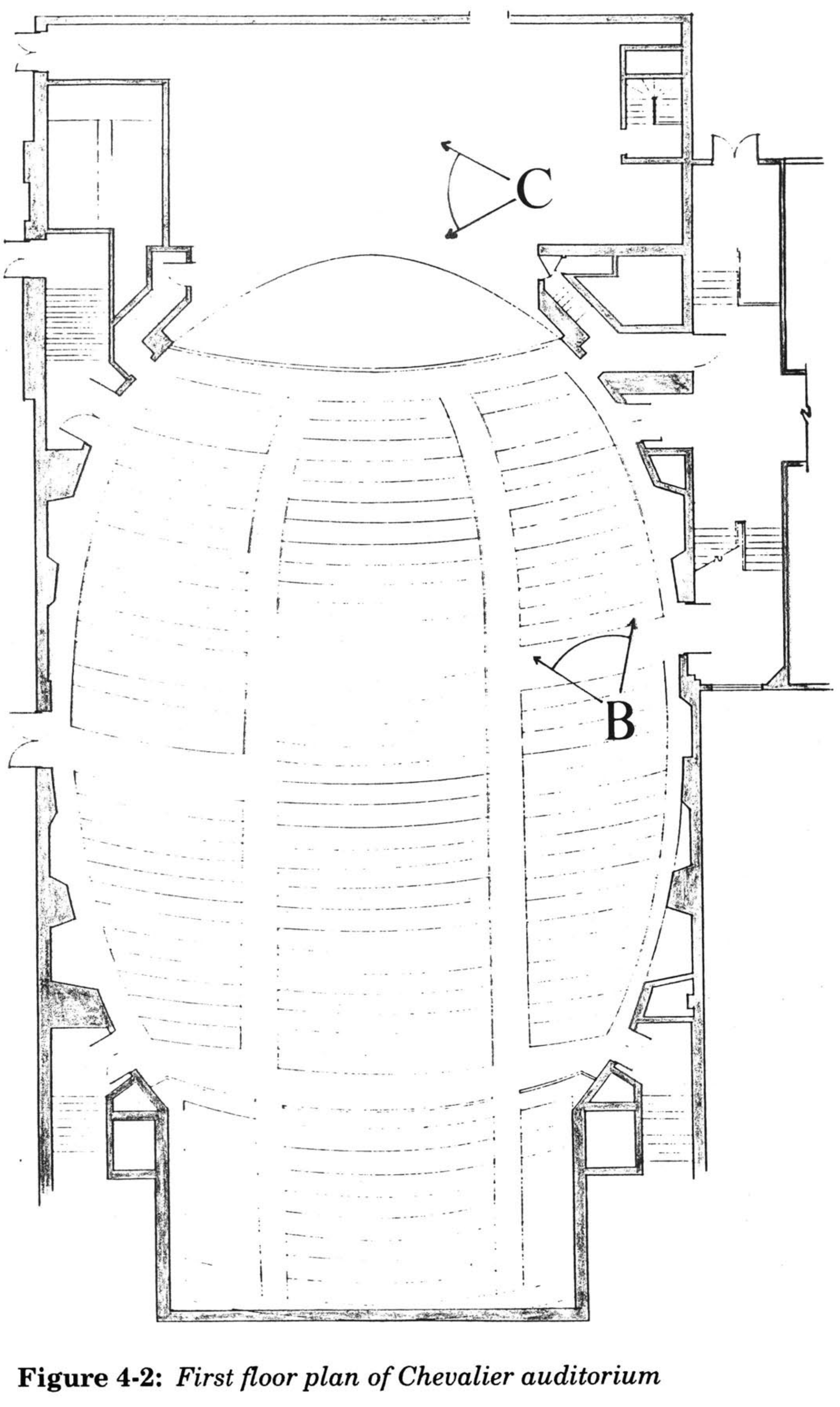


Figure 4-2: *First floor plan of Chevalier auditorium*

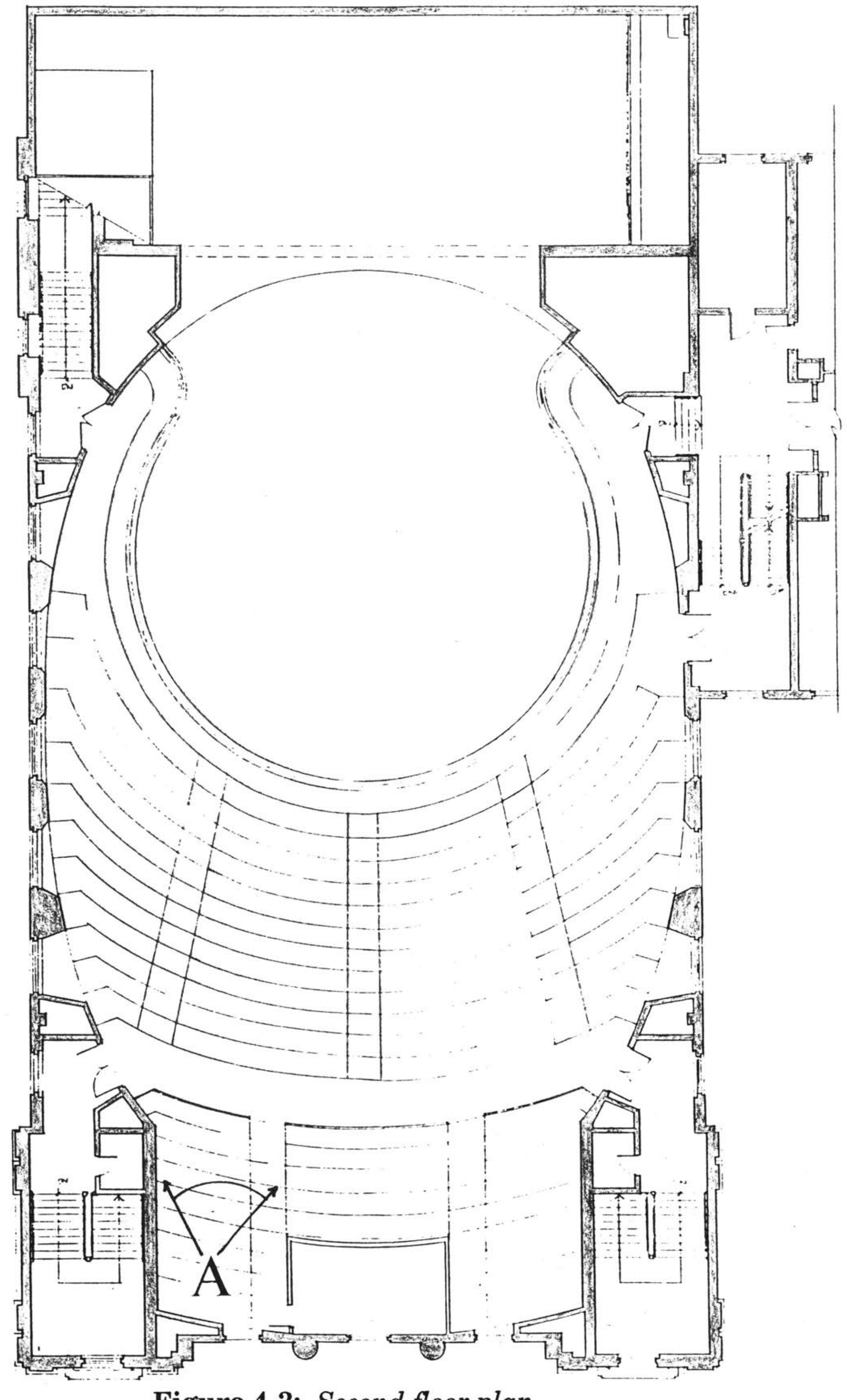


Figure 4-3: *Second floor plan*

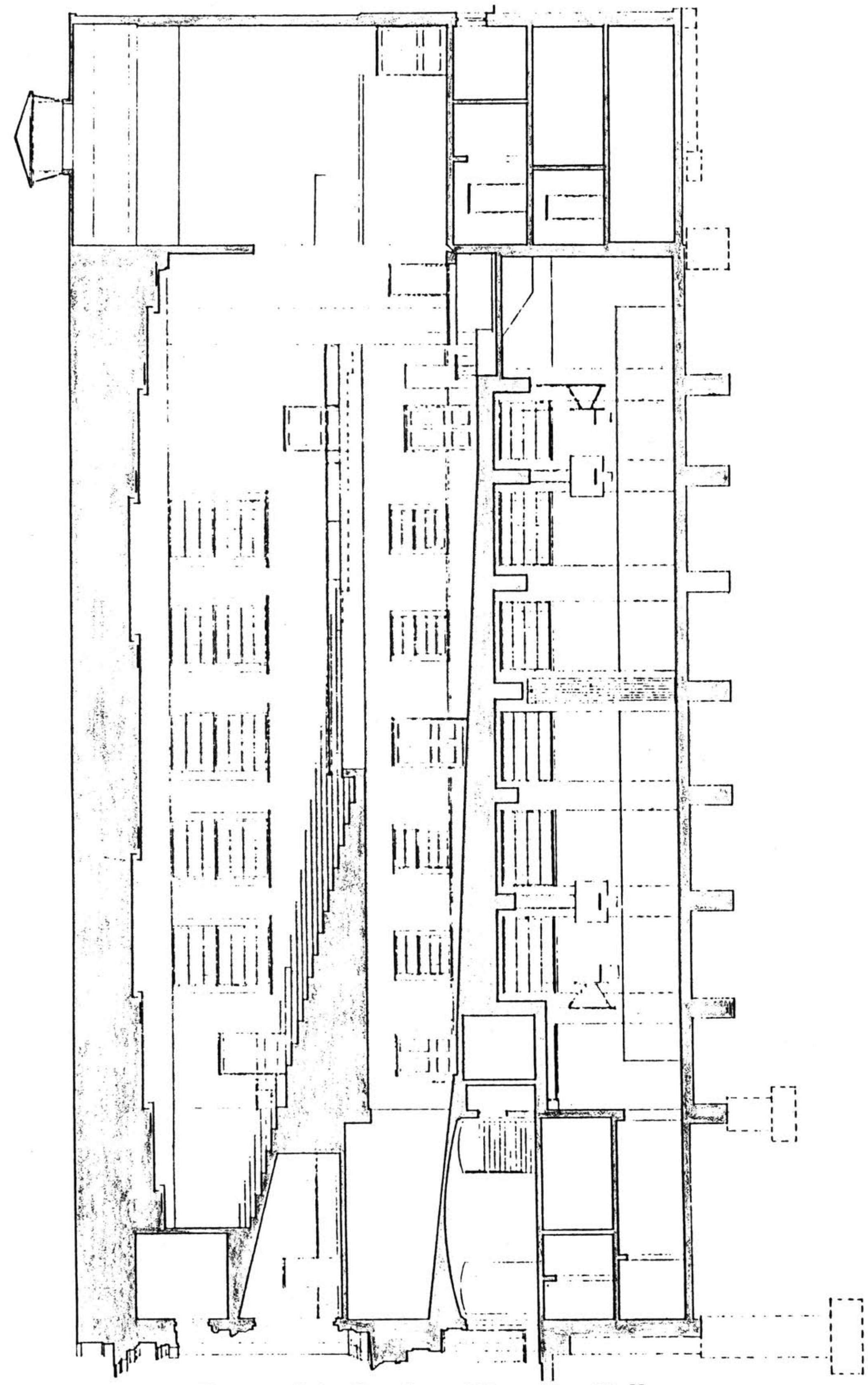
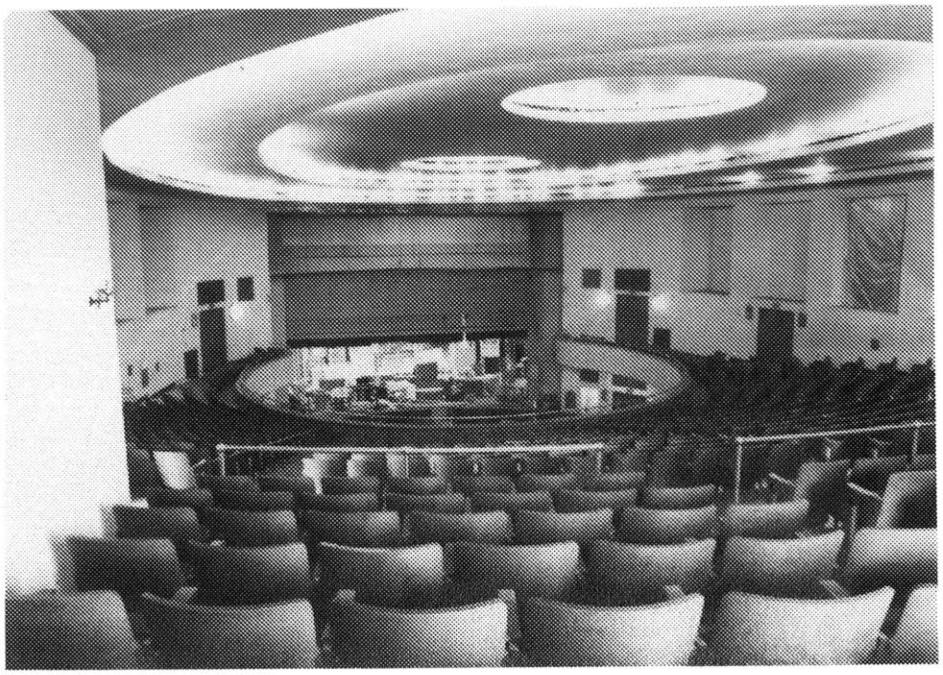


Figure 4-4: *Section - Chevalier Hall*



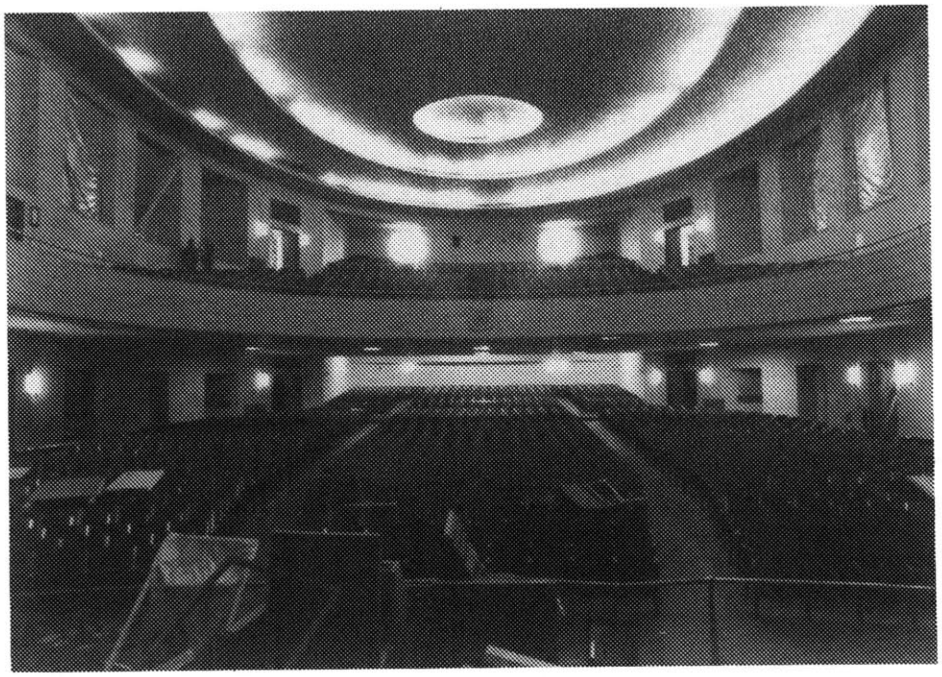
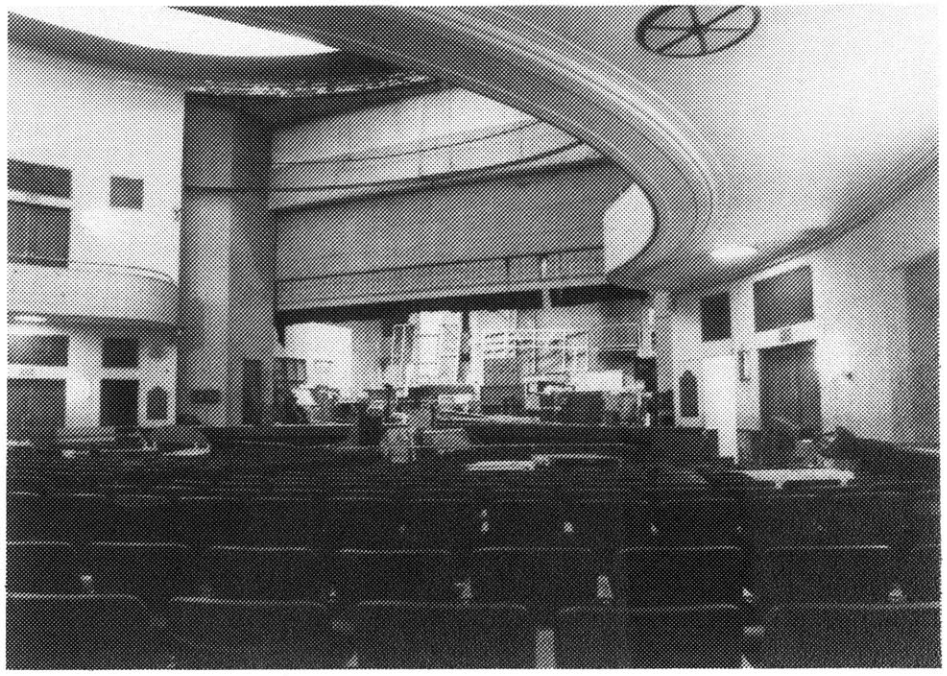


Figure 4-5: *Interior views - Chevalier auditorium*



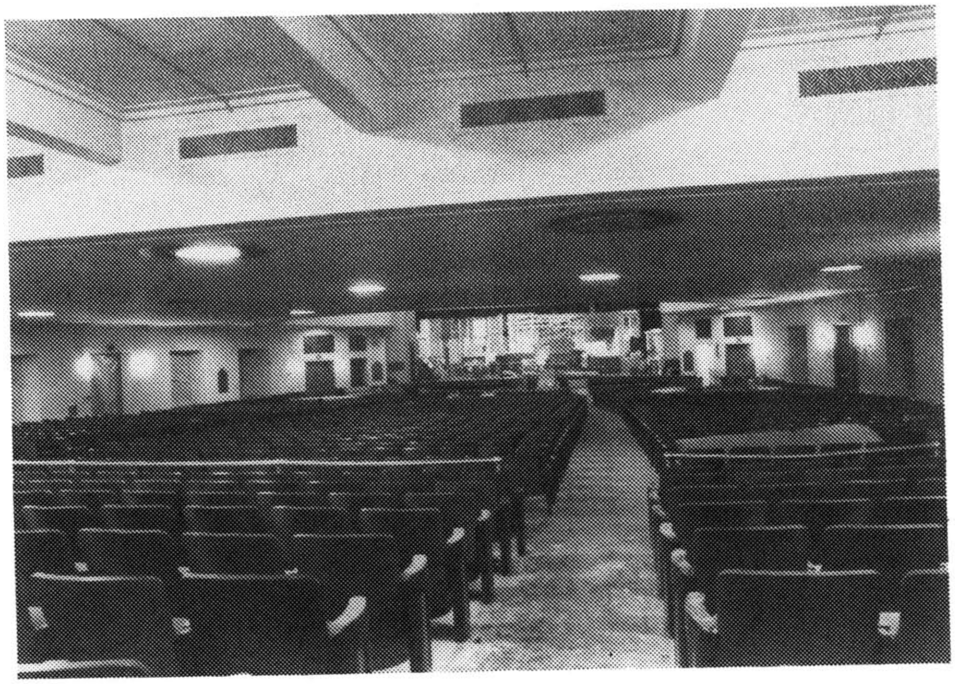


Figure 4-5, continued.

