A General Framework for Multidimensional Adaptation

David Gotz
gotz@cs.unc.edu

Ketan Mayer-Patel
kmp@cs.unc.edu

University of North Carolina at Chapel Hill
CB #3175, Sitterson Hall
Chapel Hill, NC 27599 USA

ABSTRACT

Data adaptation is an essential system component in a wide variety of application areas. To date, most applications use ad hoc methods to manage data in response to limited resources and changing system conditions. We present a generic adaptation framework that distills the common elements essential to a broad class of adaptive applications. Our framework provides an abstract data representation and defines a generic set of adaptation operations that directly support multidimensional and multimedia adaptive behavior. We present several application case studies and demonstrate the performance of our framework on an experimental prototype.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.1.1 [**Information Storage and Retrieval**]: Systems and Information Theory—*Value of information*

General Terms

Algorithms, Performance, Measurement

Keywords

Adaptation, multimedia

1. INTRODUCTION

Advances in storage and processing technologies now allow scientists to capture, simulate, or create immense collections of data. Unfortunately, advances in networking and display technologies, while impressive, have not kept pace. Although we can build multi-gigabit networks, achieving those line speeds end-to-end remains almost impossible. Similarly, high-resolution display technologies have improved but essentially remain within a factor of ten of the megapixel displays we have enjoyed for nearly twenty years. As the gap between the amount of data we can capture, store, and process and the resources we have to transmit and view that data increases, the problem of adaptation becomes increasingly central to the performance of high-bandwidth and high-volume applications such as data visualization, teleimmersion, and media streaming.

Adaptation techniques, algorithms, and frameworks have largely been developed within the context of a specific application or data type. Geographic Information Systems (GIS), for example, use multiresolutional data for visualization and representation of terrain information [7]. In graphics, multiresolutional geometry information is used to create hierarchical levels of detail (HLODs) [13]. These are used to dynamically adjust model complexity in order to achieve a target rendering rate. In multimedia, layered media encodings are used to dynamically scale media bitrates to match current network conditions [17].

The challenge we face when tackling the adaptation problem in whatever context it appears is twofold. First, how can we compactly and intuitively specify adaptation policy to support specific user-level goals? Second, given a particular adaptation policy and set of user-level goals, how can we efficiently evaluate that policy relative to available resources and the way the data are represented and organized?

While ad hoc application-specific methods may be effective when the number of dimensions for adaptation is limited, the problem can become overwhelming as the number of dimensions increases. The same is true when an adaptation decision must negotiate between data sources of fundamentally divergent natures. The complexity of these mechanisms may become very difficult to correctly manage. Expressing adaptation policy in a rule-based manner, for example, becomes painful as the number of possible tradeoffs grows.

In this paper, we present a general framework for multidimensional adaptation that makes the expression and evaluation of adaptation policies more intuitive and straightforward. We address the need to model relationships among data, compactly express adaptation policies, and dynamically evaluate utility with respect to system conditions.

We believe that providing a general framework for adaptation will help manage complexity, reduce development time, and enable the reuse of sophisticated adaptation structures in future systems.

1.1 Main Results

We present a general framework for multidimensional adaptation that is both intuitive and widely applicable. First, we distill the common elements essential to a broad class of adaptive systems. We then present our framework for general adaptation. This framework defines a graph-based representation abstraction embedded within a multidimensional utility space. Dimensional tradeoffs, cost metrics, and utility metrics are defined as spatial operations performed upon the representation graph. Adaptation is guided by iterative evaluation of the cost and utility metrics.

To illustrate the broad applicability of our framework, we present three case studies. We discuss the use of our framework in the context of a layered video streaming system, a remote 3D geometry rendering system, and a remote image-based rendering (IBR) system. We include results from an experimental evaluation of our IBR prototype.

1.2 Organization

The remainder of this paper is organized as follows. We review related work in Section 2 and discuss the general adaptation problem space in Section 3. We describe our framework in detail in Section 4. We present several application case studies in Section 5, followed by an experimental evaluation in Section 6. We conclude with a discussion of future work in Section 7.

2. BACKGROUND AND RELATED WORK

In this section, we discuss a selection of other work most related to our research. Previous work has largely addressed the problem of adaptation in the context of specific applications. We therefore present a sampling of work from a number of application areas. We then discuss some attempts to identify the common elements of adaptation and generalize the problem across multiple fields.

2.1 Application-Specific Adaptation

Different adaptation strategies have been employed in a number of specific application areas. These include multimedia systems, computer graphics, and visualization. These techniques attempt to balance the competing needs of both application and system level requirements by accessing data at various resolutions. The solutions have tended to be application specific and ad hoc.

In multimedia, adaptation is an essential task. For example, video streaming adaptation has been achieved through several different ad hoc mechanisms. These include controlling the signal-to-noise ratio (SNR) in video coding [10], frame rate adaptation [20], or combinations of the two methods [16]. Layered video coding techniques have also been used to achieve adaptive performance [17]. More general approaches for include quality of service semantics [23] and augmentation and substitution models [3].

In computer graphics, the complexity of geometric models is often much greater than typical systems can handle for real-time rendering. As a result, significant effort has been spent developing techniques for geometric simplification [13]. At runtime, models of appropriate resolution are chosen based on any of a number of heuristics, including visibility estimates [6], screen-space error computations [12], and rendering costs [8]. These techniques are especially important for remote access to large, complex datasets where communication bandwidth is at a premium [9, 21].

Visualization systems must also deal with databases that contain far too much information to graphically display. As a result, adaptive mechanisms have been widely adopted. These include multiresolution terrain models [7], multiresolution volumetric representations [11], and multiresolutional and multidimensional data query infrastructures [2].

2.2 Generalized Adaptation

While most adaptation techniques have been developed in the context of a specific application, there are some notable exceptions. Some researchers have proposed several design principles for adaptive systems [14]. Other investigators emphasize the importance of data representation in adaptation [15].

