Interactive 3D Rendering and Visualization in Networked Environments

Ioana M. Martin, James T. Klosowski, William P. Horn

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA

Abstract

Efficient delivery of 3D graphics over networks is becoming increasingly important for a number of applications, ranging from industrial design and manufacturing to entertainment. As companies make the transition from a conventional business model to an e-business model, the number of users that require access to 3D model databases is forecast to grow dramatically over the next few years. While some users may access these databases using high-performance graphics hardware over high-speed connections, others are likely to access the data with devices having limited hardware graphics support over slower connections such as busy intranets, dial-in networks, or wireless connections. In this context, there is a requirement for efficiency. This translates into ensuring that access to centralized data is provided through a unified interface cognizant of the environment conditions and capable of transparently adjusting the access mechanism in order to provide the clients with optimal access service. In this course, attendees will learn to leverage existing methods for data transfer and interactive graphics to create the next generation of 3D networked graphics software tools.

Prerequisites

This course targets an audience possessing an understanding of the basics of 3D graphics and networking. It is intended as an opportunity for the attendees to learn about ongoing efforts in both fields and to become aware of the challenges and issues involved in transmitting complex 3D models over networks.

Course Outline

1. Introduction
2. Networked Graphics: Applications, Capabilities, and Challenges
3. A Networking Primer
   - 3.1. Fundamentals of data transfer
   - 3.2. Network protocols
   - 3.3. Integrated vs. differentiated services
   - 3.4. Multimedia protocols
   - 3.5. Designing a protocol for 3D data delivery
4. Interactive Rendering of Complex 3D Models
   - 4.1. Data structures for model organization
   - 4.2. Rendering acceleration techniques
     - 4.2.1. Culling
     - 4.2.2. Simplification
     - 4.2.3. Levels-of-detail
     - 4.2.4. Impostors
   - 4.3. Model perception and representation selection
5. Overview of 3D Transfer Technologies
   - 5.1. File formats for 3D data transfer
     - 5.1.1. VRML
     - 5.1.2. Java 3D
     - 5.1.3. XGL
     - 5.1.4. MPEG-4
     - 5.1.5. MetaStream
   - 5.2. Techniques for efficient transmission of 3D data
     - 5.2.1. Compression
     - 5.2.2. Streaming
6. Adaptive System Design and Implementation
   - 6.1. Combining modalities for network rendering
   - 6.2. Environment monitoring
   - 6.3. Adaptive selection of transmission methods
   - 6.4. Support for real-time interaction
   - 6.5. Implementation issues
7. Conclusions

1. Introduction

Emerging networking infrastructures include an increasing variety of clients and servers inter-connected by communication fabrics of various types and capabilities. This heterogeneity makes it difficult for servers to provide a level of service that is appropriate for every client that requests access to multimedia content. So far, a significant body of work has been dedicated to the challenges of universal access, regarding the delivery of traditional multimedia content such as text, images, audio, and video. Less attention was focused on three-dimensional (3D) digital content, as true market opportunities for 3D graphics over networks have just recently begun to emerge.

Over the past years, a number of optimization technologies regarding both transmission and rendering of 3D models have been developed (e.g., compression, model simplification, levels-of-detail, streaming, image-based techniques). In this course, the attendees will learn to combine and leverage these existing methods to create the next generation of 3D networked graphics software tools. The objectives of this course are for the attendees to:

1. Become aware of the problems related to the rendering of 3D models over networks.
2. Learn about the various approaches that have been developed to alleviate these problems, including their strengths and limitations.
3. Understand what are the considerations and requirements that software developers should take into account when designing graphics applications for networked environments.

The course is organized into three major parts. The first two parts cover relevant topics in networking and interactive computer graphics. The third part addresses issues that arise from the merger of the first two topics and describes the technologies necessary to build interactive 3D applications in networked environments. Demos are provided to illustrate some of the concepts presented.

The networking primer (section 3) is intended as a review of relevant networking concepts. The primer includes an overview of efforts to ensure quality-of-service for multimedia applications. Typically, such applications require real-time traffic which is not naturally supported by the Internet. Several general solutions have been proposed, including traffic classification, priority allocation by application, and reservations. We cover the fundamentals of the real-time protocols (RSVP, RTP, RTCP, and RTSP) and we take a look at traffic management using quality-of-service. We analyze the requirements for the development of a protocol that supports delivery of streams containing 3D data.

The second part of the course (section 4) is dedicated to techniques that have been developed to address the challenge of rendering detailed 3D models at interactive frame rates. We review the ingredients needed to represent a model (geometry, topology, and other attributes), as well as various types of representations and data structures for organizing model components. We also describe techniques that exploit these mechanisms to build effective systems that balance interactivity and realism. In this context, we investigate model perception issues and methods to decide how various parts of a model contribute to the overall quality of the rendering and how to dynamically select an appropriate representation for each component. Complexity management techniques including culling, simplification, levels-of-detail, and impostors will be presented.

The primary focus of the course (sections 5 – 6) is on the merger of networking and interactive 3D graphics to provide solutions to the problem of visualizing models on clients with varying graphics hardware capabilities over a wide range of connection fabrics. In recent years, a variety of algorithms and storage technologies have been developed that promise to overcome difficulties in accessing and viewing large data sets. Such algorithms include compression and progressive transmission of models to reduce the perceived network delay. Storage formats that support streaming of multimedia have been enhanced to include 3D data (e.g., MPEG-4), or have been designed specifically for 3D data streaming (e.g., MetaStream). In this part of the course we provide a survey of these algorithms and technologies, and we discuss their advantages and disadvantages. We explore the (typically implicit) assumptions these methods make about their operating environments. Issues related to adaptive systems that are aware of differences in environment conditions and, at the same time, leverage existing technologies are discussed in detail. Design topics such as combining representations, adaptive selection with multiple constraints, benchmarking and dynamic environment monitoring, as well as implementation issues such as data structures, single vs. multiple threads, and scalability are addressed.

2. Networked Graphics: Applications, Capabilities, Challenges

The most familiar application of 3D graphics in networked environments, aside from games, is on-line advertising and shopping. The e-commerce dream is being able to sell everyday items via the Web by presenting them in 3D form so buyers could view them from any angle inside a virtual store. “Will that bathing suit make me look fat? Can I see home plate from a seat in Section 293 at Yankee Stadium?” Online shoppers can find answers to these and other questions on an increasing number of Web sites that are deploying interactive 3D technologies designed to reproduce real-life experiences. It’s convenient, it’s fun, and according to retail experts, 3D is seen as one of the most promising ways to convert online browsers to online buyers.

In the business world, universal network access causes community boundaries to be redefined and new models of

business organization and interaction become possible. By permitting business partners to connect to their networks, corporations are creating global inter-corporate networks. Employees access information in any part of a corporation and collaborate with geographically dispersed subsidiaries. Additionally, international mergers such as Ford Motor's acquisition of Volvo and Daimler's acquisition of Chrysler, are driving the need for global connectivity to an even higher level. Such companies have connected tens of thousands of their employees to the Internet and are using it as a way to gain a competitive advantage by providing universal access to 3D models for design and manufacturing.

In medicine, ultrasound devices used to diagnose human organs such as the heart and the liver and to observe fetus development are now moving from the traditional 2D images to 3D reconstructions. While dynamically obtaining 3D images of a beating heart remains a future goal, people have recently been able to look at 3D models of smiling faces of their unborn babies. Longstanding predictions that 3D graphics would become a simple tool of the medical office are not so far-fetched anymore.

In education, a 3D model may be worth a thousand words. A recent survey of university professors has shown that, when asked for a wish list in terms of teaching aids, assuming no technological barriers, an overwhelming majority had interactive 3D graphics on their list. From mathematics and engineering, to biology and geography, to choreography and textile design, replacing 2D images and video clips with 3D models that can be manipulated interactively enriches the learning experience.

These are just a few of the many applications of 3D graphics that could benefit from efficient and adaptive techniques for network delivery. To achieve these benefits, there are many challenges that must be addressed by networked graphics applications. Due to the increasing heterogeneity of networked environments, networked graphics applications must support access to appropriate representations of 3D models that enable clients to view these models regardless of their graphics capabilities and the network connections they use. However, accommodating multiple representations of the same model and selecting the optimal one(s) with respect to given constraints and criteria is not trivial. Bandwidth (or the lack thereof) has been a challenge for networked applications in general. As the gap between high and low-bandwidth connections increases, so do the requirements for adaptive applications. At one end of the bandwidth spectrum is the Next Generation Internet [62], that promises an order of magnitude higher bandwidths than typically available today. At the other end of the spectrum, are pervasive devices and information appliances that usually support lower bandwidth links. Last but not least, the economic and social globalization implies an explosion in the number of clients accessing information, and in particular, 3D data. In this context, scalability and security are issues that cannot be ignored.

3. A Networking Primer

In this section we review basic networking concepts such as latency, bandwidth, reliability, and protocols, we present an overview of multimedia protocols such as RSVP, RTP, RTCP, RTSP, and traffic control using quality-of-service, and we identify issues that have to be taken into account when selecting or developing a protocol for 3D graphics.

3.1. Fundamentals of data transfer

The role of a network is to provide connectivity among a set of computers. Some networks are designed to connect only a few machines (e.g., intranets), whereas others may potentially connect all the computers in the world (e.g., the Internet). Network connectivity occurs at many different levels. A simple network consists of two or more computers (nodes) directly connected by a physical medium (link). Figure 1 illustrates two examples of direct links: point-to-point (connecting a pair of nodes) and multiple-access (several nodes share the same physical link). The number of computers that can be connected with a direct link is restrictive. Fortunately, connectivity does not necessarily imply the existence of a direct physical connection. Figure 2 shows an example of a switched network in which nodes that are attached to at least two links run software that forwards the data received on one link out on another. Examples of switched networks include circuit-switched (e.g., the telephone system) and packet-switched (e.g., most computer networks). For networked graphics we are interested in the latter category. Nodes in a packet-switched network send discrete blocks of data to each other. Such blocks are usually termed *packets* or *messages* and are typically delivered using a store-and-forward strategy. As the name suggests, first a node receives a complete packet over some link, stores it in its internal memory, and then forwards the complete packet to the next node.

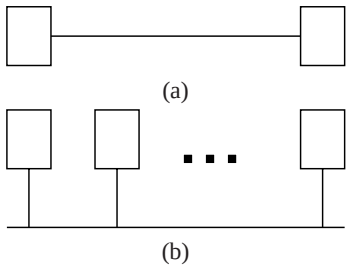


Figure 1: *(a) Point-to-point link. (b) Multiple-access link.*

A set of independent networks can be connected to form an internetwork, or internet for short. A node that is connected to two or more networks is commonly called a router or gateway and, similar to a switch, it forwards messages from one network to another. Note that an internet can be built as an interconnection of internets. Ultimately, to

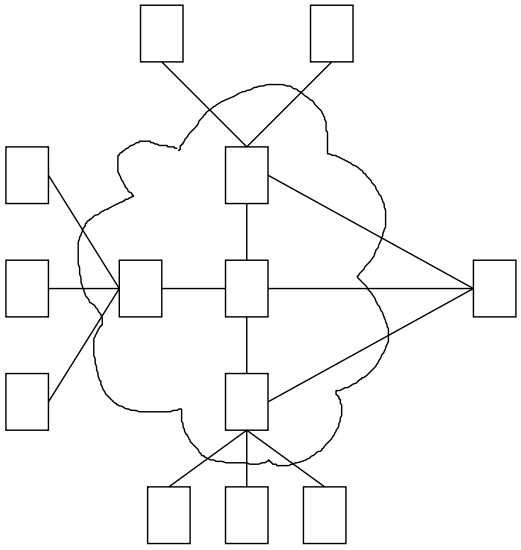


Figure 2: *Switched network: the cloud distinguishes between nodes on the inside that implement the network (switches) and nodes on the outside that use the network (hosts).*

achieve connectivity, each node must be able to specify the other nodes in the network to which it wishes to communicate. This is done by assigning an address to each node. The process of determining how to forward messages toward the destination node based on its address is called *routing*. The most popular scenario is that of a source node sending a message to a single destination node (unicast). However, sometimes it is desirable for a node to send a message to all nodes on the network (broadcast), or to a select subset (multicast). Thus, in addition to node-specific addresses, a network should also support multicast and broadcast.

Resource sharing Multiple hosts share a network by *multiplexing*. Intuitively, multiplexing is analogous to multiple jobs sharing a common CPU. There are several strategies for multiplexing multiple data flows onto one physical link.

Using *synchronous time-division multiplexing* (STDM), time is divided into equal-sized quanta and each flow is given a chance to send its data over the physical link in a round-robin fashion. Using *frequency division multiplexing* (FDM), each flow is transmitted over the physical link at a different frequency, much the same way that the signals for different TV stations are transmitted on a physical cable TV link. Both STDM and FDM have similar drawbacks. If one of the flows does not have any data to send, its share of the link remains idle, even if one of the other flows has data to transmit. Also, adding new quanta (STDM) or frequencies (FDM) may not be practical.

To solve these problems, *statistical multiplexing* is typically used. As in STDM, the physical link is shared over time. In contrast to STDM, the data is transmitted from each flow on demand, rather than in a round-robin fashion. Thus, if only one flow has data to send, it transmits that data without waiting for its turn. To ensure that other flows have a chance of transmitting, statistical multiplexing defines an upper bound on the size of the block of data that each flow is permitted to transmit at a given time. This block of data is usually termed *packet*. The term *message* is typically used for the arbitrarily large data an application transmits as a sequence of packets. A decision as to which flow will transmit next is made on a packet-by-packet basis. This decision can be made in a number of different ways. For example, a switch could be designed to service packets in a first-in-first-out (FIFO) manner. Another approach would be to service the different flows in a round-robin fashion to ensure that certain flows receive a particular share of the link's bandwidth. A third approach is for an individual application to request that its packets not be delayed for more than a certain length of time. Yet another example is to define priorities based on the class of data or aggregated traffic. A network that supports the latter two approaches is said to provide *quality of service* (QoS).

Reliability Reliable message delivery is one of the most important functions that a network can provide. There are three major classes of network failure. First, as a packet is transmitted over a network, bit errors may occur as a result of outside forces (e.g., lightning strikes, power surges). Such errors are typically rare. They are either detected and corrected, or the packet has to be retransmitted. Second, complete packets may be lost by the network (e.g., a switch runs out of buffer space or there is a bug in the software handling the packet). The main challenge in this case is distinguishing between packets that are lost and those that are late in arriving at the destination. Third, physical links may be cut or participating nodes may crash. While such failures can eventually be corrected, they can have a dramatic effect on a network for an extended period of time.

Performance Network performance is measured in two fundamental ways: *bandwidth* (or *throughput*) and *latency* (or *delay*).

The bandwidth of a network is measured in terms of the number of bits that can be transmitted over the network in a certain period of time. For example, a network might have a bandwidth of 10 million bits/second (Mbps), which means that it can deliver 10 million bits every second. Sometimes it is useful to think of bandwidth in terms of how long it takes to transmit each bit of data. For example, on a 10 Mbps network, it takes 0.1 microseconds to transmit each bit.

Latency corresponds to how long it takes a message to travel from one end of the network to the other and it is measured in terms of time. In some situations it may be useful

to know how long it takes to send a message from one end of a network to the other and back, i.e., the round-trip time (RTT) of the network. Latency can be thought of as having three components:

$$Latency = Propagation + Transmit + Queue$$

$$Propagation = Distance/SpeedOfLight$$

$$Transmit = Size/Bandwidth$$

where *Propagation* denotes the speed-of-light propagation delay, *Transmit* is the amount of time it takes to transmit a unit of data, and *Queue* is the time spent in queuing delays inside the network.

The combination of bandwidth and latency determine the performance characteristics of a given link. Their relative importance depends on the application. For example, a client that sends a 1-byte message to a server and receives a 1-byte message in return is latency bound. This application will perform differently on a transcontinental channel with 100 ms RTT than on a channel across the room with a 1 ms RTT. Whether the channel is 1 Mbps or 100 Mbps is insignificant. In contrast, consider fetching a 25 MB 3D model over a network. The more bandwidth that is available, the faster the model will be downloaded. In this case, the bandwidth dominates the performance. It will take 20 seconds on a 10 Mbps to transmit the model, whether there is a 1 ms or a 100 ms latency is insignificant.

3.2. Network protocols

To hide complexity, networks are designed with several levels of abstraction corresponding to various layers of service. The idea is to start with services offered by the hardware and to add a sequence of layers, each providing a higher, more abstract level of service. The services provided at the high layers are implemented in terms of services provided at the low levels. The advantages of layering are two-fold. First, the network software is decomposed into more manageable pieces, each solving a particular problem. Second, when adding a new service, it is easier to leverage existing lower levels.

Protocols The layers of a network consist of abstract objects called *protocols*. A protocol provides a communication service used by higher level objects to exchange messages. A protocol has two different interfaces. A *service interface* defines the operations that objects can request from a lower protocol. A *peer interface* defines the messages exchanged between the same protocol on different hosts.

The OSI architecture The Open Systems Interconnection (OSI) architecture shown in Figure 3 (a) defines a standard architecture for exchanging information between computers. Its functionality is partitioned into seven layers.

1. The physical layer handles the transmission of raw bits over a communication link.

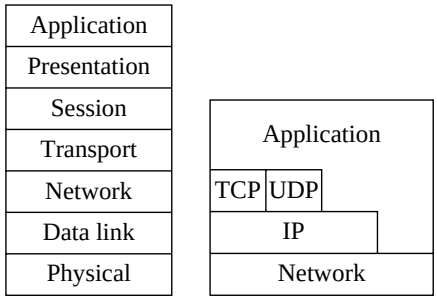


Figure 3: *(a) OSI architecture; (b) Internet architecture.*

2. The data link layer collects a stream of bits into a larger structure called a *frame*.
3. The network layer handles routing among nodes. The unit of data exchanged among nodes is typically called a *packet* (rather than frame), although they are the same thing.
4. The transport layer implements a process-to-process channel for which the unit of data exchanged is the message.
5. The session layer provides management of dialog control (e.g., one-way or two-way traffic) and synchronization (e.g., two audio and video streams part of the same teleconferencing application).
6. The presentation layer is concerned with the format of the data exchanged between peers (e.g., whether an integer is 16, 32, or 64-bits long).
7. The application layer contains a variety of protocols that are commonly needed. For example, the File Transfer Protocol (FTP) defines how file transfer applications can interoperate.

The Internet architecture The Internet architecture evolved from experiences with an early network funded by the US Department of Defense called ARPANET. The Internet architecture can be described using a four-layer model as shown on Figure 3 (b).

1. At the lowest level are a wide variety of network protocols (e.g., Ethernet, TokenRing).
2. The Internet Protocol (IP) supports the interconnection of multiple networking technologies into a single, logical internetwork.
3. The third layer contains two transport protocols: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). These protocols provide alternative logical channels to applications. TCP supports reliable delivery, whereas UDP is unreliable.
4. The fourth layer contains a number of application protocols (e.g., FTP, HTTP).

The Internet architecture does not imply strict layering. An application may bypass the transport layers and use IP

directly. In fact, programmers are free to define new abstractions that run on top of the existing protocols.