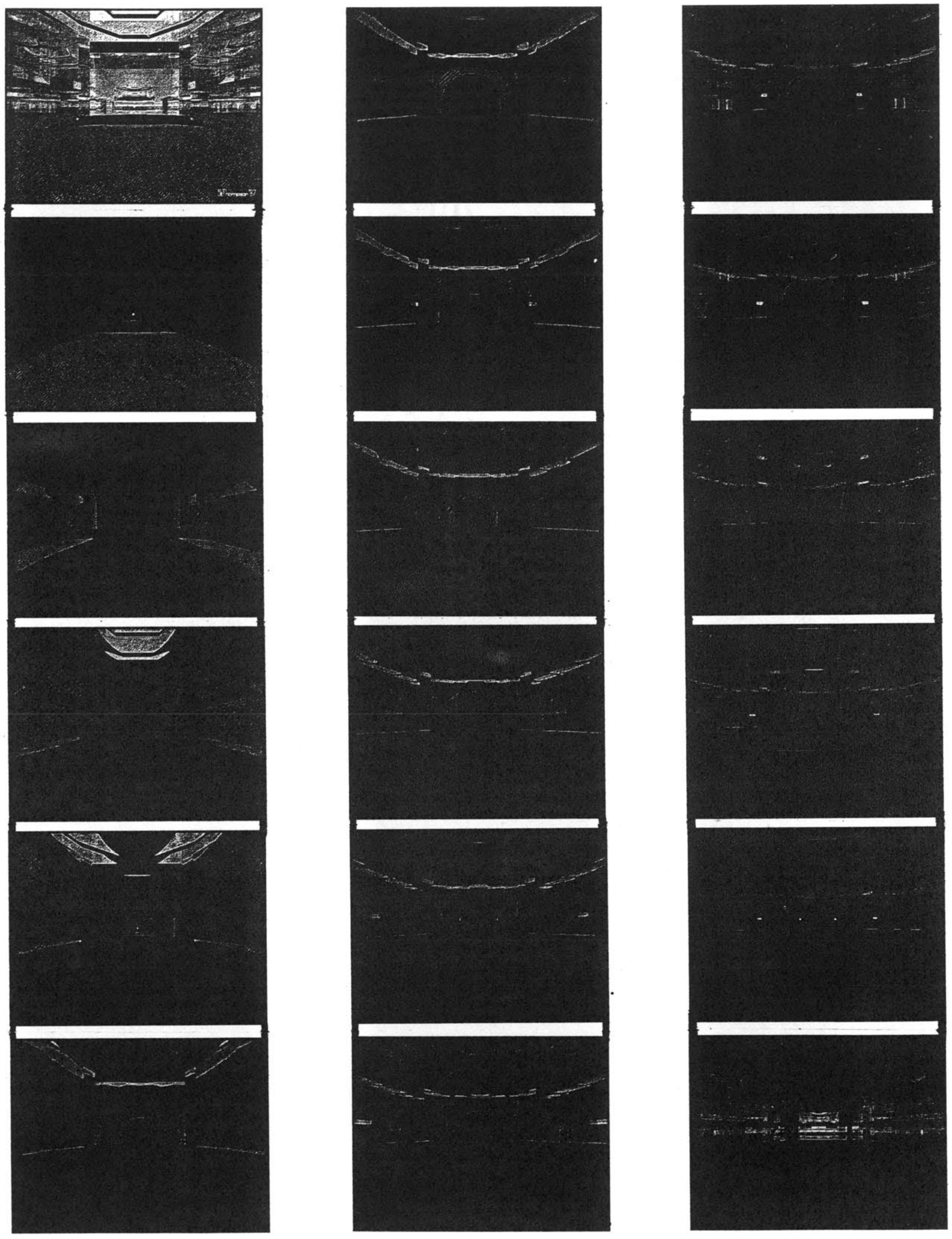


Figure 4-6: *Preliminary design - center seating analysis*

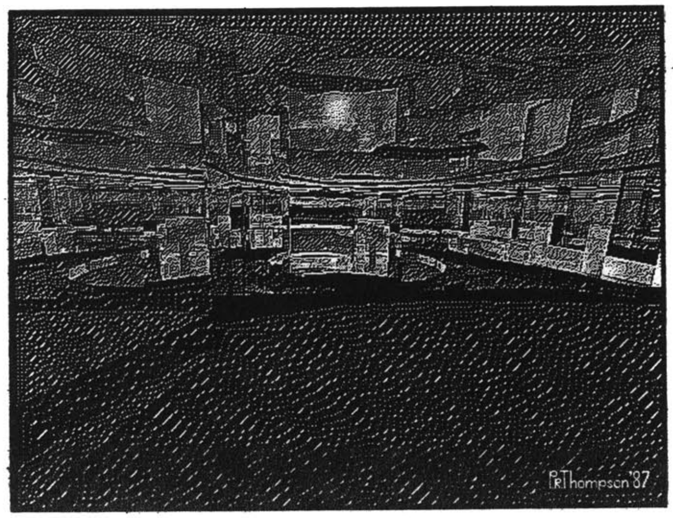
stage toward a seating location yielded poor results. It has been said that the rear seating areas provide the best intelligibility. This lower intelligibility was not directly attributable to any one specific feature of the hall, but was most likely a combination of features not properly reinforcing the intelligibility.

4.2 Analysis

From the first images is was obvious that the hall was not efficient in directing the sound to the audience. At first it was assumed that this was due to the lack of a stage enclosure. It was possible to see from the images that much of the sound was being trapped between the back stage wall and the proscenium. This was noticeable by the reverberant field as seen in center of figure 4-6. An interesting feature is the clear outline of the proscenium opening on the back stage wall. This reflected outline provided a clear view of the energy which was reflected out to the audience and the energy which became trapped in the stage area.

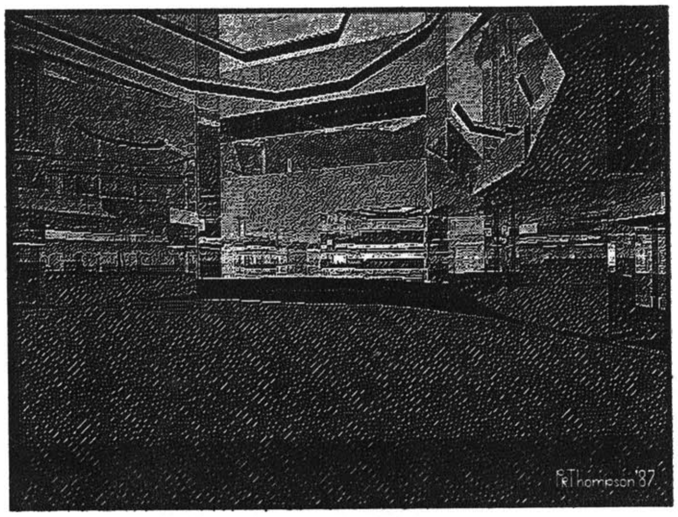
In the first images there is a noticeable amount of lateral energy coming from the main auditorium walls. When sitting in mid-audience, there are no specular and few diffuse reflections from the side walls. We see that the overall image of the space in the room is much darker than that of Symphony Hall.

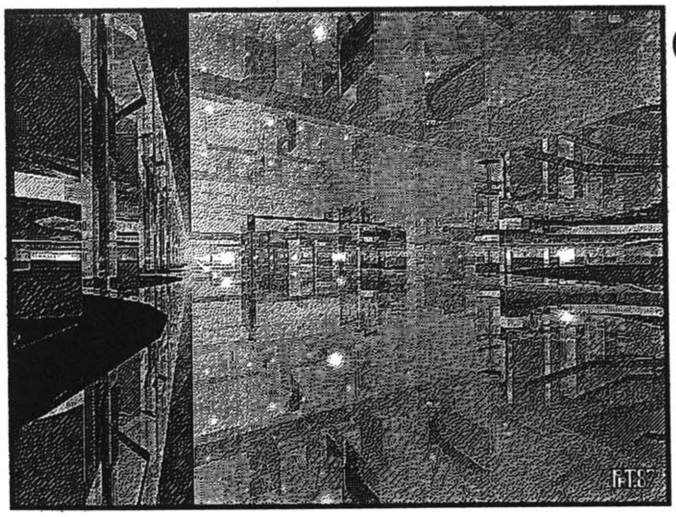
A first solution was to create a stage enclosure which would keep the sound from becoming trapped backstage, and to shape this enclosure to conform to the reflected outline on the back wall. An analysis was made of this new configuration. To my surprise, it produced relatively little change in the amount of sound reaching the audience. I suspect much of the energy was kept bouncing within the parallel walls of the enclosure, much like the modifications



A

B





C

Figure 4-7: *Chevalier Hall - Additional views. Note the top rear view with similar ray traced viewpoint in figure 4-5.*

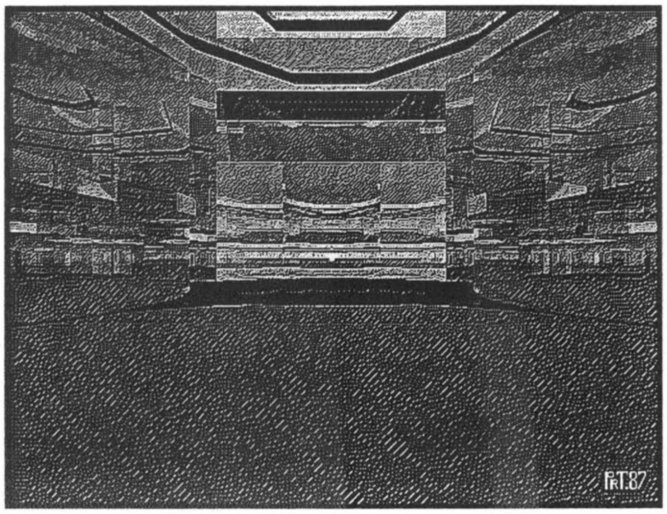


Figure 4-8: *Initial attempt at enclosing a stage*

to the Symphony hall enclosure. Although the square enclosure did prevent the intra-stage reverberations, the stage area was only slightly brighter. The rear reflection of the enclosure completely blended into the side walls, making them appear larger and without definition. This change did not affect the audience space which continued to appear dark and lacking reflections.

After looking at Symphony hall's enclosure, I decided to slope the walls and make the connection between the stage and audience larger to increase its sense of liveliness. The improvements from these relatively minor changes were immediately visible. This also produced a larger stage area with greater functionality. The modified sloped enclosure did create clear overhead and side reflections. However, this enclosure still requires some fine tuning, since there reflections did not quite arrive "in phase" as in the BSO geometry,

Again, however, this improved enclosure did not produce the desired responses within the auditorium. It seemed as though no amount of design changes from the stage area were sufficient to affect the rest of the auditorium. It was then obvious that fairly significant changes to the audience space itself

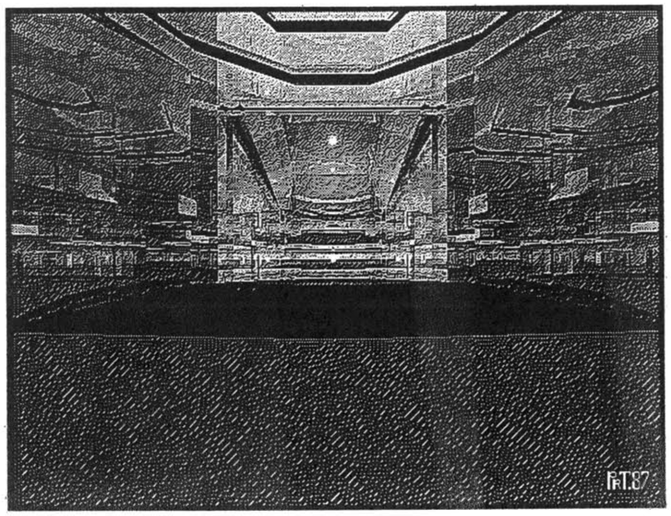


Figure 4-9: *Modified stage enclosure with the extended stage. Note the larger area in shadow in front of the stage.*

was needed. The rear seating areas under the balconies also appeared to be dark with little sound coming into them. Renderings from the rear wall indicate that the reflections that arrive at this time are very directional and arrive within a fairly short time span (fig. 4-10). This is consistent with the intelligibility and criticism often associated with these seats. There are no reflections that are indicative any sense of extension and liveness at these seats. But, these properties are indicative of good intelligibility, which is consistent with actual listener accounts.

In this deep balcony configuration, significant modifications are impossible within the scope of this rehabilitation project. Some form of amplification system will have to be used under balcony areas to create a sense of extension.[11]

[11]The consultants involved with the actual redesign plan to accomplish this through a distributed speaker system.

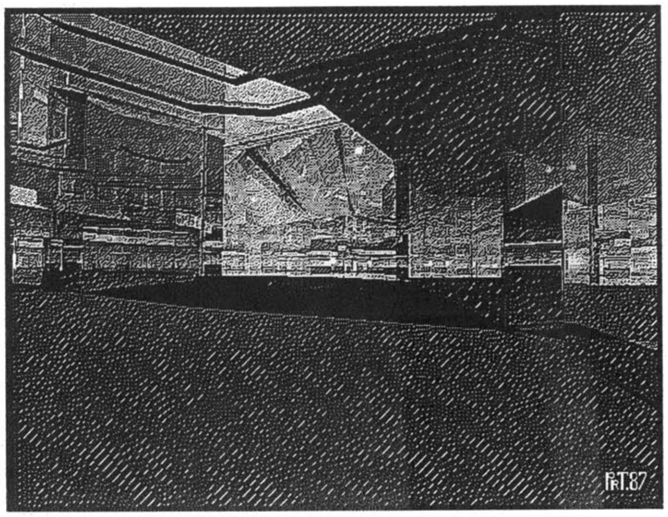


Figure 4-10: *View from the side audience under the balcony.*

4.3 Design Changes

The main problem is that the curved walls are not angled properly to direct reflections to center of the hall. As a design, the oval of the room and ceiling and circle of the balcony do not focus on the stage but rather just short of it. In this light, I decided to bring the stage out into the audience space as a slight thrust stage (fig. 4-9). The stage when not requiring an orchestra pit is then of a more traditional convex form that provides for a greater versatility in uses and relieves the narrow dimension of the back stage area. At first, it was also believed that this extension would help in reflecting more sound up to the balcony area. Subsequent analyses with this orchestra pit raised to the stage height, however, did not indicate an appreciable difference in reflected energy to the balcony level.

In order to most effectively create the lateral reflections that are needed in the audience space, I decided to make the front and side walls stepped and parallel as in figure 4-11. The most significant changes involved making the side walls step back in such a way as to maintain the oval of the original plan.

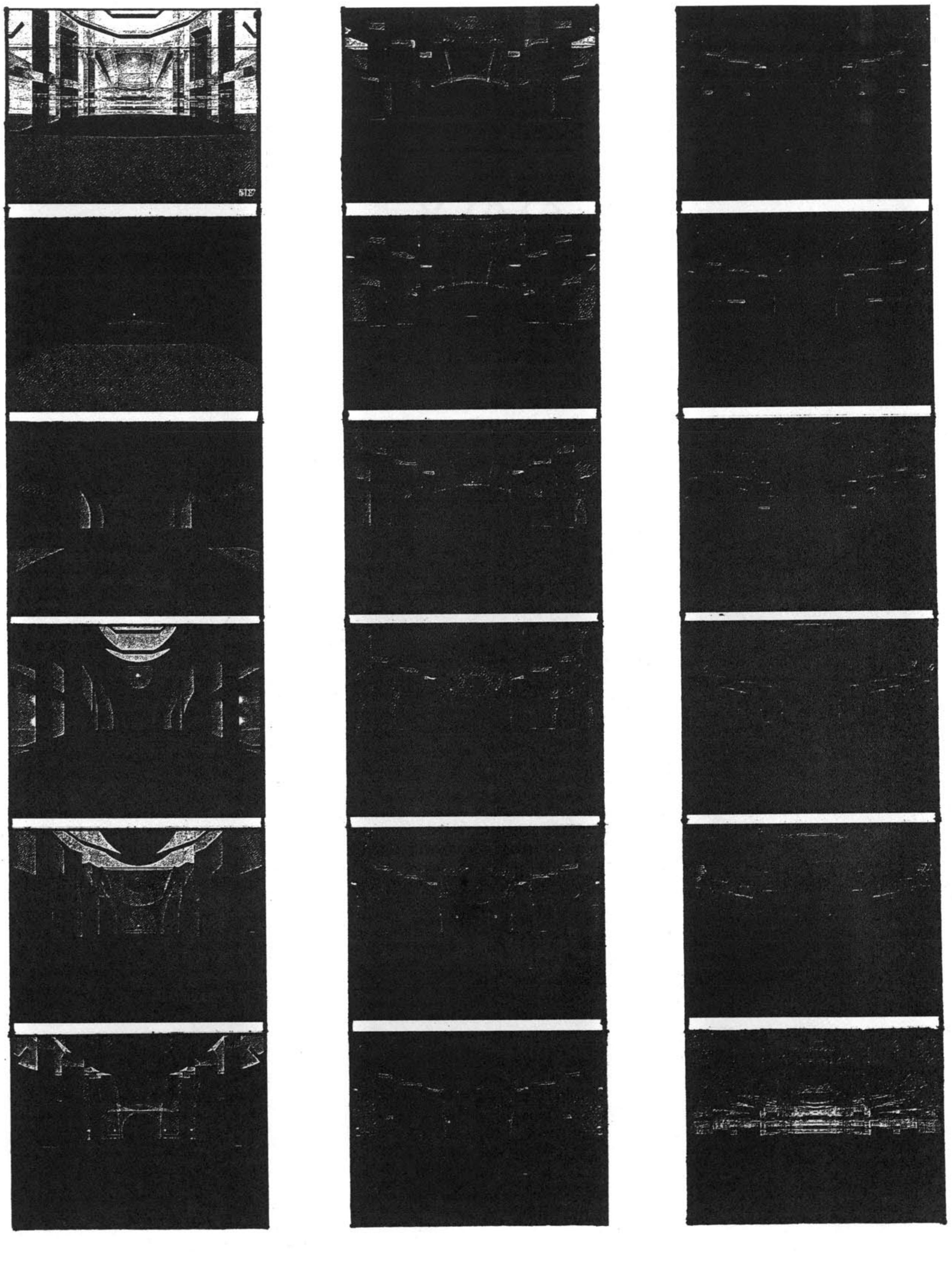
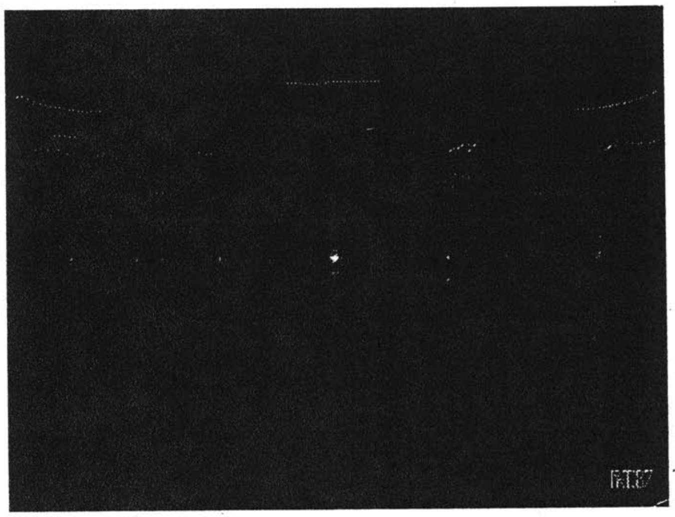


Figure 4-11: *Modifications with the new stepped walls.*

In testing these design modifications, it was apparent that large steps were made. The results immediately indicated that this was effective in delivering the sound to the mid-audience sections and preventing the sound from creeping along the side walls. The small change in wall direction was very effective in bringing more diffuse and specular sound energy to the center audience sections.



