삼차원 재구성을 위한 Data-Flow 기반의 프레임워크

김희관*
*제니텀
e-mail : albertk@zenitum.com

A data-flow oriented framework for video-based 3D reconstruction

Albert Kim
Zenitum, Inc.
e-mail : albertk@zenitum.com

Abstract

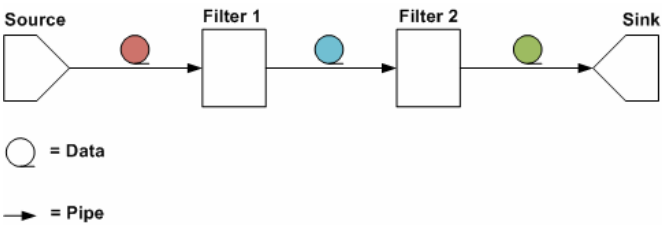
The data-flow paradigm has been employed in various application areas. It is particularly useful where large data-streams must be processed, for example in video and audio processing, or for scientific visualization. A video-based 3D reconstruction system should process multiple synchronized video streams. The system exhibits many properties that can be targeted using a data-flow approach that is naturally divided into a sequence of processing tasks. In this paper we introduce our concept to apply the data-flow approach to a multi-video 3D reconstruction system.

1. Introduction

The “Data flow architecture” a.k.a. “Pipes and Filters” architectural pattern [1][2] provides a structure for systems that process a stream of data.

Basically, a system is divided into several sequential processing steps where each step is only dependent on the output of its direct predecessor. The architecture forms a ***Pipeline*** where ***Data*** is passed down from ***Sources*** via a processing chain of adjacent ***Filters*** to ***Sinks***, all connected by Data ***Pipes***. The sequence of filters combined by pipes is the Data Processing Pipeline. The structure of the Pipeline forms a ***Data-Flow Graph*** with the Sources, Filters and Sinks as the ***Nodes*** and the Pipes as the ***Edges*** of the graph on which Data is flowing. A complex task is performed by processing data through several simpler tasks in an appropriate sequence.

As such, the dataflow paradigm is most suitable when developing applications that are themselves focused on the "flow" of data. The most readily available examples of a dataflow oriented applications come from where large data-streams must be processed, for example in video and audio processing, or for scientific visualization.



(Figure 1) Data flow architecture

Figure 1 shows an example of a simple data flow pipeline. Data flows from a source node through filter nodes to a sink node. The different colors denote possibly different data types produced or consumed by the nodes.

1.1. Related work

The data-flow paradigm has been used in various systems from different domains like digital video, hardware, music composition, multimedia, operating systems, parallel computers and scientific visualization. Widely known examples are UNIX pipes [3], the OpenGL rendering pipeline [4] and the Visualization Toolkit (VTK) [5] .

A recent example is the Network-Integrated Multimedia Middleware (NMM, [6]). NMM is a middleware for networked and distributed multimedia systems. User can specify data flow graphs using a textural description to connect networked multimedia devices like MP3 players and TVs as well as software components. NMM distributes and synchronizes the multimedia stream between the devices and components. The systems handles multimedia streams only, like audio and video, and does not deal with 3D data.

Another current approach is FlowVR [7][8] for high-performance interactive applications on PC clusters and grid environments. The main target is on virtual reality and scientific visualization applications. It consists of a core middleware which employs the data-flow oriented programming approach and a parallel rendering library. While it can be used for a wide range of virtual reality and visualization applications it is a complex framework with a much wider scope than our system.

2. Requirements

This section describes the context for our multi-video 3D reconstruction framework and identifies its requirements on an abstract level. We describe what it should be able to do and the desired properties.

Generally, we have the following situation. We have a studio for acquiring dynamic scenes from synchronized

video streams, recorded from multiple camera positions. The computer infrastructure consists of several standard PCs connected via an Ethernet network. Dependent on the setup some of the PCs will be connected to one or several video cameras through IEEE1394 interfaces.
The software part of the system has to accomplish various tasks like image capturing, image processing, 3D reconstruction and visualization of images or reconstructed geometry. The software components performing these tasks thereby should be deployed on the various hosts forming a distributed application. They should be able to communicate with each other by transferring data streams to implement the video-based 3D reconstruction.
For a developer the framework should provide the basic foundation to easily add and extend it by adding and test new components. For a user the framework should provide the possibility to easily configure and run a distributed application. In particular the system should show the following properties:

Flexibility and Extensibility:
The framework should provide the basic platform for rapid prototyping, testing and implementation of methods related to video-based 3D reconstruction and rendering. It should be easy to incorporate new modules into the system and rearrange existing modules to form families of related applications.