Sockets A popular application programming interface (API) supported by most operating systems is the so-called *socket interface* (originally provided by the Berkeley distribution of Unix).

The main abstraction of the socket interface is the *socket*, i.e., the point where an application process attaches to the network. The interface defines operations for creating a socket, attaching the socket to the network, sending/receiving messages through the socket, and closing the socket.

3.3. Integrated vs. differentiated services

Typically, Internet service is provided using a *best-effort* approach. This approach does not allow users to request an increase the quality of service they receive, even if they are willing to pay for it. However, the ability to address the various requirements of different customers is rapidly becoming important. This trend has resulted in several efforts to define mechanisms to support QoS. Internet service providers would like to maximize the sharing of the costly backbone infrastructure in a manner that enables them to control the usage of network resources according to service pricing and revenue potential. Ideally, they should be able to:

(a) isolate traffic from different customers and provide minimum bandwidth guarantees;
(b) define different levels of service for different types of traffic, in a customer-dependent manner (e.g., some customers may define voice over IP or database queries to have high priority, while others may specify FTP transfers to have low priority);
(c) allow customers to choose an extremely reliable and high-performance (possibly expensive) service.

The approaches that have been developed to provide QoS can be divided into two broad categories: *fine-grained* (to provide QoS to individual applications) and *coarse-grained* (to provide QoS to large classes of data).

Both categories have received considerable attention in recent years and Internet Engineering Task Force (IETF) working groups have been created to address QoS issues. The IETF is an open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. Solutions to the fine-grained category are referred to as *integrated services*. Solutions to the coarse-grained category are referred to as *differentiated services*.

Integrated services The term "Integrated Services" refers to a body of work produced by IETF around 1995-97. The corresponding working group developed specifications of several *service classes* designed to meet the needs of real-time applications. The group also defined how a reservation protocol (see RSVP) could be used to make reservations based on these service classes.

Two main service classes have emerged from the IETF specifications. The first one offers *guaranteed service* and it is designed for intolerant applications that require that packets never arrive late. The second class has been chosen to meet the needs of tolerant, adaptive applications, and the service provided is known as *controlled load*. In the latter case, the goal is to emulate the conditions of a lightly loaded network, even though the network may be heavily loaded by isolating controlled traffic from other traffic. As these classes are deployed, it will become clear whether additional services are needed.

Resource reSerVation Protocol (RSVP) RSVP is a network control protocol that operates on top of IP and allows a receiver to request a specific QoS for its data flows. Applications use RSVP to reserve the necessary resources along transmission paths so that the requested bandwidth is available when the transmission actually takes place. There are two things a receiver needs to know before it can make a reservation:

(a) an estimate of the amount of traffic that is to be sent, so it can make an appropriate reservation, and
(b) the path of transmission, so it can establish a reservation at each router in the path

Once this information is available, a request for reservation is made at each node along the sender-receiver path. After reservations are made, routers supporting RSVP extract the QoS class for each incoming packet and make forwarding decisions based on this information.

Differentiated services In contrast to Integrated Services which are concerned with allocating resources to individual flows, the *Differentiated Services* model allocates resources to a small number of traffic classes. The IETF Differentiated Services (diffserv) group is working towards the definition of a general conceptual model consisting of a small, well-defined set of building blocks from which a variety of aggregate behaviors may be built to support various types of applications, and specific business requirements. The basic idea is to include a small bit-pattern in each packet to mark the type of treatment it should receive at each network node and to define a common understanding about the use and interpretation of this pattern for inter-domain use, multi-vendor interoperability, and consistent reasoning about expected aggregate behaviors in a network.

Currently, the best-effort service model is enhanced by adding just one new class, offering "premium" service. One bit in the packet header could be used to distinguish between premium and regular packets. Two questions have to be addressed:

- Who sets the premium bit and under what circumstances ? For example, packets may be marked as premium by the ISP, based on the price paid for the service.
- What happens when a premium packet reaches a router ? A number of strategies have been defined, including *expedited forwarding (EF)*, in which a packet is forwarded with minimal delay and loss, and *assured forwarding (AF)*, in which packets within a specified profile are guaranteed to be delivered.

3.4. Multimedia protocols

Multimedia applications are part of a class of applications that are time sensitive and are generally termed as *real-time applications*. The idea of playing multimedia over the Internet is attractive, as the Internet has become a platform for the majority of networking activities. Users can receive data and multimedia content over the same network, without having to invest in additional hardware. However, the availability of increased bandwidth is a necessary but not sufficient condition. Multimedia data needs to be played back continuously and smoothly, and the Internet does not naturally support real-time traffic. Appropriate hardware and software infrastructures and application tools have to be developed to support real-time applications.

The TCP protocol is designed for reliable transmission of data with minimal delay constraints. However, multimedia traffic does not require reliable transmission and other protocols may be more appropriate. If a receiver has to wait for a TCP retransmission, an unacceptable gap can occur in a playback of a video (or any other delay-sensitive data). Also, since there is no predefined path for packets to flow over the Internet, there is no mechanism to ensure that the bandwidth needed is available between the sender and the receiver, so QoS cannot be guaranteed. In addition, TCP does not provide timing information, which is critical for multimedia support. Most multimedia applications are less affected by missing data than by lengthy delays caused by retransmissions. Also, they do not require in-sequence delivery. In this section, we present several protocols that have been developed to enhance the Internet architecture to support audio, video, and interactive multimedia conferencing.

Real-time Transport Protocol (RTP) is a protocol that provides support for applications transmitting real-time data over networks. Services include payload type identification, sequence numbering, and time stamping. RTP typically runs on top of UDP and it works in conjunction with an auxiliary control protocol (RTCP) to monitor packet delivery. The header of an RTP packet provides timing information necessary to synchronize data, as well as sequence numbers to place incoming packets in order. The payload type identifier indicates the type of data and the compression scheme used to deliver it. However, RTP does not provide any mechanisms to ensure timely delivery or QoS. Hence, for applications that require such guarantees, RTP must be accompanied by other mechanisms. Figure 4 shows an RTP packet encapsulated inside a UDP packet.

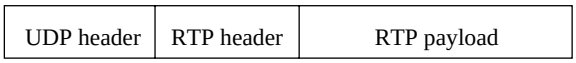


Figure 4: *The structure of an RTP packet inside a UDP packet.*

Real-Time Control Protocol (RTCP) works in conjunction with RTP to monitor performance and to issue diagnostics. RTCP control packets are periodically transmitted by each participant in an RTP session to all other participants. The primary goal is to provide information to the application regarding the quality of the data distribution so that appropriate action is taken.

Real-Time Streaming Protocol (RTSP) is a presentation protocol for streaming multimedia data over a network. Instead of downloading large multimedia files and playing them back, data is sent over the network in streams. While a packet is being played, another is being received and users have access to the multimedia content without waiting for the entire data to be received. Sources of data for streaming can include both live data feeds as well as stored clips. RTSP is an application-level protocol designed to work with lower level protocols such as RTP and RSVP. It provides VCR-style functionality for audio and video streams, including pause, fast forward, reverse, and absolute positioning.

3.5. Designing a protocol for 3D data delivery

Delivery of 3D data over networks can be viewed as an extension to the delivery of multimedia data. Certain features, such as increased bandwidth and the ability to stream are desirable for both 3D and multimedia applications. Other features, however, are specific to 3D applications and have to be addressed by specialized protocols.

Graphics applications are typically less tolerant to packet loss than video and audio applications. Although timely delivery is still important, a reliable transport protocol such as TCP is likely to be more suitable for transmission of geometric data than an unreliable protocol for which additional support would be necessary to recover the dropped packets. Depending on the format used to encode the 3D data, a lost packet may result in a minor defect in the surface of an object, or it may render useless all remaining packets. Out-of-order transmission of packets may also be a problem for geometric data. Existing streaming techniques for geometry tend to generate progressive representations consisting of a coarse model and a sequence of refinements for which the order in which they have to be applied to recover the full-resolution model is predefined. Similarly, effective compression methods typically use prediction in conjunction with entropy coding to achieve good compression rates. In this case, predictors and items predicted must be part of the same

package, which may not be easily accomplished. The overhead introduced to achieve error resilience for progressive or compressed streams will probably exceed the overhead of choosing an existing reliable protocol such as TCP over a faster, unreliable scheme.

Audio and video streams support a limited type of interactions such as start, stop, fast forward, and reverse. Interactions with a 3D scene are more complex and, among other things, include: 3D manipulation of the scene or of individual objects; restricting access and display to parts of the scene; and changing camera positions or the attributes of objects in the scene. It is desirable to take such interactions into account when downloading the content. In such cases, a feedback channel should be available to communicate the changes caused by interactive actions.

The delivery of 3D data over networks may require multiplexing the geometric information with other kinds of streams, such as images, video, and audio. Texture images are one simple example. Image impostors, as discussed in the next section, constitute another example and are used to reduce the transmission and rendering complexity of 3D scenes. A protocol supporting delivery of 3D graphics should provide synchronization with other types of streams.

SUMMARY

A network can be defined recursively as consisting of two or more nodes connected by a physical link, or as two or more networks connected by one or more nodes.

Layered architectures provide a framework for network design. The central objects in this design are network protocols. Protocols provide a communication service to higher level protocols and define the format of the messages exchanged by peers. Two popular architectures are OSI and the Internet.

The Internet uses a best-effort approach that does not allow users to request an increase in the quality of service they receive. To address this problem, two classes of approaches are considered: integrated services that provide fine-grained quality of service to applications and differentiated services that address larger classes of traffic.

When designing a protocol for 3D delivery, issues such as streaming, error resilience, packet order, and multiplexing of geometry streams with other types of data should be considered.

4. Interactive Rendering of Complex 3D Models

Users typically want to interact with 3D models in real-time. They want to load, visualize, and inspect models without having to wait minutes, or even seconds, for the images to appear on their computer screens. In addition, users also want models to look as realistic as possible. Consequently, 3D models are increasing in complexity every day. Designers, at the request of their customers, are making their models more realistic using additional details. Added realism can be achieved, for instance, by specifying colors, normals, or textures to supplement the information captured by the geometry of a model, or by using additional geometric detail to approximate objects more closely.

In parallel to an increase in model complexity, graphics hardware has also been improving, providing additional capabilities, as well as better rendering performance. Unfortunately, the increase in graphics processing power has not been able to keep up with the rapid growth of models. Moreover, there has been an explosion of new types of computing devices, such as personal digital assistants (PDAs), wearable computers, and so-called information appliances. These devices provide no hardware support for 3D graphics and some are even limited in the types of 2D images they can display. Consequently, significant amount of research is being done to develop techniques which address the challenge of rendering 3D models in real-time so that users can visualize and interact with the models.

In this section, we focus on the techniques that enable users to interact in real-time with complex 3D models. We begin by reviewing commonly used methods to represent and organize 3D models in computer graphics. We emphasize the tradeoffs which must be made between interactivity and realism and we discuss model perception issues and methods to decide how the various components of a model contribute to the overall "feel" of the rendered images. We emphasize interactive rendering techniques that provide users with the highest image quality possible, while maintaining interactivity.

4.1. Data Structures for model organization

The information used to represent a 3D model may include, but is not limited to, geometry, topology, materials, normals, and textures. In most cases, the geometry of a model is specified as vertex coordinates, and its topology specifies how to connect the vertices together, e.g., as triangles or quadrilaterals, to form the surface of the model. The topology is often described using triangles, since they are the simplest of primitives, and (typically) graphics hardware is designed to render them. Thus, for faster rendering, many models are specified using triangles to avoid conversion of other primitives to triangles by the hardware. The geometry and topology of a model could also be described using alternative schemes, such as NURBS and Bezier curves [20], or even as canonical

shapes such as cubes, spheres, cylinders, and cones. Since all of these representations can ultimately be triangulated, we focus our discussion to triangulated models.

The material properties of a model specify how light is reflected by its geometry to produce color. Examples of such properties include diffuse color, specular color, shininess, and transparency. Many models include only a simple diffuse color, specified for each of their vertices or for each of their faces. Normals may also be specified either at the vertices (for smooth shading) or per face (for flat shading). Some models use textures as a simple means of giving the appearance of a more complicated model. In such cases, texture coordinates are provided for the vertices.

Model representations For a model with n vertices and m triangles, a simple representation is to list its vertices $v_{0,\ldots,}v_{n-1}$, followed by the triangles $t_{0,\ldots,}t_{m-1}$, where each v_i is a point in $\Re^3$ and each t_j is a set of three indices into the vertex list. This scheme, known as an *indexed face set* (IFS) [5], easily supports colors, normals, or texture coordinates for additional detail. For many models, an IFS is the representation most often chosen due to its simplicity and functionality. Unfortunately, some models require more complicated descriptions than those allowed by the IFS. For this reason, we discuss an alternative representation for organizing models.

A *scene graph* is a general term given to a hierarchical data structure used to describe a model. Scene graphs are most often directed acyclic graphs, in which nodes store their data in fields. Typically there are several types of nodes in a scene graph, which may represent geometry, material properties, groups of other nodes, or even particular behaviors of the geometry, such as a motion sequence or the set of viewpoints for which the geometry should be drawn. More complicated features such as audio and video can also be incorporated so that appropriate effects occur when the viewer is close enough to a specific component of the model. Examples of well-known scene graphs include VRML [5] (which originated from Inventor [27]), IRIS Performer [67] and Java 3D [82].

The hierarchical data structures that provide the structure of a scene graph may organize a model spatially, functionally, or by some combination of the two. A spatial hierarchy organizes a model's geometry into groups based upon their location in space. Objects that are located close to each other have a better chance of being grouped together than those that are far apart. A functional (i.e., semantic) hierarchy arranges geometric components according to their function or purpose. For example, a model of an automobile may group all of the geometry which comprises the engine together, or it may even further subdivide the geometry into smaller parts, such as the spark plugs. Figure 17 illustrates an example of a semantic hierarchy. Some scene graphs use both kinds of groupings. Initially they use the functional hierarchy, since designers have traditionally built models using this framework, and then a spatial hierarchy is superimposed on top of the functional one.

Each node v of a scene graph may consist of a bounding volume, *e.g.*, a sphere or axis-aligned bounding box, which approximates all of the geometry associated with v, and all of its children. The bounding volumes are beneficial for accelerating the rendering of the model (see section 4.2.1). Geometry can be stored at any level of the hierarchy, although it is often stored only in the leaf nodes as an IFS, triangle fan, or triangle strip.

Triangle Fans and Strips As previously mentioned, due to its simplicity, the IFS is a popular choice for describing the geometry and topology of a model. However, an IFS is not the most efficient representation for rendering purposes. Each triangle is defined by three vertices, all of which must be sent to the graphics pipeline for processing. As many triangles share vertices, it seems possible to increase graphics performance by sending fewer than three vertices per triangle to the graphics pipeline. In doing so, fewer vertices and normals would need to be transformed and fewer lighting calculations would have to be computed. However if the bottleneck of the graphics processing is the number of pixels that can be filled per second, i.e., the pixel fillrate, then no performance gain would be achieved by sending fewer vertices.

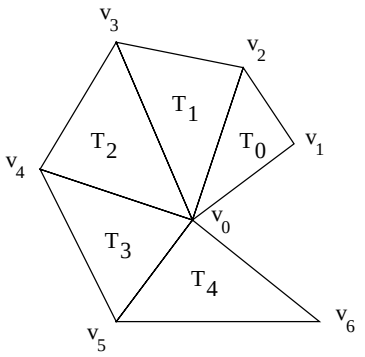


Figure 5: *Triangle fan.*

Figure 5 shows an example of a *triangle fan*, which is a set of connected triangles, all of which share a common vertex, v_0. This fan of five triangles can be completely described as an ordered list of seven vertices $v_0, v_1, \ldots, v_6$, where each triangle T_i in the fan can be specified by the three indices v_0, v_{i+1}, v_{i+2}. Thus, rather than sending three vertices per triangle down the graphics pipeline, only one vertex needs to be sent, following the three vertices which define the first triangle.

Triangle strips describe another sequence of connected triangles which require only one additional vertex per triangle (after the three vertices for the first triangle). Figure 6

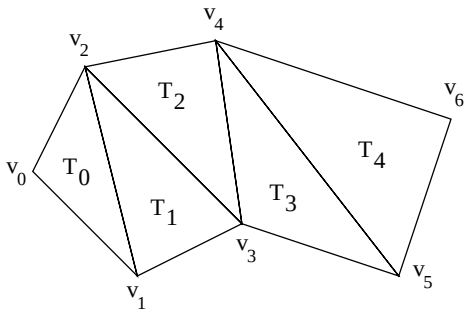


Figure 6: *Triangle strip.*

illustrates a triangle strip which is described, similarly to the triangle fan, as an ordered list of seven vertices $v_0, v_1, \ldots, v_6$. In this case however, each triangle i is defined by the three indices v_i, v_{i+1}, v_{i+2}. Initially the three vertices (v_0, v_1, v_2) of first triangle are sent down the graphics pipeline. For each subsequent triangle, only a single vertex needs to be processed since the other two vertices have already been handled with the previous triangle.

Due to their efficiency, low-level graphics APIs such as OpenGL support triangle fans and strips as rendering primitives. Consequently, considerable research has been dedicated to efficiently computing fans and strips given a triangulated model. Recent efforts by Evans, et al. [19] and Xiang, et al. [99] address this problem.

4.2. Rendering acceleration techniques

Users demand interactivity and realism when visualizing 3D models. Interactive rendering techniques are being designed to address increased model complexity with reduced or no loss in rendering performance. Whenever possible, these techniques conservatively preserve all of the detail so that the rendered image is accurate, while providing an increased level of interactivity. If the model is too complex to render at interactive rates using these techniques, approximations of the model are often used to achieve the desired interactivity, while maintaining a reasonable image quality.

The techniques discussed next include culling, simplification, levels-of-detail, and impostors. Among these, we overview conservative approaches which produce fully accurate images, as well as several approximate methods which sacrifice image fidelity for greater interactivity. Ideally, the trade-offs these approaches make will not be noticeable to the viewer. In most of the cases, however, the algorithms simply attempt to reduce the inaccuracies perceived by the human eye.

4.2.1. Culling

One technique to accelerate rendering is to avoid drawing those portions of a model that do not contribute to the final image. Once these portions (or a subset), have been computed, one can *cull* (i.e., remove), them from the rendering process. Culling is often referred to as *visibility culling*, since it involves determining what geometry is not visible from a given viewpoint and ignoring it from further consideration.

For large scenes, the number of visible triangles is typically much smaller than the total number of triangles. This is true, for instance, for urban models, as well as indoor models, since the walls of the buildings restrict the amount of visible geometry. Thus, efficient visibility algorithms can reduce the amount of geometry that must be processed considerably, thereby increasing the rendering performance.

The z-buffer [6] is one example of a visibility algorithm that is widely implemented in hardware. For each pixel p on the screen, the z-buffer maintains the depth value of the triangle fragment currently contributing to the color buffer at p. If another fragment that is closer to the viewer is rendered, the algorithm will replace the existing value in the z-buffer at p with the depth value corresponding to this fragment. One drawback to the z-buffer is that the pixels may be overwritten many times, which can slow down the graphics performance. Ideally, only the visible triangles are rendered and therefore the amount of overdrawing is minimized. Therefore, the z-buffer is typically used as a filter, and additional culling algorithms are used to approximate the visible set of triangles.