In addition to general principles, formal adaptation models such as Adaptation Spaces [4] have been developed to explicitly specify possible alternative application behaviors for reliable systems. In other work, resource-centric quality-of-service models [5] are used to adapt resource allocations to compensate for changes in the system's state.

3. PROBLEM SPACE

While adaptation must necessarily incorporate application specific knowledge, there are a number of concepts that are common across a broad class of adaptive applications. Generally speaking, adaptation is the process of adjusting access to data to reflect current system conditions. In this section, we discuss the common adaptation concepts of data representation, metrics, and operations.

3.1 Representation

Adaptive data representations, regardless of application, are typically hierarchical or multiresolutional in nature. This derives from the fundamental reason for adaptation: limited resources must be allocated in response to changing system conditions. Without multiresolutional representations there would be little flexibility in data access and adaptation would be difficult. While simple representations may adapt across a single dimension, other datasets may contain several dimensions for adaptive behavior. We define the *dimensionality* of a dataset as the number of adaptive dimensions.

For example, a simple layered video representation has a dimensionality of one when the only choice in transmission is the number of layers. An alternative representation might have a dimensionality of three: color depth, frame rate, and image resolution.

A dataset can be considered to be a collection of individual *elements* of information. An element is an atomic unit of information at the application level. For example, in a layered video representation, one layer of one frame might be treated as an element.

Despite the notional independence of individual elements, there are often encoding dependencies introduced between them. Dependencies can be introduced in several ways, ranging from specific data structural reasons (for example, the dependence between layers in a layered video stream) to storage efficiency (for example, predictive coding between video frames). Data dependencies between elements can be viewed as the *syntactic* relationship between elements.

There can also be *semantic* relationships between elements. For example, a multiple-descriptor video stream stored at several bitrates has a group of independent elements for each frame. In this example, unlike a layered representation, each element is independently encoded and there are no syntactic dependencies between different bitrate streams. However, there is still a logical relationship that relates the set of elements within the bitrate dimension.

Elements are often grouped into *clusters* when encoded. We define a cluster as an access-level data structure which combines one or more elements. Data is stored as atomic clusters of information. For example, each quality layer in a video representation could be considered a cluster. During video playback, data is accessed by choosing a certain number of layers. This is in contrast to the elemental notion of individual video frames.

Finally, it is important to note the distinction between the actual encoded data and the representation's data structure. The representation structure, defined by elements and their relationships, is used to decide which data is required during adaptation. The structure may be explicit or implicit, but it is always there. When explicitly defined, the structure has been given many names in different fields. For example, the structure has been called a scene-graph *skeleton* [22] in graphics, a *description* [9] in streaming complex media types, and a *sketch* [15] in wireless systems.

3.2 Utility-Cost Ratio

The notions of *utility* and *cost* are fundamental concepts essential to adaptation. Individual elements are either more or less useful to an application than others. We define this notion as the utility of information. At the same time, access to a unit of data comes at some cost, often measured in time or required resources.

Given a set of available elements, the process of adaptation attempts to maximize the utility of the received information and minimize the associated cost. When noted as a ratio of utility with respect to cost, adaptation becomes an attempt to maximize the *utility-cost ratio*, or UCR.

3.3 Adaptation Operations

There are a number of common adaptation operations present in most adaptive applications. Given a data structure description and an algorithm for evaluating the utility and cost of specific elements, an application must take action to adapt the flow of data.

The two primary operations, *load* and *unload*, are tightly coupled. The load operation is performed to resolve data for a specific element in the data representation. An unload operation can be used to abort an ongoing load operation if the application decides it no longer needs the associated data, or to signal the disposal of an element's associated information at some point after the load operation has completed

Another universal adaptation operation is the management of inter-dimensional tradeoffs in utility. We refer to this as *dimensional scaling*. As system conditions change, an application may need to adjust the relative importance of individual dimensions with respect to other dimensions in the dataset.

4. FRAMEWORK

This section outlines our general framework for adaptation. We develop a graph-based data representation abstraction, which is embedded within a multidimensional utility space. We then pose the task of adaptation as a maximization problem.

4.1 Illustrative Example

Throughout this section, we illustrate our framework via a simplified sample application. For each new concept, we present a formal definition followed by a concrete example. The sample application is a simplified computer graphics system where a user navigates through a one dimensional scene composed of a collection of geometric objects, each of which is stored at multiple resolutions. We assume that the geometric models are layered in the sense that lower resolution models are used to code higher resolution information. At runtime, the example application should adapt the flow of incoming data to reflect both limited rendering resources and a moving viewpoint within the one dimensional scene.

4.2 Representation Abstraction

Our representation abstraction is a graph-based structure embedded within a multidimensional utility space. In the following subsections, we describe our abstraction and outline the mapping between individual components of our abstraction and the corresponding universal data representation properties outlined in Section 3.1.

4.2.1 Utility Space

The utility space, noted as $\mathbf{U}$, is a linear space over the field $\Re$. The space $\mathbf{U}$ is an n-dimensional space where n corresponds to the dimensionality of the dataset to be represented.

In general, utility space dimensions can be categorized into a taxonomy of three types. For any instance of $\mathbf{U}$, there may be zero or more of each dimensional type. The three classifications are:

- **Navigable**: Dimensions in which an application maintains a dynamic point of interest. The application adjusts the position of the point of interest based on application conditions.

- **Static**: Dimensions in which there is a predefined and constant point of interest regardless of any dynamic system conditions. This is analogous to a navigable layer with a fixed point of interest.