Scalability:
Scalability is the ability of a system to use increasing numbers of computing resources to more efficiently process large datasets. It should be easy to add new hosts or to rearrange the distribution of software modules across these host, e.g. to support more simultaneous camera streams. The system should support data, task and pipeline parallelism (see [9]). The number of supported host and modules should be limited only by the available hardware resources.

Understandability and Maintainability
The system design should be communicated to developers and users through the use of patterns as design language. Developers or users familiar with the patterns will have an immediate idea of the basic structure and properties of the system. The use of design patterns advocates a systematic and structured system design.

Configurability and Usability:
The framework and its components should be easily configurable and controllable through a graphical front-end which provides the users all necessary information to handle the system.

3. Solution

In this section we describe our implementation for a data-flow oriented framework for video-based 3D reconstruction is introduced which meets the aforementioned requirements.

3.1. Nodes

All source, sink and processing modules are represented by *Nodes*. A node is the smallest unit of processing. Each node can have a number of input and/or output *Ports*. Each port has exactly one supported *Data Format* that the node produces or accepts at this port. Dependent on the number of ports, nodes are classified in six generic classes all derived from a basic node class:

DataSourceNode
- Creates data objects and provides them to the processing pipeline.
- Has no input port and exactly 1 output port.

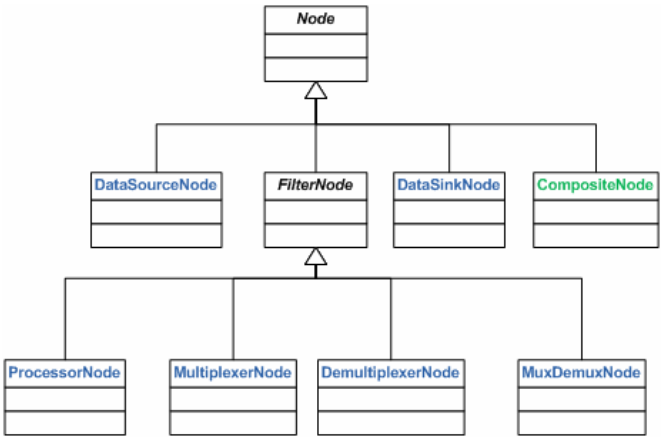
DataSinkNode
- Consumes data objects from the processing pipeline.
- Has exactly 1 input port and no output port.

ProcessorNode
- Basic filter node of the data processing pipeline.
- Has exactly 1 input port and exactly 1 output port.

MultiplexerNode
- Data filter node that has several input ports. Used to merge data from different nodes.
- Can have several input ports and exactly 1 output port.

DemultiplexerNode
- Data filter node that has several output ports. Used to split data to different nodes.
- Can have several output ports and exactly 1 input port.

MuxDemuxNode
- Data filter node that has several input and several output ports.
- Can have several output ports and several input ports.



(Figure 2) Node hierarchy

Figure 2 depicts the hierarchy of nodes. Concrete nodes are created as subclasses of the generic nodes. Each concrete node thereby generally runs as active component in its own thread. Additionally, nodes can be grouped inside a *CompositeNode* class which acts as a container for several nodes following the “Composite” pattern [10]. Grouped nodes don’t run as active components but are triggered by the

composites thread which reduces the number of concurrent threads.

3.2 Tokens

Nodes exchange information through message passing. The messages, called *Tokens,* follow the "Payloads" pattern [1] and can encapsulate all kinds of information. Tokens are self-identifying objects which provide an abstract model such that the type of information contained within each token can be easily identified by the nodes. They are composed of two components, a descriptor component and a data component. The descriptor component contains general information like the type and format of the Token. The data component contains type specific data and can be empty for some Tokens. There are two basic kinds of Tokens, Event and Data:

Event
Represent control information to be transported along the pipeline. Node classes process the events they are interested in and consume or forward them.