As described in [13], we distinguish between the exact set of all visible triangles V, a conservative visible set C (the set of all visible triangles V plus some invisible ones), and an approximate visible set A (the set of most visible triangles plus some invisible ones). Computing the exact visible set V is a very complicated and expensive operation. Therefore, culling research has focused primarily on computing conservative visible sets or approximate visible sets, using the z-buffer to determine which triangles will actually contribute to the final image. By trading off accuracy in the final rendered image, overall rendering time can often be significantly reduced. We discuss several types of culling including back-face, view frustum, detail, and occlusion culling.

Back-face culling One of the most popular forms of culling is *back-face* culling, which refers to not rendering the geometry that is facing away from the viewer. Assuming that the model is opaque and that its faces are consistently oriented so that their normals point towards the outside of the model, the viewer only sees those faces with normals directed towards the viewpoint. Hence, computing the dot-product of the face normal with a vector between the viewpoint and an arbitrary point on the face indicates whether the face is front or back-facing.

Back-face culling is included in many rendering APIs (e.g., OpenGL) and is a relatively cheap way to conservatively reduce the number of triangles that need to be rendered by roughly one-half. However, back-face culling typ-

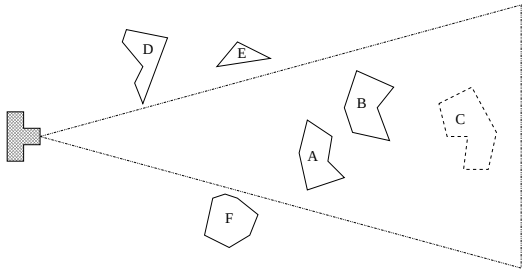


Figure 7: *Culling examples. For the current viewpoint (camera icon), only the front-facing triangles of objects A and B need to be rendered. Objects A and B have four and three back-faces respectively, which do not need to be rendered since they face away from the viewer. Object C is culled since it is occluded from the viewpoint by objects A and B. Objects D, E, and F are not within the viewer's field of view and culled by the view frustum.*

ically processes only a single triangle at a time. Kumar, et al. [49] and Hoff, et al. [35] have shown that it is possible to obtain further speedups by determining whether multiple triangles can be back-face culled using only a single test run in software.

View frustum culling When visualizing a model, the user specifies a viewpoint, a view direction, and a field-of-view (FOV). Together, these form a 3D viewing volume, also called a *view frustum*, which defines the region of space which is visible to the viewer. View frustum culling avoids rendering any geometry which is outside of this viewing volume since it would be clipped away anyway in the graphics pipeline [20]. View frustum culling is typically performed in software at the application level to avoid sending unnecessary triangles to the graphics pipeline.

View frustum culling is a popular conservative culling technique since it can be done efficiently using a hierarchy built upon the input model (see section 4.1). If such a hierarchy exists, it can be compared against the view frustum to determine which objects are outside of the viewer's FOV [9]. This process can be performed using a recursive traversal algorithm starting at the root of the hierarchy. If the current node is outside of the frustum, the traversal of the subtree starting at this node can stop since all of the geometry corresponding to it and its children is not visible to the viewer. If the node is completely contained within the frustum, all of the geometry within the node, and its children, is within the user's FOV and will need to be rendered. Although the hierarchy may still need to be traversed along this branch to render the geometry, no additional view frustum tests will need to be performed since it has already been determined that all of the nodes (along this branch) are within the view frustum. If the current node is partially within the view frustum, the traversal algorithm is called recursively for each of the children of the current node.

Detail culling Unlike the previous two culling methods, *detail culling* is an approximate method that allows faster rendering at the expense of image fidelity. The idea behind this method, also referred to as screen-size culling [60], is to avoid rendering polygons in the model that convey only small details. For an object or group of objects, a bounding volume is computed and projected onto an image plane. If the projected area of the bounding volume is below some threshold, e.g., some number of pixels, the geometry within the bounding volume is not rendered in the final image. Therefore, objects which are very far away from the viewer, and tend to contribute little to the overall image, are culled.

Occlusion culling For those objects that pass all the previously mentioned culling tests, i.e., objects that are front-facing, within the view frustum, and that contribute significantly to the scene, *occlusion culling* avoids rendering the geometry which is hidden from the viewer by other geometry and therefore does not contribute to the final rendered image. While occlusion culling is the most complicated of the culling methods covered in this tutorial, it also has the potential of providing the best results.

Due to the vast coverage of this type of culling in the literature, many classifications of these algorithms have been made. For a complete taxonomy of occlusion culling algorithms, refer to the survey by Cohen-Or et al. [13]. We classify the existing approaches into two main categories: object-space and image-space algorithms. However, in addition to these categories, we also discuss several other properties of occlusion culling algorithms:

- Preprocessing: the occlusion computations may be performed during a preprocessing step and stored, or they may be computed online, as the model is being visualized.
- Viewpoints: the occlusion computations may be valid for a single viewpoint or for a region of viewpoints.
- Occluder fusion: an object may only be occluded from the viewpoint by a collection of other objects, as opposed to any single object. Some algorithms consider only individual occluders, whereas others take into account the combined effect of occluders for more accurate occlusion determination.
- Conservativeness: some algorithms are guaranteed to cull only geometry that is not visible to the viewer, while others may sacrifice image fidelity for speed.
- Dynamic scenes: most culling algorithms are designed for static scenes. Algorithms that rely heavily on preprocessing, especially in object-space, are difficult to extend to dynamic scenes.

Most occlusion culling algorithms maintain one or more hierarchical data structures, in object-space, image-space, or both. Our classification is based on where the actual visibility determination is made.

Object-space culling Coorg and Teller have proposed several object-space culling algorithms. Their first algorithm [16] pre-computes a conservative set *C* of the visible triangles for a region of viewpoints. As the viewpoint moves, they keep track of *visual events* using separating and supporting planes between convex objects that will cause changes to *C*. The second technique proposed by Coorg and Teller [17] also uses separating and supporting planes to determine visibility, but it is much more efficient. Rather than keeping track of visual events, a small set of large occluders is dynamically selected for each viewpoint. These occluders are then used to compute those portions of a model that are within the shadows of the occluders. Both of these methods are conservative, consider only individual occluders, and are applicable to static scenes. A method similar to Coorg and Teller [17] is the *shadow frusta* work of Hudson et al. [41].

Some object-space techniques take advantage of very specific problem domains. For example, there has been a significant amount of occlusion culling research specifically targeted for walkthroughs of architectural models [88, 55, 2] and outdoor urban environments [14, 95, 48, 96]. A property of architectural models that researchers have exploited is that such models can easily be broken into *cells* and *portals*. Cells have boundaries that coincide with opaque surfaces, such as rooms and hallways. Portals correspond to the non-opaque surfaces that are present between cells, such as doors and windows. The technique of Teller and Séquin [88] creates an adjacency graph, which connects the cells via portals, that is then used to compute the cell-to-cell visibility. Each cell *c* has a list of all of the other cells that may potentially be seen from a viewpoint within *c*. This list determines the geometry which could potentially be seen while the viewpoint is within cell *c*, and constitutes a conservative visible set. Further processing can determine which objects are potentially visible from cell *c*. The method combines occluders effectively and can result in significant rendering performance improvements (e.g., a 100 times speed-up). The drawbacks of this approach are that these computations are performed as a significant preprocessing step, which precludes dynamic scenes from being considered, and the memory consumption may be fairly large.

Luebke and Georges [55] use a method similar to Teller and Séquin, although the preprocessing has been greatly reduced by dynamically computing the visibility in such a cell-partitioned model. The approach of Aliaga and Lastra [2] reduces the complexity of rendering the geometry that could be visible from a particular cell by replacing the portals with images instead of the actual geometry from the visible cells. This technique, discussed further in section 4.2.4, trades conservativeness for interactivity.

Outdoor urban environments are another special case of occlusion culling that has been well-studied [14, 95, 48, 96]. These environments are similar to architectural models in that they can often be subdivided easily into cells and portals. Another advantage is that such models can easily be thought of in only 2.5 dimensions. Building layouts are projected onto the xy-plane and a height value for each of the line segments is associated with the floor plans to indicate how high a wall actually is. In general, these techniques are conservative, compute visibility for a region of viewpoints, work well for static scenes, and combine occluders to achieve significant occlusion. Some of these methods require significant preprocessing [14, 48, 96], while others do not [95].

The Prioritized-Layered Projection (PLP) algorithm [46, 47] is a fast, approximate visibility technique for visualizing models with high depth-complexity. For each viewpoint, an estimate of the visible set of triangles is computed and rendered. PLP is an effective method for computing nearly-correct images for time-critical applications. The user can specify how many triangles may be rendered in a particular frame. PLP computes an initial spatial tessellation, that has more cells where there is more geometry and fewer cells where there is less geometry. Cells in the tessellation are assigned (for each viewpoint) a *solidity* value, which is a probabilistic measure based upon the geometry in the neighboring cells. Solidity values are computed online as the spatial tessellation is being traversed. During this traversal, cells are inserted into a priority queue, based upon their solidity, to determine which cells are most likely to be visible. Cells are removed from the queue and their geometry is rendered, until the triangle budget is reached. PLP can take advantage of occluder fusion. In its current form, it is only applicable to static scenes.

Image-space culling Although many image-space techniques also use hierarchies built in object-space, they clearly differ from object-space methods in that the occlusion computation is performed in viewing coordinates. In general, these techniques compute visibility by filling up the image as objects are rendered and culling subsequent objects against the already filled parts of the image.

Due to the discrete nature of an image, image-space techniques are usually simpler to implement and are more robust than object-space algorithms. Also, it is typically easier to approximate the visible set, for an even greater performance improvement.

Image-space techniques are often preferred over their object-space counterparts when the model is composed of many small triangles without clearly defined occluders. By projecting a large number of small triangles that are individually insignificant occluders, the accumulated effect in the image plane can be significant. A disadvantage of the image-space techniques is that they rely on reading back information from the framebuffer, which is usually inefficient.

The Hierarchical Z-buffer (HZB) of Greene, et al. [29] extends the traditional z-buffer by maintaining an *image Z pyramid* to quickly reject hidden geometry. A second data structure, an octree, is also built upon the geometry in object-

space. At run-time, the octree is traversed and each node of the tree is queried against the current HZB. If the current node is determined to be behind previously seen geometry, then this node is skipped, together with all of its children and their geometry. Otherwise, the geometry is rendered, the new z values are propagated through the HZB, and the children of the node are visited. Temporal coherence is exploited by pre-rendering previously visible nodes. The main weakness of the HZB approach is that the queries against the HZB, as well as the updates, assume the availability of special graphics hardware to achieve interactive rates. These ideas were extended in [28].

The Hierarchical Occlusion Maps (HOM) [100] approach is similar to the HZB in that it uses hierarchies within both object and image-space. Initially, the image-space HOMs are built by rendering geometry with a high potential for occlusion, i.e., geometry that is very close to the current viewpoint. To determine whether the remaining geometry is visible from the current viewpoint, the object-space hierarchy is traversed and each node is checked against the HOM, using, if necessary, a conservative depth-test to determine occlusion. This technique supports occluder fusion and, if desired, culling of objects that contribute only a few pixels to the scene. The HOM technique is one of the few existing methods that works for dynamic scenes.

Bartz et al. [4] introduce another technique based on an object-space hierarchy (a *sloppy n*-ary space partitioning tree in this case) and an image-based occlusion test. Their method uses OpenGL to scan-convert bounding volumes into a *virtual occlusion buffer.* This buffer is then sampled to detect changes triggered by nodes containing potentially visible geometry. The sampling density can be adjusted to provide adaptive, non-conservative culling.

The techniques presented in [4] may be implemented in hardware for greater efficiency, although no such implementation exists at this time. However, occlusion culling hardware does exist. The VISUALIZE fx family of graphics accelerators from Hewlett-Packard (HP) supports an extension to OpenGL that allows a query to be made against the z-buffer [76]. This query indicates whether the z-buffer would have been modified, if specified geometry had been rendered. For example, a bounding volume may be used to approximate some complex geometry. The HP extension can then determine if this bounding volume would have modified the z-buffer. If not, the complex geometry inside it need not be rendered at all.

4.2.2. Simplification

Even with all of the culling techniques described above, some 3D models are still too complex to render interactively. Another approach to address this problem, which may be used in addition to culling, is *simplification*, i.e., the process of reducing the complexity of a given model. In its most basic form, simplification takes a geometric model and produces one or more geometric representations of that model that are faster to render. This gain in rendering efficiency comes at the price of an approximate image. The rendered image will likely not be exactly the same as if the original model were rendered, although with good simplification algorithms, the user may not notice the difference. Herein lies the main problem of simplification: *can a simplified model be computed such that it is significantly faster to render than the original, and yet the appearance of the original model is preserved?*

The simplification problem can been formulated in several ways such that one can speak of optimal solutions. For a good discussion of these formulations and the difficulty that exists in solving them, the reader is referred to [91, 10]. As we have already mentioned, models tend to be very complex, which means that simplification algorithms must be very efficient. The majority of the algorithms use inexpensive local operations, applied repeatedly, until a target model complexity is reached. The running time of such algorithms is typically $O(n \log n)$, for a model with n triangles. While local operations may produce good results, they typically cannot make any guarantees as to the quality of the overall solution. There has been considerable work on the problem of simplification, and many different techniques have been applied. Next, we classify these techniques according to the type of local operation applied during the simplification process.

Vertex clustering Rossignac and Borrel [70] proposed a very simple and robust technique for reducing the complexity of a model. In their approach, the model is placed within a uniform grid of cells and the vertices within each of these cells are clustered together to form a single vertex. Only those triangles whose three vertices fall within different cells will remain in the simplified model. Vertex clustering works for all models since it makes no assumptions about their topology. However, the simplified models are not always good approximations of the original. Low and Tan [52] and Luebke and Erikson [53] independently achieved better simplification results by generalizing this method to use an adaptive spatial partitioning.

Face contraction The face contraction operation is similar to vertex clustering in that the three vertices of a triangular face of a model are contracted into a single vertex (typically resulting in four faces being removed from the model). However, the significant difference is that the underlying topology of the model determines which vertices are "clustered" together. This helps to better preserve the important features of the original model, although it also places some restrictions on what geometry is removed during simplification. Face contractions have been proposed by Hamann [32] and Gieng et al. [26].

Edge contraction For more precise control during the simplification process, the majority of recent techniques [39, 30, 68, 24, 25, 51, 38] have opted to use edge contractions

to reduce model complexity. This operation, illustrated in Figure 8, merges two adjacent vertices into one and removes all of the triangles, usually two, that contained this edge.

All methods that use edge contractions must make two important decisions. The first is in what order the edges should be contracted. Most algorithms use a greedy approach that contracts, at each iteration, the edge that is "cheapest", according to some cost function. The second decision is where to place the newly created vertex. The new location can be decided by *subset placement*, at either endpoint or the midpoint of the edge, or by *optimal placement*, at the location that minimizes the cost function [24].

Garland and Heckbert [24] generalize the edge contraction operation to allow for arbitrary pairs of vertices to be contracted. This can be advantageous in that it allows for disconnected components of the model to be fused together. However, such contractions may lead to non-manifold results.

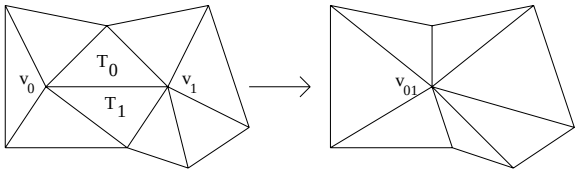


Figure 8: *The edge contraction operation. Contracting the edge between vertices v_0 and v_1 removes triangles T_0 and T_1 from the mesh. The two vertices are replaced with a single vertex v_{01}.*

Vertex decimation Another local operation for reducing a model's complexity is known as vertex decimation. A vertex of a model is selected for removal and the n adjacent triangles are also removed. This results in a hole in the model which is then re-triangulated, using only $n-2$ triangles. Such techniques have been proposed by Schroeder et al. [75] and Turk [90] and are reasonably efficient and effective, although they are inherently limited to manifold surfaces.

Recently, more attention has been placed upon preserving the overall appearance of a model during the simplification process, as opposed to simplifying only the geometry of the object. Models may have associated colors and normals (per vertex, per corner, or per face), as well as texture coordinates per vertex. By addressing these additional attributes in the model, the total overall appearance can be even greater [25, 38, 11, 12, 7].

So far, we have discussed "static" simplification algorithms which produce an independent set of representations for a given model. When dealing with such representations, an important issue is deciding which of them to render for a given viewpoint. This issue is discussed in section 4.2.3. In contrast to static methods, research has also been done on "dynamic" simplification, which occurs at run-time and is usually dependent upon the viewpoint and view direction. While view dependent methods [37, 54, 98] may be more computationally expensive and may actually require more memory than the static approaches, they have the advantage that they provide a more realistic image, without many of the artifacts which may result from switching between static representations.

There are several surveys on simplification algorithms. Heckbert and Garland [34] provide a good taxonomy of simplification algorithms. The main classification upon which they distinguish algorithms is the types of models upon which the algorithms run, e.g., manifolds, non-manifolds, or height fields. Cignoni et al. [8] present a characterization of several simplification methods, based on the simplification strategy, the error management policy, and the capability to preserve mesh characteristics. Additionally, they report the results obtained using six simplification codes that are in the public domain. A comparison of the computational cost and the approximation accuracy of the output meshes is included. A more recent survey on the simplification problem is presented in [10].

4.2.3. Levels-of-detail

Static simplification algorithms produce an independent sequence of representations for a model, each of which are computationally less expensive to render than the previous one. Since each representation is faster to render, and therefore contains fewer details of the model, they are often called *levels-of-detail* (LODs). LODs are typically used when a model is moving away from the viewer. In such cases, small details in the model are no longer perceived by the viewer and therefore need not be rendered.

Recall that simplification algorithms, and LODs, are used as an interactive rendering technique when a model is too complex to render in real-time. When using LODs to trade accuracy in the images for greater rendering performance, an important issue to address is which LOD should be rendered for a given viewpoint. The most naive approach is to select LODs based upon their distance from the viewer. Although this is a very efficient technique, which determines distance and selects an appropriate LOD using a look-up table, disturbing artifacts may occur in the rendered images (e.g., popping when switching between LODs).

Rather than discretely selecting a single LOD to render for each viewpoint, alternative schemes can be used to improve the overall visual quality at the expense of additional rendering time. One example of such a scheme is alpha-blending, which fades from one LOD to another over the course of several frames. During these frames, the LODs are rendered with the blending coefficients slowly changing to fade from one LOD to the other. Another technique to smoothly transition between LODs is to use *morphing*, which actually changes the geometry that gets rendered from one LOD to

the other during the transition. Geomorph LODs [36], essentially animate the edge collapse operation performed during the simplification algorithm. When going from one representation to the next simpler one, the two vertices that are being contracted are animated so that the transition between LODs is seamless. Geomorphs can only be used for simplification algorithms based on edge contractions and are a good example of dynamic levels-of-detail.