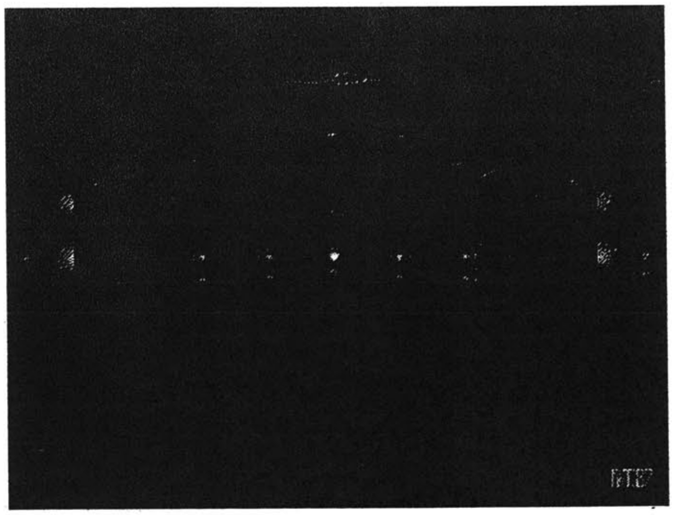


Figure 4-12: *Cumulative specular reflections only. Before and after modifications.*

The rough stepping nature did produce visibly dark vertical areas of shadows. The issue of whether these shadows are audibly perceptible remains to be tested. Shadow and shade areas are the products of modeling the diffuse components, of which the significance remains to be determined. Again, most acoustical models and research are primarily concerned with the specular components. There is an obvious and significant improvement in the sound properties of the space when only the specular reflections are rendered (fig. 4-12), as there are now reflections arriving where previously there were none. Comparisons were made of the differences between the original and modified design by summing the intensities of the images with and without diffuse components. When diffuse sound components are taken into account, the advantages are not so clear. The increase in intensity is offset by the dark shadow areas, making the image only moderately brighter overall. The overall intensity sums of the two different configurations yields the following intensities[12]:

	before	after	% change
Specular only:	97,545	134,972	+38.37
Diffuse and Specular:	10,361,892	11,660,133	+12.53
Symphony Hall:	-	12,060,669	-

I also attempted to increase the liveliness of the space by making the stage less detached from the audience (fig. 4-11). I enlarged the stage's floor area and the opening of proscenium between the auditorium and stage, by making eliminating some of the back stage rooms. This made the sound less directional and increased liveliness.

[12]Values are in pixel units.

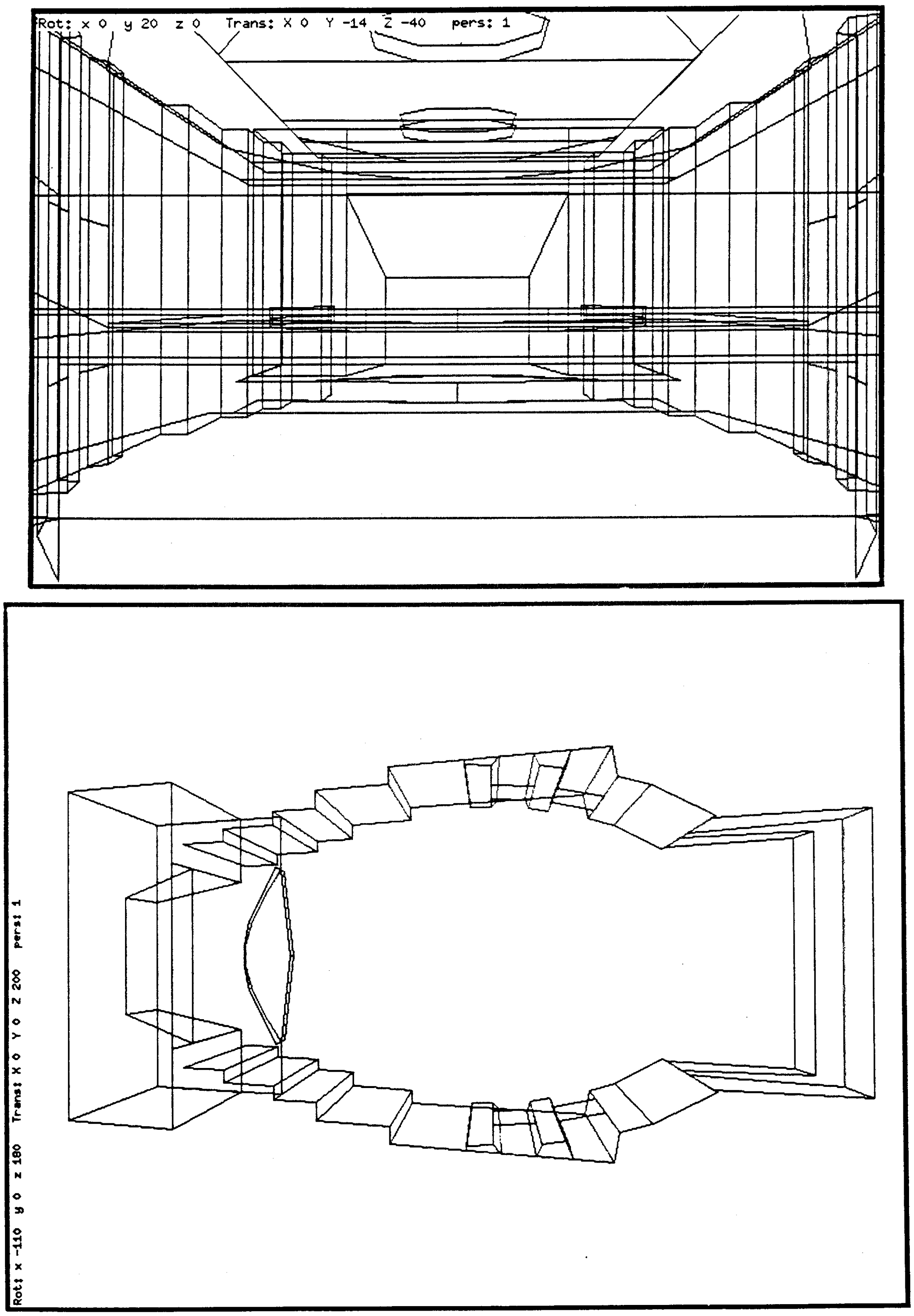
4.4 Empirical Experience

The experience of using this program has brought several points to light. This program appears to be very effective in pointing out many potential acoustical pitfalls in design. Such a visual analysis by nature of its comprehensiveness is very useful in keeping the designer from overlooking specific weaknesses or strengths which might otherwise go undetected. The graphic nature of information also makes it difficult to misinterpret properties.

Designers today, like the designer of Chevalier Hall, continue to design plans without a full understanding of the non-visual consequences. Even to the acoustical engineer of the actual project, the possibility of overlooking some aspects of the design exists.

Manual ray tracing, because of its speed and ease of use, will not be eliminated in the preliminary sketch phases of a design. It is a quick and general enough to express a concept, but it is ineffective in providing a precise evaluation. The curved walls make approximations difficult and the three dimensional analyses are not feasible.

Using this program to analyze an oval shaped space clearly shows the issues which plague such geometries. It is very difficult to visually determine where reflections will fall. First, the perception of the shallow curve of the walls is deceptive when standing in the space. The engineer here assumed that the curve was not a problem for getting reflections to the center audience sections. Second, it is easy to ascribe properties to features of a design without testing them. In this case it was believed that the deep window recesses were sufficient to diffuse the sound into the space. Neither of these predictions are consistent with the program results. The windows are either too deep or too oblique to provide an effective surface area to the audience.



Rot: x 0 y 20 z 0 Trans: X 0 Y -14 Z -40 pers: 1

Rot: x -110 y 0 z 180 Trans: X 0 Y 0 Z 200 pers: 1

Figure 4-13: *Perspective views allowing for preliminary and selective viewing of the Chevalier Hall database.*

In making modifications to the design, I found it was desirable to work as much as possible within this computer environment. Unlike a traditional architectural setting where most of the work is done on paper, it is easiest to make most changes directly on the computer and test them immediately. It is not practical to switch and incorporate changes between paper and electronic media. A three-dimensional viewing program proved invaluable for quickly and graphically previewing the database (fig. 4-13). Although the viewing program did not provide editing capabilities, it provided perspective and plan views and selective views of the database. A graphics editor would have provided a much faster interface for creating the database. From this I have found that it is desirable to work and make changes directly in the three dimensional database and visually confirm the changes in a wire frame drawing.

Because of this analysis tool, I believe that the problems of the original design are made very clear. The comprehensiveness of the information presented has allowed me to work more effectively in the redesign of Chevalier Hall. Ultimately, the program made the actual properties of this design much clearer.

Chapter 5

An Idealized Environment

The program developed here presents a significant achievement in providing a new means for understanding acoustics. Only one year ago, when the ideas of ray tracing were conceived, there was uncertainty as to success of the recursive ray tracing model used in producing the pictorial results that have been presented here. In the time that these original concepts have been applied, the potential applications of this program have far exceeded its original expectations as a simple visualizing tool. One of the primary benefits of the use of images has been the development of a visual interface for its effective integration into the design process. The long-range potential of these acoustical recursive ray concepts has grown enormously. However, I will touch on some of the more immediate needs of future applications.

For this thesis, these steps were performed manually. The interpretation and the exact extent to which data can be derived from the parametric description will have to be further explored along with other interactive tools. Access to a graphics editor will not take place within the time frame of this thesis. Hopefully, what will be set out is an abstract model of the design process into which this fits. There will be some notion as to which decision should be manual and which should be automatic, and the interface that they require.

The visualizing of acoustics is not only effective within the architectural design process, it is also powerful as an educational tool. The original goal of these images was to increase our understanding of acoustics by *seeing* its effects on a design. It does not rely on a tacit knowledge of the symbolic nature of traditional conventions and denotations.

5.1 Immediate Improvements

Future developments will concentrate on display and interface techniques using the *X Window System*[13]. *X* is a display interface protocol that provides a standard for displaying graphics across a wide variety of computer architectures. It provides a network transparent windowing system which runs under many operating systems, further increasing portability and display possibilities. This allows for images to be generated at one location and be displayed over a network at another location without concern as to display types and machine systems.

What does exist are methods to interact with the images on the computer's screen and mouse. When the mouse is pointed to a specific location on an image it is possible to get a numeric breakdown and parametric description of how a particular intensity came to be. This parametric description involves the paths of rays in the tree, the surfaces they hit, their respective intensities, and cumulative intensities up the tree. Additionally, most of the interactions should take place by means of a pointer on the images or the database through a graphic editor. Hopefully, in the future, the database will be manipulated only through a graphics editor, such as a CAD program or a higher level artificial intelligence system. To date neither of these utilities exist in a package or individually and any database manipulations will have to be done manually.

The present computer model is only preliminary. The accuracy of the images is still subject to much field testing. The acoustical intensity model and

[13]The X Window System is a trademark of MIT

the ray tracer are both deficient in the specific areas of accuracy and speed, respectively. The intensity model needs work in extending its accuracy over a variety of factors, such as issues of phase, incidence angle on absorptive surfaces, transparent objects, and frequency distribution.

The ray tracing algorithm used to date is simple but very slow. Formation of the ray tree for each pixel of the screen is by far the slowest part of the program. Renderings require anywhere from a few minutes to a full day[14] in the case of the more complex spatial description for Boston Symphony Hall. The subdivision of the ray trees into time segments does not add significant processing time.

Much work is being done in computer graphics to find more efficient ray tracing algorithms. A promising technique involves the spatial sorting of objects in order to eliminate having to run through the entire object list for each ray-surface intersection test [Fujimoto 85, Kay 86]. Similar spatial sorting, or octree, algorithms have been implemented with extremely complex scenes being processed in the same time other conventional computer graphic methods. The times are relatively independent of the number of objects. Scenes containing thousands of polygons have been traced in approximately two hours on a VAX[15] 11/750 [Fujimoto 86].

Another approach is to use more powerful machines. Because each pixel of an image is calculated independently, ray tracing is especially vectorizable and well suited to parallel processing. This ray tracer has been recently ported

[14]Computed on a Digital Equipment Corp. VS2 workstation.

[15]Vax is a trademark of the the Digital Equipment Corporation.

to a Butterfly[16] parallel processing computer. No benchmarks have been made to date; however, I would project processing times of two hours or faster using a 128 processor machine for buildings much more detailed then Symphony Hall.

Parallel processing computers also have an enormous amount of memory which should be able to simultaneously store all of the time related information from the ray trees. This will make for a much finer and wider range of time analyses to explored. One goal is to combine a faster algorithm and a faster machine, which could produce near real time rendering performance.

Work is currently being done is representing information in other numeric and graphics formats. Most programs and models to date have been designed to investigate one or a few particular aspects of acoustics. This program, due to its fairly comprehensive model of the environment, is able to extract data covering aspects of acoustics which are usually the product of several individual programs.

As more interactive features of the program are developed, the X display protocols will become an integral part of the program. Through the *X* windowing system it is possible to display a variety of information other than the images produced to date. Multiple windows can provide the means for simultaneously viewing additional acoustical information, and and greater facility for inputing sound parameters of source directivity and power output. Given the time it takes to run this program it would be useful to preview the input parameters through wire frame drawings. Additional windows, displayed next to the images, can provide more quantitative information through radial and

[16]The Butterfly is a trademark of BBN Advanced Computers, Inc. and a subsidiary of Bolt, Beranek and Newman Inc..

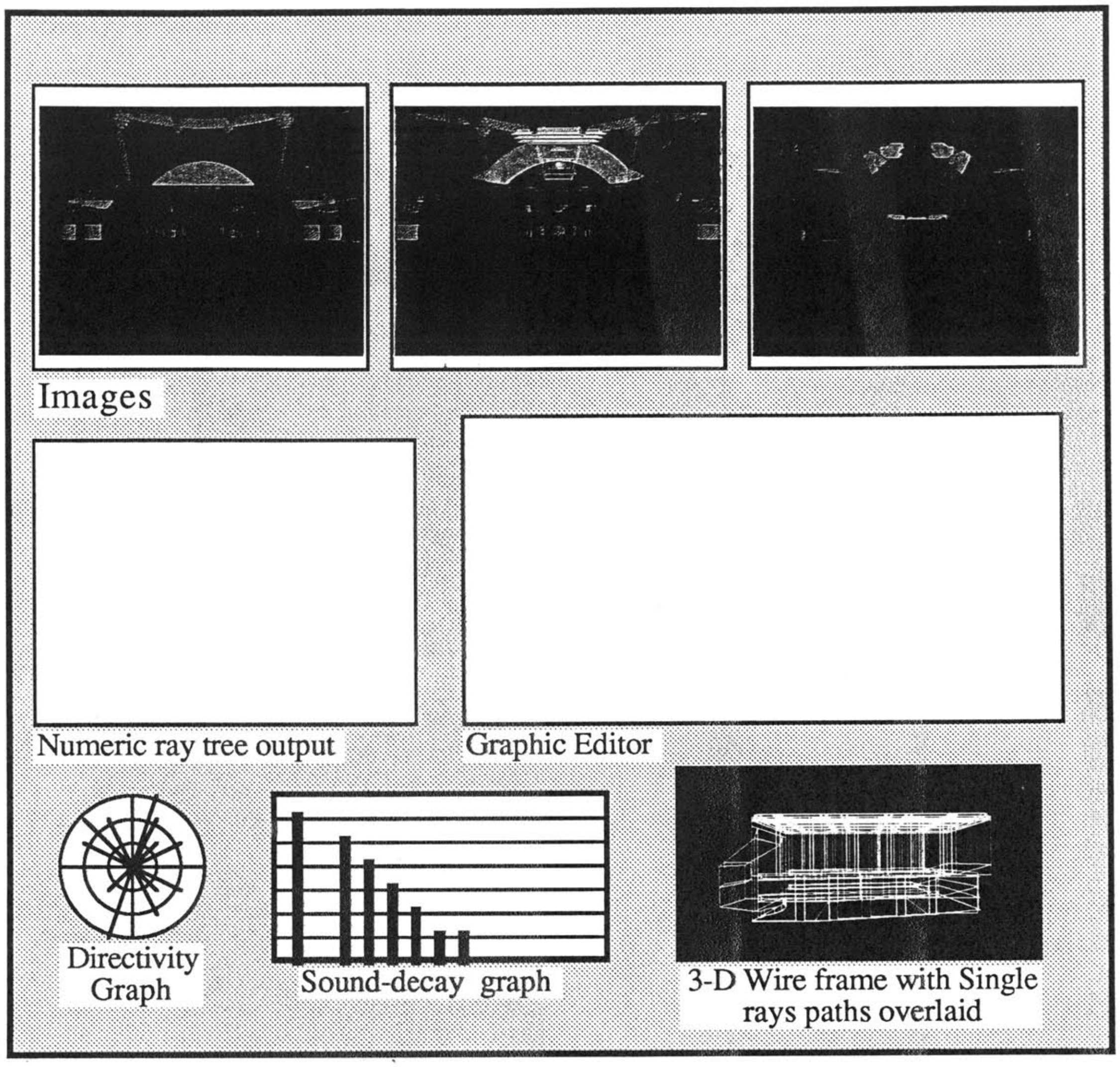


Figure 5-1: *A possible display interface for displaying and interacting with additional information.*

time delay graphs. These would help to decipher the sometimes complicated interactions and reduce some of the subjective nature of interpreting images.

Reverberation times should be able to be deduced and represented numerically without significant modifications. Other simpler approaches in computer ray tracing seem to indicate a strong feasibility in this direction [Wayman 80, Benedetto 84].

5.2 Future Applications

While the production of visual information has been an essential aspect exploited in these ray tracing concepts, these methods can also produce other types of information.

A notable aspect of the comprehensive information produced here is the possibility of synthesizing audio output from the images produced. This would provide a direct acoustical experience of listening to a design's properties. Images could either be converted directly into audio output, in which case, a person would hear the impulse response, or transformed into software parameters to digital sound field control equipment[17]. This equipment would synthesize the sound field structure of the concert hall or auditorium, and the auditory effect of each reflected sound field that comprises this sound field. Once these properties of a design are entered into a sound field modeler, it would be possible to listen to music as it would be heard.

5.3 Knowledgeable Environments

The current environment provides only the capabilities to work in one direction, that is to visualize properties according to a manually defined geometric and material database. Once a designer evaluates a space and decides to change some feature of that space, he must return to the data file, find the geometric and material specifications, make the changes according to his approximations with a text editor, and run the program again. This can be a long and tedious process before he can observe the consequences of his decisions.

[17]This is a similar approach as the Yamaha DSP-1, but the sound field properties would come from a synthesis of a space, rather than from empirical measurements.

The images used here open up an extremely large range of interactive and interface capabilities. Yet, it is unlikely that the graphic form of communication developed here for people will be of value to computers. Numeric information will have to be extracted from the ray tracing concepts.

Presently, the evaluation of the output remains in the hands of the designer. Future design decisions will benefit from the assistance of artificial intelligence systems. Constraint managers will automate and optimize through a detailed understanding of cause and effect relationships. In a complete interactive system, the designer will be able to work backwards, from the images to the spatial description, as well as forwards. This would involve the use of a constraint manager to produce an environment where the image would not be the only end product. Constraint managers operate on the principle that an image is the product of a set of a precise set of relations which can also be traced backwards. A constraint based system would be able to take a desired property and a model of the relations which generated an image, to work from changes in the image back to changes in the database.

The main liaison between the geometric description and images will be the ray program, which will communicate to designers with graphics, and a knowledge base system using parametric descriptions of the ray paths. The constraint manager will in a way perform the inverse of the ray tracer. For example, the ray program may show a gap in the arrival times of a reflected sound. If the designer judged that some sounds arrived too late for intelligibility, he could choose to move a wall by pointing at the wall in the image. The ensuing parametric descriptions of that particular ray path would be input to the constraint manager, which, it in turn, would go into the database and move the selected wall (or walls) as would be necessary to achieve the desired reflection time.

Qualitative assessments as to whether the information represented is good or bad remains to be determined by the user. At some later date, these psycho-acoustical judgments may be encoded and combined with the physical model used here, to produce a comprehensive design package.

5.4 Conclusion

The profound acoustical understanding this program brings to the design process will, I hope, provide a new, self-confident approach in manipulating the invisible properties of sound. This visual nature of the images produced is especially effective in reducing the possible misunderstanding and ascribing design intentions with the actual performance. The simulation capabilities, I believe, will produce some long term savings in time, by condensing the heuristic nature of design. I doubt this methodology will guide designers to converge on one optimal solution for a configuration, but it will allow for a much greater range of alternatives to be explored. Accurate simulations will provide the means for quicker iterations, quicker referencing and greater experience to be acquired without the process of building. This tool will help the architect to express acoustical constraints in design with a greater clarity and certainty.

Before redesigning Chevalier auditorium, I believed it would have been possible to incorporate the acoustical response, as visible in the images, into the design. This would correlate in a most direct way the acoustical interactions with the visual experience of the design. A problem with this idea, though, is that it is only effective for a particular source and listener relationship. The physical configuration of a hall must be general enough to accommodate everyone, so this is not easily possible. Given the newness of this technique though, I would not say that it is impossible, and the chance of coupling the acoustical and visual experiences may one day be possible. This may require some new technology, such as walls and reflectors that track to the source as it moves about on stage in order to maintain an optimal source and listener relationship. I doubt many will create a hall of mirrors with equally optically and visually reflective surface properties and use a light source to denote the specular properties. As an experiment, however, the effects could prove interesting.