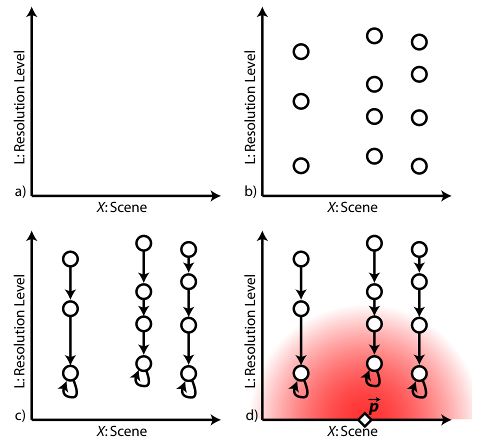

Figure 1: The sample application's representation graph.

- **Relational**: Dimensions which relate two or more distinct media subspaces (defined below) to form the global utility space $\mathbf{U}$. Relational dimensions provide a mechanism to express the relative importance between two or more independent media objects.

For multimedia applications, each media object is embedded within an independent *media subspace*. We note these as $\mathbf{M}_i$. The set $\{\mathbf{M}_1, \cdots, \mathbf{M}_n\}$ is unified via one or more relational dimensions to form the global utility space $\mathbf{U}$. Within $\mathbf{U}$, we define the *navigable subspace*, $\mathbf{N}$, as the region defined by the set of all navigable dimensions.

For example, our sample application has a dimensionality of two. There is a single navigable dimension, X, which defines the spatial position of geometric objects within the scene. The second dimension, L, is an static dimension and is used to reflect the resolution level of each representation of an object. The static value is zero, representing the lowest possible level, because we are interested in resolving low resolution data before high resolution information.

We therefore define the utility space as $\mathbf{U} = (X \times L)$ and the navigable subspace as $\mathbf{N} = (X)$. There is only one media type in this example, so $\mathbf{M_1} = \mathbf{U}$. We illustrate $\mathbf{U}$ in Figure 1(a).

4.2.2 Nodes

In our abstraction, nodes map to individual elements and represent an atomic unit of information. Utility evaluation (see Section 4.4) is performed at node-level granularity. For a given dataset, there is a set of all nodes, S. Each node $n_i \in S$ has a number of individual properties.

First, a node n_i is associated with a single media object and is located at a specific point $\mathbf{Pos}\{n_i\}$ within a single media subspace $\mathbf{M}_j$. A node has no defined position within all other media subspaces $\{\mathbf{M}_k\}, k \neq j$, and $\mathbf{Pos}\{n_i\}$ is therefore more correctly described as a hyperplane within $\mathbf{U}$ rather than a point. The relative positions between nodes are used to express the semantic relationship between individual elements of information.

Each node is assigned one of four states. This state, $\mathbf{State}\{n_i\}$, can change over time through state transitions. The possible states and valid transitions are described in Section 4.3. A node also maintains lists of both arriving edges, $\mathbf{Arr}\{n_i\}$, and departing edges, $\mathbf{Dep}\{n_i\}$. We present the concept of edges in Section 4.2.3.

In our running example, we use nodes to represent the model of an individual geometric object at a specific resolution. Due to our multiresolutional dataset, each object is represented by multiple nodes. All nodes for a given object share a common position

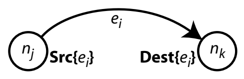

Figure 2: Edge e_i represents the data required to decode node $\mathbf{Src}\{e_i\}$ given that node $\mathbf{Dest}\{e_i\}$ is already resolved.

in the X dimension. However, they each have different resolution levels and are therefore positioned differently in the L dimension. In Figure 1(b), we illustrate three geometric objects at different positions in the scene.

4.2.3 Edges

Data dependencies between nodes are represented by directed edges. The set of all edges is noted as E. An edge $e_i \in E$ has both a source node $\mathbf{Src}\{e_i\}$ and a destination node $\mathbf{Dest}\{e_i\}$. An edge in our abstraction both expresses data dependence and corresponds to the specific bytes of information needed to resolve $\mathbf{Src}\{e_i\}$ given that we have already resolved $\mathbf{Dest}\{e_i\}$. An edge therefore expresses the syntactic relationship between a pair of nodes as depicted in Figure 2.

Not all nodes are dependent on data from other nodes for resolution. The data required to resolve a node without any prior knowledge is expressed via a *self-edge*. A self-edge is defined as an edge e_i where $\mathbf{Src}\{e_i\} = \mathbf{Dest}\{e_i\}$. We define the set of *base nodes*, $B \subset S$, as the set of all nodes with a self-edge.

Edges are assigned to a specific *cluster*, $\mathbf{Clust}\{e_i\}$. While each cluster may have several assigned edges, each edge belongs to exactly one cluster. We discuss clusters in Section 4.2.4.

In our example application, edges represent dependencies between different resolution models of the same geometric object, as shown in Figure 1(c). The model with the greatest error is fully encoded without any data dependencies. This leads to the inclusion of a self-edge for nodes representing an object at the coarsest level. The predictive relationships between nodes are expressed through directed edges pointing from higher resolution nodes to lower resolution nodes.

4.2.4 Cluster

We define a *cluster* as a group of one or more edges. Each cluster c_i contains a list $\mathbf{Edges}\{c_i\}$ of all edges assigned to it. In addition, a cluster maintains a cost estimate $\mathbf{Cost}\{c_i\}$ which measures the cost of loading. When performing load operations, the data associated with all edges in $\mathbf{Edges}\{c_i\}$ is treated as an atomic unit. Adaptation of the datastream is therefore performed at cluster-level granularity.

In our sample application, we need to load each resolution of each object independently. We therefore assign every edge to its own unique cluster.