4.2.4. Impostors

Despite the use of culling and simplification algorithms, some models are still too complex to render at interactive rates even when only the simplest LOD is used for all of the objects in the model. In such cases, other representations are needed to reduce the rendering complexity even further, while still preserving the overall appearance of the model. One example of such a representation was mentioned in section 4.2.1, as images were used to replace the geometry that was visible through the portals within architectural models. Such images are known as *impostors*.

Maciel and Shirley [56] introduced the notion of an impostor, which has come to mean an image of complex geometry that is texture mapped onto a rectangle. To prevent occluding objects that may be present behind the impostor, the image is opaque where the geometry is present and is transparent everywhere else. To be effective, an impostor must be faster to render than the geometry it is replacing, it should closely resemble the geometry, and it should be reusable for several viewpoints. Since the movement of a projected image of geometry diminishes with an increased distance from the viewer, impostors can best be used if the geometry is slowly moving and if it is located far from the viewer. When the distance from the viewer to the impostor is small, the fixed resolution of the impostor may become obvious and the individual pixels will become apparent. Before this disturbing effect occurs, the impostor should either be replaced with another representation of the geometry, or another impostor should be selected.

The original presentation of an impostor by Maciel and Shirley was more general than the simple description given above. They divided impostors into two classes: view dependent and view independent. One example of view dependent impostors include the images of some geometry mapped onto the appropriate faces of the geometry's bounding box. Then, depending upon which of the six faces of the bounding box was visible to the viewer, the appropriate impostor would be displayed. Another well-known example is a *billboard* [60], which works well for symmetric objects such as pine trees since it is designed to rotate so that it always faces the viewer. View independent impostors include static LODs and simple bounding boxes colored using a representative color for the geometry within the box.

An extension of the notion of an impostor is the *nailboard* [72], which is an impostor that also maintains a depth buffer of the same size as the impostor image. Nailboards are superior to impostors because they can avoid visibility problems when the impostor rectangle intersects nearby geometry. The depth buffer values of the nailboard are used as offsets to correctly determine occlusion. A related concept is the *layered depth image* introduced by Shade et al. [77], which may have several depth values per pixel.

Hierarchical image caching (HIC), invented independently by Schaufler and Stürzlinger [73] and Shade et al. [78], uses impostors in a hierarchy for improved performance. Initially, the model is partitioned into a hierarchy of boxes, each of which has an impostor created for it. Additional impostors are created for the parents in the hierarchy. HIC typically works for static scenes, although it could be combined with nailboards to handle dynamic objects.

4.3. Model perception and representation selection

We have presented several techniques for accelerating the rendering of 3D models. In some cases (e.g., back-face culling), the algorithms are conservative and the rendered images are completely accurate. In other cases (e.g., simplification or impostors), the algorithms tradeoff accuracy for increased interactivity. Ideally, the inaccuracies introduced by the latter will go unnoticed by the viewer. Related to this topic, we discuss the issue of model perception. By being aware of the factors that influence perception, we can design more effective interactive rendering systems. For those cases when accuracy must be sacrificed to achieve interactivity, understanding how a person perceives a rendered image, can influence the design of a system that will select the most appropriate representation (e.g., LOD or impostor) to render for a model component. Two approaches that take such perception issues into account are discussed next [23, 56].

The purpose of rendering images is to convey information to a viewer. This information can be as simple as what a model looks like, or as complex as what properties (e.g., stress, vibration) the model will exhibit during a simulation. By understanding how people perceive what they see, we can design more effective rendering systems. There are several good references on the study of visual perception and its application to computer graphics [43, 44, 33]. Each of these presents fundamental findings on how we perceive certain aspects (e.g., color or shape) within rendered images. These findings enable us to more effectively convey information as we create different representations of the components of a model and then choose which of these representations to render given certain viewing parameters. For example, such information is useful when selecting which level-of-detail or which impostor to use for a group of model components. Next, we discuss several characteristics that humans perceive and why knowing how we perceive them is important to rendering systems. These include color, motion, texture, and shape.

Using color to add detail to a model is both natural and useful. For instance, a viewer can immediately identify geometry that belongs to the same part of a model making it easier to understand what is being displayed. One popular example of a color model is the hue-lightness-saturation model (HLS) [20], where hue is the basic quality of a color, lightness is the amount of light emitted, and saturation is the vividness of the color. When creating a model, or any representation to replace the model, it is important to know that when objects differ only in hue and saturation, but not in lightness, then certain visual information (e.g., perspective depth cues) is lost to the viewer and the rendered images are not as effective. This information is particularly valuable to have when initially creating a model, as well as when simplifying a model to create LODs.

Another aspect that conveys significant information to the viewer is that of motion. For example, as components of a model are moving, information about their relative depth and shape, as well as how they are grouped together, is passed on to the viewer. Perhaps the most important result of motion is to realize that as an object moves into our periphery, we are no longer able to interpret the structure of that object. This aspect, also referred to as *focus*, is important when selecting which LODs to use for objects that are not in focus. It is also useful when designing impostors: if the impostor is not the main focus in the image, then a lower resolution texture map could be used, thereby saving computation time and memory.

Texture maps are well-known to be an effective means of conveying complicated information to the viewer. Rather than modeling a brick wall exactly, an image of a brick wall can be mapped onto a simple rectangle to give the same appearance to the viewer. It is important to know if textures are being used in conjunction with geometry when LODs are created. The textures will still convey a great deal of information to the viewer, so we might want to simplify the geometry more than we would have in the absence of textures.

During the simplification process, one of the most important features to maintain is the overall shape of the object. This is especially true around the silhouette of the object, since changes in the objects silhouette edges may be distracting to the viewer. Thus, with this additional knowledge, simplification algorithms can be designed to limit the amount of disturbing effects that may result when looking at coarser models. Another important feature in maintaining the shape of an object is the shading [44].

In addition to the individual effects of the characteristics perceived by the human visual system, there is also a combined effect in some cases. That is, perceptual characteristics are not independent. The perception of one characteristic may very well be influenced by another. As an example, studies have shown that the color of an object can influence the perceived size of an object [44].

Funkhouser and Séquin An interactive rendering system that guarantees a constant frame rate regardless of model complexity was designed by Funkhouser and Séquin [23]. When the complexity is extremely high, this system trades accuracy for interactivity. This is accomplished by maintaining several representations (in this case LODs) for each of the model components, and then deciding which representation to render to maximize the quality of the displayed image without exceeding the available frame time.

Funkhouser and Séquin initially apply view frustum culling to the components of a model. A heuristic algorithm is then applied that adapts the selection of representations for all components being considered. This algorithm is predictive in that it makes its selection based on the desired frame rate and which components are within the viewer's field of view, as opposed to a reactive algorithm which makes its selections based upon the time it took to render the previous frame. To maximize the image quality without exceeding the available frame time, *cost* and *benefit* functions are utilized. For each component in the model, a benefit-cost ratio is computed and the components are selected for rendering in a decreasing order of their benefit-cost ratio.

The cost function attempts to measure how expensive it is to render a particular representation of a component of the model. It is based upon the per-primitive costs (e.g., coordinate transformations, lighting calculations) and per-pixel costs (e.g., rasterization, depth computations). The coefficients of the cost function are determined by the particular graphics hardware used.

The benefit function estimates how well a representation contributes to the final image. To precisely compute this benefit would require taking many factors into account, such as visibility and human perception. As this becomes much too complicated, Funkhouser and Séquin have simplified the function to use a linear combination of the following characteristics: size, importance, accuracy, focus, motion, and hysteresis. We discuss each of these below.

The *size* of an object refers to the projected screen-space coverage. An object with a large screen-space size is determined to be more important than an object with a small size. For certain models, some objects are inherently more important than others. For example, the walls and floors of an architectural model may provide much more information to the user than the actual office furniture. In such cases, the user can specify that these objects have a greater *importance* and should be more likely to be rendered with a high-quality representation. The *accuracy* of a representation measures how similar it is to the original object, in terms of color, motion, shape, etc. The *focus* function has a higher value for objects near the center of the screen. As previously discussed, the small details of an object are not perceived when the object is in our periphery. Objects moving very quickly are determined to be not as important as slowly moving ones, since due to *motion* they may appear on the screen for only a short

amount of time. *Hysteresis* assumes that if the representation is changing from the previously used one, the benefit will be smaller because of potentially disturbing artifacts (e.g., popping.)

Maciel and Shirley A drawback of the previous system is that due to the greedy heuristic that determines which representation should be rendered for each object, it is possible that many objects with low benefit-cost ratios may not be displayed at all, leaving large blank spots in the final image. Maciel and Shirley [56] identified this problem and used impostors to accelerate rendering.

In general, the system of Maciel and Shirley can be considered an extension to the previous work by Funkhouser and Séquin. The systems differ, however, in several important ways. First, in [56] the entire model is organized into a single hierarchy that contains impostors (including LODs) for objects, as well as for clusters of objects. By clustering objects together, a single impostor can be used and a significant reduction in the model complexity can be achieved. Another crucial difference between these systems is that in [56] for every viewpoint every object within the view frustum, or a cluster that contains that object, will be rendered. Thus, no large gaps in the final rendered image will exist. Another difference is that Maciel and Shirley assume that their system runs on a multiprocessor machine that has texture mapping capabilities. They also rely on the hardware to automatically create the hierarchy, generate the impostors, and compute their rendering cost.

In determining the contribution of a model component to the rendered image, the two systems have many similarities, but several important distinctions. Maciel and Shirley break the contribution up into two categories: the contribution intrinsic to a component and the contribution intrinsic to a representation of that component. Similar to Funkhouser and Séquin, the factors that are intrinsic to a component include its size, focus, motion, and importance. The per-component benefit is a weighted average of these factors and is used to select the appropriate representation to render for the object. Intrinsic to the representation of an object is its accuracy to the full detail object. A major difference between the two systems is how the accuracy of the representation is actually computed. Since Maciel and Shirley use impostors as representations of components or clusters of components, they must compute the accuracies of both view dependent and view independent representations. Next, we describe how accuracies are computed.

To select between the view dependent and view independent representations, Maciel and Shirley determine the accuracies of each and then select the one with the highest accuracy-cost ratio, where the cost quantifies how expensive it is to render a particular representation. The accuracy of the representations is computed based upon the viewing angle and the distance to the object. The space of all viewing directions is discretized and an orthographic projection is used to compute the accuracy of the representation. For each of the sampled view directions, the accuracy is measured and recorded in a table, based upon the representation and view direction. To measure the accuracy, they use simple image processing techniques. They do not immediately do pixel-by-pixel comparisons, since two very similar but slightly off images would have a very low similarity value. Instead, since the achromatic channel of vision is the most important to shape recognition, they obtain a gray scale version of the images by averaging the RGB components. A Laplacian operator is used to determine edges followed by a blurring step, which increases the probability of matching the two images. Finally, the images are compared pixel-by-pixel.

Maciel and Shirley note that a cluster of components typically conveys more information than the sum of the individual components. However, since it is typically difficult to account for this phenomenon without knowledge of the models, they make a simplifying assumption and merely sum up the individual contributions to compute the benefit of a cluster. They also note that in some cases, impostor selection is very easy. For example, if the image-space size of a component is below some threshold, a simple impostor (e.g., average color bounding box) may be used. Otherwise, if an object's image-space size is above the threshold, the full detail model is used. Additional threshold values may be used to select various levels-of-detail.

SUMMARY

Interactive rendering of complex models is a fundamental problem in computer graphics. Users demand realistic-looking models and the ability to interact with them in real-time. Many algorithms have been developed to address this problem. Culling is a technique that avoids rendering geometry that does not contribute to the final image. Examples of culling include back-face, view frustum, detail, and occlusion culling. In some cases, culling is not sufficient to provide interactivity. Consequently, the complexity of models can be reduced by using a process called simplification. Simplification can be used to produce various representations of a model, each of which are more efficient to render than the previous one. While these representations, also called levels-of-detail, can increase the interactivity, they tradeoff accuracy of the final rendered image. An alternative approach to reducing complexity is to replace the geometry of a model with an image that is texture mapped onto a rectangle. This image is commonly referred to as an impostor.

To design a more efficient rendering system, it is important to understand the factors that influence human perception. Knowing how a human perceives color, motion, shape, and texture can influence the rendering techniques selected to provide interactivity and realism.

---

5. Overview of 3D Transfer Technologies

Efficient delivery of 3D models is dependent upon the data structures used to represent the data, as well as on the method(s) selected for transmission. In this section we describe several file formats that have emerged for storage and transmission of 3D data and we discuss alternative strategies to en-masse downloading of models.

5.1. File formats for 3D data transfer

In this section, we present several file formats for storage and transmission of 3D models over networks. These formats include VRML, Java 3D, XGL, MPEG-4, and MetaStream.

5.1.1. VRML

Most often, 3D models are transmitted over networks using the Virtual Reality Modeling Language (VRML). VRML is an ASCII text format in which models are fully described in terms of their geometry and their attributes. VRML is designed to encode polygonal models composed of vertices and planar faces. In addition, it offers the ability to specify commonly used semantics found in today's 3D applications, such as light sources, material properties, texture mapping, fog, hierarchical transformations, viewpoints, and animation.

Among the advantages of VRML are:

- Popularity: a large number of applications use VRML as a graphics interchange format.
- Integration: VRML provides technology that integrates 2D and 3D graphics with text and other types of multimedia.
- Distributed content: one of the key features of VRML is its support of the World Wide Web. VRML has several nodes which use URLs to connect the scene graph to the network. These nodes include fetch-on-demand and hyperlinks to other VRML content and Uniform Resource Locators (URLs). Figure 9 sketches the flow of information in a network environment during the processing of a VRML file.
- Standardization: VRML is at the basis of the MPEG-4 standard for transmission of graphics content; also, VRML files may contain references to data in many other standard file formats (e.g., JPEG, PNG, GIF for textures, WAV, MIDI for sound).
- HTML-compatible: VRML files may be embedded in Web pages using the HTML <OBJECT> tag.
- Reusability: it is relatively easy to combine VRML files created by various people or tools to create new files.

As 3D models required or generated by various applications become routinely more complex, the use of VRML for transmission becomes highly inefficient. Representing and downloading data as ASCII text is typically time-consuming and requires considerable amounts of network bandwidth

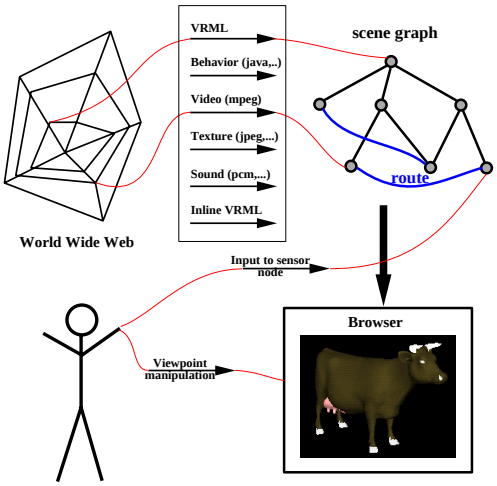


Figure 9: *Flow of information while processing a VRML file.*

and storage on the receiving computers. The transmission of large VRML files in their entirety, without the possibility of displaying the intermediate data, causes significant delays between the time users request a model and the time they are able to view it. In many cases, even after the entire data is downloaded, the amount of data is typically to large to be processed locally, and additional delays occur as the user interacts with the model. Finally, VRML implementations do not usually differentiate between necessary and unnecessary information. Given the limited size of a computer screen, not all parts of a complex model are relevant at any given time. A lot of valuable time and resources could potentially be saved during model transfer by taking into account only relevant data. Appendix A contains a brief note on the history of the VRML format, as well as an example of a simple 3D object described in VRML.

VRML200X-X3D While little content has been generated which exploits the VR aspects of VRML 2.0 (animation, interaction, and behavior), the format continues to enjoy considerable success as the standard way to express 3D worlds.

In early 1999, the consortium that manages VRML issued a press release expressing their intent to create VRML200X-X3D, or X3D for short. The stated goals of X3D are backward compatibility with VRML 2.0, integration with XML, componentization, and extensibility.

5.1.2. Java 3D

Java 3D is a proprietary, high-level, platform-independent, 3D graphics programming API designed to enable high-performance implementations across a wide range of platforms. Java 3D has been designed to handle everything from the simple display of 3D logos to the complex navigation of large "moving" virtual worlds. The API provides functionality for real-time 3D simulations such as those found in

VRML 2.0 and advanced 3D games. Java 3D is not an authoring application, its sole purpose is to provide the fastest possible runtime rendering of potentially very complex environments.

As a high-level API Java 3D is designed to exploit traditional low-level APIs such as OpenGL, Direct3D, and QuickDraw3D. Prior to running Java 3D applications, users must download and install a runtime environment for Java 3D and Java.

Recently there has been some significant cooperation between the SUN Java 3D community and the VRML200X-X3D community. A tangible result is Xj3D, a set of (java-based) interfaces, library extensions, and applications that enable VRML 2.0 and X3D viewing in a Java 3D runtime environment. Additional details can be found in [92].

5.1.3. XGL

Typically, VRML is rendered using one of three low-level mechanisms: OpenGL, Direct3D, or QuickDraw3D. The XGL file format uses the XML 1.0 syntax to encode geometry and properties for subsequent rendering with OpenGL. Details on the XGL project can be found in [97]. Since it uses XML, XGL files are easy to both create and parse. Also, a number of free parsers are readily available.

The XGL format is complete in the sense that a program can use the format to losslessly encode all of its OpenGL objects, transmit them to another program, and render the identical OpenGL content on the receiving machine. An example of an XGL file is illustrated in Appendix B.

Because the XGL format uses XML syntax, it is easy to write applications that read and write XGL. To export XGL, an application needs only write out a text file using the XML syntax. XGL supports a variety of mechanisms for arranging and referencing data within the file, so it should be easy to find a method of exporting data that is easy to implement for any graphics system. There are a number of free XML parsers available, eliminating the need to implement a complex parser when building an XGL reader.

5.1.4. MPEG-4

The Moving Picture Coding Experts Group (MPEG) pursues international standards for compression, decompression, processing and coded representation of moving pictures, audio and their combination. MPEG-4 is a standard for multimedia applications based on the scene graph technology of VRML 2.0. The joint use of this technology represents a historical convergence between computer graphics and multimedia.

Unlike VRML 2.0, the scene graph in MPEG-4 has a binary encoding. The MPEG-4 scene graph encoding is known as the Binary Format for Scene Descriptions (BIFS). The standard adds many additional nodes (100 vs. 54) and complexity to the already very complex VRML 2.0 standard. Although VRML and MPEG-4 have much in common, perhaps the biggest difference is in the network-delivery mechanisms. In MPEG-4 all information is conveyed using streams. In contrast, VRML2.0 does not specify any requirements or mechanisms for streams and content is usually delivered by using TCP to perform a file transfer.