The long term effects of this comprehensive graphic model will no doubt provide for a greater correlation between the visual and auditory experiences of a performance. It is also possible that designs involving curved walls and fan-shaped plans can be effectively treated, as in Chevalier Hall, to *not* exhibit these configurations acoustically. Design trends that arise from this methodology, if any, will be the result of much empirical experience which cannot be readily extrapolated here. The graphic analysis and ray based synthesis model presented here will, however, continue to develop as an aid in the conceptualization of the building performance.

Appendix A
Technical Notes and Listing

The program in appendix A was written in the C programming language. The code was developed and runs under the UNIX[18] operating system. However, it has been kept as portable as possible and can run under a variety of alternate operating systems.

The images produced by the program are in a file of runlength encoded one byte intensity values. The file can then be displayed, through separate programs, on a variety of computer monitors. Display devices should have a facility for using a pointer, such as a mouse, for interacting with the image. Presently, the interaction functions are limited and they serve only to determine a pixel's device coordinates on a screen. These coordinates can then be passed into different ray analysis programs which perform other specific tasks, such as providing precise ray tree information or integrating of energy over specific areas.

[18]UNIX is a trademark of AT&T Bell Laboratories

A.1 Source Code

```
/************************************************************************/
/*                                                                      */
/*  program:    ray input.crl outfile [-x xRes][-y yRes]                */
/*                                                                      */
/*  purpose:    render an image of objects using recursive              */
/*              ray tracing and an illumination model.                  */
/*                                                                      */
/*  input:      object description file containing lists of             */
/*              polygonal and quadric surfaces.                         */
/*                                                                      */
/*  output:     a file of runlength encoded hex color table             */
/*              index values with a descriptional header.               */
/*                                                                      */
/*  author:     Philip R. Thompson, ($Author: phils $)                  */
/*              $Date: 88/01/30 17:51:28 $      $State: Exp $           */
/*                                                                      */
/*  based on:   JTWhitted, 10-Jan-84                                    */
/************************************************************************/
#ifndef LINT
static char ray_cast_id[] =
        "$Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $";
#endif LINT

#include  <stdio.h>
#include  "rays.h"

#define X_RES   512         /* default terminal resolution, can be  */
#define Y_RES   384         /* overridden by command line arguments */

char *progName;

main(argc,argv)
int argc;
char **argv;
{
    Ray     root;                     /* primary ray from the viewpoint */
    Object *thisObject, *readObj();  /* pointer to object list */
    int     xRes=X_RES, yRes=Y_RES;   /* actual # of rays sent */
    int     j;
    double maxy, maxx;                /* viewport coordinates */
    double xIndex, yIndex;            /* ray direction coordinates */
    char   *out_file, *strrchr(), *calloc();
    FILE   *view_file = stdin;
#ifndef COLOR
    char   *strcpy(), *strcat(), *malloc();
    int     outFile;                  /* picture output file descriptor */
    byte    *scanLine, grey_norm();   /* color indices */
#else COLOR
    int outFile_red, outFile_grn, outFile_blu;
    byte   *scanLine_red, *scanLine_grn, *scanLine_blu;
    byte   color_norm();
#endif COLOR
```

```
    progName = argv[0];
    if (argc < 3)
        error("usage: %s in.crl out [-x xRes][-y yRes] [-i input]",
            progName);

    thisObject = readObj(argv[1]);
    if (strcmp(strrchr(argv[2],'.'),".crl") == 0)
        error ("can't overwrite input file: %s", argv[2]);
    else
        out_file = argv[2];

    for (argc -= 3, argv += 3; argc > 0; argc--, argv++) {
        if (**argv == '-')
            switch (*++(*argv)) {
            case 'x':
                xRes = atoi(*(++argv));
                argc--;
                if (xRes > MAXRES)
                    error("max X resolution is %d",(char *)MAXRES);
                fprintf(stderr,"\nxRes = %d\n",xRes);
                break;
            case 'y':
                yRes = atoi(*(++argv));
                argc--;
                if (yRes > MAXRES)
                    error("max Y resolution is %d",(char *)MAXRES);
                fprintf(stderr,"yRes = %d\n",yRes);
                break;
            case 'i':
                if ((view_file=fopen(*(++argv), "r")) == NULL)
                    error("Can't read input file: %s.", *argv);
                argc--;
                break;
            default:
                error("Unknown command line option: -%s", *argv);
            }
        else
            error("usage: %s in.crl out [-x res][-y res][-i input]",
                progName);
    }
    maxx = (xRes-1)/2.0;
    maxy = (yRes-1)/2.0;
    init_ray(view_file, maxx, maxy);

#ifdef COLOR
    scanLine_red = (byte *)calloc((unsigned)xRes, sizeof(byte));
    scanLine_grn = (byte *)calloc((unsigned)xRes, sizeof(byte));
    scanLine_blu = (byte *)calloc((unsigned)xRes, sizeof(byte));
    if (scanLine_red==NULL || scanLine_grn==NULL || scanLine_blu==NULL)
        error("NO more memory - calloc error", "\0");

    outFile_red = ecreat(strcat(strcpy(malloc((unsigned)
            strlen(out_file)+4),out_file),".red"), 0644);
    outFile_grn = ecreat(strcat(strcpy(malloc((unsigned)
            strlen(out_file)+4),out_file),".grn"), 0644);
    outFile_blu = ecreat(strcat(strcpy(malloc((unsigned)
            strlen(out_file)+4),out_file),".blu"), 0644);
    init_outFile(outFile_red, xRes, yRes, RED);
    init_outFile(outFile_grn, xRes, yRes, GREEN);
    init_outFile(outFile_blu, xRes, yRes, BLUE);
#else
    if ((scanLine =(byte *)calloc((unsigned)xRes,sizeof(byte))) == NULL)
        error("NO more memory - calloc() error", "\0");
    outFile = ecreat(out_file, 0644);
    init_outFile(outFile, xRes, yRes, GREY);
#endif COLOR

    /*  loop for each scan line */
    for (yIndex=maxy; yIndex >= -maxy; yIndex -= 1.0) {

        /*  loop for each pixel  */
        for (xIndex = -maxx, j=0; xIndex <= maxx; xIndex += 1.0, j++) {

            viewTrans(&root, xIndex, yIndex);

             /* the real work of the program takes place
             *  in the procedures rayHit() and raySahde()
             *  - intersect ray with objects and calculate
             *  intensities, perhaps recursively
             */
            rayHit (&root, thisObject);
            shade_root(&root, thisObject);

#ifndef COLOR
            scanLine[j] = grey_norm(&root);
        }
        runl_encode(outFile, scanLine, xRes);
#else
            scanLine_red[j] = color_norm(root.intensity.red);
            scanLine_blu[j] = color_norm(root.intensity.blue);
            scanLine_grn[j] = color_norm(root.intensity.green);
        }
        runl_encode(outFile_red, scanLine_red, xRes);
        runl_encode(outFile_grn, scanLine_grn, xRes);
        runl_encode(outFile_blu, scanLine_blu, xRes);
#endif COLOR
        printf("Scanline number: %d\n", (int)yIndex);
    }
#ifndef COLOR
    if (close(outFile) != 0)
        error("error in CLOSING outFile %d", (char *)outFile);
#else
     (void)close(outFile_red);
     (void)close(outFile_grn);
     (void)close(outFile_blu);
#endif COLOR
    printf("\n%c%s finished A-okay!!!\n", 7, progName);
    exit(0);
}  /* end main */
```

```
/* if the camera moves, rotates, tilts, etc., the transformation
 * is done here
 */
viewTrans(this, x, y)
Ray  *this;
double  x, y;
{
    double z = 1.0, normalizeVec();          /* distance to viewport */
    extern Point eye;

    bzero((char *)this, sizeof(Ray));
    Transform(&x,&y,&z);

    this->direction.x = x;
    this->direction.y = y;
    this->direction.z = z;
    (void)normalizeVec(&(this->direction));

    this->origin.x = this->head.x = eye.x;
    this->origin.y = this->head.x = eye.y;
    this->origin.z = this->head.x = eye.z;

    this->rayType = PRIMARY;
    this->ident = EOL;
    this->cSpec = 1.0;
    this->t = HUGEREAL;
}


#ifndef COLOR
byte grey_norm(thisRay)      /* get a grey value, 0 - MAX_RGB */
Ray *thisRay;
{
    float intensity;
    int  norm_intensity;

    intensity = (thisRay->intensity.red + thisRay->intensity.green +
            thisRay->intensity.blue) / 3.0;
    if (intensity > 1.0)
        intensity = 1.0;
    norm_intensity = (int)(intensity*MAX_RGB + 0.5);
#else COLOR
byte color_norm(intensity)
float intensity;
{
    int norm_intensity;

    if (intensity > 1.00)
        intensity = 1.00;
    norm_intensity = (int)(intensity*MAX_RGB + 0.5);
#endif COLOR
    return((byte)norm_intensity);
}
```

```
/* freeing function for Ray types
*/
freeChildren(this)
Ray *this;
{
    if (this->shadow != NULL) {
        free ((char *)this->shadow);
        this->shadow = NULL;
    }
    if (this->refracted != NULL) {
        free ((char *)this->refracted);
        this->refracted = NULL;
    }
    if (this->reflected != NULL) {
        free ((char *)this->reflected);
        this->reflected = NULL;
    }
}

/***  end  cast.c  ***/
```

```
/**********************************************************************/
/*                                                                    */
/*  purpose:    a recursive shader and illumination model             */
/*                                                                    */
/*  author:     Philip R.  Thompson,    ($Author: phils $)            */
/*              $Date: 88/01/30 17:51:28 $      $State: Exp $         */
/*                                                                    */
/*  based on:   JTWhitted,  1/17/84                                   */
/**********************************************************************/
#ifndef LINT
static char  ray_shade_id[] =
        "$Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $";
#endif LINT

#include  <stdio.h>
#include  "rays.h"

extern int  numlights;
extern Light  *light;

/* rayShade() - a recursive shading procedure
*/
rayShade (thisRay, thisObject)
Ray  *thisRay;
Object  *thisObject;
{
    Ray  *sRay(), *tRay(), *lRay();

    if (thisRay->ident != EOL) {
        if (thisObject[thisRay->ident].theseProps->type & REFLECTED)
            thisRay->reflected = sRay (thisRay, thisObject);

        if (thisObject[thisRay->ident].theseProps->type & REFRACTED)
            thisRay->refracted = tRay (thisRay, thisObject);

        if (thisObject[thisRay->ident].theseProps->type & SHADOW)
            thisRay->shadow = lRay (thisRay, thisObject);
    }
    shade (thisRay, thisObject);

    /* once shade for a ray is computed,
    *  all descendent rays can be freed
    */
    freeChildren (thisRay);
}


/* sRay() - specular reflection ray generator.
*           returns pointer to reflected ray
*/
Ray *sRay (parentRay, object)
Ray  *parentRay;
Object *object;
{
    Ray  *this, *newRay();
    double  normalizeVec();

    if ((parentRay->cSpec < 0.05) || (parentRay->level >= MAXLEVEL))
        return (NULL);
    this = newRay();
    this->rayType = REFLECTED;
    this->parent = parentRay;
    this->level = parentRay->level+1;
    this->cSpec = object[parentRay->ident].theseProps->kSpec *
            parentRay->cSpec;

    /*  origin of reflected ray is point of intersection of
    *   incident ray with reflecting surface.
    */
    this->origin.x = parentRay->head.x;
    this->origin.y = parentRay->head.y;
    this->origin.z = parentRay->head.z;

    /*  reflected ray direction is V' + 2*Norm
    *   (see CACM, vol.23, no.6, page 344 for explanation)
    */

    this->direction.x = parentRay->direction.x / -parentRay->incDot +
                                        2.0*parentRay->normal.x;
    this->direction.y = parentRay->direction.y / -parentRay->incDot +
                                        2.0*parentRay->normal.y;
    this->direction.z = parentRay->direction.z / -parentRay->incDot +
                                        2.0*parentRay->normal.z;
    (void)normalizeVec(&(this->direction));

    /*  this is where the recursion is the process takes place,
    *   this new ray is run through the mill to see what intensity
    *   it picks up before shading for its parent is computed.
    */
    rayHit (this, object);
    return (this);
}


/*  tRay() - transparency ray generator.
*           returns pointer to refracted ray
*   note:   relative index of refraction is ratio of refractive
*           indices at interface between two media.
*/

Ray *tRay (parentRay, object)
Ray  *parentRay;                    /*  parent ray */
Object *object;
{
    Ray  *this, *newRay();
    double  kref, Vt1, Vt2, normalizeVec(), sqrt();
    Point  Vprime, Vtemp;

    if ((parentRay->cSpec < 0.05) || (parentRay->level >= MAXLEVEL))
        return(NULL);
    this = newRay();
    this->rayType = REFRACTED;
    this->level = parentRay->level + 1;
    this->parent = parentRay;
    this->cSpec = object[parentRay->ident].theseProps->kSpec *
            parentRay->cSpec;

    this->origin.x = parentRay->head.x;
    this->origin.y = parentRay->head.y;
    this->origin.z = parentRay->head.z;

    Vprime.x = parentRay->direction.x / -parentRay->incDot;
    Vprime.y = parentRay->direction.y / -parentRay->incDot;
    Vprime.z = parentRay->direction.z / -parentRay->incDot;
    Vtemp.x = Vprime.x + parentRay->normal.x;
    Vtemp.y = Vprime.y + parentRay->normal.y;
    Vtemp.z = Vprime.z + parentRay->normal.z;

    Vt1 = sqrt(sqr(Vprime.x) + sqr(Vprime.y) + sqr(Vprime.z));
    Vt2 = sqrt(sqr(Vtemp.x) + sqr(Vtemp.y) + sqr(Vtemp.z));
    kref = sqrt(sqr(object[parentRay->ident].theseProps->kRefr) *
            sqr(Vt1) - sqr(Vt2));
    this->direction.x = kref*(parentRay->normal.x + Vprime.x) -
                                parentRay->normal.x;
    this->direction.y = kref*(parentRay->normal.y + Vprime.y) -
                                parentRay->normal.y;
    this->direction.z = kref*(parentRay->normal.z + Vprime.z) -
                                parentRay->normal.z;
    (void)normalizeVec(&(this->direction));

    rayHit (this, object);
    return (this);
}


/*  lRay() - light ray(s) generator.
 *          returns a pointer to light ray(s) if visible
 *  note:  doesn't call rayHit and 't' is 0.0 for obscured rays
 */

Ray *lRay(parentRay, object)
Ray  *parentRay;
Object *object;
{
    int     i;
    Ray     *this;
    char    *calloc();
    Object *j;
    double ldist, dotProduct(), normalizeVec();

    if ((parentRay->cSpec < 0.05) || (parentRay->level > MAXLEVEL))
        return(NULL);
    if ((this=(Ray *)calloc((unsigned)numlights, sizeof(Ray))) == NULL)
        error("lRay: calloc() error", "\0");
    for (i=0; i < numlights; i++) {
        this[i].rayType = SHADOW;
        this[i].parent = parentRay;
        this[i].level = parentRay->level+1;
        this[i].origin.x = parentRay->head.x;
        this[i].origin.y = parentRay->head.y;
        this[i].origin.z = parentRay->head.z;
        this[i].head.x = light[i].x;
        this[i].head.y = light[i].y;
        this[i].head.z = light[i].z;
        this[i].direction.x = this[i].head.x - this[i].origin.x;
        this[i].direction.y = this[i].head.y - this[i].origin.y;
        this[i].direction.z = this[i].head.z - this[i].origin.z;
        this[i].t = ldist = normalizeVec(&(this[i].direction));
        this[i].ctotal = parentRay->ctotal + ldist;
        if ((this[i].incDot = dotProduct(&(this[i].direction),
                &(parentRay->normal))) <= 0.0) {
            this[i].t = this[i].ctotal = 0.0;
            continue;
        }
        this[i].intensity.red = this[i].intensity.green =
                this[i].intensity.blue = light[i].intensity;
        for (j=object; j != NULL; j=j->next) {
            switch (j->objType) {
            case POLYGON:
                interPoly (&(this[i]), j);
                break;
            case SPHERE:
                interSphere (&(this[i]), j);
                break;
            case QUADRIC:
                interQuad (&(this[i]), j);
                break;
            }
            if (this[i].t < ldist) {
                if (object[this[i].ident].theseProps->type & REFRACTED){
                    this[i].intensity.red *=
                            object[this[i].ident].theseProps->kTrans;
                    this[i].intensity.green *=
                            object[this[i].ident].theseProps->kTrans;
                    this[i].intensity.blue *=
                            object[this[i].ident].theseProps->kTrans;
                    this[i].t = ldist;
                } else {        /* object is opaque */
                    this[i].t = this[i].ctotal = 0.0;
                    break;
                }
            }
        }
    }
    return(this);
}


/* shade()  - computes shade at each node of the ray tree
```

```
*/
shade (thisRay, thisObject)
Ray     *thisRay;
Object  *thisObject;
{
    Point  R;
    int    i;
    Properties *surfProp;
    double dotNL, dotVL, dotProduct(), pow(), normalizeVec();
    float tmp, phong = 1000.0;
    extern float  Iamb;

    surfProp = thisObject[thisRay->ident].theseProps;

    if (thisRay->t == HUGEREAL) {              /* doesn't hit anything, */
        thisRay->intensity.red = 0.082;        /* give it a backgroud */
        thisRay->intensity.green = 0.071;      /* color 'RG&B' */
        thisRay->intensity.blue = 0.066;
        return;
    } else {
        /*  the ray intersects a surface,
        *   compute shade with a local model.
        */
        thisRay->intensity.red = Iamb * surfProp->color.red;
        thisRay->intensity.green = Iamb * surfProp->color.green;
        thisRay->intensity.blue = Iamb * surfProp->color.blue;
    }
    if (thisRay->reflected != NULL) {
        /*  there is a reflected ray, add contribution from
        *   mirror reflection.
        */
        thisRay->intensity.red += thisRay->reflected->intensity.red *
