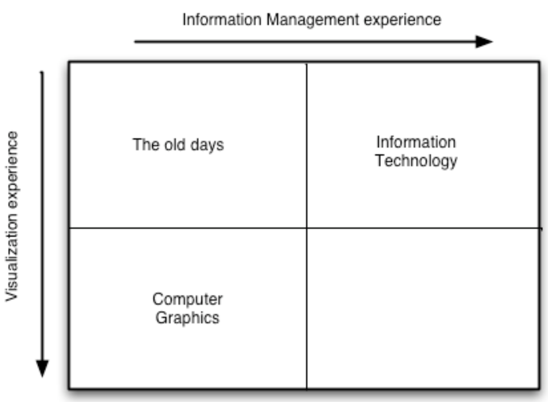
3D Graphics as an IT Application

Extended abstract

Julian E Gómez
Research Center for Advanced Computer Science
NASA Ames Research Center
Moffett Field, CA 94035

Introduction

In the computer graphics world there is decades of experience with visualizing data but little experience with data management. In the IT world there is decades of experience with managing data but very little experience with visualizing it.



Frequently a 3D application is the product of bipolar development, where a company's knowledge is incorporated in a database that has been developed over some time, and the knowledge must be transformed to a state that the 3D application can work with, and then transformed back afterwards to become part of the knowledge base. Transforming data is an expensive proposition; in many cases there are economies of scale, but there are no economies of knowledge, i.e. even though a process is learned, it still has to be executed in full the next time it is needed. It also raises the problem of the data on one side of the transformation becoming inconsistent with the data on the other side.

Another issue with transmitting 3D data is which format to use. Some vendors' formats have become de facto standards. Some open formats, like VRML, have gained moderate acceptance, but don't offer adequate breadth. Only recently has there been an ISO possibility, with X3D and its derivatives offering a standard and extensible means of describing 3D data and scenes.

Basics

A scene graph is a directed acyclic graph ("dag"), and an RDBMS is a directed graph ("digraph"). Since dags are a proper subset of digraphs, then scene graphs can be stored within an RDBMS. This paper proposes that 3D information constructs be treated as basic information units, akin to numbers and strings, and discusses some ramifications of doing so. The idea is significantly more detailed than asset management, where the 3D constructs are stored as BLOBS – the tables should be designed to represent the nature of a scene graph, and arcs from the scene graph stored as relations in the tables.

For this paper, we note that, in general, a scene graph or scene graph fragment is a subgraph when inserted into another graph, so the remainder of the discussion will treat all of them as subgraphs.

Figure 2 shows a general model of a table storing subgraphs. In general, a node can have *n* children, which are shown as a reflexive one-to-many mapping.

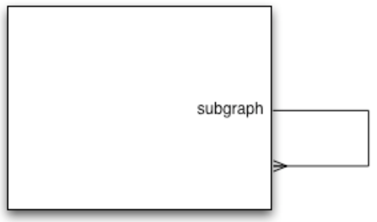


Figure 2. Table that understands subgraphs.

With scene graph fragments in a database, it is an easy matter to share the subgraphs. In addition, clients using the same subgraphs will actually be referring to the same pieces of information, instead of distributed copies of those subgraphs. This also means that there is an improved abstraction level between server and client, since they are dealing directly with 3D constructs instead of translated versions of those constructs.

This mechanism has been derived independently of any programming language or 3D graphics API. The 3D entities are stored as generic constructs. Since all scene graph APIs have a common theoretical basis, if a client needs a result in a particular language or set up for a particular API, the translation can be done as part of the query processing.

Subgraph vectorization

A very important possibility arises from storing subgraphs, which is to store the node structure separately from the parameters to the nodes.

The subgraph is composed of a dataflow network of nodes, and each one of those nodes has a parameter vector of multiformat data. A vector of all of those vectors forms a state description of the subgraph. The structure of the subgraph can be stored separately from the state description, and in addition, multiple state descriptions for that subgraph can be stored. Figure 3 shows an example subgraph, its structure vector (i.e. row), and multiple state vectors (also rows).