MPEG-4 covers the encoding of a large variety of audio-visual media objects. These media objects can be either natural or synthetic. Composition and spatial relationships are defined using a VRML-style scene graph. The leaf nodes of the scene graph contain media objects. The standard provides support for the streaming of the scene graph structure, streaming of updates to the scene graph, and for the streaming of content to individual nodes. In contrast, the VRML 2.0 standard does not contain any streaming specifications. Similar to VRML 2.0, the scene graph structure and the node attributes are not necessarily static. In MPEG-4, nodes, node fields, and graph structure can all change as the result of user interactions or transmitted (streamed) updates. MPEG-4 specifies how streams are multiplexed and synchronized. The standard provides a generic set of QoS descriptors for MPEG-4 media, but it does not attempt to specify how these parameters are exploited in the network layers. The standard also specifies a back channel to enable the user to communicate with the transmitter. Appendix C summarizes the architecture of an MPEG-4 receiver. Figure 10 illustrates how an MPEG-4 transmitter and an MPEG-4 receiver would use the Internet to communicate.

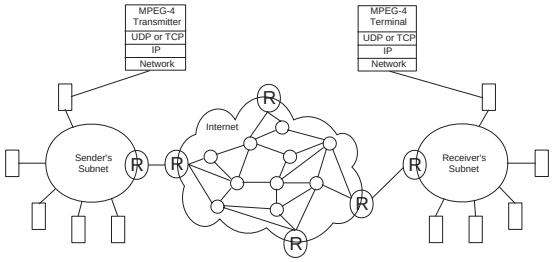


Figure 10: *Streaming MPEG-4 contents on the Internet (the letter R indicates router nodes).*

5.1.5. MetaStream

MetaStream (MTS) is a binary file format developed by the MetaCreations corporation as a multiresolution storage format for 3D models and their textures. The format combines the advantages of progressive representations with geometric compression, to deliver 3D graphics over the Internet.

In MTS, data is organized hierarchically into *blocks*. Several standard block types (e.g., image, 3D geometry) are provided and user-defined block types may be defined to extend

the format as needed. In addition, blocks are classified as *carrier blocks* intended for storage and organization of other blocks, and *data blocks* for storage of the actual data. Data of different kinds may stored together into *group blocks* using one of several strategies that define how data blocks of each kind are to be interleaved with others. These strategies allow the creator of an MTS file to decide upon the streaming behavior of the file [1].

3D models are encoded into MTS files by specifying the so-called *base mesh* (i.e., the coarsest resolution mesh), a sequence of *refinement records* that lead from the base mesh to the final model, and *plug-in properties* that allow extra information to be associated with the models. For compression purposes, a combination of techniques is used to encode the data into a compact format. These include quantization, entropy coding, and the use of special data formats.

There are many advantages offered by the MTS format, some of the most compelling ones being good compression ratios, streaming, and high visual quality of the final renderings. Additionally, the format supports animation and interactivity and its creators have designed it to integrate with HTML and XML.

5.2. Techniques for Efficient Delivery of 3D Data

A number of techniques have been developed in recent years to reduce the delay perceived when 3D data is transferred over a network. Such methods can be classified into two broad classes. On one hand, compression methods reduce the amount of data to be transferred and hence, the time to transfer it, by encoding the data in a form that is more compact than the original representation. On the other hand, streaming methods deliver the data progressively and aim to reduce the time between the request of a model and the display of a first meaningful representation of the model. In this section, we define the basic concepts related to compression and streaming and we overview some of the existing techniques in each category.

5.2.1. Compression

Data compression is concerned with representing information in a compact form by identifying structures that exist in the data. An early example of compression is the Morse code developed in the 19th century by Samuel Morse. Letters sent by telegraph were encoded with dots and dashes and Morse noticed that certain letters occur more often than others. To reduce the average time for the delivery of a message, he assigned shorter sequences to letters occurring more frequently, and longer sequences to those encountered less often. In addition to exploiting the structure of the data (e.g., statistical structure in the previous example), the characteristics of the users of the data may also be exploited for compression purposes. For instance, when delivering audio and video data, perceptual limitations are taken into account to achieve compression by discarding information that is irrelevant.

Compression is routinely used in a variety of applications for compression of different kinds of data, including text, audio, video, and, more recently, 3D geometry. Before we overview the most recent developments in techniques for compression of 3D data, we briefly introduce basic concepts and terms related to compression in general.

Fundamentals In general, the term *data compression* is used to denote two algorithms: one that generates a compressed representation X_c of a given data set X, and one that operates on the compressed representation X_c to recover a reconstruction X_r of X. Thus compression schemes can be broadly classified into *lossless* if the recovered data X_r is identical to the original X, or *lossy* if X_r is different from X [71].

The performance of a compression technique is typically evaluated in several ways. The *compression ratio* measures the number of bits required to represent the data before compression with respect to the number of bits required after compression. The *rate* of compression is another performance indicator that reflects the average number of bits required to represent a single sample. For lossy schemes, it is also useful to quantify the difference between the original and the reconstruction, that is, to evaluate the *fidelity* or the *quality* of the reconstruction.

Entropy is a concept introduced by Claude Shannon [79] as a quantitative measure of the information associated with a given experiment. For a given set of independent events A_i which are outcomes of a random experiment, the entropy of the experiment (i.e., the amount of information associated with it) is:

$$H = -\sum P(A_i) \log P(A_i)$$

Shannon showed that, if an experiment is a source that outputs symbols A_i from a set A, then the entropy is a measure of the average number of binary symbols needed to code the output of the source. In particular, he showed that the best a lossless compression scheme can do is to encode the output using an average number of bits equal to the entropy of the source.

In general, determining the entropy of a physical source is not feasible and estimates must be used instead, based on assumptions about the structure of the data. These assumptions constitute the *model* for the data, and good models typically lead to efficient compression schemes.

Coding means assigning binary sequences (*codewords*) to the elements of an alphabet. The set of the binary sequences constitutes a *code*. If the same number of bits is used to encode every symbol of an alphabet, the corresponding code is termed *fixed-length*. However, as mentioned previously, it is a good idea to use fewer bits to represent symbols that occur less often, that is, to use a *variable-length* code. For

such codes, an important property is *unique decodability*, i.e., any given sequence of codewords can be decoded in a unique way. For example, a code with the property that none of the codewords is a prefix of another codeword satisfies this property. Such a code is called a *prefix code*. A simple way to represent a prefix code is using binary trees: at each node, the left branch corresponds to the bit 0 and the right branch corresponds to the bit 1 (or vice-versa). Codewords are associated with the leaves of the tree.

Symbol	Probability	Codeword
a	0.4	0
b	0.2	10
c	0.2	110
d	0.1	1110
e	0.1	1111

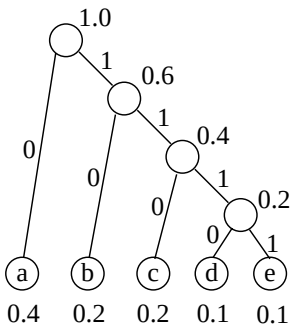


Figure 11: *The Huffman encoding procedure: starting with the alphabet symbols as the leaves of a binary tree, the two nodes with the smallest probabilities are always combined first. The probability of a parent is equal to the sum of the probabilities of the children. The codeword corresponding to each symbol is derived by traversing the tree from the root to the corresponding leaf and assigning a* 0 *to each left branch and a* 1 *to each right branch.*

Huffman codes are prefix codes and were developed by D. Huffman [42] as part of a class assignment! The procedure for generating these codes exploits the ideas that symbols that have a higher probability of occurrence should occur more frequently and that the two symbols that occur least frequently must have the same length. The only additional constraint added to these ideas by the Huffman procedure is that the two lowest probability codes differ only in the last bit. An example of how to build a Huffman code using a binary tree is shown in Figure 11 for a small alphabet. It has been shown that these codes are optimal for a given statistical model.

Arithmetic coding [71] is another popular method for generating variable length codes. It is especially suitable for small alphabets and alphabets with highly skewed probabilities. The idea is to tag a sequence of symbols with a unique identifier to distinguish it from other sequences. The identifier is chosen to be a real number in the interval [0, 1). The cumulative distribution function associated with the source is used to partition the unit interval into subintervals. The first symbol in the sequence restricts the tag to one of the subintervals, which is subdivided in the same proportions as the original. The process is repeated for succeeding symbols. Major advantages of arithmetic coding include ease of adapting codes to changing input statistics and the ability to separate modeling and coding procedures for increased flexibility.

Compression of 3D models A number of powerful techniques have emerged over the past few years for compression of 3D polygonal and, more recently, tetrahedral meshes. The majority of existing schemes are termed as lossless, but lossy techniques have also been proposed.

Most 3D compression schemes are based on efficient coding of the topological information. They use connectivity to predict the position of a vertex with respect to its neighbors and then variable length codes are used to encode corrections to the predicted positions. Vertex coordinates are typically quantized to finite precision (10-14 bits). More recent approaches investigate the use of signal processing techniques to compress geometry. In the remainder of this section, we provide pointers to some of the most recent work published in this area, and we overview in detail the compression method of Taubin and Rossignac that has become part of the MPEG-4 standard. For additional details on geometric compression, we recommend the SIGGRAPH and Eurographics 2000 courses on this topic.

In 1995, M. Deering introduced the concept of *geometry compression* and the idea of representing polygonal data with fewer bits than using conventional representations, for only a slight loss in quality [18]. Polygonal data (i.e., triangles) is converted to an efficient linear strip format that allows for compact representation of geometry while maintaining a linear data structure. Based on this representation, vertex positions and attributes (i.e., normals and colors) are quantized. Subsequently, the differences between neighboring quantized values are encoded using variable-length Huffman codes. Compression ratios achieved with this method are between 6:1 and 10:1, depending on the original representation and the final quality level desired. The speeds reported are 3,000 triangles per second for compression, and 10,000 triangles per second for decompression.

In 1996, Taubin and Rossignac introduced the *topological surgery* procedure for compression of 3D meshes based on spanning trees that can be encoded into extremely compact form [87]. Topological surgery expands upon the work of Deering, by providing lossless encoding and higher compression ratios for the connectivity information, better organization of the mesh vertices for compression, as well as the means to generate long triangle strips for efficient rendering using graphics adapters.

According to this scheme, the triangles of a mesh are viewed as forming one or more connected components. The connectivity information pertaining to each component is encoded using a *vertex spanning tree* in the graph of vertices and edges of that component. Since often times proximity in the vertex spanning tree implies geometric proximity, ancestors in the tree are used to predict vertex positions and only the differences between predicted and actual positions are encoded. When vertex coordinates are quantized, the corrections typically have smaller magnitude than the absolute positions and can therefore be encoded with fewer bits. The corrections are entropy encoded using Huffman or arithmetic coding.

To encode the connectivity, the mesh is first cut through a subset of its edges, including all the edges of the spanning tree (see Figure 12). When treated as a topological boundary, the cut edges define the mesh as a set of triangle runs connected by branching triangles. Compressing a simple mesh (without attributes) involves:

- constructing and encoding the vertex tree that encodes a spanning tree of the graph defined by the vertices and edges of the mesh;
- compressing the vertex position corrections representing deviations between predicted and actual positions;
- encoding the triangle tree which consists of triangle strips; and
- computing and compressing the marching pattern defining left-right movements along the triangle strips.

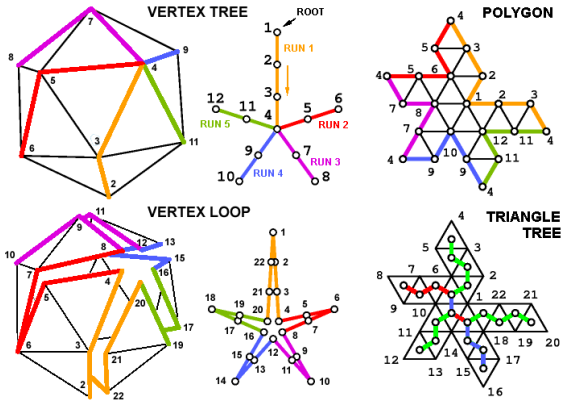


Figure 12: *Representation for compression using the topological surgery method: the vertex spanning tree is composed of vertex runs; cutting through the edges of this tree yields a simply connected polygon with the vertex loop as its boundary. Figure courtesy of G. Taubin, reproduced with permission.*

The results reported for topological surgery include compression/decompression speeds of 60,000 to 90,000 triangles per second and compression ratios ranging between 26:1 and 97:1.

The pioneering ideas formulated in [18] and [87] have been exploited by a number of other techniques. For instance, in a 1998 paper, Gumhold and Strasser [31] described an algorithm that improves on the performance of Deering's technique, both on the compression ratios and the time to compress/decompress the data. Their *cut-border* algorithm encodes an arbitrarily connected and oriented triangle mesh in one pass, by defining a set of encodable operations. These operations describe the order of traversal of the triangle mesh and uniquely encode the connectivity of the mesh, starting from a seed triangle. They are Huffman encoded into a bitstream, together with additional vertex data. For the models used as benchmarks in Deering's paper, Gumhold and Strasser report compression ratios between 7.4:1 and 12:1. The running times reported are between 300,000 and 500,000 triangles per second for compression, and approximately twice as much for the decompression. Similar techniques, that focus on connectivity compression are [89] and [69].

An alternative approach to connectivity compression is that proposed by Karni and Gotsman in [45]. By analogy with compression techniques for images, the authors exploit elements of spectral theory pertaining to 3D meshes for lossy compression purposes. Using terminology borrowed from signal processing, the main observation is that relatively smooth models can be recovered with little loss in visual quality from a relatively small number of low-frequency basis functions. The adaptation of spectral theory to 3D meshes is taken from [84] and consists of computing the mesh Laplacian and its eigenvectors and eigenvalues. These may be viewed as the natural vibration modes of the surface and the associated natural frequencies, respectively. However, the computation of the Laplacian for the entire mesh is numerically instable for large meshes. The authors, have therefore proposed a partitioning of the mesh into submeshes. The results reported (at the time of the writing of this tutorial) are compared only to the technique reported in [89] which they seem to outperform in terms of visual quality of the decoded models. No timing results or comparisons with other techniques were available.

3D Compression in MPEG-4 The Hierarchical3DMesh node of MPEG-4 supports the progressive streaming of connectivity, geometry, and properties of a 3D polygonal mesh. It uses the topological surgery compression scheme to reduce the size of the polygonal mesh and the progressive forest split mechanism (see section 5.2.2) to progressively transmit the mesh.

The mesh connectivity provides a spatially coherent ordering which permits the efficient encoding of coordinate and property data. A logic diagram for the MPEG-4 mesh encoder is shown in Figure 13 and a logic diagram for the MPEG-4 mesh decoder is shown in Figure 14. As illustrated in these figures, connectivity information of the 3D mesh is used to guide both the encoding of the vertex and property

data. Optionally, to reduce total size, properties are quantized prior to compression.

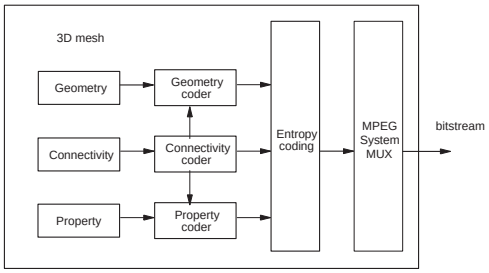


Figure 13: *MPEG-4 3D mesh encoder.*

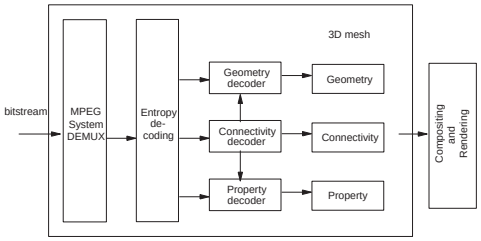
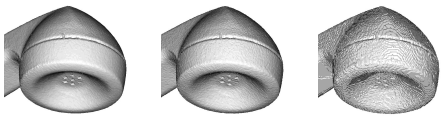


Figure 14: *MPEG-4 3D mesh decoder.*

The sequence of images in Figure 15 shows how the geometry of a model (without properties) changes as a function of the number of bits per vertex coordinate used during compression. Similar effects can be observed by varying the number of bits per normal, per color component, and per texture coordinate.

5.2.2. Streaming

Streaming is a technique for transferring data such that it can be processed as a steady and continuous stream that can be accessed before the entire data is transmitted. Streaming technologies offer two major advantages to transmission of multimedia data: shorter times between the request for the data and the receipt of an initial representation, the ability to play (and possibly, to interact with) the data at intermediate stages during transmission. For streaming to work, the client receiving the data must be able to collect it and send it as a steady stream to the application that is processing it. If the client receives the data faster than required, it has to save the excess data in a buffer. Conversely, if the data is received at a slower rate, its presentation may not appear to be smooth.



(a) (b) (c)

Figure 15: *(a) The original model. (b) The same model quantized to 11 bits per coordinate. (c) The same model quantized to 9 bits per coordinate. Reproduced from [86] with permission.*

Considerations Several issues have to be considered when streaming geometry. As previously mentioned, one of the main goals of streaming is to minimize the perceived delay in the transmission of a large model by sending an initial representation of reduced complexity which is subsequently refined. However, converting a 3D model from its original format to a representation that can be streamed is non-trivial. For instance, the resulting stream may have a total size that exceeds that of the original model or of the original model compressed using some compression technique. For large models it is usually preferable to receive something as soon as possible and to receive the entire model over a relatively longer period of time, than to receive nothing for a while, and then to obtain the entire model in one piece. Nevertheless, a stream that has not been carefully optimized may trickle down the network for a long time, making the receipt of the full model quite painful. Competitive streaming techniques use compression to reduce the size of the stream. A related question is what should be part of the initial representation. Is it better to have a (possibly very) coarse representation of the entire model, or should the initial representation consist of only the parts of the model that are “most important”? Also, where should the first refinements be made: at arbitrary locations or viewpoint dependent? How often should the model rendering be updated with data being streamed in? In the case of a 2D image being downloaded progressively, updates may be inserted at the correct pixel locations. In the case of updating a 3D model, the entire scene has to be redrawn to create an updated frame, and thus, caching of the received data becomes important.

In the case of video and audio data, a typical optimization of transmission consists of using an unreliable transport protocol such as UDP, since for this type of data there is usually a certain tolerance to loss of packets. Transmission of packets out of order is another characteristic of unreliable transmission. In the case of 3D data, depending on the application, it may be unacceptable to lose data during transmission, and the use of a reliable protocol such as TCP may be required. Also, depending on the method used to create the streamable representation, out of order refinement of the model may not be possible.

Streaming techniques In [36], Hoppe introduced the concept of a *progressive mesh* (PM). His idea was to simplify an arbitrary polygonal mesh through a sequence of *edge-collapse* operations and during this process, to record the sequence of inverse transformations (i.e., *vertex splits*) necessary to reconstruct the original mesh from the simplified one (see also section 4.2.2). For transmission over a network, one would first send the simplified mesh as an initial approximation of a model, followed by a streamed sequence of vertex splits required to fill-in the full-resolution mesh. Building such a PM representation typically involves an optimization phase to ensure that the simplified meshes generated are good approximations of the original. Such an optimization may involve the use of error metrics as well as various searching techniques to determine the order in which edge collapse operations are performed. A number of such optimizations have been developed [24], [50]. In addition to being naturally suited for network transmission, PM representations offer an alternative scheme for compression of meshes, even though not as effective as [18], [87]. The key observation is that each vertex split operation defines a local perturbation of the mesh that can be compactly encoded. Hoppe has also extended the PM approach to handle simplification/refinement of a mesh based on viewpoint position [37].