                surfProp->kSpec;
        thisRay->intensity.green += thisRay->reflected->intensity.green
                * surfProp->kSpec;
        thisRay->intensity.blue += thisRay->reflected->intensity.blue *
                surfProp->kSpec;
    }
    if (thisRay->refracted != NULL) {

        /*  there is a refracted ray, add contribution
        *   from transmitted light.
        */
        thisRay->intensity.red += thisRay->refracted->intensity.red *
                surfProp->kTrans;
        thisRay->intensity.green += thisRay->refracted->intensity.green
                * surfProp->kTrans;
        thisRay->intensity.blue += thisRay->refracted->intensity.blue *
                surfProp->kTrans;
    }
    if (thisRay->shadow != NULL) {
        for (i=0; i < numlights; i++) {
            if ((thisRay->shadow[i].t <= 0.0) ||
                    ((dotNL=thisRay->shadow[i].incDot) <= 0.0))
                continue;
            /* calculate diffuse surface normal shading */
            thisRay->intensity.red += (float)dotNL * surfProp->kDiff *
                    thisRay->shadow[i].intensity.red;
            thisRay->intensity.green += (float)dotNL * surfProp->kDiff *
                    thisRay->shadow[i].intensity.green;
            thisRay->intensity.blue += (float)dotNL * surfProp->kDiff *
                    thisRay->shadow[i].intensity.blue;

            if (thisRay->reflected != NULL)
                dotVL = dotProduct(&(thisRay->shadow[i].direction),
                        &(thisRay->reflected->direction));
            else {
                /* find direction of highlight reflection (R)
                *  in order to calculate phong highlights
                */
                R.x = -thisRay->shadow[i].direction.x/dotNL +
                                        2.0*thisRay->normal.x;
                R.y = -thisRay->shadow[i].direction.y/dotNL +
                                        2.0*thisRay->normal.y;
                R.z = -thisRay->shadow[i].direction.z/dotNL +
                                        2.0*thisRay->normal.z;
                (void)normalizeVec(&R);
                dotVL = -dotProduct(&(thisRay->direction), &R);
            }
            if (dotVL > 0.0) {
                tmp = (float)pow(dotVL,phong) * surfProp->kSpec;
                thisRay->intensity.red +=
                        thisRay->shadow[i].intensity.red * tmp;
                thisRay->intensity.green +=
                        thisRay->shadow[i].intensity.green * tmp;
                thisRay->intensity.blue +=
                        thisRay->shadow[i].intensity.blue * tmp;
            }
        }
    }
}


/*  code to make the source directly visible from
*   the view point
*/
shade_root(thisRay, objList)
Ray *thisRay;
Object *objList;
{
    int  i;
    Ray  this;
    Object  *j;
    float  tmp;
    double  dotVL, pow(), dotProduct();
    double  ldist, normalizeVec();

    bzero((char *)&this, sizeof(Ray));
    this.rayType = SHADOW;
    this.level = thisRay->level;
    this.parent = thisRay;
    this.origin.x = thisRay->origin.x;
    this.origin.y = thisRay->origin.y;
    this.origin.z = thisRay->origin.z;
    for (i=0; i < numlights; i++) {
        this.head.x = light[i].x;
        this.head.y = light[i].y;
        this.head.z = light[i].z;
        this.direction.x = this.head.x - this.origin.x;
        this.direction.y = this.head.y - this.origin.y;
        this.direction.z = this.head.z - this.origin.z;
        this.t = this.ctotal = ldist = normalizeVec(&(this.direction));

        if ((dotVL=dotProduct(&(thisRay->direction),
                &(this.direction))) <= 0.0)
            continue;
        this.intensity.red = this.intensity.green =
            this.intensity.blue = light[i].intensity;
        for (j=objList; j != NULL; j=j->next) {
            switch (j->objType) {
            case QUADRIC:
                interQuad (&this, j);
                break;
            case POLYGON:
                interPoly (&this, j);
                break;
            case SPHERE:
                interSphere (&this, j);
                break;
            }
            if (this.t < ldist) {
                if (objList[this.ident].theseProps->type & REFRACTED) {
                    this.intensity.red *=
                            objList[this.ident].theseProps->kTrans;
                    this.intensity.green *=
                            objList[this.ident].theseProps->kTrans;
                    this.intensity.blue *=
                            objList[this.ident].theseProps->kTrans;
                    this.t = ldist;
                } else {      /* object is opaque */
                    this.t = 0.0;
                    this.ctotal = 0.0;
                    break;
                }
            }
        }
#ifdef SRAY
        printRay(&this);
#endif SRAY
        tmp = (float)pow(dotVL, 50000.0);
        thisRay->intensity.red += this.intensity.red * tmp;
        thisRay->intensity.green += this.intensity.green * tmp;
        thisRay->intensity.blue += this.intensity.blue * tmp;
    }
}
```

```
/*** end shade.c ***/
```

```
/***********************************************************************/
/*                                                                     */
/*  purpose:    intersects tRay (perhaps recursively) with all         */
/*              surfaces in the object description and returns         */
/*              distance and object id. of the nearest intersection    */
/*                                                                     */
/*  author:     Philip R. Thompson, ($Author: phils $)                 */
/*              $Date: 88/01/30 17:51:28 $     $State: Exp $           */
/*                                                                     */
/*  based on:   JTWhitted, 1-17-84                                     */
/***********************************************************************/
#ifndef LINT
static char ray_hit_id[] =
        "$Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $";
#endif LINT

#include <stdio.h>
#include "rays.h"


rayHit(tRay, thisObject)
Ray *tRay;
Object  *thisObject;
{
    Object  *i;
    double  dotProduct(), normalizeVec();

    /* loop once per surface element */
    for (i = thisObject; i != NULL; i = i->next) {
        switch (i->objType) {
        case POLYGON:
            interPoly (tRay, i);
            break;
        case SPHERE:
            interSphere (tRay, i);
            break;
        case QUADRIC:
            interQuad (tRay, i);
            break;
        default:
            error("RayHit: Unknown object type.", "\0");
        }
    }
    if (tRay->t < HUGEREAL) {
        (void)normalizeVec(&(tRay->normal));
        /* calculate dot product of parent ray and surface normal
         * to a) determine surface orientation, and b) generate
         * Vprime for sRay(), tRay() and lray();
         */
        if ((tRay->incDot = dotProduct(&(tRay->direction),
                    &(tRay->normal))) > 0.0) {
            tRay->incDot = -tRay->incDot ;
            tRay->normal.x = -tRay->normal.x;
            tRay->normal.y = -tRay->normal.y;
            tRay->normal.z = -tRay->normal.z;
        }
    }
    if (tRay->rayType == PRIMARY)
        tRay->ctotal = tRay->t;
    else
        tRay->ctotal = tRay->parent->ctotal + tRay->t;

    /* calculate shade (perhaps recursively)
    *  returned by this ray
    */
    rayShade(tRay, thisObject);
}


/*
l_rayHit(tRay)
Ray *tRay;
*/


interQuad (tRay, tObj)
Ray *tRay;
Object *tObj;
{
    Quadric *surf;
    double  acoef;                  /* square term coefficient */
    double  bcoef;                  /* linear term coefficient */
    double  ccoef;                  /* constant term coefficient */
    double  a,b,c,d,e,f,g,h,j,k;
    double  disc, tTmp, sqrt();
    double  thisX, thisY, thisZ;    /* intersection point on surface  */
    Point   *rDir;                  /* this ray direction */
    Point   *rOrg;                  /* this ray origin */

    rDir = &(tRay->direction);
    rOrg = &(tRay->origin);
    surf = tObj->ptr.quadric;

    a = surf->a;  b = surf->b;  c = surf->c;
    d = surf->d;  e = surf->e;  f = surf->f;
    g = surf->g;  h = surf->h;  j = surf->j;
    k = surf->k;

    /*  substitute ray equation into surface equation
    */
    acoef = a * rDir->x * rDir->x + b * rDir->x * rDir->y +
            c * rDir->x * rDir->z + e * rDir->y * rDir->y +
            f * rDir->y * rDir->z + h * rDir->z * rDir->z;
    bcoef = 2.0 * a * rOrg->x * rDir->x +
              b * (rOrg->x * rDir->y + rDir->x * rOrg->y) +
              c * (rOrg->x * rDir->z + rDir->x * rOrg->z) +
              d * rDir->x + 2.0 * e * rOrg->y * rDir->y +
              f * (rOrg->y * rDir->z + rDir->y * rOrg->z) +
              g * rDir->y + 2.0 * h * rOrg->z * rDir->z + j * rDir->z;
    ccoef = a * rOrg->x * rOrg->x + b * rOrg->x * rOrg->y +
            c * rOrg->x * rOrg->z + d * rOrg->x +
            e * rOrg->y * rOrg->y + f * rOrg->y * rOrg->z +
            g * rOrg->y + h * rOrg->z * rOrg->z + j * rOrg->z + k;

    if (acoef == 0.0)
        return;
    else {
         /* use the quadratic formula to sorve for 't'
         */
        disc = bcoef*bcoef - 4.0*acoef*ccoef;

        if (disc < 0.0)                  /* no REAL solution */
            return;
        else
            tTmp = (-bcoef - sqrt(disc))/(2.0 * acoef);
        if (tTmp < 0.0)                  /* t must be positve */
            tTmp = (-bcoef + sqrt(disc))/(2.0 * acoef);

        /* this is the blatant, but essential fudge
         * to insure that the origin of a ray is
         * not mistaken for a valid point of intersection
         */
        if (tTmp < EPSILON)
            return;
    }
    /* if this is the closest value of t so far,
     * then record t and the identity of the surface
     * and compute the surface normal (yes, it would
     * be more efficient to compute the surface normal
     * outside of this loop)
     */
    if (tTmp < tRay->t) {
        tRay->t = tTmp;
        tRay->ident = tObj->objIdent;
        thisX = tRay->head.x = tRay->origin.x + tTmp*tRay->direction.x;
        thisY = tRay->head.y = tRay->origin.y + tTmp*tRay->direction.y;
        thisZ = tRay->head.z = tRay->origin.z + tTmp*tRay->direction.z;
        tRay->normal.x = 2.0*a*thisX + b*thisY + c*thisZ + 2.0*d;
        tRay->normal.y = 2.0*e*thisY + b*thisX + f*thisZ + 2.0*g;
        tRay->normal.z = 2.0*h*thisZ + c*thisX + f*thisY + 2.0*j;
    }
}

interSphere(tRay, tObj)
Ray *tRay;
Object *tObj;
{
    Sphere *sphr;
    Point  delta;
    double a,b,c,d;
    double tTmp, dotProduct(), sqrt();

    sphr = tObj->ptr.sphere;

    a = (sqr(tRay->direction.x) + sqr(tRay->direction.y) +
         sqr(tRay->direction.z));
    delta.x = tRay->origin.x - sphr->center.x;
    delta.y = tRay->origin.y - sphr->center.y;
    delta.z = tRay->origin.z - sphr->center.z;
    b = 2.0 * dotProduct(&(tRay->direction), &delta);
    c = sqr(delta.x) + sqr(delta.y) + sqr(delta.z) - sqr(sphr->radius);
    d = b*b - 4.0*a*c;

    /* check if intersection exists */
    if (d < 0.0)
        return;
    else
        tTmp = (-b - sqrt(d)) / (2.0*a);
    if (tTmp < 0.0)
        tTmp = (-b + sqrt(d)) / (2.0*a);
    if (tTmp < EPSILON)
        return;

    if (tTmp < tRay->t) {
        tRay->t = tTmp;
        tRay->ident = tObj->objIdent;
        tRay->head.x = tRay->origin.x + tTmp*tRay->direction.x;
        tRay->head.y = tRay->origin.y + tTmp*tRay->direction.y;
        tRay->head.z = tRay->origin.z + tTmp*tRay->direction.z;
        tRay->normal.x = 2.0*(tRay->head.x - sphr->center.x);
        tRay->normal.y = 2.0*(tRay->head.y - sphr->center.y);
        tRay->normal.z = 2.0*(tRay->head.z - sphr->center.z);
    }
}


/*   Intersect a ray with a polygon
 *   On entry:
 *       surf = definitions for a polygon and a ray
 *   On exit:
 *       tRay = ray with new normal and length (t)
 */
interPoly(tRay, tObj)
Ray *tRay;
Object *tObj;
{
    register double  a,b,c,d;
    register double  thisX,thisY,thisZ;
    register double  tTmp, dotVN;
    Point   *rOrg, *rDir;

    tTmp = HUGEREAL;
    rDir  = &(tRay->direction);
    rOrg  = &(tRay->origin);

    /* Get plane equation & normal of the polygon */
    a = tObj->ptr.polygon->a;
    b = tObj->ptr.polygon->b;
    c = tObj->ptr.polygon->c;
    d = tObj->ptr.polygon->d;
    if ((dotVN = a*rDir->x + b*rDir->y + c*rDir->z) == 0.0)
        return;        /* polygon on edge, not visible */

    if ((tTmp=(a*rOrg->x + b*rOrg->y + c*rOrg->z + d)/-dotVN) < 0.0)
        tTmp = -tTmp;

    /* check that the origin of the ray is not mistaken for point of
    *  intersection and distance is closest so far.
    */
    if ((tTmp < tRay->t) && (tTmp > EPSILON)) {
        /* The ray intersects the plane of the polygon. Find the
            point of intersection */
        thisX = rOrg->x + (rDir->x * tTmp);
        thisY = rOrg->y + (rDir->y * tTmp);
        thisZ = rOrg->z + (rDir->z * tTmp);

        /* first check if point is within the object's bounding box */
        if (thisX > tObj->objmin.x && thisX < tObj->objmax.x &&
            thisY > tObj->objmin.y && thisY < tObj->objmax.y &&
            thisZ > tObj->objmin.z && thisZ < tObj->objmax.z)

            /* it is, so do the actual tests and calculations */
            if (inpoly(thisX,thisY,thisZ,tObj->ptr.polygon)) {
                tRay->t = tTmp;
                tRay->ident = tObj->objIdent;
                tRay->head.x = thisX;
                tRay->head.y = thisY;
                tRay->head.z = thisZ;
                tRay->normal.x = a;
                tRay->normal.y = b;
                tRay->normal.z = c;
            }
    } else
        return;
}


/* The algorithm counts intersections between the polygon and a ray
*  starting at the given point.  The 2 bit "angle" calculation
*  technique is used
*/
inpoly(x,y,z,polyptr)
double x,y,z;
Polygon *polyptr;
{
    double p2x,p2y;
    double p1x,p1y;
    double sign, tempd, fabs();
    int count, i, j, polytype;
    int to, from, temp;

    tempd = fabs(polyptr->c);
    polytype = 1;

    if (fabs(polyptr->b) > tempd) {
        tempd = fabs(polyptr->b);
        polytype = 2;
    }
    if (fabs(polyptr->a) > tempd) {
        tempd = fabs(polyptr->a);
        polytype = 3;
    }
    switch (polytype) {
    case 1:
        p1x = polyptr->vertex[0]->x - x;
        p1y = polyptr->vertex[0]->y - y;
        break;
    case 2:
        p1x = polyptr->vertex[0]->x - x;
        p1y = polyptr->vertex[0]->z - z;
        break;
    case 3:
        p1x = polyptr->vertex[0]->y - y;
        p1y = polyptr->vertex[0]->z - z;
        break;
    }
    if (p1x >= 0.0) {
        if  (p1y < 0.0)
            from = 3;
        else if (p1y >= 0.0 && p1x > 0.0)
            from = 0;
        else
            from = 1;
    } else {
        if (p1y <= 0.0)
            from = 2;
        else
            from = 1;
    }
    count = 0;       /*  initialize intersect counter */
    for(i=0; i < polyptr->numvtx; i++) {
        j = (i+1 < polyptr->numvtx)? i+1 : 0;
        switch (polytype) {
        case 1:
            p2x = polyptr->vertex[j]->x - x;
            p2y = polyptr->vertex[j]->y - y;
            break;
        case 2:
            p2x = polyptr->vertex[j]->x - x;
            p2y = polyptr->vertex[j]->z - z;
            break;
        case 3:
            p2x = polyptr->vertex[j]->y - y;
            p2y = polyptr->vertex[j]->z - z;
            break;
        }
        if  (p2x >= 0.0) {
            if  (p2y < 0.0)
                to = 3;
            else if (p2y >= 0.0 && p2x > 0.0)
```

```
                to = 0;
            else
                to = 1;
        } else  {
            if (p2y <= 0.0)
                to = 2;
            else
                to = 1;
        }
        temp = (to - from) & 3;
        if  (temp == 1)
            count++;
        else if (temp == 3)
            count--;
        else if (temp == 2) {
            sign = p1x*p2y - p2x*p1y;
            if  (sign > 0.0)
                count += 2;
            else if (sign < 0.0)
                count -= 2;
            else
                return(TRUE);
        }
        p1x = p2x;
        p1y = p2y;
        from = to;
    }
    if ((count != 0) && ((count % 4) == 0))
        return(TRUE);
    else
        return(FALSE);
}

/*** end hit.c ***/
```

```
/*************************************************************************/
/*                                                                       */
/*  purpose:     contains math functions for vectors and                */
/*               3-D coordinate transformations                          */
/*                                                                       */
/*  author:      Philip R. Thompson, ($Author: phils $)                  */
/*               $Date: 88/01/30 17:51:28 $      $State: Exp $          */
/*                                                                       */
/*                    3-D GRAPHIC ROUTINES                               */
/*************************************************************************/
#ifndef LINT
static char ray_math_id[] =
    "$Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $";
#endif LINT

#include <math.h>
#include "rays.h"

double  TransM[4][4];          /* Transformation Matrix    */
double  SV[4][4];              /* Saved View matrix        */
double  cx,cy,cz;              /* Current cursor position  */
double  screen_scale, screen_ctrx, screen_ctry;


/*  MATRIX TRANSFORMATION ROUTINES */

PrintMatrix (M)
double M[4][4];
{
    register x,y;

    for (x=0;x<4;x++) {
        for (y=0;y<4;y++)
            printf("%10.2f",M [x][y]);
        printf("\n");
    }
    printf("\n");
}


Set (M)
double M[4][4];
{
    register x,y;

    for (x=0;x<4;x++)
        for (y=0;y<4;y++)
            TransM[x][y] = M[x][y];
}


Save (M)
double M[4][4];
{
    register x,y;
    for (x=0;x<4;x++)
        for (y=0;y<4;y++)
            M[x][y] = TransM[x][y];
}


ResetMatrix (M)
double M[4][4];
{
    register x,y;

    for (x=0;x<4;x++)
        for (y=0;y<4;y++)
            if (x == y)
                M[x][y] = 1.0;
            else
                M[x][y] = 0.0;
}

MatrixMult(M1,M2)
double M1[4][4];
double M2[4][4];
{
    register x,y,z;
    double R[4][4];

    for (x=0;x<4;x++)
        for (y=0;y<4;y++) {
            R[x][y] = 0.0;
            for (z=0;z<4;z++)
                R[x][y] += M1[x][z] * M2[z][y];
        }
    Set(R);
}


Transform(newx,newy,newz)