4.2.5 Point of Interest and Prediction Vector

An adaptive application must maintain a *point of interest* which moves within $\mathbf{N}$. This point is part of a larger *prediction vector*, $\vec{p}$, which contains both the current point of interest and zero or more predictions of future interest points. Each vector entry $\vec{p}[i]$ pairs a point $\mathbf{Pos}\{\vec{p}[i]\}$ in $\mathbf{U}$ with a confidence value $\mathbf{Con}\{\vec{p}[i]\} \in (0, 1]$. The point of interest is noted as $\vec{p}[0]$. We restrict $\mathbf{Con}\{\vec{p}[0]\} = 1$ indicating full confidence in the position of the current point of interest.

Our simplified example uses a prediction vector of length one where only $\vec{p}[0]$ is defined. By definition, $\mathbf{Con}\{\vec{p}[0]\} = 1$. The static resolution level value, $l = 0$, and the user's current position in the scene, $x \in X$, are used to define $\mathbf{Pos}\{\vec{p}[0]\} = (x, 0)$. The prediction vector is illustrated in Figure 1(d) as a diamond marker.

Variable	Description
$\mathbf{U}$	Utility Space
$\mathbf{M}_i \subset \mathbf{U}$	Media Subspaces
$\mathbf{N} \subset \mathbf{U}$	Navigable Subspace
S	Set of all nodes
n_i	A node from the set S
$\mathbf{Pos}\{n_i\}$	The position of n_i
$\mathbf{State}\{n_i\}$	The state of n_i
$\mathbf{Arr}\{n_i\}$	The list of arriving edges at n_i
$\mathbf{Dep}\{n_i\}$	The list of departing edges at n_i
B	The set of base nodes (nodes with a self-edge)
A	The availability front (all nodes in state $Available$)
E	Set of a edges
e_i	An edge from the set E
$\mathbf{Src}\{e_i\}$	The source node for e_i
$\mathbf{Dest}\{e_i\}$	The destination node for e_i
$\mathbf{Clust}\{e_i\}$	The cluster to which e_i belongs
C	The set of all clusters
c_i	A cluster from the set C
$\mathbf{Edges}\{c_i\}$	The list of edges in c_i
$\mathbf{Cost}\{c_i\}$	The cost estimate for c_i
$\vec{p}$	The prediction vector
$\vec{p}[i]$	The ith element of $\vec{p}$
$\mathbf{Pos}\{\vec{p}[i]\}$	The position of $\vec{p}[i]$
$\mathbf{Con}\{\vec{p}[i]\}$	The confidence value for $\vec{p}[i]$
α	The set of scale dimensional factors
$\alpha_i \in \alpha$	An individual scale factor

Table 1: The parameters of our representation graph model.

4.3 State Transitions

Each node in our representation abstraction exists in a particular state. We express the process of adaptation through node state transitions over time. In this subsection, we outline the overall state space, discuss fundamental transition invariants, and present the set of allowable state transitions.

4.3.1 State Space

Each node in our representation is assigned to one of four possible states. A node's state may change over time, but it has just a single state at any particular moment in time. A node's state reflects the current status of the information represented by that node. The set of possible states is ordered to allow relational comparisons between nodes. In increasing order, the valid states are as follows:

- **Idle**: The information for this node is not resolved. Nor is it possible to resolve without resolving some other node first.
- **Available**: The information for this node is not resolved. However, it is possible to resolve without resolving some other node first.
- **Active**: The information for this node is in the process of being resolved.
- **Resolved**: The information for this node is resolved.

We call the set of available nodes the *availability front* and note it as A. The availability front has special significance during utility evaluation and is discussed in more detail in Section 4.4.

4.3.2 Invariants

There are three invariant conditions which are guaranteed to be true at all times. Following any node's change in state, these invariants must be enforced. Only state transitions which maintain these conditions are considered legal in our framework.

Self-Edge Invariant: A node with a self-edge may not be idle. Idle nodes require the resolution of other nodes prior to being resolved. A self-edge indicates that a node can be resolved without dependencies. Therefore such a node can never be idle.

Coherent Front Invariant: A node's state is less than or equal to the highest state among the set of predictors. This rule maintains

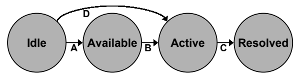

Figure 3: Promotion Transitions

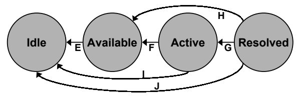

Figure 4: Demotion Transitions

a coherent front through the representation graph based on the data dependence relationships expressed by edges.

Active Predictor Invariant: A node can not be idle if it has one or more predictors that are in state $Active$ or greater. This implies that an $Idle$ node must transition to $Available$ upon the transition of any predictor to $Active$.

4.3.3 *Valid Transitions*

The state of each node can change over time by either *promotion* to a higher state or *demotion* to a lower state. In this subsection, we describe the set of valid transitions. For clarity, we introduce the notation $Trans(n_i, StateA, StateB)$ to mark a transition of node n_i from $StateA$ to $StateB$. We further classify transitions into two categories. *Primary transitions* occur as a direct result of an adaptation operation. Following a primary transition, a number of *secondary transitions* may ripple through the representation graph as invariants are reinforced.

Promotion Transitions. There are four valid promotion transitions, as shown in Figure 3. Nodes in the $Idle$ state can be promoted along one of two paths. First, $Trans(n_i, Idle, Available)$ occurs as a secondary transition when one of n_i's predictors is promoted to $Active$. The other promotion transition from $Idle$ is $Trans(n_i, Idle, Active)$ which occurs as a secondary transition when a cluster containing any of the edges in $\mathbf{Dep}\{n_i\}$ is loaded.

The third promotion transition is $Trans(n_i, Available, Active)$. This is a primary transition and can occur during a load operation when an $Available$ node is targeted for resolution along some edge. This transition can trigger a number of secondary transitions due to the Active Predictor Invariant and can even occur as a secondary transition itself.

The final promotion transition is $Trans(n_i, Active, Resloved)$. This is a primary transition and does not trigger any secondary transitions. It occurs when a node receives enough data to be be resolved and is no longer actively receiving data.