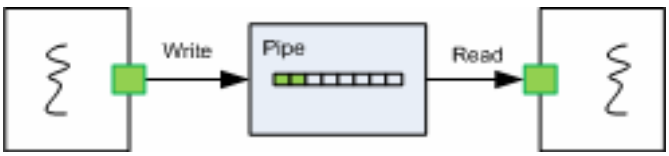
Data
Encapsulates data processed by the nodes. There are several generic data types like Image, Volume and Mesh. Subtypes of these classes specify the concrete *Data Format* that can be supported by Nodes at their input and output ports.

For network transfer tokens provide methods for serialize and deserialize them to and from byte streams.

3.3. Pipe

A pipe connects exactly one output port of a node with one input port of its successor. For the nodes to be connected the output port of a node has to be plug-compatible with the input-port of its successor. This means their supported data formats (or plug-formats) have to match.

Pipes provide a uniform interconnection mechanism. A pipe is implemented as a FIFO queue of token objects. Tokens are not interpreted by the pipe which allows adding new token types at a later time. Figure 3 shows a queue connecting two nodes. A producer node fills the pipe while the consumer node reads from the pipe. The pipe thereby performs synchronization of the two active components. Tokens are passed as references which avoids the overhead of copying.



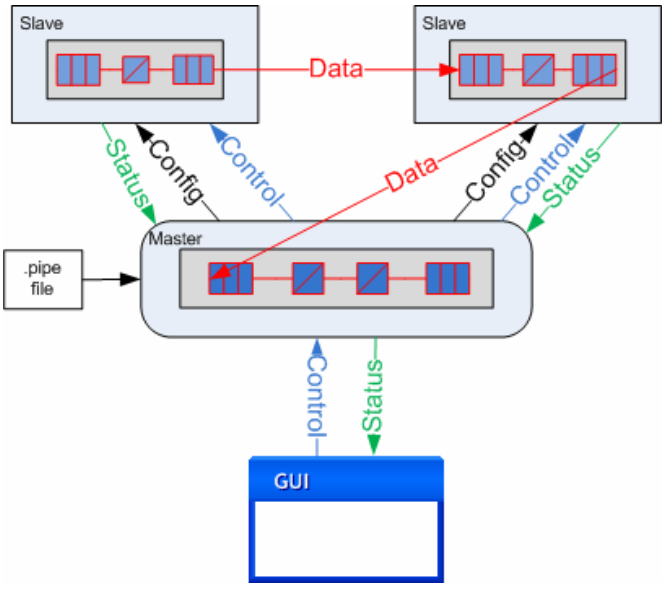
(Figure 3) Pipe

3.4. Pipeline Segment

A pipeline segment is one part of a distributed pipeline, deployed on a specific host. It is a completely connected data-flow pipeline from source to sink. The minimum pipeline segment thereby consists of one source node connected to one sink node. Nodes have to be added to a pipeline segment to be part of a distributed pipeline. To be valid all ports of the segment's nodes have to be connected properly and no port has to stay unconnected.

3.5. Distributed Pipeline

A distributed pipeline consists of at least one pipeline master and a number of pipeline slaves. Each slave manages one pipeline segment. The segments of the distributed pipeline are connected through special network source and sink nodes, where the network sink node of one segment is connected to the network source node of its successor segment. The master is responsible for managing the whole distributed pipeline. He reads the pipeline configuration file and checks it for consistency, i.e. ensures that all node connections are set appropriately. He initializes the slaves with the configuration information and controls them during run-time. Furthermore he can also manage one pipeline segment by himself.



(Figure 4) Distributed Pipeline

Figure 4 shows a simple diagram of a distributed pipeline. The master reads and evaluates the configuration information for the pipeline and distributes it to the participating slaves. Each slave then builds its own pipeline segment using the configuration information and connects its network sink nodes to the network source nodes of its successor segment. A user controls the master through a graphical front-end to start the pipeline and display status information from master and slaves.

3.6. Pipeline Configuration

For creating new distributed pipelines a XML data-flow graph description format (.pipe) is defined. The format consists of two main parts. In the first part the pipelines connectivity is specified, i.e. how nodes and their ports are connected to each other. In the second part the properties of the nodes are set.