double *newx,*newy,*newz;
{
  double  x = *newx, y = *newy, z = *newz;

  *newx=TransM[0][0]*x + TransM[1][0]*y + TransM[2][0]*z + TransM[3][0];
  *newy=TransM[0][1]*x + TransM[1][1]*y + TransM[2][1]*z + TransM[3][1];
  *newz=TransM[0][2]*x + TransM[1][2]*y + TransM[2][2]*z + TransM[3][2];
}


translateXYZ(x,y,z)
double x,y,z;
{
    double M[4][4];

    ResetMatrix (M);
    M [3][0] = x;
    M [3][1] = y;
    M [3][2] = z;
    MatrixMult(TransM,M);
}


rotateX(rad)                /* rotation counter-clockwise in radians */
double rad;
{
    double M[4][4];

    ResetMatrix (M);
    M[1][1] = cos(rad);
    M[1][2] = sin(rad);
    M[2][1] = -sin(rad);
    M[2][2] = cos(rad);
    MatrixMult(TransM,M);
}

rotateY(rad)
double rad;
{
    double M[4][4];

    ResetMatrix (M);
    M[0][0] = cos(rad);
    M[0][2] = -sin(rad);
    M[2][0] = sin(rad);
    M[2][2] = cos(rad);
    MatrixMult(TransM,M);
}

rotateZ(rad)
double rad;
{
    double M[4][4];

    ResetMatrix (M);
    M[0][0] = cos(rad);
    M[0][1] = sin(rad);
    M[1][0] = -sin(rad);
    M[1][1] = cos(rad);
    MatrixMult(TransM,M);
}

scaleXYZ(sx,sy,sz)
double sx,sy,sz;
{
    double M[4][4];

    ResetMatrix (M);
    M[0][0] = sx;
    M[1][1] = sy;
    M[2][2] = sz;
    MatrixMult(TransM,M);
}
```

```
rotateAxis(x1, y1, z1, x2, y2, z2, rad)  /* rotate abt arb axis */
double x1, y1, z1, x2, y2, z2, rad;
{
    double a, b, c, d, D;              /* temporary variables */
    double r1, r2, PolarAngle();       /* temporary angles */

    a = x2 - x1;
    b = y2 - y1;
    c = z2 - z1;
    d = sqrt(a*a + b*b + c*c);
    a /= d;
    b /= d;
    c /= d;
    D = sqrt(b*b + c*c);

    ResetMatrix(TransM);

    translateXYZ(-x1,-y1,-z1);
    r1 = PolarAngle(c,b);
    rotateX(r1);
    r2 = PolarAngle(D,a);
    rotateY(r2);
    rotateZ(rad);
    rotateY(-r2);
    rotateX(-r1);
    translateXYZ(x1,y1,z1);
}


/* VIEWING TRANSFORMATION */

perspective(P)     /* Avoid this */
double P;
{
    double M [4][4];

    ResetMatrix(M);
    M [2][3] = P;
    M [3][3] = P;
    MatrixMult(TransM,M);
}


LeftHand()
{
    double M [4][4];

    ResetMatrix(M);
    M [2][2] = -1;
    MatrixMult(TransM,M);
}


view(Ex,Ey,Ez)
double Ex,Ey,Ez;
```

```
{
    double hypo, PolarAngle();

    ResetMatrix(TransM);
    translateXYZ(-Ex,-Ey,-Ez);
    rotateX(90.0);
    hypo = sqrt(Ex*Ex+Ey*Ey);
    if ((Ex != 0.0) || (Ey != 0.0))
         rotateY(PolarAngle(-Ey,Ex));
    rotateX(PolarAngle(hypo,-Ez));
}


/*  DISPLAY ROUTINES  */

Init_View(wind_w, wind_h)
int wind_w, wind_h;
{
    ResetMatrix(TransM);
    screen_scale = (double)wind_w;
    screen_ctrx = (double)wind_w / 2.0;
    screen_ctry = (double)wind_h / 2.0;
}


Display_2D(x, y, z)
double *x, *y, z;
{
    *x = (*x / z) * screen_scale + screen_ctrx;
    *y = (*y / z) * -screen_scale + screen_ctry;
}


ShowLine(x1,y1,z1,x2,y2,z2)
double x1,y1,z1,x2,y2,z2;
{
    double tx1=x1, ty1=y1, tx2=x2, ty2=y2;

    Display_2D(&tx1, &ty1, z1);
    Display_2D(&tx2, &ty2, z2);
#ifdef THREE_D
    DrawLine(trunc(tx1),trunc(ty1),trunc(tx2),trunc(ty2));
#endif THREE_D
}


#define LEFT     0x01
#define RIGHT    0x02
#define BOTTOM   0x04
#define TOP      0x08
#define BACK     0x10
#define FRONT    0x20
#define ZMIN     0.1

Code(x,y,z,c)
```

```
double x,y,z;
unsigned *c;
{
    *c = 0;
    if (x < -z)
        *c = LEFT;
    else if (x > z)
        *c |= RIGHT;
    if (y < -z)
        *c |= BOTTOM;
    else if (y > z)
        *c |= TOP;
}

ClipLine(x1,y1,z1,x2,y2,z2)
double x1,y1,z1,x2,y2,z2;
{
    unsigned  c, c1, c2;
    double  x, y, z, t;

    if (z1 < ZMIN) {
        if (z2 < ZMIN)
            return;
        t = (ZMIN - z1) / (z2 - z1);
        x1 += (x2 - x1) * t;
        y1 += (y2 - y1) * t;
        z1 = ZMIN;
    } else if (z2 < ZMIN) {
        t = (ZMIN - z2) / (z1 - z2);
        x2 += (x1 - x2) * t;
        y2 += (y1 - y2) * t;
        z2 = ZMIN;
    }
    Code(x1,y1,z1,&c1);
    Code(x2,y2,z2,&c2);

    while (c1 | c2) {
        if (c1 & c2)
            return;
        c = (c1 != 0) ? c1 : c2;

        if (LEFT & c) {
            t = (z1 + x1)/((x1 - x2) - (z2 - z1));
            z = t*(z2 - z1) + z1;
            x = -z;
            y = t*(y2 - y1) + y1;
        } else if (RIGHT & c) {
            t = (z1 - x1)/((x2 - x1) - (z2 - z1));
            z = t*(z2 - z1) + z1;
            x = z;
            y = t*(y2 - y1) + y1;
        } else if (BOTTOM & c) {
            t = (z1 + y1)/((y1 - y2) - (z2 - z1));
            z = t * (z2 - z1) + z1;
            x = t * (x2 - x1) + x1;
            y = -z;
        } else if (TOP & c) {
            t = (z1 - y1)/((y2 - y1) - (z2 - z1));
            z = t * (z2 - z1) + z1;
            x = t * (x2 - x1) + x1;
            y = z;
        }
        if (c == c1) {
            x1 = x; y1 = y; z1 = z;
            Code (x,y,z,&c1);
        } else if (c == c2) {
            x2 = x; y2 = y; z2 = z;
            Code (x,y,z,&c2);
        }
    }
    ShowLine(x1,y1,z1,x2,y2,z2);
}


move_abs3(x, y, z)
double x,y,z;
{
    double  nx = x, ny = y, nz = z;

    Transform(&nx, &ny, &nz);
    cx = nx;
    cy = ny;
    cz = nz;
}


line_abs3(x, y, z)
double x, y, z;
{
    double  nx = x, ny = y, nz = z;

    Transform(&nx, &ny, &nz);
    ClipLine(cx, cy, cz, nx, ny, nz);
    cx = nx;
    cy = ny;
    cz = nz;
}


/*  ARC ROUTINES  by  Cheng-Haam Tham  */

arcX (r, theta, theta_stop, d_theta, x)
double r, theta, theta_stop, d_theta, x;
{
    theta = toRadians(theta);
    theta_stop = toRadians(theta_stop);
    d_theta = toRadians(d_theta);

    move_abs3(x, (r*cos(theta)), (r*sin(theta)));
    for (theta += d_theta; theta <= theta_stop; theta += d_theta)
```

```
        line_abs3(x, (r*cos(theta)), (r*sin(theta)));
}


arcY (r, theta, theta_stop, d_theta, y)
double r, theta, theta_stop, d_theta, y;
{
    theta = toRadians(theta);
    theta_stop = toRadians(theta_stop);
    d_theta = toRadians(d_theta);

    move_abs3((-r*cos(theta)), y, (r*sin(theta)));
    for (theta += d_theta; theta <= theta_stop; theta += d_theta)
        line_abs3 ((-r*cos(theta)), y, (r*sin(theta)));
}


arcZ (r, theta, theta_stop, d_theta, z)
double r, theta, theta_stop, d_theta, z;
{
    theta = toRadians(theta);
    theta_stop = toRadians(theta_stop);
    d_theta = toRadians(d_theta);

    move_abs3((r*cos(theta)), (r*sin(theta)), z);
    for (theta += d_theta; theta <= theta_stop; theta += d_theta)
        line_abs3((r*cos(theta)), (r*sin(theta)), z);
}


/*  TEXT ROUTINE  */

text (x,y,z,str)
double x,y,z;
char str[];
{
    double  nx = x, ny = y, nz = z;

    Transform(&nx, &ny, &nz);
    Display_2D(&nx, &ny, nz);
#ifdef THREE_D
    DrawText(trunc(nx), trunc(ny), str);
#endif THREE_D
}


/* VECTOR ROUTINES */

double normalizeVec(vec)        /* get the unit vector */
Point *vec;
{
    double denom;

    denom = sqrt(sqr(vec->x) + sqr(vec->y) + sqr(vec->z));
    vec->x /= denom;
    vec->y /= denom;
    vec->z /= denom;
    return (denom);          /* the magnitude */
}


double dotProduct (v1, v2)
Point *v1, *v2;
{
    return(v1->x * v2->x +
           v1->y * v2->y +
           v1->z * v2->z);
}


double PolarAngle(x,y)          /* returns the polar angle of (x,y) */
double x,y;
{
    double length, theta;

    length = sqrt(x*x + y*y);
    if ((length == 0.0) || ((theta = fabs(x/length)) >= 1.0))
        return(0.0);
    else
        theta = acos(theta);
    if ((x >= 0.0) && (y >= 0.0))
        return(theta);
    else if ((x <= 0.0) && (y >= 0.0))
        return(PI - theta);
    else if ((x <= 0.0) && (y <= 0.0))
        return(-PI + theta);
    else
        return(-theta);
}


double VectorsAngle(v1, v2)   /* angle between two vectors*/
Point *v1, *v2;
{
    double theta = 0.0;
    double d1, d2, dotProduct();

    d1 = sqrt(sqr(v1->x) + sqr(v1->y) + sqr(v1->z));
    d2 = sqrt(sqr(v2->x) + sqr(v2->y) + sqr(v2->z));
    if ((d1 != 0.0) && (d2 != 0.0)) {
        theta = dotProduct(v1,v2) / (d1*d2);
        theta = acos(theta);
        return(theta);
    }
    return(theta);
}

/*** end 3d_math.c ***/
```

```
/***********************************************************************/
/*                                                                     */
/*  purpose:    read the data from an ASCII crl format file            */
/*                                                                     */
/*  author:     Philip Thompson,    ($Author: phils $)                 */
/*              $Date: 88/01/30 17:51:28 $      $State: Exp $          */
/*                                                                     */
/*  based on:   Peter Jurgensen    12-June-1985                        */
/*                                                                     */
/*  note:       reads only the surface and curve definitions           */
/*              of the old CRL files                                   */
/***********************************************************************/
#ifndef LINT
static char ray_read_id[] =
        "$Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $";
#endif LINT

#include <stdio.h>
#include "rays.h"

#define toupper(c) (('a'<=(c) && (c)<='z') ? ((c)-('a'-'A')) : (c))
#define UEOfile {printf("Unexpected end of file.\n"); return;}

Point *point_ptr[MAXPNTS];
Object thisObject[MAXOBJ];
Properties properties[MAXOBJ];
int nObjs = EOL;                    /* total number of objects */
int nProps = EOL;                   /* total number of properties */
int points = 0;                     /* total number of points */


Object *readObj(fileName)
char *fileName;
{
    int obcode;
    FILE *fopen(), *inFile;             /* the file pointer */
    char  *tmp_name, *strrchr(), *strcpy(), *strcat(), *malloc();

    /* check for ".crl" extension on object data file
    */
    if (strcmp(strrchr(fileName,'.'),".crl") != 0)
        tmp_name = strcat(strcpy(malloc((unsigned)
                    strlen(fileName)+4),fileName),".crl");
    else
        tmp_name = fileName;

    if ((inFile=fopen(tmp_name,"r")) <= NULL)
       error("readObj: can't OPEN %s", tmp_name);

    /* read the header of the file */
    if ((obcode = fgetc(inFile)) != '>')
        error("readObj: not a crl file - no header","\0");
    else
       dalheader(inFile);
    /* loop through all the records in the file */
    while (((obcode = fgetc(inFile)) != EOF) && (nObjs < MAXOBJ)) {
        switch(toupper(obcode)) {
        case ';':               /* COMMENT */
            dalcomma(inFile);
            break;
        case 'V':               /* VERTEX */
            dalpoina(inFile);
            break;
        case 'P':               /* POLYGON */
            dalsurfa(inFile);
            break;
        case 'Q':               /* QUADRIC */
            dalcurva(inFile);
            break;
        case 'S':               /* SPHERE */
            dalsphera(inFile);
            break;
        case 'F':               /* LIBRARY FINISH */
            dalfinia(inFile);
            break;
        case 'E':               /* EDGE */
            daledgea(inFile);
            break;
        case '>':               /* HEADER */
            dalheader(inFile);
            break;
        default:
            error("readobj: bad object code %c\n", (char *)obcode);
        }
    }
    if (nObjs == MAXOBJ)
        fprintf(stderr,"Too many objects: %d / %d\n", nObjs, MAXOBJ);

    thisObject[nObjs].next = NULL;
    if (fclose(inFile) != NULL)
        error("readObj: error CLOSING inFile","\0");
    init_polys(thisObject);
    return(thisObject);
}


dalcomma(file_ptr)              /* DALCOMM */
FILE *file_ptr;
{
    char record[MAXSTR];

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
}


dalheader(file_ptr)             /* DALHEADER */
FILE *file_ptr;
{
    char record[MAXSTR];

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
}


dalpoina(file_ptr)              /* DALPOIN */
FILE *file_ptr;
{
    char record[MAXSTR];
    double Xreal, Yreal, Zreal;
    Point *makepnt();

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    if (sscanf(record+1,"%lf %lf %lf", &Xreal, &Yreal, &Zreal) != 3)
        error("DALPOIN: incorrect number of points %d",(char *)points);
    points++;
    point_ptr[points] = makepnt(Xreal, Yreal, Zreal);
}


daledgea(file_ptr)              /* DALEDGE */
FILE *file_ptr;
{
  char record[MAXSTR];

        if (fgets(record, MAXSTR-1, file_ptr) == NULL)
            UEOfile
}


dalsurfa(file_ptr)              /* DALSURF */
FILE *file_ptr;
{
    char record[MAXSTR], *malloc();
    int next, local_index;
    int vertices, finish;
    Point **vtx;
    Polygon *poly;

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    if (sscanf(record+1,"%d %*d %d",
                &vertices, &finish) != 2)
        error("DALSURF: incorrect vertex and finish","/0");

    vtx = (Point **)malloc((unsigned)vertices*sizeof(Point *));
    for (next=0; next < vertices; next++) {
        if (fgets(record, MAXSTR-1, file_ptr) == NULL)
            UEOfile
        if (sscanf(record,"%d", &local_index) != 1)
            error("DALSURF: incorrect point index","/0");
        vtx[next] = point_ptr[local_index];
    }
    poly = (Polygon *)malloc((unsigned)sizeof(Polygon));
    poly->numvtx = vertices;
    poly->vertex = vtx;

    nObjs++;
    thisObject[nObjs].objType = POLYGON;
    thisObject[nObjs].objIdent = nObjs;
    thisObject[nObjs].ptr.polygon = poly;
    thisObject[nObjs].theseProps = &(properties[finish-1]);
    thisObject[nObjs].next = &(thisObject[nObjs+1]);
}


dalcurva(file_ptr)          /* DALCURV */
FILE *file_ptr;
{
    char    record[MAXSTR], *malloc();
    Quadric *surf;
    int     finish;

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    if (sscanf(record+1,"%d", &finish) != 1)
        error("DALCURV: incorrect finish argument","/0");
    surf = (Quadric *)malloc (sizeof(Quadric));
    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    if (sscanf(record,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
            &(surf->a),&(surf->b),&(surf->c),&(surf->d),&(surf->e),
            &(surf->f),&(surf->g),&(surf->h),&(surf->j),&(surf->k))!= 10)
        error("DALCURV: incorrect surface arguments","/0");
    nObjs++;
    thisObject[nObjs].objIdent = nObjs;
    thisObject[nObjs].objType = QUADRIC;
    thisObject[nObjs].theseProps = &(properties[finish-1]);
    thisObject[nObjs].ptr.quadric = surf;
    thisObject[nObjs].next = &(thisObject[nObjs+1]);
}


dalsphera(file_ptr)         /* DALSPHERA */
FILE *file_ptr;
{
    char    record[MAXSTR], *malloc();
    Sphere  *sphr_ptr;
    int     finish;

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    if (sscanf(record+1,"%d", &finish) != 1)
        error("DALSPHERA: incorrect finish argument","\0");
    sphr_ptr = (Sphere *)malloc((unsigned)sizeof(Sphere));
    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    if (sscanf(record,"%lf %lf %lf %lf", &sphr_ptr->center.x,
            &sphr_ptr->center.y, &sphr_ptr->center.z,
            &sphr_ptr->radius) != 4)
        error("DALSPHERA: incorrect sphere description", "\0");
    nObjs++;
    thisObject[nObjs].objIdent = nObjs;
    thisObject[nObjs].objType = SPHERE;
    thisObject[nObjs].theseProps = &(properties[finish-1]);
    thisObject[nObjs].ptr.sphere = sphr_ptr;
    thisObject[nObjs].next = &(thisObject[nObjs+1]);
}


dalfinia(file_ptr)          /* DALFINI */
FILE *file_ptr;
{
    char record[MAXSTR];
    float dRed, dGreen, dBlue;
    int type;

    if (fgets(record, MAXSTR-1, file_ptr) == NULL)
        UEOfile
    nProps++;
    if (sscanf(record+1,"%f %f %f   %f %f %f %f %d",
             &dRed, &dGreen, &dBlue,
             &(properties[nProps].kDiff), &(properties[nProps].kSpec),
             &(properties[nProps].kTrans), &(properties[nProps].kRefr),
             &type) != 8)
        error("DALFINI: incorrect finish count","/0");
    properties[nProps].color.red = dRed;
    properties[nProps].color.green = dGreen;
    properties[nProps].color.blue = dBlue;
    properties[nProps].type = (unsigned)type;
}


Point *makepnt(x,y,z)
double x,y,z;
{
    char *malloc();
    Point *ptr;

    if ((ptr=(Point *)malloc(sizeof(Point))) == NULL)
        error("makepnt: malloc() error","/0");
    ptr->x = x;
    ptr->y = y;
    ptr->z = z;
    return (ptr);
}


/* Initialize the object list, calculate the plane equations for all the
 * polygons in the data base, and calculate the bounding cubes for each
 * polygon.
 */
```

```
init_polys(firstobj)
Object *firstobj;
{
    Object  *objptr;
    Polygon  *polyptr;
    Point  **vtx;
    int  i;
    double  a,b,c,d, x1,y1,z1, x2,y2,z2, x3,y3,z3, normx,normy,normz;
    double  minx, miny, minz, maxx, maxy, maxz;
#ifdef DEBUG
    double  normd;
#endif DEBUG


    /* Calculate the plane equation of the polygons */
    objptr = firstobj;
    while (objptr != NULL) {

#ifdef DEBUG
        printf("Object Ident: %d\n", objptr->objIdent);
        printf("type %c  rgb %.2f %.2f %.2f\n", objptr->type,
                objptr->theseProps->color.red, objptr->theseProps->
                color.green, objptr->theseProps->color.blue);
#endif DEBUG

        if (objptr->objType != POLYGON) {   /* if obj isn't a poly */
            objptr = objptr->next;          /* get the next one   */
            continue;
        }
        polyptr = objptr->ptr.polygon;        /* Get a ptr to poly   */
        vtx = polyptr->vertex;

        /* Calculate bounding cube of the polygon */
        minx = miny = minz = HUGEREAL;
        maxx = maxy = maxz = -HUGEREAL;
        for (i = 0; i < polyptr->numvtx; i++) {
            minx = min(vtx[i]->x, minx);
            miny = min(vtx[i]->y, miny);
            minz = min(vtx[i]->z, minz);
            maxx = max(vtx[i]->x, maxx);
            maxy = max(vtx[i]->y, maxy);
            maxz = max(vtx[i]->z, maxz);
        }
        objptr->objmin.x = minx - EPSILON;
        objptr->objmin.y = miny - EPSILON;
        objptr->objmin.z = minz - EPSILON;
        objptr->objmax.x = maxx + EPSILON;
        objptr->objmax.y = maxy + EPSILON;
        objptr->objmax.z = maxz + EPSILON;

        /* Get vertices of a polygon */
        x1 = vtx[0]->x; y1 = vtx[0]->y; z1 = vtx[0]->z;
        x2 = vtx[1]->x; y2 = vtx[1]->y; z2 = vtx[1]->z;
        x3 = vtx[2]->x; y3 = vtx[2]->y; z3 = vtx[2]->z;
        /* Calculate the plane equation */
        a = y1*(z2-z3) + y2*(z3-z1) + y3*(z1-z2);
        b = -(x1*(z2-z3) + x2*(z3-z1) + x3*(z1-z2));
        c = x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2);
        d = x1*(z2*y3-y2*z3) + x2*(z3*y1-z1*y3) + x3*(z1*y2-z2*y1);

        /* Calculate the normal vector */