A generalization of Hoppe's PM technique is the *progressive forest split* (PFS) of Taubin et al. [85]. The PFS format shares with PM and similar methods the representation of a polygonal mesh as a low resolution model and a sequence of refinement operations and the ability to smoothly interpolate between consecutive levels of detail. However, at the expense of granularity, it renders itself better to compression, by using a more complex refinement strategy: the *forest split*. Using the terminology from the Topological Surgery compression method described in the previous section, a forest split operation is represented by a forest in the graph of vertices and edges of a mesh, a sequence of simple polygons, and a sequence of vertex displacements. Applying such an operation involves cutting the mesh through the forest edges, splitting the resulting boundaries apart, and filling each of the resulting boundary loops with a simple polygon.

In addition to geometry, it is often necessary to stream additional types of data, such as texture images. Although the techniques previously mentioned have been or can be modified to include properties such as texture coordinates, they do not address the need for streaming the actual texture images. In principle, one could view these as separate streams that may be transmitted independently either before or after the geometry is transmitted. However, the visual quality on the receiving side may be seriously impacted by using such a strategy, especially for scenes which are heavily populated with textured objects.

To address this issue, Cohen-Or et al. [15] have developed a streaming technique that exploits frame-to-frame coherence in 3D animation sequences of data sets dominated by textures. This technique is view-dependent, in that it identifies a superset of the visible parts of the model at each frame, and it streams geometric and texture information for these. Instead of using the original textures of a model, a set of view-dependent textures are used. Given a camera position, a new frame is generated on the client based on data streamed up to a certain point, which includes the visible polygons and a number of nearby views. Given a set of view-dependent textures, the highest quality one is selected for each polygon. Other streaming techniques that focus on the integration of geometric and other types of data are proprietary and are incorporated into products [58], [66], [93].

3D – an extension to streaming multimedia Over the last decade, a number of competing technologies and industry standards have emerged for streaming of multimedia data such as text, audio, and video. As the examples below illustrate it, serious efforts are now being made not only to add 3D data to this list, but also to integrate 3D content seamlessly with other media types.

As described in the previous section, the MPEG-4 multimedia standard supports 3D content via a subset of the VRML scene graph structure using the BIFS binary encoding. The recently introduced Hierarchical3DMesh node type supports the full functionality of the 3D mesh compression by topological surgery, including the streaming of polygonal data into the scene graph through a separate thread. An example of streaming 3D data and textures in MPEG-4 is shown in Figure 16. Commands to create the scene graph containing the Hierarchical3DMesh node are transmitted via the Scene Description stream. Commands to create the two object descriptors linking the scene graph leaf nodes with the appropriate elementary stream are transmitted in the Object Descriptor stream. The Progressive Forest Split Compressed Geometry elementary stream loads the Hierarchical3DMesh node and the Texture elementary stream loads the Texture node. In this example, an Intellectual Property Management and Protection (IPMP) stream is also attached to the Hierarchical3DMesh node. The content of this stream is used to protect the intellectual property contained in the compressed geometry stream [61].

HotMedia is a toolkit for enhancing e-business applications with special effects, such as interactive multi-track animations, panoramas, rotations, zooms and scrolling, and streaming audio synchronized with HTML or Javascript. 3D content is now being added to this list. HotMedia files are served from standard HTTP servers and require no plug-ins. HotMedia dynamically determines which players are needed and downloads them "just-in-time". The data (e.g., images, animations) are downloaded progressively, so that the user experience begins right away, without waiting for the entire file to be received [40].

Pulse Entertainment, a provider of interactive rich media technology for the Internet, announced earlier this year (May

2000) [64] that Pulse-powered content will stream to RealNetworks' RealPlayer. Such content includes interactive 3D animations which are typically designed in 3D Studio MAX and translated into Internet-ready content using a 3D Studio MAX translator developed by Pulse.

The MetaCreationsm Corporation, Virtue3D, Inc., RealityWave, Inc., and Vuent, Inc. are just a few additional examples of companies that focus on Internet visualization technologies and 3D rendering services for online access. They own proprietary streaming technologies which they license to their customers, and in addition, they offer tools and services for creating 3D content and integrating it into customer sites.

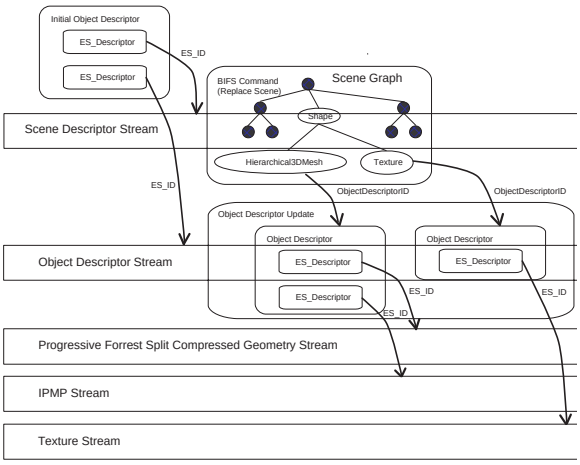


Figure 16: *Progressive streaming of 3D polygonal meshes in MPEG-4.*

SUMMARY

Efficient storage and delivery of 3D data is dependent upon the format used to represent the data. VRML is one popular 3D file format that supports descriptions of basic geometry, as well as complex attributes and behaviors of 3D models. However, it is not suitable for transmission of complex scenes over networks, as the size of the VRML files required to store such scenes is prohibitively large. Alternative formats (e.g., MPEG-4, MTS) have been designed to allow for efficient encoding and streaming of 3D data.

A number of compression and streaming techniques have been developed to optimize 3D data delivery. Compression techniques are concerned with representing data in a compact form by exploiting structures within the data. In the case of 3D models, most compression schemes focus on efficient encoding of model topology, which drives the encoding of geometry and other attributes. Streaming methods deliver data as a steady stream that can be accessed and processed before the entire contents is transmitted. The progressive mesh is a popular representation that allows streaming of 3D models. Considerations such as what data should be part of the initial representation, in what order should the refinements be made, and the use of compression to reduce the total size of the stream should be taken into account.

6. Adaptive System Design and Implementation

A significant body of work has been dedicated to the challenges of adaptive delivery of traditional multimedia content such as text, images, audio, and video [59], [21], [22]. For instance, in the TranSend project [21], [22], the adaptation is termed "distillation" and it is achieved by image compression, reduction of image size and color space, and video conversion to different frame rates and encodings. In [59], a multimodal progressive hierarchy called the InfoPyramid is used to represent multimedia Web content items such as text, images, video, and audio. This hierarchy is used to select, for a given set of client resources, the version of a content item that delivers the most value. The types of client devices considered range from workstations to cellular phones. For traditional multimedia types, *transcoding* has proven successful in serving variations of the same object at different sizes and using different modalities. For instance, in [59], a video item may be transcoded into a sequence of images for clients that are not capable of displaying video. In general, transcoding is defined as a transformation that is used to convert multimedia content from one form to another. It can be naturally extended to 3D data. By their very nature, 3D models are amenable to access through various representation modalities, that typically imply tradeoffs between complexity, interaction, and download times. As in the case of the other types of multimedia, adaptive delivery of 3D content reduces the end-to-end latency perceived by clients.

6.1. Combining Modalities for Network Rendering

In section 4 we have presented various data structures and organization schemes for 3D models. These schemes are also important for management of 3D data for transmission purposes. Unless models are downloaded en-masse, they have to be partitioned into units of content that allow a finer resolution processing for the purposes of transmission and rendering. In the remainder of this section, we regard complex models as collections of components. We define a model *component* as an atomic part of a model that corresponds to a visually meaningful entity that can be individually transmitted from a server to a client. The partitioning of the model into components may be defined at model creation time or on-the-fly. The partitioning may range from a simple decomposition into connected components to a complex scene graph describing semantic groupings, spatial groupings, or a combination of the two. Figure 17 shows an example of a scene graph in which the components have been partitioned based on semantic similarities. Semantic partitions are characteristic to CAD/CAM applications where components that

belong to the same assembly are typically grouped together. Groupings based on spatial tessellations are common in environment navigation applications and are typically used for models that have a high depth complexity for pruning occluded regions.

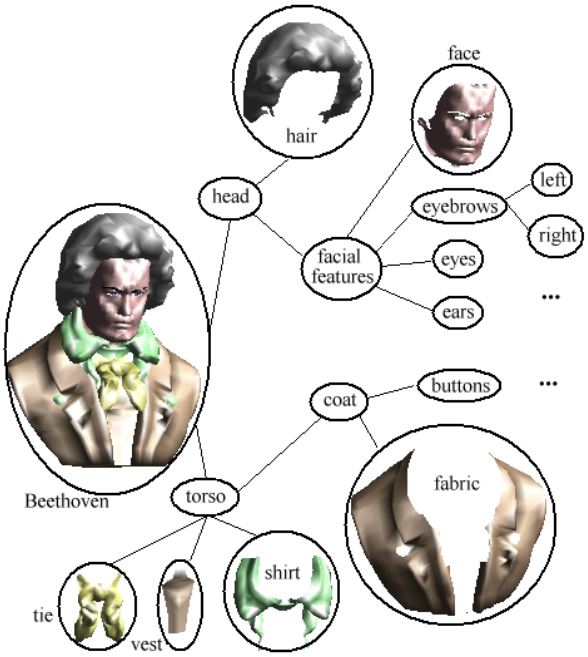


Figure 17: *Example of a scene graph in which model components have been grouped according to semantic criteria.*

The advantages of partitioning a model into components include efficient data management, support for implementing refined schemes for perceptual measurements, and most important, the ability to associate several representation modalities with each component. The use of different modalities permits an application to perform trade-offs between image quality and transmission and rendering performance. Examples of modalities include simple polygonal mesh representations, progressive meshes [36], 2D images, depth images, bounding boxes (with or without textures), and canonical shapes (e.g., sphere, cylinder, cone, box). These modalities may be computed and delivered at various levels of resolution.

Under this organizational paradigm, the selection of a representation to be delivered to a client entails prioritizing model components according to some criterion (e.g., importance to the final rendering) and determining, given a resource budget, the most appropriate modality to be used for each component. In addition, user preferences may influence the selection of specific modalities, resource budget allocation, and the resolution of various tradeoffs.

Figure 18 illustrates the adaptive client-server environment proposed in [57]. In this paradigm, 3D models are stored in a database which is connected to a server. When a client makes a request for a 3D model to the server, it first receives basic information about the model requested (i.e., meta-data). The *meta-data* allows the client to define selection criteria and, based on these, to steer the downloading of the model. In Figure 18, the selection of modalities takes place on the client. Alternatively, the selection may be performed on the server. The tradeoffs involved will be discussed in section 6.5.

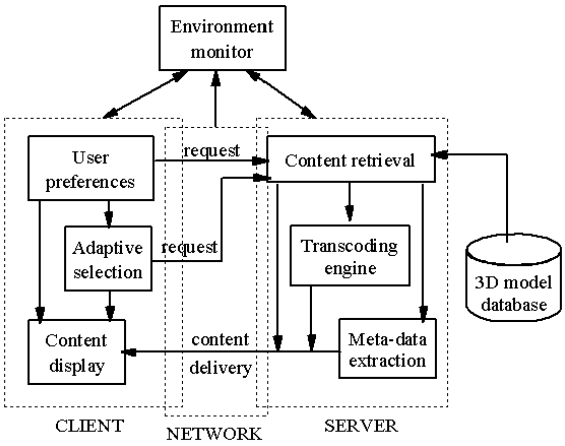


Figure 18: *Logical flow of control in an adaptive client-server setup: a monitoring tool records the characteristics of the environment, such as server load, network delay, and client and server rendering capabilities. This data is used in conjunction with information about the model to select suitable modalities for transmission and rendering of the model components. Reproduced from [57] with permission.*

6.2. Environment Monitoring

To dynamically adapt content, a tool is necessary to monitor the environment in which model transfer and rendering occurs. This tool provides quantitative information to the selection process. The state of a particular client-server setup may be characterized in terms of *state parameters*. Some of the most important parameters to be evaluated within an adaptive framework may include:

a) the rendering capability of the client,
b) the rendering capability of the server,
c) the load on the server, and
d) the performance of the communication link.

There are several ways to record measurements for such parameters. For example, one may extract values of the performance counters maintained by the operating system

(e.g., the Performance Monitor on Windows NT). Alternatively, one may record measurements that are easier to interpret from an application's perspective [57]. For instance, one could use the average frame rate to describe the rendering capabilities of a machine (client or server) for a discrete range of model sizes and several types of rendering. The server load could be measured in terms of the average time between the receipt of a request by the server and the processing of that request. The network performance may be measured in terms of latency and bandwidth. The main advantage of using application measurements is not having to interpret the values of low-level performance counters. Numbers related to CPU, memory, I/O, and network behavior may be difficult to map to values corresponding to the application-level state parameters considered. Instead, history information may be recorded dynamically for each of the state parameters. This information could be used to predict future behavior and to estimate performance. Next, we describe two schemes for measuring performance in the case of two of the parameters previously mentioned. Similar schemes may be devised for other parameters that may be of importance during transmission and rendering.

Measuring client rendering performance A benchmark test can be used to determine the frame rates achievable on a given client for a preset number of rendering types (e.g., wireframe, shaded, with or without textures). The goal is to capture initial measurements that reflect the client capabilities. Alternatively, if the benchmark is not performed, the history may be initialized with default values (e.g., measurements recorded when the client is installed). A benchmark could be designed, for instance, to render different size data sets for a preset number of frames. The average of the measurements collected for each set may be entered in the history. As models are downloaded, rendered, and manipulated by users, the frame rate history can subsequently be updated with measurements performed on these models. Various replacement strategies (e.g., least-recently used) may then be used to limit the size of the history and to ensure that only the most recent data is used for performance estimation.

Measuring network performance As described in section 3, network performance is measured in two fundamental ways: bandwidth and latency. The former is measured by the number of bits that can be transmitted over the network in a certain period of time, whereas the latter corresponds to how long it takes a message to travel from one end of the network to the other. In networked graphics, we are interested in estimating the time necessary to transfer a given modality from a server to a client. As in the case of rendering performance, benchmarks may be performed to measure transfer times for different size packets. To reduce inaccuracies due to path asymmetries and differences in the source and destination clocks, round-trip delays of messages may be more useful than one-way latency. According to [63], the relationship between transfer time, bandwidth, and data size can be expressed as:

$$TransferTime = RTT + TransferSize/Bandwidth,$$

where RTT represents the round-trip time of the network and is used to account for a request message being sent across the network and the data being sent back. Figure 19 illustrates the benchmark measurements taken for two client-server configurations over 10Mbps and 100Mbps connections, respectively. For small data sizes, the transfer time is latency bound, and the bandwidth available does not significantly influence the transfer time. In contrast, for large data, the more bandwidth there is, the faster the data is delivered. One way to use these measurements is to approximate the transfer time of a modality of a given size with the shortest time recorded for that size [81]. If the actual size does not have a corresponding entry in the history, the transfer time may be approximated by the next largest size for which history data is available.

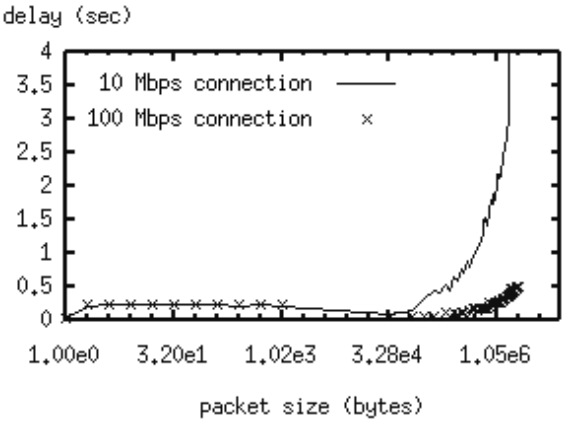


Figure 19: *Benchmark measurements for two client-server configurations with different bandwidth connections. A logarithmic scale was used to show relative performance. Reproduced from [57] with permission.*

6.3. Adaptive Selection

When several modalities are available for the transmission of a model component, a selection algorithm must be employed to decide which modality to choose. A transcoding engine is then necessary to perform the conversion of the corresponding 3D content to a format that is consistent with the modality selected. The content conversion may be done either on-line, upon selection of a desired modality, or off-line at model creation time.

6.3.1. The Performance Model

In general, modality selection is performed to determine "the best" representation of the 3D data to be sent to a requesting

computer. To compare modalities, one must define a set of performance parameters that allow comparisons in terms of the resources required by various modalities and the values they offer. For example, if m denotes a modality associated with a model component, the following performance parameters may be considered [74]:

1. $T(m)$: the total estimated time to deliver m to a client,
2. $Q(m)$: the estimated quality associated with rendering m,
3. $I(m)$: the degree of interaction supported by m.

The estimated delivery time T is defined as the sum of estimates of the time T_g it takes the transcoding engine to generate m, the time T_t to transfer it over the network, and the time T_r to render it for the first time on the client:

$$T(m) = T_g(m) + T_t(m) + T_r(m).$$

For modalities generated off-line and cached with the model, the generation time is equivalent to the time to retrieve them from the database, which we approximate with a constant. For modalities generated on-the-fly, the transcoding time is estimated based on information about the rendering capabilities of the server (if the generation involves rendering on the part of the server) and/or the worst-case complexity of the transcoding method as a function of modality size.

The estimated quality Q reflects how closely the rendering of a model component using a particular modality resembles the rendering of the full-detail data. In general, it is difficult to find a common measure of fidelity to be used for a variety of 2D and 3D modalities without actually rendering them and comparing the resulting images. One way would be to define quality as a dimensionless number between 0.0 and 1.0 that is modality specific. For instance, the modality corresponding to the full-detail representation of a model component may be assigned a quality of 1.0; the quality of a level-of-detail representation may be expressed as a percentage of the number of vertices in a level with respect to the number of vertices in the full-detail representation; the quality of a depth image could be 1.0 if the image is rendered at the same (or higher) resolution as on the client, or proportionally less, otherwise.

The degree of interaction I represents the number of degrees of freedom when interacting with a particular modality. It can assume values between 0 and 7 for the three principal axes of rotation, translation, and the field-of-view. Geometric modalities typically have all degrees of freedom, whereas image-based representations usually allow for restricted forms of interaction only (e.g., panoramas typically allow rotation about a point, but no translation).

6.3.2. The Selection Process

The meta-data information describes the basic characteristics of a model and may be customized according to the capabilities of the clients. In this section we focus on adaptive delivery of 3D models to clients with varying degrees of support for 3D rendering (either in software or in hardware) for which the main challenge is to determine an optimal mixture of modalities to represent a model. Clients that can display only text or 2D images are not discussed in this course. For 3D capable clients, the meta-data information may include the model structure (e.g., relationships between components or a model hierarchy), limited geometric information (e.g., bounding box, coarse mesh) for each of the model components, and the number, types, and characteristics of all modalities available for each component [57]. For instance, the characteristics of a modality may be the data necessary to evaluate the performance parameters T, Q, and I for that modality and include size (e.g., number of geometric primitives or pixel dimensions), and number of degrees of freedom supported.