For user convenience a visual programming tool can assist

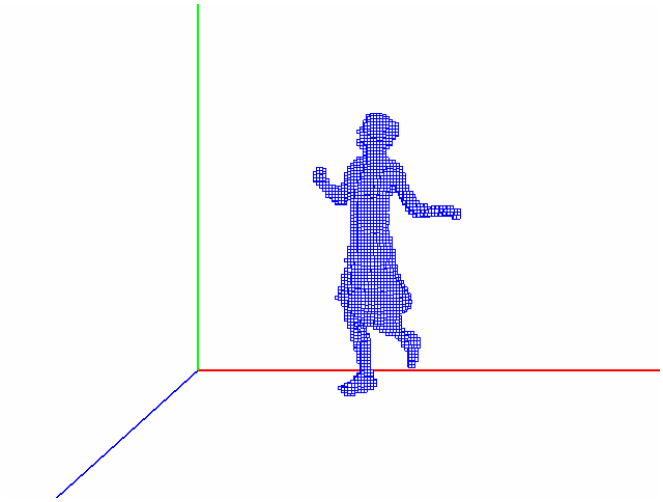
in the creation of data-flow graphs. This is used to simply connect modules and set their parameters without the need for any other programming or manually editing of XML files.

3.7. Node Development

To simplify node development a XML node description format (.node) is provided. Developers specify the generic type, the supported data formats and the specific properties of the new node in the description file. A node compiler generates C++ stub classes with the basic code skeleton for integration into the framework. Developers build subclasses of the stub to implement the nodes functionality. Nodes are compiled into shared libraries and placed into a special plugin directory to be loaded by the frameworks runtime.

4. Application

As an example we implemented a volume-based 3D reconstruction application for our framework. It uses the images from 8 synchronized video streams to reconstruct a voxel model of a dancing female actor. The applications consists of 66 nodes from which many, like the ones for reading images, background subtraction and JPEG encoding and decoding, can be used to build similar applications. Figure 5 shows a screenshot of the reconstructed model.



(Figure 5) Volume based 3D reconstruction

5. Conclusion

In this paper, we presented a data-flow oriented framework for video-based 3D reconstruction. It is designed with flexibility, extensibility and scalability in mind to provide a platform for rapid prototyping and implementing 3D reconstruction algorithms and applications. We outlined the core design decisions, components and tools of our system. The framework allows it to easily develop, integrate and test new algorithms and data types and distribute them flexible over several computers. A prototype of the framework is used in our video-based 3D reconstruction studio.

6. Acknowledgments

This research was supported by a grant(07KLSGC05) from Cutting-edge Urban Development – Korean Land Spatialization Research Project funded by Ministry of Construction & Transportation of Korean government.

References

[1] Dragos-Anton Manolescu, *A Data Flow Pattern Language*. In Proceedings of the 4th Pattern Lanuages of Programming Conference (PLoP 1997), Monticello, Illinois, USA, September 3-5 1997.

[2] Frank Buschman, Regine Meunier, Hans Rohnert, Peter Sommerlad, and MichaelStal. Pattern-Oriented Software Architecture—A System of Patterns. John Wiley & Sons,July 1996. ISBN 0-47195-869-7.

[3] Dennis M. Ritchie. *A stream input-output system*, AT&T Bell Laboratories Technical Journal, 63(8):1897–1910, October 1984.

[4] Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, *OpenGL Programming Guide*, 6th Edition, Addison-Wesley, 2007, ISBN 0321481003

[5] Will Schroeder, Ken Martin, Bill Lorensen, *The Visualization Toolkit. An Object-Oriented Approach To 3D Graphics*, 4th Edition, Kitware, Inc., 2006, ISBN 1-930934-19-X

[6] M. Lohse, F. Winter, M. Repplinger, and P. SlusallekRoger. *Network-Integrated Multimedia Middleware (NMM)*. In Proceedings of ACM Multimedia 2008(ACM MM 2008), Vancouver, Canada, October 27-31, 2008

[7] T. Arcila, J. Allard, C. Menier, E. Boyer and B. Raffin. *FlowVR: A Framework for Distributed Virtual Reality Application*. In: 1ière journées de l'Assoiation Française de Réalité Virtuelle, Augmentée, Mixte et d'Intera tion 3D, Roquenourt, Frane(November 2006)

[8] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert. *FlowVR: a middleware for large scale virtual reality applications*. In Euro-Par 2004 Parallel Processing: 10th International Euro-Par Conference, pages 497–505, Pisa, Italia, August 2004.

[9] James Ahrens, Charles Law, Will Schroeder, Ken Martin, Michael Papka, *A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets*, Technical Report, Los Alamos National Laboratory, 2000

[10]Erich Gamma, Richard Helm, Ralph Johnson, and JohnVlissides. Design Patterns—Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995. ISBN 0-201-63361-2.