        x1 -= x2; y1 -= y2; z1 -= z2;
        x3 -= x2; y3 -= y2; z3 -= z2;

        normx = (y1*z3 - y3*z1);
        normy = (x3*z1 - x1*z3);
        normz = (x1*y3 - x3*y1);

        /* Adjust the sign of the plane equation so that it
         * agrees with the normal vector
         */
        if  ((normx > 0.0 && a > 0.0) ||
             (normx < 0.0 && a < 0.0)) {
            polyptr->a = a;
            polyptr->b = b;
            polyptr->c = c;
            polyptr->d = d;
        }
        else if ((normy > 0.0 && b > 0.0) ||
                 (normy < 0.0 && b < 0.0)) {
            polyptr->a = a;
            polyptr->b = b;
            polyptr->c = c;
            polyptr->d = d;
        }
        else if ((normz > 0.0 && c > 0.0) ||
                 (normz < 0.0 && c < 0.0)) {
            polyptr->a = a;
            polyptr->b = b;
            polyptr->c = c;
            polyptr->d = d;
        }
        else {
            polyptr->a = -a;
            polyptr->b = -b;
            polyptr->c = -c;
            polyptr->d = -d;
        }
#ifdef DEBUG
    normd = -(normx*x2 + normy*y2 + normz*z2);
    printf("a:%8.2f b:%8.2f c:%8.2f d:%8.2f\n",polyptr->a,polyptr->b,
        polyptr->c,polyptr->d);
    printf("x:%8.2f y:%8.2f z:%8.2f d:%8.2f\n",normx,normy,normz,normd);
#endif

        objptr = objptr->next;
    } /* Done with all the objects */
}
```

```
/*** end read.c ***/
```

```
/*************************************************************************/
/*                                                                       */
/*  purpose:     housekeeping functions                                  */
/*                                                                       */
/*  author:      Philip R. Thompson,  ($Author: phils $)                 */
/*               $Date: 88/01/30 17:51:28 $     $State: Exp $            */
/*************************************************************************/
#ifndef LINT
static char ray_utils_id[] =
        "$Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $";
#endif LINT

#include <stdio.h>
#include "rays.h"
#include "ImageHeader.h"

#ifndef NONUNIX
#include <pwd.h>
#include <sys/time.h>
#include <sys/types.h>
#endif NONUNIX

static char  view_str[100];
Light  *light;
int  numlights, seg_length, seg_num, seg_start;
byte  *seg_array;
Point  eye;
Point  view_pnt;
double  z_rotate, viewangle;
float  Iamb;

/* this allocates internal memory for each ray
*/
Ray *newRay()
{
    char *malloc();
    Ray  *this;

    if ((this=(Ray *)malloc(sizeof(Ray))) == NULL)
        error("newRay: malloc() error","\0");
    bzero((char *)this, sizeof(Ray));
    this->ident = EOL;
    this->t = HUGEREAL;
    return (this);
}


error(s1, s2)
char *s1, *s2;
{
    extern int errno, sys_nerr;
    extern char *sys_errlist[], *progName;
    FILE  *fopen(), *fp;
    char  fname[16], *err_file, *strcat(), *strcpy();
    err_file = strcat(strcpy(fname, progName), ".err");
    if ((fp=fopen(err_file, "w")) == NULL)
        exit(2);

    fprintf(fp,"%s ERROR->\n", progName);
    fprintf(fp, s1, s2);
    fprintf(stderr,"%c%s ERROR->\n%c", BELL, progName, BELL);
    fprintf(stderr, s1, s2);
    if (errno > 0  && errno < sys_nerr) {
        fprintf(fp," (%s)", sys_errlist[errno]);
        fprintf(stderr," (%s)", sys_errlist[errno]);
    }
    fprintf(fp,"\n");
    fprintf(stderr,"\n");
    exit(1);
}


int ecreat(ofilename, pmode)
char *ofilename;
int pmode;
{
    int ofildes;

    if ((ofildes = creat(ofilename, pmode)) <= NULL)
        error ("can't CREATE %s", ofilename);
    return (ofildes);
}


#ifdef SRAY
init_from_header(infile, x, y)
FILE  *infile;
int *x, *y;
{
    int  i;
    ImageHeader  head;
    long file_pos, ftell();

    if (fread((char *)&head, sizeof(head), 1, infile) != 1)
        error("Unable to read file header.","\0");
    if (atoi(head.header_size) != sizeof(ImageHeader))
        error("Header size mismatch", "\0");
    if ((atoi(head.file_version)) != IMAGE_VERSION)
        error("Incorrect Image_file Version.","\0");
    *x = atoi(head.pixmap_width);
    *y = atoi(head.pixmap_height);
    fprintf(stderr,"\nResolution: %d x %d\n", *x, *y);

    file_pos = ftell(infile);
    if (fseek(infile, 152L, 0) == -1)
        error("Improper fseek call", "\0");

     (void)fscanf(infile,"%lf %lf %lf %lf %lf %lf %lf %lf %d %f",
        &eye.x, &eye.y, &eye.z, &view_pnt.x, &view_pnt.y, &view_pnt.z,
        &viewangle, &z_rotate, &numlights, &Iamb);

    light = (Light *)malloc((unsigned)numlights*sizeof(Light));
    for (i=0; i < numlights; i++) {
        (void)fscanf(infile,"%lf %lf %lf %f", &light[i].x,
            &light[i].y, &light[i].z, &light[i].intensity);
        fprintf(stderr,"source[%d]: %4.1lf %4.1lf %4.1lf  %4.2f\n", i,
            light[i].x, light[i].y, light[i].z, light[i].intensity);
    }
    (void)fseek(infile, file_pos, 0);
}
#endif SRAY


init_ray(infile, x, y)
FILE *infile;
double  x, y;
{
    int  i;
    char tmp[48], *strncat(), *strncpy();

    Get_viewdata(infile);
    init_ViewTrans(x, y);

    (void)sprintf(tmp,
        "%.1lf %.1lf %.1lf\n%.1lf %.1lf %.1lf\n%.1lf\n%.1lf\n%d %.1f\n",
        eye.x, eye.y, eye.z, view_pnt.x, view_pnt.y, view_pnt.z,
        viewangle, z_rotate, numlights, Iamb);
    (void)strncpy(view_str, tmp, strlen(tmp));
    for (i=0; i < numlights; i++) {
        (void)sprintf(tmp,"%.1lf %.1lf %.1lf %.1f\n", light[i].x,
            light[i].y, light[i].z, light[i].intensity);
        (void)strncat(view_str, tmp, strlen(view_str));
        fprintf(stderr,"source[%d]: %4.1lf %4.1lf %4.1lf  %4.2f\n", i,
            light[i].x, light[i].y, light[i].z, light[i].intensity);
    }
    (void)sprintf(tmp,"%d %d %d\n", seg_start,seg_num,seg_length);
    (void)strncat(view_str, tmp, strlen(view_str));
}

Get_viewdata(infile)
FILE *infile;
{
    int  i;
    char *malloc();
    FILE *outfile, *fopen();

    if (infile == stdin)
        outfile = stdout;
    else
        outfile = fopen("/dev/null", "w");

    fprintf(outfile,"Enter eye coordinates <from>: ");
    (void)fscanf(infile,"%lf  %lf  %lf", &eye.x, &eye.y, &eye.z);
    fprintf(outfile,"Enter view coordinates <to>: ");
    (void)fscanf(infile,"%lf %lf %lf", &view_pnt.x, &view_pnt.y,
            &view_pnt.z);
    fprintf(outfile,"Enter viewangle <45>: ");
    (void)fscanf(infile,"%lf", &viewangle);
    fprintf(outfile,"Enter screen orientation <0>: ");
    (void)fscanf(infile,"%lf", &z_rotate);
    fprintf(outfile,"Enter number of light sources <0-%d>: ", MAXLIGHT);
    (void)fscanf(infile,"%d %f",&numlights, &Iamb);
    if (numlights < 0  && numlights > MAXLIGHT)
        error("Incorrect light sources. MAX is %d", (char *)MAXLIGHT);
    if ((light = (Light *)malloc((unsigned)numlights *
            sizeof(Light))) == NULL)
        error("light & malloc() no more light space", "\0");
    for (i=0; i < numlights; i++) {
        fprintf(outfile,"coords & intensity of %d <5 5 0  1.0>: ",i+1);
        (void)fscanf(infile,"%lf %lf %lf %f",&light[i].x,
            &light[i].y,&light[i].z,&light[i].intensity);
    }
    fprintf(outfile, "Start, Number & lengths of time segments: ");
    (void)fscanf(infile,"%d %d %d",&seg_start, &seg_num, &seg_length);
    if (seg_num > MAX_SEG)
        error("Too many segments. MAX is %d", (char *)MAX_SEG);
}

init_ViewTrans(x, y)
double  x, y;
{
    Point  hi, low;        /* vectors to diagonal viewport corners */
    Point  view_dir;
    double PolarAngle(), VectorsAngle();
    double sqrt(), tan(), cos();
    double viewfactor, rotate;
    extern double  TransM[4][4];

    viewfactor = toRadians(viewangle/2.0);
    viewfactor = sqrt(sqr(x)+sqr(y)) / tan(viewfactor);

    hi.x = x/viewfactor;  low.x = -x/viewfactor;
    hi.y = y/viewfactor;  low.y = -y/viewfactor;
    hi.z = low.z = 1.0;

    view_dir.x = view_pnt.x - eye.x;          /* viewing direction */
    view_dir.y = view_pnt.y - eye.y;
    view_dir.z = view_pnt.z - eye.z;
    viewangle = toDegrees(VectorsAngle(&hi, &low));

    fprintf(stderr,"\nEye point: %5.2lf %5.2lf %5.2lf\n",
            eye.x, eye.y, eye.z);
    fprintf(stderr,"View dir:  %5.2lf %5.2lf %5.2lf\n",
            view_dir.x, view_dir.y, view_dir.z);
    fprintf(stderr,"viewAngle: %.2lf degrees.\n", viewangle);

    /* set viewport transformation matrix
    */
    ResetMatrix(TransM);
    scaleXYZ(1.0/viewfactor, 1.0/viewfactor, 1.0);
    fprintf(stderr,"rotationZ = %4.2lf deg.\n",z_rotate);
    rotateZ(toRadians(z_rotate));
    rotate = PolarAngle(view_dir.z, view_dir.x);
    fprintf(stderr,"rotationY = %4.2lf deg.\n",toDegrees(rotate));
    rotateY(rotate);
    view_dir.z *= cos(rotate);
    rotate = -PolarAngle(view_dir.z, view_dir.y);
    fprintf(stderr,"rotationX = %4.2lf deg.\n",toDegrees(rotate));
    rotateX(rotate);
}



init_outFile(outfile, width, height, color)
int outfile, width, height, color;
{
    register int  x;
    extern char  *progName;
    ImageHeader  header;
#ifndef NONUNIX
    char  *time_ptr, *login_ptr, *ctime(), *getlogin();
    struct passwd  *getpwnam(), *getpwuid(), *pwd;
    struct timeval  time_val;
    struct timezone  time_zone;
    uid_t  getuid();

    (void)gettimeofday(&time_val, &time_zone);
    time_ptr = ctime(&time_val.tv_sec);
    time_ptr[strlen(time_ptr)-1] = '\0';
    login_ptr = getlogin();
    if (*login_ptr)
        pwd = getpwnam(login_ptr);
    else
        pwd = getpwuid((int)getuid());

    bzero((char *)&header, sizeof(ImageHeader));
    (void)sprintf(header.creator,"%s", pwd->pw_gecos);
    (void)sprintf(header.date,"%s", time_ptr);
#endif NONUNIX

    (void)sprintf(header.file_version,"%d", IMAGE_VERSION);
    (void)sprintf(header.header_size,"%d", sizeof(ImageHeader));
    (void)sprintf(header.pixmap_width,"%d", width);
    (void)sprintf(header.pixmap_height,"%d", height);
    (void)sprintf(header.num_colors,"%d", MAX_RGB+1);
    (void)sprintf(header.creat_program,"%s", progName);
    (void)sprintf(header.view_info,"%s", view_str);

    switch (color) {
    case GREY:
        for (x = 0; x <= MAX_RGB; x++)
            header.c_map[x][0] = header.c_map[x][1] =
                header.c_map[x][2] = (byte)x;
        break;
    case RED:
        for (x = 0; x <= MAX_RGB; x++) {
            header.c_map[x][0] = (byte)x;
            header.c_map[x][1] = header.c_map[x][2] = 0;
        }
        break;
    case GREEN:
        for (x = 0; x <= MAX_RGB; x++) {
            header.c_map[x][1] = (byte)x;
            header.c_map[x][0] = header.c_map[x][2] = 0;
        }
        break;
    case BLUE:
        for (x = 0; x <= MAX_RGB; x++) {
            header.c_map[x][2] = (byte)x;
            header.c_map[x][0] = header.c_map[x][1] = 0;
        }
        break;
    }
    writeBytes(outfile, (char *)&header, sizeof(ImageHeader));
}

runl_encode (outfile, scanline, xres)
int outfile, xres;
byte *scanline;
{
    byte runlength, lastcolor;
    int i;

    runlength = 0;
    lastcolor = scanline[0];
    for (i=1; i < xres; i++) {
        if ((runlength == 255) || (scanline[i] != lastcolor)) {
            writeBytes (outfile, (char *)&runlength, sizeof(byte));
            writeBytes (outfile, (char *)&lastcolor, sizeof(byte));
            runlength = 0;
            lastcolor = scanline[i];
        } else
            runlength++;
    }
    writeBytes (outfile, (char *)&runlength, sizeof(byte));
    writeBytes (outfile, (char *)&lastcolor, sizeof(byte));
}


writeBytes(fd,buf,nbytes)
int fd, nbytes;
char *buf;
{
    if (write(fd, buf, nbytes) != nbytes)
        error("problem WRITING to outFile", "\0");
}

/***  end utils.c  ***/
```

```
/*  Author: Philip R. Thompson
 *  Address:  phils@athena.mit.edu, 9-514
 *  Note:  size of header is 1024 bytes
 *  $Header: appendix1.mss,v 1.2 88/01/30 17:51:28 phils Exp $
 *  $Date: 88/01/30 17:51:28 $
*/

#define IMAGE_VERSION    2
#define MAX_INDEX        256

typedef struct ImageHeader {
    char file_version[8];   /* header version  */
    char header_size[8];    /* Size of file header (bytes)  */
    char pixmap_width[8];   /* Width of the raster image  */
    char pixmap_height[8];  /* Height of the raster imgage (lines)  */
    char num_colors[16];    /* actual number of colors in file  */
    char data_size[16];     /* size of runlength encoded data (bytes) */
    char creator[48];       /* Name of who made it  */
    char date[32];          /* Date and time image was made  */
    char creat_program[8];  /* program that created this file  */
    char view_info[104];    /* viewing parameters for this image */
    unsigned char c_map[MAX_INDEX][3];  /* RG&B values of the indices */
} ImageHeader;

typedef struct Line {
    unsigned short x, y;
    unsigned char length, color;
} Line;

/*** end ImageHeader.h ***/
```

```
/***********************************************************************/
/*                                                                     */
/*  purpose:    header file for ray casting routines.                 */
/*                                                                     */
/*  author:     Philip R. Thompson  ($Author: phils $)                */
/*              $Date: 88/01/30 17:51:28 $     $State: Exp $           */
/*                                                                     */
/*  based on:   JTWhitted, copyrighted 1984                            */
/***********************************************************************/

#define MAXOBJ   1000    /* maximum number of surface elements */
#define MAXPNTS  2000    /* maximum number of vertices for surfaces */
#define MAXRES   1024    /* maximum horizontal resolution    */
#define MAXSTR   256     /* arbitrary maximum string length  */
#define MAXLEVEL 10      /* maximun depth for this branch of tree */
#define MAXLIGHT 4       /* maximum number of light sources */
#define MAX_SEG  20      /* maximum number of ray segments and files */
#define MAX_RGB  255     /* max. intensities for r,g & b  */

#define EOL          -1
#define HUGEREAL     100000000.0
#define EPSILON      0.0000001

#define PRIMARY      0x0
#define REFLECTED    0x1
#define REFRACTED    0x2
#define SHADOW       0x4
#define RED          1
#define GREEN        2
#define BLUE         3
#define GREY         4

#define PI           3.14159265358979323846
#define DEG          57.29577951308232087680
#define BELL         7
#define FALSE        0
#define TRUE         1

#define sqr(x)       (x * x)
#define trunc(x)     ((int)((x)+0.5))
#define abs(x)       (x > 0 ? x : -x)
#define max(x,y)     (x > y ? x : y)
#define min(x,y)     (x < y ? x : y)
#define toRadians(theta) ((theta) / DEG)
#define toDegrees(theta) ((theta) * DEG)


typedef unsigned char  byte;

typedef enum {POLYGON, QUADRIC, SPHERE, LINE, POINT} ObjType;

typedef struct Point {       /* point and vector structures */
        double  x,y,z;
} Point;
```

```
typedef struct Pixel {
        float  red, green, blue;      /* R,G&B color components  */
} Pixel;


/*  each quadric surface element is defined by a coefficient
*   array for general quadric surface of form:
*/
typedef  struct Quadric {
         double a,b,c,d,    /* a*x*x + b*x*y + c*x*z + d*x     */
                  e,f,g,    /*       + e*y*y + f*y*z + g*y     */
                    h,j,    /*               + h*z*z + j*z     */
                      k;    /*                       + k = 0;  */
} Quadric;


/*  polygon description contains lists of plane coefficients, points,
*   and various flags
*/
typedef struct Polygon {
        int numvtx;             /*  number of vertices  */
        double a,b,c,d;         /*  plane equation  */
        Point  **vertex;        /*  table of vertex indixes */
        struct Polygon  *next;  /*  polygon list pointer  */
} Polygon;


typedef struct Sphere {
        Point  center;          /* center of sphere */
        double radius;          /* radius "    "    */
} Sphere;


typedef struct Light {
        double  x,y,z;          /* location of light source */
        float   intensity;      /* color intensity of light */
} Light;


typedef struct Properties {
        Pixel   color;          /* color properties */
        float   kDiff;          /* diffuse reflection coefficient */
        float   kSpec;          /* specular    "            "       */
        float   kTrans;         /* transmission coefficient */
        float   kRefr;          /* index of refraction between media */
        unsigned type;          /* gross class. of surface props - */
} Properties;                    /* reflective, transparent or both */


/*  object description contains lists of
*   surfaces and surface properties
*/
typedef struct  Object {
        struct Object  *next;          /* object list ptr  */
        int            objIdent;       /* object indentifier  */
        Properties     *theseProps;    /* list of surface properties */
        ObjType        objType;        /* type of object  */
        union {
           Polygon *polygon;           /* pointers to surface coefs */
           Quadric *quadric;
           Sphere  *sphere;
        } ptr;
        Point objmin;                   /* bounding box of object   */
        Point objmax;
} Object;


/*  each ray is a straight line defined by:
*       ray = origin + t*direction
*   starting from the viewer's position, a tree of rays is generated
*   by a recursive ray casting a procedure.  shadow rays for each
*   point of intersection are grouped in a linked list
*/
typedef  struct Ray {
         unsigned  rayType;     /*  reflected, refracted or shadow */
         int  ident;            /*  identifier of intersected surface*/
         short  level;          /*  depth of this branch of ray tree  */
         Point  origin;         /*  ray origin  */
         Point  direction;      /*  ray direction  */
         Point  head;           /*  coord at intersection point */
         Point  normal;         /*  surface normal at intersect point */
         Pixel  intensity;      /*  accumulated reflection  */
         double t;              /*  distance to nearest intersect pnt*/
         double ctotal;         /*  cumulative distance travelled */
         double cSpec;          /*  cumulative specular attenuation */
         double incDot;         /*  dot product of incident or reflect
                                        ray and surface normal */
        struct  Ray *parent;    /*  ptr to parent  */
        struct  Ray *reflected; /*  ptr to reflected ray  */
        struct  Ray *refracted; /*  ptr to refracted ray  */
        struct  Ray *shadow;    /*  ptr to shadow ray  */
} Ray;

/***  end rays.h  ***/
```