Demotion Transitions. Unlike promotion, there are no restrictions on demotion. This is illustrated in Figure 4 which shows that all six possible paths are valid. Demotion transitions are needed when data is flushed from a system. Typically, data is stored in a cache with limited space. As new data arrives, a cache management algorithm must decide which data can be evicted to make room for the new information. Demotion transitions are also needed to support the early termination of ongoing load operations.

A resolved node n_i can transition to any of the three other states. The specific transition is determined by the state invariants. For example, $Trans(n_i, Resolved, Active)$ is performed if a cluster containing any of the edges in $\mathbf{Dep}\{n_i\}$ is currently loading. Otherwise, n_i may transition to either $Idle$ or $Available$. The specific transition to occur is determined by the Coherent Front Invariant.

The demotion transitions presented so far are primary transitions that occur as a result of flushing a node's data or unloading a cluster. The remaining demotion transitions are secondary transitions and occur as side effects as the state invariants are reinforced. Following any demotion transition of node n_i, the nodes linked to n_i through edges $\mathbf{Arr}\{n_i\}$ must all be checked to ensure compliance with the state invariants.

4.4 Supporting Utility and Cost Evaluations

Both the utility of information and the cost of acquiring it are application specific properties. Our framework provides general tools to make these evaluations but leaves the formulation of specific metrics to the application designer. To evaluate the UCR, an application must first define two metrics. The first metric measures the utility of an individual node. The second metric determines the cost of resolving that node.

4.4.1 *Utility Metric*

We define an abstract utility metric, $UtilMetric$, used to evaluate the usefulness of each node $n_i \in A$. $UtilMetric$ evaluates the utility of a single node as a function of the node itself, the overall utility space, the set of all nodes, and the prediction vector. The implementation of this metric must be defined by the application to meet system-specific needs.

For example, in our sample application we need a metric that reflects our need for both low-error and nearby models. This can be achieved by computing the inverse of the distance between the $\vec{p}[0]$ and a node n_i as defined in Equation 1. Using this metric, nodes closer to the point of interest are assigned a higher utility. This behavior is shown by the shaded region in Figure 1(d).

$$UtilMetric(n_i, \mathbf{U}, S, \vec{p}) = \frac{1}{Dist(n_i, \vec{p}[0])} \tag{1}$$

4.4.2 *Cost Metric*

We define an abstract cost metric, $CostMetric$, used to evaluate the minimum cost of resolving a node n_i. A node can be resolved through any edge $e_i \in \mathbf{Dep}\{n_i\}$ where $\mathbf{Dest}\{e_i\} > Available$. For e_i, the cost is typically a property of the associated cluster $\mathbf{Clust}\{e_i\}$. $CostMetric$ evaluates the utility of a single node as a function of node itself.

The cost metric in our running example is simply the number of bytes associated with the node in question. In our representation, each node has exactly one departing edge. We can therefore define define the cost metric as shown in Equation 2.

$$CostMetric(n_i) = \mathbf{Cost}\{\mathbf{Clust}\{e_i\}\} | e_i \in \mathbf{Dep}\{n_i\} \tag{2}$$

4.4.3 *Utility-Cost Ratio*

A utility-cost ratio function, as described in Section 3.2, is used is used to drive adaptation. Given our utility and cost functions, we define the function UCR in Equation 3.

$$UCR(n_i, \mathbf{U}, S, \vec{p}) = \frac{UtilMetric(n_i, \mathbf{U}, S, \vec{p})}{CostMetric(n_i)} \tag{3}$$

4.5 Adaptation Operations

We support two classes of adaptation operations. First, the loading and unloading of clusters is performed through the UCR metric and state transitions. Second, the management of dimensional tradeoffs is implemented through scaling operations on the utility space. In this section, we describe these operations in more detail.

4.5.1 *Load and Unload Operations*

At runtime, an application will be busy loading a working set of clusters. The size of this set is determined by the application's budget and the cost associated with the loading clusters. An application must manage the working set to ensure that the must useful information is being loaded and supplied to the system.

Clusters are added to the working set through the $Load(n_i)$ operation. Typically, a system loads the node n_i with the highest UCR value and subscribes to lowest cost cluster associated with $\mathbf{Dep}\{n_i\}$ that would resolve n_i.

Clusters are removed from the working set in one of two ways. First, a cluster is removed from the set when it is completely loaded. When all the data for a cluster has arrived, the cluster is removed from the working set and communication resources can be reallocated to a new cluster.

Second, a cluster can be removed prior to resolution through $Unload(n_i)$. This signals that the active cluster for n_i is no longer needed and can be removed from the active cluster set. All active nodes associated with the cluster must be dropped from $Active$ to either $Idle$ or $Available$ depending on the state of the representation graph.

The adaptation logic in a typical application will repeatedly evaluate the UCR metric on all nodes $n_i \in A$ to determine the best available node. At the same time, the UCR metric will be evaluated on the set of $Active$ nodes to determine the worst active node. If the best available node has a higher UCR value than the worst active node, then the active node is unloaded in favor of the best available node. Otherwise, the working set remains the same.

4.5.2 Managing Dimensional Tradeoffs

Our representation abstraction maintains a set of nodes located within a multidimensional utility space $\mathbf{U}$. We express the utility of an individual node n_i as a geometric function of $\mathbf{Pos}\{n_i\}$ in $\mathbf{U}$. We can therefore manage the tradeoffs between dimensions in our utility space by performing scaling operations on individual dimensions. We define the set of scale factors as $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$, where $n = |\mathbf{U}|$.

Geometrically scaling $\mathbf{U}$ in a specific dimension will result in biasing the $UtilMetric$ evaluation and alter the importance of a specific dimension. The relative importance between dimensions is expressed through these scaling operations. An application must maintain the α factors for each dimension of $\mathbf{U}$ and adjust them to match the application requirements as they change over time.