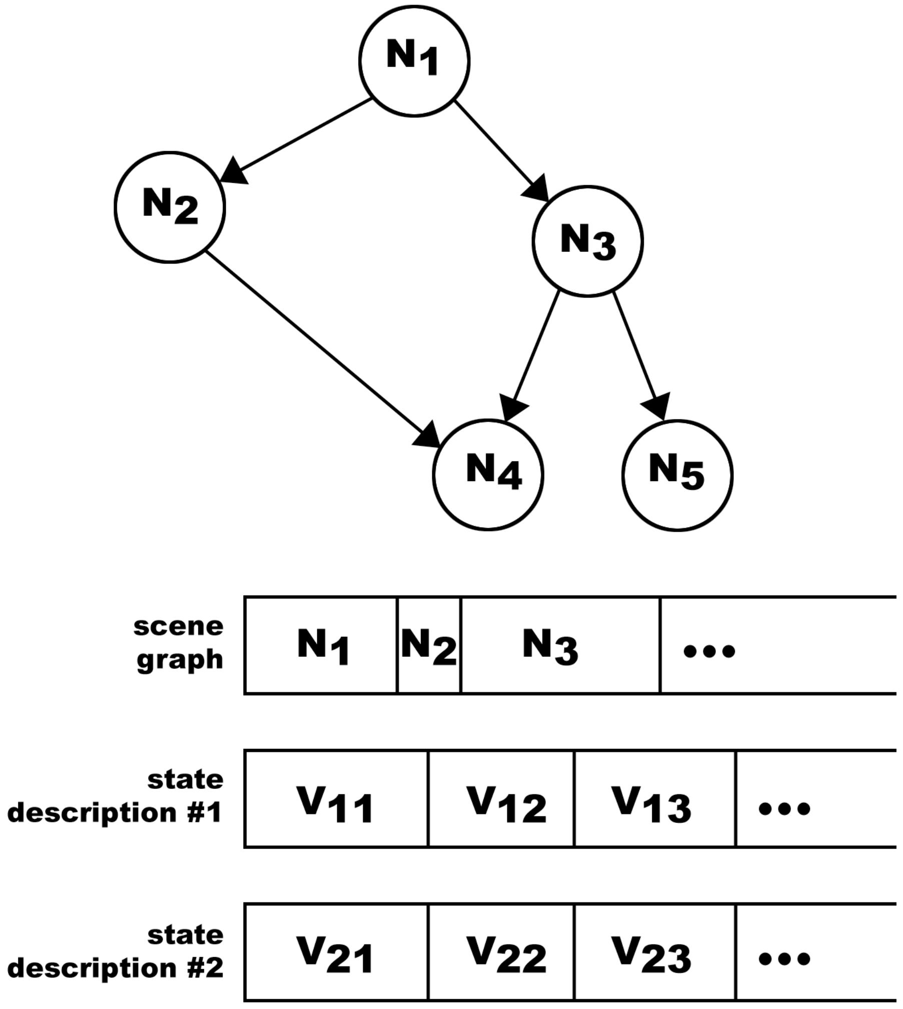


Figure 3. Storing scene graph vectors and state descriptions.

The advantage of storing the parameter vectors separately from the subgraph is that the subgraph can be defined once, and then alternate state descriptions loaded rapidly. Just as photographers will do bracketing of an exposure, the ability to load different state descriptions allows quick viewing of alternatives. An immediate benefit is in realtime applications, where it is very expensive to optimize a scene graph. Once it is optimized, then alternative values can be loaded without disturbing the scene graph.

Another possibility is animation. The state descriptions are tied to points in time, and then either the database or the client creates interpolated values for the parameter vectors. Because the 3D constructs are integrated with the knowledge base, this kind of animation allows more tightly coupled viewing of the overall system behavior; the previously required back and forth transformations disappear. This is a general benefit; it’s not limited to just doing animation.

Subgraphs as components

Following from the concept of a subgraph as a basic entity is that they are also components, in the IT sense of the word. There is a fairly direct relationship between graph descriptions and functional (i.e. dataflow) descriptions, so a subgraph will quite likely have a component representation as well.

As an example, a subgraph that performs rotation in an arbitrary plane could be encapsulated as a component, and that component used later in another subgraph. The query response would handle expanding the scene graphs as necessary.

The benefit of encapsulating subgraphs as components is that it promotes reuse. With the subgraphs being more manageable units, they're easier to incorporate in other subgraphs, or to be accessed by clients.

Note that this process does not imply modeling every node in the subgraph as its own component, nor does it assume that the only good representation is one component for the entire subgraph. The granularity of componentization remains to be studied.

Information access

Having 3D as a basic information unit changes the nature of the information access. Drilling down and sideways are accepted practices, but 3D allows in addition drilling in any direction. Furthermore, non-spatial data such as metadata or phase space representations could be stored as peers to basic information, allowing new ways of understanding the stored data, which could then be considered drilling in any dimension.

In general, an improved abstraction level between the client and the datastore means that no bilateral translation will have to be done. Also, since the 3D data is in the datastore with the business intelligence, the clients don't have to assemble and match different parts of information; they are inherently associated.

Example engineering application

As business intelligence moves from being a database to a knowledge base system, incorporating knowledge management technology, it becomes more important to have data abstractions at a higher level. A common engineering situation is a CAD/CAM environment. Parts are designed and managed by a PLM system. However, there is much more 3D information about a part than its design and history; examples include stress profiles, systems connectivity, and phase portraits of its failure profile. PLM can address only so many of these.

The common element among those examples is that they have a direct 3D visualization. Although it is possible to introduce that data, it's a significant development effort. However, with 3D data as a primitive, those elements then become basic information, and can be incorporated into the part information along with the other data that was defined earlier.

Future work

Future work includes looking for optimization possibilities, how to identify and describe dependencies between nodes, and analysis of the best granularity level, i.e. where does it make sense to treat a scene graph node atomically, as one row in the database. Too fine a granularity creates tremendous overhead as many rows are fetched and many objects are created, but too coarse a granularity means there won't be enough control over the subgraph.

Summary

As a company's information base moves from an RDBMS to a knowledge base, 3D graphics becomes a more important and relevant technology. 3D itself is frequently part of the information base, but just as importantly, it provides a mechanism to view and understand the complex relationships between the compound information constructs in a knowledge base. If 3D is a basic piece of information, the process of storing and visualizing can leverage from that power to become less complex. Benefits arise from vectorizing subgraphs to make them more flexible, from encapsulating subgraphs as components, and from being able to drill in multiple dimensions and information contexts.

References

X3D – Extensible 3D Specification. Web3D Consortium.

Gómez, Julian. The Convergence of IT and 3D. *Proc. Experiential E-commerce.* 2001.