Appendix B
Geometric Spatial Databases

The following is a CRL file describing the geometry and surface properties of Chevalier Auditorium.

```
> CRL VERSION 1
F  0.8 0.8 0.8  0.2 0.8 0.0 0.0 5
V  -29.50 0.00 -2.00
V  -29.50 0.00 2.00
V  29.50 0.00 2.00
V  29.50 0.00 -2.00
V  23.00 0.00 -2.00
V  15.00 0.00 -6.00
V  5.00 0.00 -8.00
V  -5.00 0.00 -8.00
V  -15.00 0.00 -6.00
V  -23.00 0.00 -2.00
P  10 0 1
1
2
3
4
5
6
7
8
9
10
V  -21.00 1.00 28.00
V  21.00 1.00 28.00
P  4 0 1
2
11
12
3
V  -21.00 27.50 28.00
V  -29.50 44.50 2.00
P  4 0 1
2
11
13
14
V  21.00 27.50 28.00
V  29.50 44.50 2.00
P  4 0 1
3
12
15
16
```

```
P  4 0 1
11
13
15
12
V  -21.00 14.00 28.00
V  -22.40 16.00 24.00
V  22.40 16.00 24.00
V  21.00 14.00 28.00
P  4 0 1
17
18
19
20
P  4 0 1
13
14
16
15
V  -29.50 -4.00 -2.00
V  -29.50 0.00 0.00
V  -29.50 -4.00 0.00
P  4 0 1
21
1
22
23
V  29.50 -4.00 -2.00
V  29.50 0.00 0.00
V  29.50 -4.00 0.00
P  4 0 1
24
4
25
26
V  -23.00 -4.00 -2.00
P  4 0 1
21
1
10
27
V  23.00 -4.00 -2.00
P  4 0 1
24
4
5
28
V  -15.00 -4.00 -6.00
P  4 0 1
27
10
9
29
V  15.00 -4.00 -6.00
P  4 0 1
```

```
28
5
6
30
V  -5.00 -4.00 -8.00
P  4 0 1
29
9
8
31
V  5.00 -4.00 -8.00
P  4 0 1
30
6
7
32
P  4 0 1
31
8
7
32
F  0.3 0.3 0.3  0.3 0.2 0.0 0.0 5
V  -22.40 30.00 24.00
V  22.40 30.00 24.00
P  4 0 2
18
33
34
19
F  0.7 0.7 0.7  0.5 0.1 0.0 0.0 5
V  -29.50 51.00 0.00
P  4 0 3
22
35
14
2
;; front upper slope
V  29.50 51.00 0.00
P  4 0 3
25
36
16
3
P  4 0 3
14
35
36
16
F  0.3 0.3 0.3  0.5 0.2 0.0 0.0 5
V  -38.50 7.00 -3.50
V  -38.50 10.50 -3.50
V  -34.00 10.50 -3.50
V  -34.00 7.00 -3.50
P  4 0 4
37
```

```
38
39
40
V  38.50 7.00 -3.50
V  38.50 10.50 -3.50
V  34.00 10.50 -3.50
V  34.00 7.00 -3.50
P  4 0 4
41
42
43
44
V  -29.50 10.50 -6.50
V  -29.50 7.00 -6.50
P  4 0 4
40
39
45
46
V  29.50 10.50 -6.50
V  29.50 7.00 -6.50
P  4 0 4
44
43
47
48
V  -29.50 12.50 -99.00
V  -29.50 9.00 -99.00
P  4 0 4
46
45
49
50
V  29.50 12.50 -99.00
V  29.50 9.00 -99.00
P  4 0 4
48
47
51
52
V  -25.50 12.50 -104.50
V  -25.50 9.00 -104.50
P  4 0 4
50
49
53
54
V  25.50 12.50 -104.50
V  25.50 9.00 -104.50
P  4 0 4
52
51
55
56
V  -16.00 12.50 -108.00
```

```
V  -16.00 9.00 -108.00
P  4 0 4
54
53
57
58
V  16.00 12.50 -108.00
V  16.00 9.00 -108.00
P  4 0 4
56
55
59
60
P  4 0 4
58
57
59
60
V  -40.00 9.00 -99.00
V  -40.00 7.00 -3.50
P  5 0 4
46
50
61
62
40
V  40.00 9.00 -99.00
V  40.00 7.00 -3.50
P  5 0 4
48
52
63
64
44
V  40.00 9.00 -130.50
V  -40.00 9.00 -130.50
P  10 0 4
65
66
61
50
54
58
60
56
52
63
V  -38.50 19.50 -3.50
V  -38.50 23.00 -3.50
V  -34.00 23.00 -3.50
V  -34.00 19.50 -3.50
P  4 0 4
67
68
69
```

```
70
V  38.50 19.50 -3.50
V  38.50 23.00 -3.50
V  34.00 23.00 -3.50
V  34.00 19.50 -3.50
P  4 0 4
71
72
73
74
V  -31.00 23.00 -5.50
V  -31.00 19.50 -5.50
P  4 0 4
70
69
75
76
V  31.00 23.00 -5.50
V  31.00 19.50 -5.50
P  4 0 4
74
73
77
78
V  -31.00 24.50 -108.00
V  -31.00 21.00 -108.00
P  4 0 4
76
75
79
80
V  31.00 24.50 -108.00
V  31.00 21.00 -108.00
P  4 0 4
78
77
81
82
V  -25.50 24.50 -114.00
V  -25.50 21.00 -114.00
P  4 0 4
80
79
83
84
V  25.50 24.50 -114.00
V  25.50 21.00 -114.00
P  4 0 4
82
81
85
86
V  -19.00 24.50 -117.00
V  -19.00 21.00 -117.00
P  4 0 4
```

```
84
83
87
88
V  19.00 24.50 -117.00
V  19.00 21.00 -117.00
P  4 0 4
86
85
89
90
P  4 0 4
88
87
89
90
V  -40.00 21.00 -108.00
V  -40.00 19.50 -3.50
P  5 0 4
76
80
91
92
70
V  40.00 21.00 -108.00
V  40.00 19.50 -3.50
P  5 0 4
78
82
93
94
74
V  40.00 21.00 -130.50
V  -40.00 21.00 -130.50
P  10 0 4
95
96
91
80
84
88
90
86
82
93
F  0.1 0.1 0.1  0.9 0.0 0.0 0.0 5
V  -40.00 -4.00 -36.00
V  -40.00 -4.00 -0.00
V  40.00 -4.00 -0.00
V  40.00 -4.00 -36.00
P  4 0 5
97
98
99
100
```

```
V  -40.00 -0.50 -101.00
V  40.00 -0.50 -101.00
P  4 0 5
101
97
100
102
V  -40.00 2.00 -130.50
V  40.00 2.00 -130.50
P  4 0 5
103
101
102
104
V  38.50 22.00 -117.00
V  38.50 28.00 -138.00
V  -38.50 28.00 -138.00
V  -38.50 22.00 -117.00
P  4 0 5
105
106
107
108
V  -31.00 21.50 -117.00
V  -38.50 24.50 -117.00
V  -31.00 20.00 -3.50
P  4 0 5
109
110
68
111
V  31.00 21.50 -117.00
V  38.50 24.50 -117.00
V  31.00 20.00 -3.50
P  4 0 5
112
113
72
114
V  38.50 9.50 -108.00
V  38.50 15.00 -130.50
V  -38.50 15.00 -130.50
V  -38.50 9.50 -108.00
P  4 0 5
115
116
117
118
V  -29.50 9.50 -108.00
V  -38.50 12.50 -108.00
V  -29.50 7.50 -3.50
P  4 0 5
119
120
38
```

```
121
V  29.50 9.50 -108.00
V  38.50 12.50 -108.00
V  29.50 7.50 -3.50
P  4 0 5
122
123
42
124
F  0.8 0.8 0.8  0.4 0.6 0.0 0.0 5
V  -38.50 22.00 0.00
V  -38.50 56.00 0.00
V  -38.50 56.00 -138.00
V  -38.50 22.00 -138.00
P  4 0 6
125
126
127
128
V  38.50 22.00 0.00
V  38.50 56.00 0.00
V  38.50 56.00 -138.00
V  38.50 22.00 -138.00
P  4 0 6
129
130
131
132
F  0.8 0.8 0.8  0.4 0.6 0.0 0.0 5
V  -38.50 -4.00 0.00
V  -38.50 51.00 0.00
P  4 0 7
133
134
35
23
V  38.50 -4.00 0.00
V  38.50 51.00 0.00
P  4 0 7
135
136
36
26
V  -38.50 54.00 0.00
V  38.50 54.00 0.00
P  4 0 7
134
137
138
136
F  0.8 0.8 0.8  0.3 0.7 0.0 0.0 5
V  -38.50 22.00 -25.00
V  -38.50 -4.00 -25.00
P  4 0 8
133
```

```
125
139
140
V  38.50 22.00 -25.00
V  38.50 -4.00 -25.00
P  4 0 8
135
129
141
142
V  -40.00 22.00 -25.00
V  -40.00 -4.00 -25.00
P  4 0 8
140
139
143
144
V  40.00 22.00 -25.00
V  40.00 -4.00 -25.00
P  4 0 8
142
141
145
146
V  -40.00 22.00 -30.00
V  -40.00 -4.00 -30.00
P  4 0 8
144
143
147
148
V  40.00 22.00 -30.00
V  40.00 -4.00 -30.00
P  4 0 8
146
145
149
150
V  -38.50 22.00 -30.00
V  -38.50 -4.00 -30.00
P  4 0 8
148
147
151
152
V  38.50 22.00 -30.00
V  38.50 -4.00 -30.00
P  4 0 8
150
149
153
154
V  -38.50 22.00 -44.00
V  -38.50 -4.00 -44.00
P  4 0 8
```

```
152
151
155
156
V  38.50 22.00 -44.00
V  38.50 -4.00 -44.00
P  4 0 8
154
153
157
158
V  -38.50 -3.50 -44.00
V  -40.00 22.00 -44.00
V  -40.00 -3.50 -44.00
P  4 0 8
159
155
160
161
V  38.50 -3.50 -44.00
V  40.00 22.00 -44.00
V  40.00 -3.50 -44.00
P  4 0 8
162
157
163
164
V  -40.00 22.00 -49.00
V  -40.00 -3.50 -49.00
P  4 0 8
161
160
165
166
V  40.00 22.00 -49.00
V  40.00 -3.50 -49.00
P  4 0 8
164
163
167
168
V  -38.50 22.00 -49.00
V  -38.50 -3.50 -49.00
P  4 0 8
166
165
169
170
V  38.50 22.00 -49.00
V  38.50 -3.50 -49.00
P  4 0 8
168
167
171
172
```

```
V  -38.50 -3.00 -49.00
V  -38.50 22.00 -63.00
V  -38.50 -2.50 -63.00
P  4 0 8
173
169
174
175
V  38.50 -3.00 -49.00
V  38.50 22.00 -63.00
V  38.50 -2.50 -63.00
P  4 0 8
176
171
177
178
V  -40.00 22.00 -63.00
V  -40.00 -2.50 -63.00
P  4 0 8
175
174
179
180
V  40.00 22.00 -63.00
V  40.00 -2.50 -63.00
P  4 0 8
178
177
181
182
V  -40.00 22.00 -68.00
V  -40.00 -2.50 -68.00
P  4 0 8
180
179
183
184
V  40.00 22.00 -68.00
V  40.00 -2.50 -68.00
P  4 0 8
182
181
185
186
V  -38.50 22.00 -68.00
V  -38.50 -2.50 -68.00
P  4 0 8
184
183
187
188
V  38.50 22.00 -68.00
V  38.50 -2.50 -68.00
P  4 0 8
186
```

```
185
189
190
V  -38.50 -2.00 -68.00
V  -38.50 22.00 -83.00
V  -38.50 -1.50 -83.00
P  4 0 8
191
187
192
193
V  38.50 -2.00 -68.00
V  38.50 22.00 -83.00
V  38.50 -1.50 -83.00
P  4 0 8
194
189
195
196
V  -40.00 22.00 -83.00
V  -40.00 -1.50 -83.00
P  4 0 8
193
192
197
198
V  40.00 22.00 -83.00
V  40.00 -1.50 -83.00
P  4 0 8
196
195
199
200
V  -40.00 22.00 -88.00
V  -40.00 -1.50 -88.00
P  4 0 8
198
197
201
202
V  40.00 22.00 -88.00
V  40.00 -1.50 -88.00
P  4 0 8
200
199
203
204
V  -38.50 22.00 -88.00
V  -38.50 -1.50 -88.00
P  4 0 8
202
201
205
206
V  38.50 22.00 -88.00
```

```
V  38.50 -1.50 -88.00
P  4 0 8
204
203
207
208
V  -38.50 -1.00 -88.00
V  -38.50 22.00 -102.00
V  -38.50 -0.50 -102.00
P  4 0 8
209
205
210
211
V  38.50 -1.00 -88.00
V  38.50 22.00 -102.00
V  38.50 -0.50 -102.00
P  4 0 8
212
207
213
214
V  -40.00 22.00 -102.00
V  -40.00 -0.50 -102.00
P  4 0 8
211
210
215
216
V  40.00 22.00 -102.00
V  40.00 -0.50 -102.00
P  4 0 8
214
213
217
218
V  -40.00 22.00 -106.00
V  -40.00 -0.50 -106.00
P  4 0 8
216
215
219
220
V  40.00 22.00 -106.00
V  40.00 -0.50 -106.00
P  4 0 8
218
217
221
222
V  -38.50 22.00 -106.00
V  -38.50 -0.50 -106.00
P  4 0 8
220
219
```

```
223
224
V  38.50 22.00 -106.00
V  38.50 -0.50 -106.00
P  4 0 8
222
221
225
226
V  -38.50 0.00 -106.00
V  -38.50 22.00 -130.00
V  -38.50 0.00 -130.00
P  4 0 8
227
223
228
229
V  38.50 0.00 -106.00
V  38.50 22.00 -130.00
V  38.50 0.00 -130.00
P  4 0 8
230
225
231
232
V  38.50 2.00 -130.50
V  38.50 23.00 -130.50
V  -38.50 23.00 -130.50
V  -38.50 2.00 -130.50
P  4 0 8
233
234
235
236
V  38.50 54.00 -138.00
V  -38.50 54.00 -138.00
P  4 0 8
106
237
238
107
F  0.8 0.8 0.8  0.5 0.5 0.0 0.0 5
V  -38.00 58.00 -135.00
V  -38.00 58.00 -3.00
V  38.00 58.00 -3.00
V  38.00 58.00 -135.00
P  4 0 9
239
240
241
242
;stairs
V  -37.17 54.00 -138.00
V  -37.17 54.00 0.00
P  4 0 9
```

```
137
238
243
244
V  -37.17 55.33 -138.00
V  -37.17 55.33 0.00
P  4 0 9
244
243
245
246
V  -35.83 55.33 -138.00
V  -35.83 55.33 0.00
P  4 0 9
246
245
247
248
V  -35.83 56.67 -138.00
V  -35.83 56.67 0.00
P  4 0 9
248
247
249
250
V  -34.50 56.67 -138.00
V  -34.50 56.67 0.00
P  4 0 9
250
249
251
252
V  -34.50 58.00 -138.00
V  -34.50 58.00 0.00
P  4 0 9
252
251
253
254
;stairs
V  37.17 54.00 -138.00
V  37.17 54.00 0.00
P  4 0 9
138
237
255
256
V  37.17 55.33 -138.00
V  37.17 55.33 0.00
P  4 0 9
256
255
257
258
V  35.83 55.33 -138.00
```

```
V  35.83 55.33 0.00
P  4 0 9
258
257
259
260
V  35.83 56.67 -138.00
V  35.83 56.67 0.00
P  4 0 9
260
259
261
262
V  34.50 56.67 -138.00
V  34.50 56.67 0.00
P  4 0 9
262
261
263
264
V  34.50 58.00 -138.00
V  34.50 58.00 0.00
P  4 0 9
264
263
265
266
;stairs
V  38.50 54.00 -137.00
V  -38.50 54.00 -137.00
P  4 0 9
238
237
267
268
V  38.50 55.33 -137.00
V  -38.50 55.33 -137.00
P  4 0 9
268
267
269
270
V  38.50 55.33 -136.00
V  -38.50 55.33 -136.00
P  4 0 9
270
269
271
272
V  38.50 56.67 -136.00
V  -38.50 56.67 -136.00
P  4 0 9
272
271
273
```

```
274
V  38.50 56.67 -135.00
V  -38.50 56.67 -135.00
P  4 0 9
274
273
275
276
V  38.50 58.00 -135.00
V  -38.50 58.00 -135.00
P  4 0 9
276
275
277
278
;stairs
V  38.50 54.00 -1.00
V  -38.50 54.00 -1.00
P  4 0 9
137
138
279
280
V  38.50 55.33 -1.00
V  -38.50 55.33 -1.00
P  4 0 9
280
279
281
282
V  38.50 55.33 -2.00
V  -38.50 55.33 -2.00
P  4 0 9
282
281
283
284
V  38.50 56.67 -2.00
V  -38.50 56.67 -2.00
P  4 0 9
284
283
285
286
V  38.50 56.67 -3.00
V  -38.50 56.67 -3.00
P  4 0 9
286
285
287
288
V  38.50 58.00 -3.00
V  -38.50 58.00 -3.00
P  4 0 9
288
```

287
289
290

Bibliography

[Allen 79] Jont B. Allen and David A. Berkley.
Image method for efficiently simulating small-room acoustics.
Journal of the Acoustical Society of America 65(4):943-50, April, 1979.

[Amanatides 84] John Amanatides.
Ray Tracing with Cones.
Computer Graphics 18(3):129-35, April, 1984.
Proc. Association for Computing Machinery (ACM) Siggraph'84.

[Benedetto 84] Giuliana Benedetto and Renato Spangolo.
Statistical distribution of free pathlengths in the acoustics of enclosures.
Journal of the Acoustical Society of America 75(5):1519-21, May, 1984.

[Benedetto 85] Giuliana Benedetto and Renato Spangolo.
Reverberation time in enclosures: The surface reflection law and the dependence of the absorption coefficient on the angle of incidence.
Journal of the Acoustical Society of America 77(4):1447-51, April, 1985.

[Beranek 62] Leo L. Beranek.
Music, Acoustics & Architecture.
John Wiley & Sons, Inc., 1962.

[Beranek 86] Leo L. Beranek.
Acoustics.
American Institute of Physics, Inc., 1986.
Copyright by the Acoustical Society of America.

[Borish 84] Jeffrey Borish.
Extension of the image model to arbitrary polygedra.
Journal of the Acoustical Society of America 75(6):1827-36, June, 1984.

[Cohen 85] Michael F. Cohen and Donald P. Greenerg.
The Hemi-Cube, A Radiosity Solution for Comlpex Environments.
In *Computer Graphics*. ACM Siggraph '85, July, 1985.
Quarterly report of ACM Siggraph, Vol. 19, Number 3.

[Cook 82] Robert L. Cook and Kenneth E. Torrance.
A Reflectance Model for Computer Graphics.
ACM Transaction on Graphics 1(1):7-24, January, 1982.
Also in *ACM Siggraph'85 Conference* course notes, Vol. 12.

[Dennett 81] Daniel C. Dennett.
The Nature of Images and the Introspective Trap.
Bradford Book. Imagery.
M.I.T. Press, 1981, pages 51-61.

[Fujimoto 85] Akari Fujimoto and Kansei Iwata.
Accelerated Ray Tracing.
In *Computer Graphics Tokyo'85.* Graphica Computer Corp., 1985.

[Fujimoto 86] Akira Fujimoto, Takajuki Tanaka, and Kansei Iwata.
ARTS: Accelerated Ray-Tracing System.
IEEE Computer Graphics & Applications ?(?):16-26, April, 1986.

[Furrer 64] Willi Furrer.
Room and Building Acoustics and Noise Abatement.
Butterworth Inc., 1964.

[Jordan 80] Vilhelm Lassen Jordan, M.Sc., Ph.D.
Acoustical Design of Concert Halls and Theatres.
Applied Science Publishers, LTD., 1980.

[Jurgensen 87] Peter Jurgensen and James Anderson.
A Knowledge Representation Schema for Design Support Systems
Computer Resource Laboratory at the School of Architecture and Planning, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[Kay 79] Douglas S. Kay and Donald Greenberg.
Transparency for Computer Synthesized Images.
Computer Graphics 13():158-164, , 1979.

[Kay 86] Timothy L. Kay and James T. Kajiya.
Ray Tracing Complex Scenes.
Computer Graphics 20(4):269-78, August, 1986.
Proc. ACM Siggraph'84.

[Knuth 75] Donald Knuth.
Computer Science and Mathematics.
American Scientist 61():709, , 1975.

[Nishita 85]	Tomouki Nishita and Eihachiro Nakame. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics*. ACM Siggraph '85, July, 1985. Quarterly report of ACM Siggraph, Vol. 19, Number 3.
[Phong 75]	Bui-Tong Phong. Illumination for Computer Generated Images. *Communications of the ACM* 18():311-17, , 1975.
[Pylyshyn 81]	Zenon Pylyshyn. The Imagery Debate - Analog Media versus Tacit Knowledge. *Bradford Book. Imagery.* M.I.T. Press, 1981, pages 151-206.
[Rogers 85]	David F. Rogers. *Procedural Elements for Computer Graphics.* McGraw-Hill Book Company, 1985.
[Schwartz 81]	Robert Schwart. Imagery - There's More to It Than Meets the Eye. *Bradford Book. Imagery.* M.I.T. Press, 1981, pages 109-30.
[Wayman 80]	J. L. Wayman. *Computer simulation of sound fields using ray methods.* PhD thesis, University of California, Santa Barbara, July, 1980.
[Weghorst 84]	Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved Computational Methods for Ray Tracing. *ACM Transaction on Graphics* 3(1):52-69, January, 1984.
[Whitted 80]	Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM* 23(6):343-49, June, 1980.
[Whitted 85a]	Turner Whitted and Rob Cook. A Comprehensive Shading Model. In *Image Rendering Tricks*. ACM Siggraph '85, July, 1985. Conference course notes, Vol. 12.
[Whitted 85b]	Turner Whitted. Simple Ray Tracing. In *Image Rendering Tricks*. ACM Siggraph '85, July, 1985. Conference course notes, Vol.12.
[Wojtowicz 86]	Jerzy Wojtowicz and William Fawcett. *Archictecture: Formal Approach.* Academy Editions, 1986.

[Yamaha 86] *Sound Field Creation, The Yamaha DSP-1* Yamaha, 1986.