5. APPLICATION CASE STUDIES

The benefit of an abstract adaptation framework is the applicability of a common set of functions and representations across a large set of adaptive applications. In this section, we present a series of case studies where we apply our framework to concrete adaptation problems. We frame each application as a spatial adaptation problem inside a multidimensional utility space. Our examples demonstrate both cross-media adaptation and high-dimensional utility spaces. We examine a multicast video streaming application, a 3D geometry system, and a image-based rendering streaming system.

5.1 Video Streaming

Quality adaptation for audio and video has often been achieved using layered representations [18]. In this case study, we consider a multicast-based application with independent video and audio streams. The distribution of each layer-encoded stream is split across multiple clusters.

Using our adaptation model, we define a utility space of three dimensions. There is a video layer dimension, V, an audio layer dimension, A, and a single relational dimension to facilitate intermedia tradeoffs, Γ. This forms two one-dimensional media subspaces, $\mathbf{M_1} = V$ and $\mathbf{M_2} = A$, and a utility space $\mathbf{U} = (V \times A \times \Gamma)$. We set treat both V and A as static dimensions, leaving $\mathbf{N} = \emptyset$. The lack of a navigable subspace implies that $\vec{p}$ is constant throughout the adaptation process and is located at the origin of $\mathbf{U}$.

Each layer is mapped to a node, implying that adaptation is performed by adding or dropping layers. The coding dependencies between layers are represented by edges. Each edge e_i belongs to an independent cluster corresponding to a multicast group. The representation graph and utility space for this design are shown in

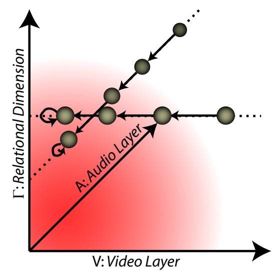

Figure 5: Three-Dimensonal Video Streaming Utility Space

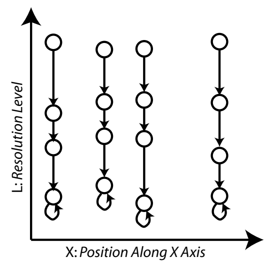

Figure 6: Utility Space for Geometric Application

Figure 5.

Our utility metric is a simple inverse Euclidean distance metric between a nodes position and the origin of $\mathbf{U}$. Our cost metric is the bitrate of transmission for the corresponding multicast group. The streaming application manages the tradeoffs between audio layers and video layers by changing the positions of $\mathbf{M_1}$ and $\mathbf{M_2}$ within the relational dimension Γ.

5.2 Geometric Models

Suppose we have a 3D streaming application with a scene composed of a collection of geometric objects. For each object, we have a series of levels-of-detail (LOD) that represent the object at various degrees of accuracy. We assume that the base LOD for each object is fully encoded while more accurate approximations are encoded predictively.

Using our adaptation model, we define a five dimensional utility space $\mathbf{U} = (X \times Y \times Z \times L \times \Delta)$ comprised of the three spatial dimensions (X, Y, Z), the resolution level dimension (L), and a geometric size dimension (Δ). The navigable subspace is $\mathbf{N} = (X \times Y \times Z)$. Both L and Δ are static dimensions. Because there is only a single media type, there is just one media subspace defined as $\mathbf{M_1} = \mathbf{U}$.

We map each individual LOD to a single node because adaptation is performed by managing specific LODs. We represent the encoding dependencies between different LODs of the same geometric object with edges. We illustrate a 2D subset of the overall representation graph in Figure 6.

Each edge is assigned to a different cluster to allow access to each LOD individually. We set $\mathbf{Cost}\{c_i\}$ equal to the size in bytes of the corresponding LOD.

We define a prediction vector $\vec{p}$ of length two. The first entry, $\vec{p}[0]$, is the point of interest. In the navigable subspace, $\mathbf{Pos}\{\vec{p}[0]\}$ is determined by the application and is based on the virtual camera's position. We use $\vec{p}[1]$ to represent the gaze direction and speed for the user under the assumption that any future movement will likely be in that same direction.

We implement $UtilMetric$ as a modified inverse distance measure between $\vec{p}[0]$ and a node n_i in space $\mathbf{U}$ after it has been scaled by the appropriate α scale factors. Once the scaled distance has been computed, we bias the utility based on the dot product between the view direction and the object direction (see the δ factor below).

$$\begin{gathered} UtilMetric(n_i, \vec{p}) = \tfrac{1}{\delta\sigma} \\ \delta = Dist(\alpha\mathbf{Pos}\{n_i\}, \alpha\mathbf{Pos}\{\vec{p}[0]\}) \\ \sigma = 2 + (|\alpha\mathbf{Pos}\{n_i\} - \alpha\mathbf{Pos}\{\vec{p}[0]\}| \cdot |\alpha\mathbf{Pos}\{\vec{p}[1]\}|) \end{gathered} \tag{4}$$

5.3 Image-Based Rendering

In the previous examples, we discussed the potential use of our framework in the context of two hypothetical applications. This third example illustrates a real-world utilization of our adaptation policies in a working prototype designed to deliver image-based rendering environments to large user groups [9]. In this subsection, we discuss the adaptive design for our prototype. We present experimental results in Section 6.

Our experimental prototype supports remote access to Sea of Images [1] datasets, which consist of several thousand high resolution images annotated with camera pose information. The camera position for all images is restricted to a plane at eye-level. During reconstruction, a virtual image is synthesized by interpolating across the three images closest to a virtual eyepoint.

In our prototype, we defined a five-dimensional utility space, $\mathbf{U} = (X \times Y \times \Theta \times \Delta \times \Gamma)$, corresponding to the multidimensional representation we developed for the application. There are three navigable dimensions, $\mathbf{N} = (X \times Y \times \Theta)$. Two dimensions, (X, Y), define the plane containing the set of camera positions. The Θ dimension corresponds to the image view direction measured as an angle from the $+X$ axis. There is only one media subspace $\mathbf{M_1} = \mathbf{U}$.