Estimating perceptual importance Initially, the geometric information contained in the meta-data can be used to display an outline of the requested model (see Figures 20 (a)-(b)) and to allow users on the client side to define camera parameters. Next, to determine the order in which model components should be considered for modality selection and downloading, the visibility and perceptual importance of each component are estimated. The perceptual importance is a measure of the contribution of a component to the perception of a final rendering of the model. It is difficult to design good heuristics for evaluating importance in a way that closely mimics the partition made by a human eye into what is important and what is not. However, benefit heuristics such as those proposed in [56] for interactive navigation of large 3D data sets may also be used to predict importance in the context of adaptive transmission. Visibility and perceptual importance may be approximated from the coarse geometric information. A simple way of estimating visibility is to perform view-frustum culling at the bounding box level. Perceptual importance can be roughly approximated by the projected area of the bounding box. Alternatively, one could employ an image-based method to estimate visibility and perceptual importance in one pass. For a given viewing position, the collection of bounding boxes representing a 3D model could be rendered into an off-screen buffer (e.g., the back buffer). Each box would then be rendered using a color that uniquely identifies it (see Figure 20 (c)). The resulting image is processed to compute a histogram of the colors in it. Model components corresponding to the colors found, i.e., those whose bounding boxes are visible from the current viewpoint, are sorted in decreasing order of their histogram levels and are considered by the adaptive selection algorithm in this order. Components left out by this algorithm are considered to be context data and are processed last. This approach has the advantage that it is fast to compute and can be later augmented to incorporate additional parameters [23] such as distance from the center of the screen (i.e., the focus of attention) to more accurately predict perceptual importance.

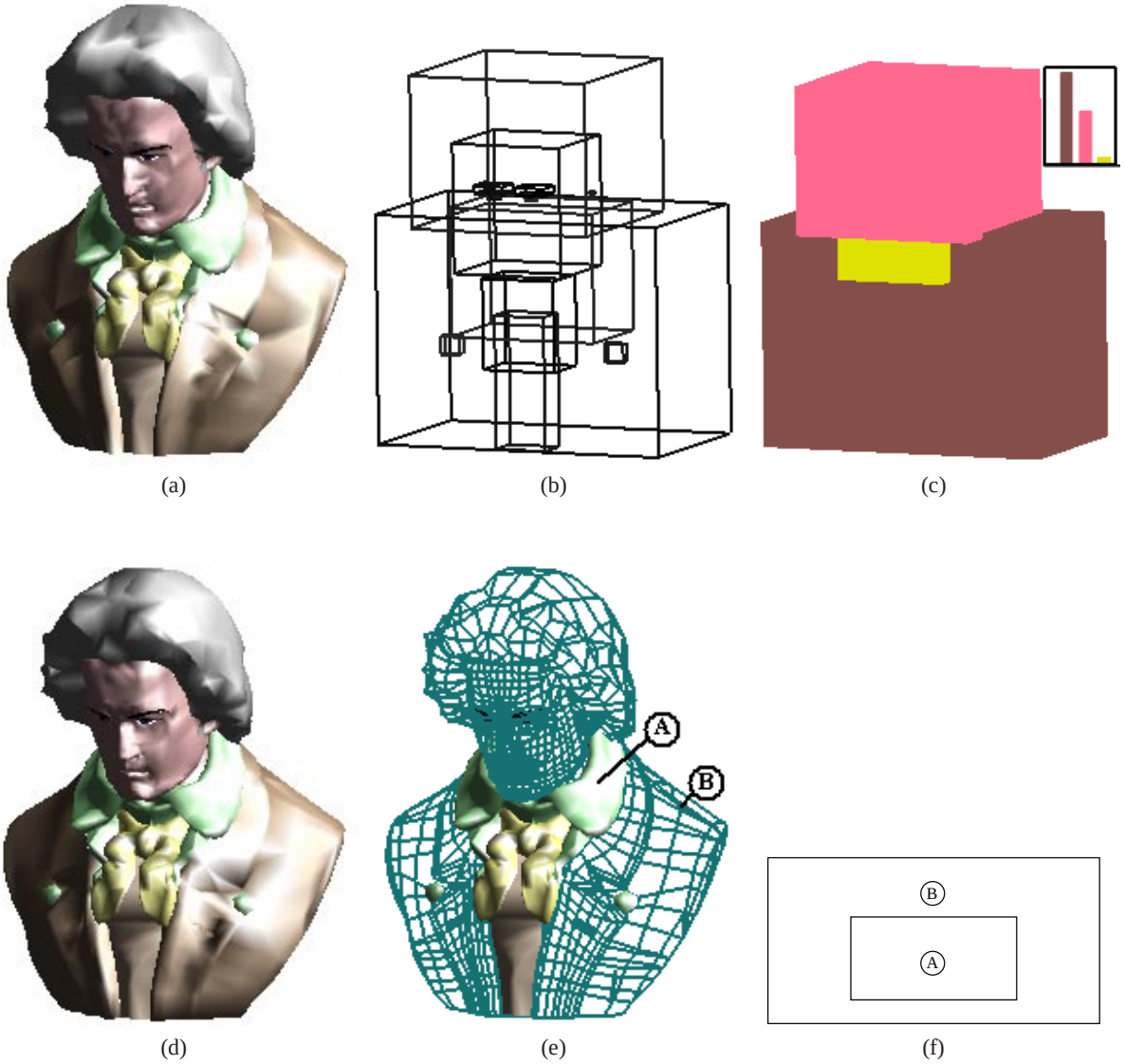


Figure 20: *Example of adaptive selection of components to be downloaded to a client based on estimated perceptual importance and visibility from the current viewing position. (a) Original model. (b) Meta-data information includes a collection of bounding boxes that can be manipulated in 3D to select a viewing position. (c) Once a view is selected, the bounding boxes of the components are rendered into an off-screen buffer. The histogram levels of the resulting image (shown in the upper right corner) determine the components that will be downloaded and the order in which they are processed. In this example, three components are downloaded. (d) Hybrid rendering on the client combining the components downloaded with a depth image generated on the server. (e) Same as (d), except the three components downloaded are shown as wireframe. (f) Top view of the bounding boxes corresponding to components A and B. Using a non-layered approach, component A is never considered by the selection algorithm due to occlusion by the larger box corresponding to component B. Reproduced from* [57] *with permission.*

Automatic selection User preferences and/or information provided by the environment monitor and meta-data may be used to determine the time budget B_T available for transmission and rendering. Starting with the component with the highest importance value, a selection algorithm could proceed to identify the most suitable modality for a component C, as follows. The performance parameters T, Q, and I are evaluated for all modalities available for C, based upon the characteristics of C defined in its meta-data. Among the modalities with $T \leq B_T$, the one with the highest quality Q is selected. If several modalities have the highest quality, the one that supports the highest degree of interaction I is chosen. This type of selection assumes that timely delivery of the model data is most important to the client, followed by the quality of the modalities received, and lastly by the degree of interaction they offer. However, depending on the application, alternative prioritization schemes may be more suitable. For instance, clients may be willing to concede on the waiting time, as long as the quality level of all components received is larger than a threshold value B_Q. In this case, modalities are first selected based on the quality level they provide, and if several modalities offer the same quality they are differentiated based on their associated delivery times, and lastly, based on their corresponding degrees of interaction. Different prioritization schemes corresponding to the permutations of T, Q, and I may be considered [57].

6.4. Support for Real-Time Interaction

When multiple representations are combined to render a model, it is likely that some of them are view independent, whereas others are generated with respect to a particular viewing position. In an interactive client session, the viewpoint may change quite often, and the challenge is to update the view-dependent representations accordingly, with as little delay as possible, to maintain consistency between all rendered parts of the model. For example, assuming that a given model component is represented by a depth image, manipulating the model in real time implies updating the image to reflect changes in its position with respect to the viewer. If a new image is generated on the server for every frame, the client will receive the data with delay due to latency in the generation process and in the transmission over the network. Moreover, rendering a large number of frames may overburden the server, it could severely impact the traffic over the network, and it would affect the interaction on the client side, as the client spends precious cycles receiving and decoding the data. In principle, there are several updating alternatives to the brute-force approach of generating view-dependent representations from scratch, each time the position of the model changes.

A simple technique is to display only view-independent representations as the object moves, and to send a request for a view-dependent representation to be generated only for the final position of the model, after the motion has stopped. For example, one could use bounding boxes to manipulate the model, and when a desired orientation is chosen, data is generated and transmitted from the server, under the assumption that the object will remain in that position for a certain period of time.

A more elaborate alternative would make use of view-dependent data previously downloaded to synthesize a new view-dependent representation locally on the client. For example, if several 2D images are available from several viewing positions, a new image corresponding to a new position could be synthesized by warping the available images (e.g., [3], [65]). A drawback is that accumulating view-dependent representations on the client may require a considerable amount of storage. Additionally, accurate warping may require complex processing that may not be feasible in real time.

A different class of techniques are so-called *dead reckoning* techniques (or protocols) that attempt to predict the position of the model at future moments in time [80]. Based on this prediction, a view-dependent representation is generated consistent with the state of the model at the estimated time of receipt by the client. In between updates the state is still inconsistent, but some overhead is eliminated by determining in advance what will be needed for display by the time data arrives at the client. In the remainder of this section we briefly describe the underlying philosophy of such protocols.

Historically, a sailing ship's speed over a nautical mile was measured by means of a rope with knots tied to a log. A sand filled timing glass was used to measure the time from leaving the log dead (much as a dead man might appear) in the water (dead reckoning) and the number of evenly spaced knots passed along the rope. Some argue that the term dead-reckoning is derived from the navigational practice of starting from a point that was dead in the water. From this point the direction and time would be used to deduce location along the route as it crossed longitudinal lines. Others contend that dead reckoning comes from "deduced reckoning", also from sailing, which is a simple mathematical procedure for determining the present location of a ship by advancing some previous position through known course and velocity information over a given length of time.

A dead reckoning protocol involves two aspects: prediction and convergence. *Prediction* is the method by which the position of the object is inferred based on previous history. However, the prediction is just an estimate of the true position, which may have to be corrected to match the true position. *Convergence* defines how the position is to be corrected. Such corrections are usually done gradually, over several frames, to avoid popping artifacts.

6.5. Implementation Issues

Implementing an adaptive system for delivering 3D models over networks is a non-trivial task. The complexity arises

primarily from the necessity to combine techniques from different domains into a common framework. These domains include:

1. *Networking*: the adaptive system must take into account the heterogeneity of the environment and it must contend with the various issues that networked applications face: network resource management, error resilience, and fault tolerance.
2. *Rendering*: since the end goal is to display and visualize the 3D models on the computers which have requested them, techniques developed in the graphics field must be exploited to create visual representations that meet the exigencies of the application for which they are used.
3. *Virtual manipulation/navigation*: the main reason behind using 3D models and not 2D images is the need for interacting with the model in real-time. Therefore, algorithms that have been developed for rendering acceleration to support interactivity must be folded into the system as well.
4. *Database management*: while simple models may be stored in single files, others may consist of collections of files corresponding to various types of information associated with the models. Alternatively, for processing and transmission purposes, caching of different representations associated with a model may result in multi-element collections. In such cases, the system has to deal with issues such as search, retrieval, and access to these files.
5. *Security*: is paramount to the success of such systems, as it becomes a major concern for almost all applications of networked graphics. Technologies for authentication, encryption, and watermarking are just a few examples of what may be required as part of the system.

While we emphasize the importance of all of the above topics when building an adaptive networked graphics framework, we restrict our discussion here to implementation concerns related to network transmission and rendering.

The action that initiates model transfer between a client computer and a remote repository is usually the request of a model of interest from the repository by a user or an application. Typically, the request goes to either a server in a client-server type of architecture or to a proxy server that identifies the requested data on nodes in a distributed environment. Unless the original representation of a model is used or unless the representation(s) to be delivered has been cached, the data has to be loaded for processing at the source nodes. In the case of complex models, loading a model into memory may take a very long time. Hence, if the resources are available, it may be more efficient to dedicate certain nodes (or processes) to servicing complex models that are most frequently requested. Thus, a model is loaded once and various requests for that particular model are routed and handled by the process dedicated to that model, as opposed to loading the model upon each request.

Server scalability is another important issue. 3D models are becoming commodity media items, which implies that millions of requests have to be handled simultaneously. A distributed server may be one solution for such situations.

A related concern is the type of connection and the amount of bookkeeping that has to be maintained on the server for each client. For example, if a point-to-point communication using a TCP socket is established between a server and each of its clients, then the alternatives are: (a) to keep an open connection for each client for the duration of a client session at the price of allowing only a limited number of clients to have access to this server, or (b) to open a connection for each request and to close it after the request is serviced, thus introducing an overhead for setting up connections. Bookkeeping may also impact scalability. If only the server has access to the models and their structure, then any adaptive processing that may be required will have to be done on the server. Choosing to transmit some information about the model to clients (e.g., meta-data) may enable them to make decisions locally and, on that basis, to request data from the servers.

The implementation of an adequate *transmission protocol* for the geometry and related data (textures, materials, annotations, etc.) is important for dealing with some of the issues mentioned in section 5.2.2, such as reliable vs. lossy transmission and buffer caching.

On the client side, there are a number of design and implementation issues to be considered, as well. For example, it may be desirable to separate communication with the server, the decoding of the data received, and the rendering of the model. A common solution is to dedicate a separate thread to each of these activities, with the caveats of possibly having to synchronize them and to manage the communication between these threads. Rendering the data as soon as it becomes available may not be efficient, as updating a frame may require redrawing an entire scene. Accumulating data and not displaying it, however, may impact the interactivity and the ability of the user to steer the downloading process. Adequate buffering strategies that compromise between these alternatives must therefore be implemented.

Finally, an important issue is where the bulk of the selection work is performed: on the server or on the client. If the selection is performed on the server, the advantage is that model information necessary to compute estimates of the performance parameters does not have to be transmitted to the client. However, doing all the selection work on the server considerably increases the server load, which has an impact on scalability. If the selection is performed on the client, additional data about the model has to be downloaded to help make an informed decision. The latter approach may be preferable if good performance estimates can be derived without excessively increasing the size of the meta-data.

SUMMARY

In multimedia jargon, transcoding defines the process that is used to convert multimedia content from one form to another. By extension, transcoding of 3D content implies the use of different representation modalities to deliver models adaptively to various clients.

We regard models as collections of components that can be individually transmitted and rendered. This allows for efficient data management and fine-grained schemes for measuring perceptual importance. Combining different modalities for representing each component typically leads to better performance than the use of a "one-size-fits-all" strategy.

The selection of the most appropriate representation to be sent to requesting clients should account for the resources available, the importance of various components to the final rendering, as well as for user preferences.

A monitoring tool is a necessary part of an adaptive environment that provides quantitative information about the state of the environment. Preferably, such a tool captures information about the environment dynamically, so that the application framework can adapt to changes that may have an impact on the overall performance.

7. Conclusions

In this course, we have described the main issues related to the delivery of 3D models over networks. We started by reviewing basic notions of networking and we presented a survey of some of the most important technologies developed for optimizing rendering and transmission of 3D models. By formulating the problems of universal access to non-trivial 3D models and by contrasting different methods, we illustrated the importance of adaptive approaches. We conveyed the fact that graphics and networking are both mature fields, which have been traditionally considered separately, but are now beginning to converge. Our main goal was to emphasize the various possibilities for leveraging existing technologies from both areas, to create powerful networked graphics environments.

Acknowledgements

The authors would like to thank all of those who contributed to the preparation of the material for this tutorial. In particular, we would like to thank Dirk Bartz, Fausto Bernardini, Paul Borrel, Josh Mittleman, Bengt-Olaf Schneider, Claudio Silva, Frank Suits, and Gabriel Taubin.

References

1. V. Abadjev, M. del Rosario, A. Lebedev, A. Migdal, and V. Paskhaver. Metastream. In *VRML 99*, pages 53–62, February 1999. 20
2. D. Aliaga and A. Lastra. Architectural walkthroughs using portal textures. In *IEEE Visualization 97*, pages 355–362, October 1997. 12
3. S. Avidan and A. Shashua. Novel view synthesis by cascading trilinear tensors. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):293–306, 1998. 30
4. D. Bartz, M. Meißner, and T. Hüttner. OpenGL-assisted occlusion culling for large polygonal models. *Computers & Graphics*, 23(5):667–679, October 1999. 13
5. R. Carey and G. Bell. *The Annotated VRML 2.0 Reference Manual*. Addison-Wesley Developers Press, Reading, MA, 1997. 9
6. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974. 10
7. P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A general method for preserving attribute values on simplified meshes. In *IEEE Visualization 98*, pages 59–66, October 1998. 14
8. P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22:37–54, 1998. 14
9. J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976. 11
10. J. Cohen. Model simplification. In *Interactive WalkThrough of Large Geometric Datasets (SIGGRAPH 99 Course Notes #20)*, August 1999. 13, 14
11. J. Cohen, D. Manocha, and M. Olano. Simplifying polygonal models using successive mappings. In *IEEE Visualization 97*, pages 395–402, 1997. 14
12. J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *SIGGRAPH 98 Conference Proceedings*, pages 115–122. ACM SIGGRAPH, July 1998. 14
13. D. Cohen-Or, Y. Chrysanthou, and C. Silva. Visibility problems for walkthrough applications. In *Visibility: Problems, Techniques, and Applications (SIGGRAPH 00 Course Notes #4)*, July 2000. 10, 11
14. D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–253, 1998. 12

15. D. Cohen-Or, Y. Mann, and S. Fleishman. Deep compression for streaming texture intensive animations. In *SIGGRAPH 99 Conference Proceedings*, pages 261–268. ACM SIGGRAPH, 1999. 24

16. S. Coorg and S. Teller. Temporally coherent conservative visibility. In *12th Annual ACM Symposium on Computational Geometry*, pages 78–87, 1996. 12

17. S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, 1997. 12

18. M. Deering. Geometry compression. In *SIGGRAPH 95 Conference Proceedings*, pages 13–20. ACM SIGGRAPH, 1995. 21, 22, 24

19. F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization 96*. IEEE, October 1996. 10

20. J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990. Overview of research to date. 8, 11, 16

21. A. Fox and E. A. Brewer. Reducing www latency and bandwidth requirements by real-time distillation. In *Proc. 5th Intl. WWW Conference*, Paris, France, 1996. 25

22. A. Fox, S. Gribble, E. Brewer, and E. Amir. Adapting to network and client variation using infrastructural proxies: Lessons and perspectives. *IEEE Personal Communications*, 40:10–19, 1998. 25

23. T. Funkhouser and C. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH 93 Conference Proceedings*, pages 247–254. ACM SIGGRAPH, August 1993. 15, 16, 28

24. M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, pages 209–216. ACM SIGGRAPH, August 1997. 13, 14, 24

25. M. Garland and P. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization 98*, pages 263–269, October 1998. 13, 14

26. T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, and I. J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, April 1998. 13

27. Silicon Graphics. *Iris Inventor Programming Guide*, 1992. 9

28. N. Greene. Hierarchical polygon tiling with coverage masks. In *SIGGRAPH 96 Conference Proceedings*, pages 65–74. ACM SIGGRAPH, August 1996. 13