The remaining dimensions are static dimensions. Our representation uses a JPEG2000-based multiresolution image representation and organizes the set of images in a spatial quadtree hierarchy which arranges the images into a progressive order in terms of spatial density within $\mathbf{N}$. We define Δ as the spatial density dimension and Γ as the image resolution dimension.

We segment our dataset into groups of images by first partitioning the dataset using a regular grid in $\mathbf{N}$. Within each partition, we further divide the data along the Δ and Γ dimensions.

We map each of these divisions to a node within a representation graph. For compression, our representation introduces inter-element dependencies within a partition which are represented with edges. Each edge belongs to its own cluster and $\mathbf{Cost}\{\mathbf{Clust}\{e_i\}\}$ is set equal to the number of bytes in the cluster $\mathbf{Clust}\{e_i\}$. The prediction vector and utility metric are identical to those used in the geometric model example.

The similarity in $UtilMetric$ across all three case studies speaks to the power of a common adaptation abstraction. The same mechanisms and metrics can be used and reused across a wide range of applications. The individual application requirements are simply mapped to the available controls exposed by our framework.

6. EVALUATION

We performed several experiments to evaluate the performance properties of our proposed adaptation framework. We implemented a working prototype for a remote rendering engine for the Sea of Images (SOI) algorithm. The prototype was designed to support scalable streaming for large heterogeneous user groups [9]. We utilize the data representation and metrics defined in Section 5.3. We map each cluster to a unique transmission channel and data adaptation is performed by managing a working set of channels.

We tested system performance for our prototype under a variety of network conditions using the Dummynet network emulator [19]. For all experiments, we emulated a single user navigating a predefined path through a static image database consisting of 1,947 panoramic images. Each image has a full 360° field-of-view and a resolution of 2047×512. The raw size of the database is approximately six gigabytes.

Every experimental session lasted for 330 seconds and followed an identical path though the digitized environment. All experiments used a fixed channel bandwidth of 10,000 Bytes/sec. We refer readers to [9] for a more detailed description of our prototype and experimental configuration, and for the results of additional experiments. In the remainder of this section, we discuss the Area Factor quality metric and present results from our experiments.

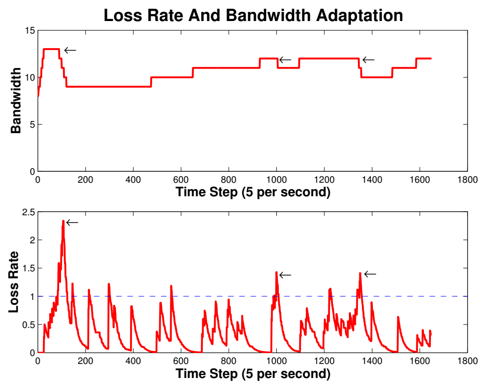

Figure 7: Bandwidth adaptation in response to estimated loss.

6.1 The Area Factor Metric

The Area Factor (AF) is the quality metric used in our experiments. The AF metric is derived from the SOI algorithm's use of image triplets. For any synthesized viewpoint, we can identify two triplets: the *ideal triplet* and the *available triplet*. The ideal triplet consists of the three images closest to the viewpoint using the entire database as input. The available triplet consists of the three images closest to the viewpoint available at the time of rendering.

The image positions of each triplet forms a triangle. We measure the quality of reconstruction by computing AF as the ratio of the areas of these triangles. The range for AF is restricted to $AF \geq 1$. Ideal reconstruction is represented by $AF = 1$. Larger values of AF signify greater disparity between the ideal and available triangles, implying poorer reconstruction quality.

6.2 Adaptation to Available Bandwidth

To test our framework's adaptive performance to changing network conditions, we measured both the size of the active cluster set in channels and the estimated loss rate for a complete session. During this session, we set the available bandwidth to 100,000 Bytes/sec. The results are plotted in Figure 7.

Client bandwidth hovered at about 10 channels. During the session, there were three loss events that caused decreases in the cluster set size. A decrease is triggered by the detection of loss rates above a threshold for an extended period of time. The three loss events are highlighted in the figure by arrows. Note that extended spikes in the loss rate were matched by drops in bandwidth. Following extended periods of relatively low loss, the network is probed for additional bandwidth by increasing the cluster set size.

6.3 Quality and Bandwidth

The target user population is heterogeneous in nature. Not only will users navigate the digitized space independently, the user pool will contain both high and low bandwidth clients. We tested our framework's performance over a number of different bandwidth connections. We computed the average AF value for each session and plotted it against the available bandwidth. As bandwidth increases, AF decreases, signifying improved quality at higher network capacities. This trend is evident in Figure 8.

The observed behavior shows that our framework yields improved quality for higher-speed connections. Adaptation is made possible by the multiresolutional utility space and the UCR metric allows for graceful quality degradation by intelligently allocating available bandwidth to the most useful data.

6.4 Quality and Latency

We simulated various levels of subscription latency by adding a fixed delay to all subscribe requests. We varied this additional delay from 20ms to 750ms. The delay values are in addition to any other network delays such as queue time. At each configuration, we computed the average AF for the session. The results are shown in Figure 9.

The results show that AF trends upward as subscription latency grows, signifying a drop in reconstruction quality. There are two reasons for this behavior. First, longer subscription latencies make adaptation harder by increasing the time for subscribe requests. Second, the system adjusts the α factors to compensate for longer latencies by increasing the importance of the navigable dimensions.

7. CONCLUSIONS AND FUTURE WORK

We have presented a general framework for multidimensional adaptation. The framework is applicable to a large class of adaptive applications with multiple media types and adaptive dimensions.