29. N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993. 12

30. A. Guéziec. Surface simplification with variable tolerance. In *International Symposium on Medical Robotics and Computer Assisted Surgery*, pages 132–139, November 1995. 13

31. S. Gumhold and W. Strasser. Real-time compression of triangle mesh connectivity. In *SIGGRAPH 98 Conference Proceedings*, pages 133–140. ACM SIGGRAPH, 1998. 22

32. B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11:197–214, 1994. 13

33. C. Healey, V. Interrante, and P. Rheingans. Fundamental issues of visual perception for effective image generation. In *SIGGRAPH 99 Course Notes #6*, August 1999. 15

34. P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *Multiresolution Surface Modeling (SIGGRAPH 97 Course Notes #25)*, August 1997. 14

35. K. Hoff III. Backface cluster culling using normal-space partitioning. Technical report, Computer Science Dept., Univ. of North Carolina at Chapel Hill, 1996. 11

36. H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108. ACM SIGGRAPH, August 1996. 15, 24, 26

37. H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, pages 189–198. ACM SIGGRAPH, 1997. 14, 24

38. H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *IEEE Visualization 99*, pages 59–66, October 1999. 13, 14

39. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH 93 Conference Proceedings*, pages 19–26. ACM SIGGRAPH, August 1993. 13

40. HotMedia. www.ibm.com/hotmedia. 24

41. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. In *13th Annual ACM Symposium on Computational Geometry*, pages 1–10, 1997. 12

42. D. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40:1098–1101, 1951. 21

43. V. Interrante, P. Rheingans, J. Ferwerda, R. Gossweiler, and T. Filsinger. Principles of visual perception and their applications to computer graphics. In *SIGGRAPH 97 Course Notes #33*, August 1997. 15

44. V. Interrante, P. Rheingans, J. Ferwerda, R. Gossweiler, and C. Healey. Applications of visual perception in computer graphics. In *SIGGRAPH 98 Course Notes #32*, July 1998. 15, 16

45. Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH 00 Conference Proceedings*. ACM SIGGRAPH, 2000. To appear. 22

46. J. Klosowski and C. Silva. Rendering on a budget: A framework for time-critical rendering. In *IEEE Visualization 99*, pages 115–122, October 1999. 12

47. J. Klosowski and C. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2), June 2000. 12

48. V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. In *Eurographics Workshop on Rendering 00*, June 2000. 12

49. S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical back-face computation. In *Eurographics Workshop on Rendering 96*, pages 235–244, New York City, NY, June 1996. Eurographics, Springer Wein. 11

50. P. Lindstrom. Out-of-core simplification of large polygonal models. In *SIGGRAPH 00 Conference Proceedings*. ACM SIGGRAPH, 2000. To appear. 24

51. P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization 98*, pages 279–286, October 1998. 13

52. K. Low and T. Tan. Model simplification using vertex-clustering. In *Symposium on Interactive 3D Graphics*, pages 75–82. ACM SIGGRAPH, April 1997. 13

53. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings*, pages 199–208. ACM SIGGRAPH, August 1997. 13

54. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings*, pages 199–208. ACM SIGGRAPH, August 1997. 14

55. D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, April 1995. 12

56. P. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics*, pages 95–102. ACM SIGGRAPH, April 1995. 15, 17, 28

57. I. Martin. Adaptive rendering of 3d models over networks using multiple modalities. Technical Report RC 21722(97821), IBM Research, April 2000. 26, 27, 28, 29, 30

58. MetaCreations Corporation. www.metacreations.com. 24

59. R. Mohan, J. Smith, and C.-S. Li. Adapting multimedia internet content for universal access. *IEEE Transactions on Multimedia*, 1(1):10–19, 1999. 25

60. T. Möller and E. Haines. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, 1999. 11, 15

61. MPEG-4 information technology - coding of audio-visual objects - Part 1: Systems, 1999. ISO/IEC JTC 1/SC 29/WG 11 N 2501. 24

62. Next Generation Internet Initiative. www.ngi.gov. 3

63. L. Peterson and B. Davie. *Computer Networks*. Morgan Kaufmann, 2000. 27

64. Pulse Entertainment. www.pulse3d.com. 25

65. M. Rafferty, D. Aliaga, and A. Lastra. 3d image warping in architectural walkthroughs. In *Proceedings of VRAIS 98*, pages 228–233, 1998. 30

66. RealityWave Inc. www.realitywave.com. 24

67. J. Rohlf and J. Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In *SIGGRAPH 94 Conference Proceedings*, pages 381–395. ACM SIGGRAPH, July 1994. 9

68. R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):67–76, August 1996. 13

69. J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999. 22

70. J. Rossignac and P. Borrel. Multi-resolution 3D approximation for rendering complex scenes. In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993. Genova, Italy. 13

71. K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, San Francisco, CA, 1996. 20, 21

72. G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In *Eurographics Workshop on Rendering 97*, pages 151–162, June 1997. 15

73. G. Schaufler and W. Stürzlinger. A three dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):227–236, August 1996. 15

74. B.-O. Schneider and I. Martin. An adaptive framework for 3d graphics over networks. *Computers and Graphics*, 23:867–874, 1999. 28

75. W. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. In *SIGGRAPH 92 Conference Proceedings*, pages 65–70. ACM SIGGRAPH, July 1992. 14

76. N. Scott, D. Olsen, and E. Gannett. An overview of the VISUALIZE fx graphics accelerator hardware. *The Hewlett-Packard Journal*, pages 28–34, May 1998. 13

77. J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH 98 Conference Proceedings*, pages 231–242. ACM SIGGRAPH, July 1998. 15

78. J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82. ACM SIGGRAPH, August 1996. 15

79. C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948. 20

80. S. Singhal and M. Zyda. *Networked Virtual Environments*. Addison-Wesley, 2000. 30

81. Q. Snell, A. Mikler, and J. Gustafson. Netpipe: A network protocol independent performance evaluator. In *Proceedings of IASTED ISSM Intl. Conference*, pages 129–134, 1996. 27

82. H. Sowizral, K. Rushforth, and M. Deering. *The Java 3D API Specification*. Addison-Wesley, Reading, MA, 1998. 9

83. I. Sutherland. Sketchpad: a man-machine graphical communication system. *SJCC*, 1963. 35

84. G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH 95 Conference Proceedings*, pages 351–358. ACM SIGGRAPH, 1995. 22

85. G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH 98 Conference Proceedings*, pages 123–132. ACM SIGGRAPH, 1998. 24

86. G. Taubin, W. Horn, J. Rossignac, and F. Lazarus. Geometry coding and VRML. *Proceedings of the IEEE, Special issue on Multimedia Signal Processing*, 86(6):1228–1243, June 1998. 23

87. G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998. 21, 22, 24

88. S. Teller and C. Séquin. Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH 91 Conference Proceedings*, pages 61–69. ACM SIGGRAPH, July 1991. 12

89. C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34, June 1998. 22

90. G. Turk. Re-tiling polygonal surfaces. In *SIGGRAPH 92 Conference Proceedings*, pages 55–64. ACM SIGGRAPH, July 1992. 14

91. A. Varshney. *Hierarchical Geometric Approximations*. PhD thesis, Computer Science Dept., Univ. of North Carolina at Chapel Hill, 1994. 13

92. Web 3D consortium. www.vrml.org. 19

93. Vuent Inc. www.vuent.com. 24

94. J. Wernecke. *The Inventor Mentor*. Addison-Wesley, Reading, MA, 1993. 35

95. P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, 18(3):51–60, September 1999. 12

96. P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. Technical Report TR-186-2-00-06, Institute of Computer Graphics, Vienna University of Technology, 2000. 12

97. Xgl file format specification. www.xglspec.org. 19

98. J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), April–June 1997. 14

99. X. Xiang, M. Held, and J. Mitchell. Fast and effective stripification of polygonal surface models. In *Symposium on Interactive 3D Graphics*, pages 71–78. ACM SIGGRAPH, April 1999. 10

100. H. Zhang, D. Manocha, T. Hudson, and K. Hoff III. Visibility culling using hierarchical occlusion maps. In *SIGGRAPH 97 Conference Proceedings*, pages 77–88. ACM SIGGRAPH, August 1997. 13

Appendix A: A Brief History of VRML

The Virtual Reality Modeling Language (VRML) was the first significant effort to focus on the delivery of 3D graphics across the World Wide Web. While the arrangement of graphical structures into directed acyclic graphs (scene graphs) can be traced back to Sutherland's work on Sketchpad [83], many of the other features and conventions in VRML can be traced back to SGI's Inventor product. The IRIS Inventor 3D toolkit [94] was introduced by SGI in 1992. Inventor is a C++ 3D toolkit with an object-oriented design. It is based on a hierarchical scene graph and it provides many features including a full set of tools to support: creation, modification, and interaction with scene graphs, geometry, events, and manipulators (e.g., trackball). In addition to providing these tools, Inventor also introduced a file format to store its scene graph.

The initiative behind VRML can be traced back to a spring of 1994 Birds of a Feather session at the first annual World Wide Web Conference in Geneva, Switzerland. As a result of this meeting, an ad-hoc group formed and the VRML mailing list was created to discuss the issues. One of the early goals was to create a specification that was analogous to HTML by exploiting WWW hyperlinks. Indeed, VRML originally stood for "Virtual Reality Markup Language". The analogy between VRML and HTML was not that strong and word "Modeling" was eventually replaced "Markup". Eventually, three requirements were chosen for VRML 1.0: platform independence, extensibility, and the ability to work well over low-bandwidth connections.

A consensus was reached and version 1.0 specification for a file format was released in Spring of 1995. The format was essentially a stripped-down version of Inventor's ASCII format in which only fundamental nodes were retained. The only features in VRML that were not supported by inventor were the WWWInline (read from URL) and WWWAnchor (WWW hyperlink) nodes. Indeed, if these two nodes are not present, a VRML file can be changed to an inventor file by changing "#VRML V1.0 ascii" to "#Inventor V1.0 ascii".

VRML 1.0 was the first successful attempt to provide a standard file format for the interchange of 3D models on the Internet. VRML 1.0 enjoyed considerable success. Several freely available browsers were released and a considerable number of VRML worlds were available for download. Perhaps the biggest deficiency of VRML 1.0 is its inability to satisfy the third design requirement. Bandwidth and throughput were major limitations in the late 1990s (as they still are today) and accessing large VRML models meant very long delays.

In the fall of 1995 several long-term contributors to the VRML community formed the VRML Architecture Group (VAG) to guide an effort to create a VRML 2.0 specification. VRML 1.0 had inherited some deficiencies from Inventor. A major problem with the basic scene graph strategy of Inventor is the excessive overloading of the parent-child relationship. In addition to providing a state mechanism to support the model instancing matrix, inheritance was also used to provide basic properties such as normals, coordinates, and materials. The excessive overloading of state produced scene graphs that were difficult to optimize for display.

In addition to improvements in the design of the scene graph, the VAG believed that the consensus of VRML community was that VRML 1.0 lacked key features of animation, interaction, and behavior. The perception was that these features were necessary to create the "moving worlds" required for a more impressive virtual reality

experience. Desires to move toward capabilities such as multi-user worlds where avatars could conduct “virtual sword fights and collaborative data mining could take place” dominated the discussion on the requirements for VRML 2.0.

VRML 2.0 (also known as VRML 97) was completed in August 1996. The changes in the treatment of state were well thought out and led to an improved representation for the scene graph. However, in hindsight, the complexity of the new features (animation, interaction, and behavior) were disastrous. Resources diverted to the exploitation and implementation of these features were squandered and, as a result, many of the key commercial ventures based on VRML 2.0 did not succeed.

VRML 2.0 had the same network throughput limitations of VRML 1.0. The throughput limitations were exacerbated by three major deficiencies in the format. First, because there was no support for a binary format or geometric compression, large models were up to two orders of magnitude larger than they had to be. Second, no attempt was made to efficiently store levels of detail. Finally, no provision was made for progressive transmission. Proposals were made to address these issues, but, for a variety of reasons, none of them made their way into the VRML specification. Despite its deficiencies, VRML 2.0 remains a popular file format for representing 3D virtual worlds (see Figure 21).

```
#VRML V2.0 utf8
Shape
 appearance Appearance
  material Material
   ambientIntensity  0.5
   diffuseColor      0.1837 0.1837 0.1837


 geometry IndexedFaceSet
  coord Coordinate
   point [
      0.94  0.00 -0.33 ,
     -0.47  0.81 -0.33 ,
     -0.47 -0.81 -0.33 ,
      0.00  0.00  1.00 ,
    ]

  coordIndex [
     2, 1, 0, -1,
     3, 2, 0, -1,
     1, 3, 0, -1,
     2, 3, 1, -1,
    ]
  color Color
   color [
    1.00 0.62 0.00,
    1.00 0.00 0.00,
    0.87 0.00 0.87,
    0.37 0.37 1.00,
   ]

  colorPerVertex FALSE
  colorIndex [ 0 1 2 3 ]
```

Figure 21: *A simple VRML file.*

Appendix B: The XGL File Format

```
<WORLD>
   <LIGHTING>
      <DIRECTIONALLIGHT>
         <DIFFUSE>1.000,1.000,1.000</DIFFUSE>
         <SPECULAR>0.100,0.100,0.100</SPECULAR>
         <DIRECTION>-0.302,0.302,0.905</DIRECTION>
      </DIRECTIONALLIGHT>
      <AMBIENT>0.000,0.000,0.000</AMBIENT>
   </LIGHTING>
   <MESH ID="0">
      <MAT ID="0">
         <AMB>0.000,1.000,0.000</AMB>
         <DIFF>0.000,1.000,0.000</DIFF>
         <SPEC>1.000,1.000,1.000</SPEC>
         <EMISS>0.000,0.000,0.000</EMISS>
         <SHINE>51.200</SHINE>
         <ALPHA>1.000</ALPHA>
      </MAT>
      <P ID="0">-1.0,-0.5,0.0</P>
      <P ID="1">-1.0,-0.5,1.0</P>
...
      <N ID="5">0.0,0.0,-1.0</N>
      <F>
         <MATREF>0</MATREF>
         <FV1>
            <PREF>0</PREF>
            <NREF>0</NREF>
         </FV1>
         <FV2>
            <PREF>1</PREF>
            <NREF>0</NREF>
         </FV2>
         <FV3>
            <PREF>2</PREF>
            <NREF>0</NREF>
         </FV3>
      </F>
...
   </MESH>
   <OBJECT>
      <TRANSFORM>
         <FORWARD>0.000000,0.000000,1.000000</FORWARD>
         <UP>0.000000,1.000000,0.000000</UP>
         <POSITION>0.000000,0.900000,-0.000000</POSITION>
         <SCALE>1.000000</SCALE>
      </TRANSFORM>
      <MESHREF>0</MESHREF>
   </OBJECT>
   <OBJECT>
...
      <TRANSFORM>
         <FORWARD>0.000000,0.000000,1.000000</FORWARD>
         <UP>0.000000,1.000000,0.000000</UP>
         <POSITION>-0.900000,-0.000000,0.000000</POSITION>
         <SCALE>1.000000</SCALE>
      </TRANSFORM>
      <MESHREF>0</MESHREF>
   </OBJECT>
   <BACKGROUND>
      <BACKCOLOR>1.0,1.0,1.0</BACKCOLOR>
   </BACKGROUND>
</WORLD>
```

Figure 22: *A simple XGL file.*

Appendix C: The MPEG-4 Receiver Architecture

The decoder model enables a transmitter to predict how a receiving terminal will behave in terms of buffer management and synchronization when reconstructing the transmitted scene. The decoder model includes a timing model and buffer model.

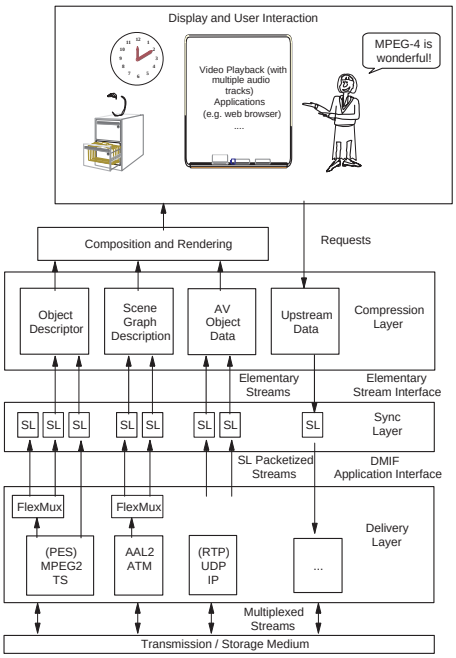


Figure 23: *System view of an MPEG-4 receiving terminal.*

The timing model defines the temporal behavior of the scene. Two sets of timing information are defined MPEG-4: clock references and time stamps. The clock reference conveys the time base to the receiving terminal. Time stamps permit the scheduling of decoding or composition for individual media objects.

The buffer model enables the transmitter to monitor and control the buffer resources that are needed to decode the elementary streams. The buffer model permits the scheduling of data transmission in a way that insures that buffers at the receiving end to not underflow or overflow.

As shown in Figure 23, instead of using its own transport mechanisms, MPEG-4 is designed to exploit existing transport protocols to transmit and receive content via streams. For example, MPEG-4 content can be delivered using: the Asynchronous Transfer Mode (ATM) Adaptation Layer2 (AAL2), an MPEG-2 stream, or transport control / Internet protocol (TCP/IP). MPEG-4 specifies a multiplexing tool called FlexMux to recover elementary streams from multiplexed streams. The MPEG-4 interface to the Delivery layer is the DMIF Application Interface(DAI). The Delivery Multimedia Integration Framework (DMIF) provides an abstraction of network, broadcast and file access in order to allow content providers to develop content once for different transport technology instances.

MPEG-4 specifies mechanisms to describe, group, and to synchronize elementary streams. Each elementary stream contains only one type of data. Individual streams or groups of streams are identified and characterized by object descriptors. Object descriptors are themselves delivered by elementary streams. The metadata contained in object descriptors indicates the format of a stream and parameters required for decoding. Object descriptors may be used to select a subset of its referenced streams for a particular representation or encoding of a media object.

The description of the scene graph and the description of the elementary streams are separated in MPEG-4. Media objects in the scene graph are not concerned with the details of the the streams used to deliver their content. Likewise, streams know nothing of the scene graph structure. Object descriptors are used to connect streams with media objects. The separation of media objects and streams facilitates the authoring of content by separating the details of a particular media stream from the construction (authoring) of the scene graph.

The synch layer (SL) extracts timing information (clock references) and synchronization data (time stamps) from SL-packetized streams and reconstructs elementary streams from individual access units (e.g. a frame of video). The operations performed by the SL are not specific to a particular media type, for example, the mechanisms used to create SL-packetized streams for video are the same as those used for audio.

Elementary streams are decoded using stream-specific decoders in the compression layer. The receiving terminal then composes and renders the decoded streams. Finally, the system architecture permits the receiving terminal to communicate with the transmitter through a backup channel.

MPEG-4 contains a mechanism for intellectual property management and protection (IPMP). Although beyond the scope of this course, IPMP provides important functionality for content providers to restrict the use of their material. For example, this mechanism could be used to restrict the capability of a receiver to locally store a 3D mesh.