We defined an abstract graph-based data representation that explicitly models multidimensional and multiresolutional properties. Our abstraction can express both semantic and syntactic data relationships. We discussed graph-based state transitions as a mechanism for maintaining system state and data availability.

Our framework exports an interface to allow application-specific conditions to be incorporated into adaptive behavior. This is supported via dimensional scaling operations, the prediction vector, and custom $UtilMetric$ and $CostMetric$ metrics.

We have demonstrated the applicability of our adaptive framework to several media types. This includes both conceptual design and a real-world prototype designed for remote access to image-based models.

Using a common, generic framework allows a system designer to reuse the basic adaptation mechanisms and metrics by mapping their data and application requirements to the exposed interface. By posing the problem as a geometric exercise, the framework allows a simpler conceptual vision of adaptation for complex systems by viewing it as a spatial measure.

There are several avenues for future work. We would like to develop a common set of $UtilMetric$ and $CostMetric$ metrics to serve as a toolbox for system designers. In the same spirit, we plan to develop an open-source middleware library to facilitate more timely system implementation. This will include a defined API for adaptive applications.

We are also interested in exploring systems that require high-dimensional adaptive behavior. This includes our image-based rendering prototype and teleimmersive systems that stream several independent multiresolutional media objects of variable complexity.

8. REFERENCES

[1] D. G. Aliaga, T. Funkhouser, D. Yanovsky, and I. Carlbom. Sea of images. In *Proc. of IEEE Visualization*, 2002.

[2] M. Beynon, R. Ferreira, T. M. Kurc, A. Sussman, and J. H. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *IEEE Symposium on Mass Storage Systems*, pages 119–134, 2000.

[3] S. Boll, W. Klas, and J. Wandel. A cross-media adaptation strategy for multimedia presentations. In *Proc. of ACM Multimedia*, 1999.

[4] S. Bowers, L. Delcambre, D. Maier, C. Cowan, P. Wagle, D. McNamee, A.-F. L. Meur, and H. Hinton. Applying adaptation spaces to support quality of service and survivability. In *DARPA Information Survivability Conference and Exposition*, 2000.

[5] S. Chatterjee, J. Sydir, B. Sabata, and T. Lawrence. Modeling applications for adaptive qos-based resource management. In *Proc. of IEEE High Assurance Systems Engineering Workshop*, 1997.

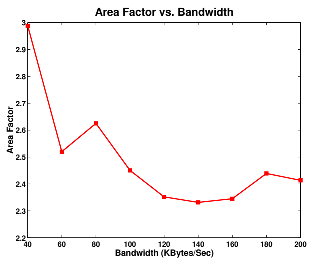

Figure 8: Adapting quality in response to changing bandwidth availability.

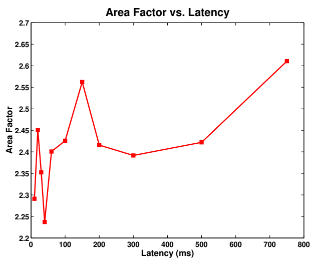

Figure 9: Adapting quality in response to changing latency estimates.

[6] D. Cohen-Or, Y. Chrysanthou, and C. Silva. A survey of visibility for walkthrough applications. *SIGGRAPH Course Nodes # 30*, 2001.

[7] L. D. Floriani and P. Magillo. Regular and Irregular Multi-Resolution Terrain Models: A Comparison. In *Proc. of 10th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'02)*, pages 143–148, 2002.

[8] T. Funkhouser and C. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proc. of ACM SIGGRAPH*, 1993.

[9] D. Gotz and K. Mayer-Patel. A framework for scalable delivery of digitized spaces. *To Appear in the International Journal on Digital Libraries*. Special Issue on Digital Museums.

[10] H. Kanakia, P. Mishra, and A. Reibman. An adaptive congestion control scheme for real-time packet video transport. In *Proc. of ACM SIGCOMM*, 1993.

[11] E. C. Lamar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proc. of IEEE Visualization*, 1999.

[12] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time continuous level of detail rendering of height fields. In *Proc. of ACM SIGGRAPH*, pages 109–118, 1996.

[13] D. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, pages 24–35, May 2001.

[14] M. McIlhagga, A. Light, and I. Wakeman. Towards a design methodology for adaptive applications. In *Proc. of ACM/IEEE International Conf. on Mobile Computing and Networking*, 1998.

[15] C. Policroniades, R. Chakravorty, and P. Vidales. A data repository for fine-grained adaptation in heterogeneous environments. In *International Workshop on Data Engineering for Wireless and Mobile Access*, 2003.

[16] R. S. Ramanujan, J. A. Newhouse, A. A. M. N. Kaddoura, E. R. Chartier, and K. J. Thurber. Adaptive streaming of mpeg video over ip networks. In *Proc. of the IEEE Conference on Computer Networks*, 1997.

[17] R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled video playback over the internet. In *Proceedings of ACM SIGCOMM*, pages 189–200, 1999.

[18] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for internet video streaming. *IEEE Journal on Selected Areas of Communications (JSAC)*, Winter 2000. Special issue on Internet QoS.

[19] L. Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[20] L. Rowe and B. Smith. A continuous media player. In *Network and Operating System Support for Digital Audio and Video*, 1992.

[21] E. Teler and D. Lischinski. Streaming of complex 3d scenes for remote walkthroughs. In *Proc. of Eurographics*, 2001.

[22] G. Varadhan and D. Manocha. Out-of-core rendering of massive geometric environments. In *Proc. of IEEE Visualization*, 2002.

[23] J. Walpole, C. Krasic, L. Liu, D. Maier, C. Pu, D. McNamee, and D. Steere. Quality of service semantics for multimedia database systems. *Database Semantics: Semantic Issues in Multimedia Systems*, 1999.