

Interactive Grid-based Free-Form Shape Modeling

Anthony Chong S. K.¹, Konstantin Levinski² and Alexei Sourin³
School of Computer Engineering, Nanyang Technological University, Singapore

Abstract— Geometric shape modeling becomes increasingly complex and resource-demanding task. In this paper we propose a method to leverage the power of Grid to provide users with high-precision free-form shape modeling environment.

Function-based representation allows for defining shapes with an unlimited level of detail while keeping a small size of the model. The model compactness allows us to create optimal infrastructure for distributing work between processing nodes. A novel parallel surface extraction algorithm was proposed to avoid a uniform sampling of the defining function. Only points located close to the surface of the shape are sampled.

A protocol for interactive communication between the user of the modeling system and the processing modules in Grid has been developed to support shape modification, visualization with arbitrary precision and dynamic load-dependent allocation of computing resources during the modeling session.

Index Terms—Free-form shape modeling, Grid Computing

I. INTRODUCTION

Free-form shape modeling is a method of construction of geometric shapes in any possible way rather than from a finite set of basic shapes. Such freedom of modeling requires new shape representations capable of expressing arbitrary modifications.

A shape capable of free-form modifications can be efficiently defined with an explicit function. An explicit function $g=f(\mathbf{p})$ defines a scalar field over the \mathbf{R}^3 space, where point $\mathbf{p} \in \mathbf{R}^3$. The surface of the shape is defined with an equation $f(\mathbf{p})=0$. Using this representation, we can simulate various crafting techniques such as sculpting, carving, embossing, engraving, etc. [1]. The original shape and thousands of modifications applied to it are expressed as mathematical functions. The function-defined model is extremely compact compared to commonly used shape definitions with polygons and voxels. However, thousands of interactive operations make the defining function so complex that the modeling becomes non-interactive on a single computer. In this project we use Grid to get the required computational power for high-precision free-form shape modeling. The compactness of the representation allows us to avoid installing Grid client on the end-user computer. A small

web-based Java3D application is sufficient to receive the data from Grid and send the modifications introduced by the user back for processing.

II. GRID-BASED SHAPE MODELING

In a single-processor environment, the principle of surface extraction is simple: starting from the initial point \mathbf{p} such as $f(\mathbf{p})=0$, we follow the surface, keeping track of already extracted area. This method is not feasible when several processors are used because a real-time coordination is required to keep track of the processed areas. To parallelize the extraction process, we propose a 2-step approach. At the first step, a very low detail extraction is performed to find out where the main part of the surface is and how the processing should be distributed among the available resources. The second step is performed by all the available nodes simultaneously using the cells obtained at the first step. The algorithm for this two-step extraction is illustrated in Figure 1. It works as follows:

- 1) The cell involved in polygonization is processed using a computing node.
- 2) Cells are distributed among the nodes for computation.
- 3) New cells might have to be computed after step 2. For example, cell 3 in Figure 3 will be detected only after cell 2, which is below it, is processed.
- 4) If no new cells can be found, the process is considered to be completed pending modification requests. Otherwise we repeat from step 3.

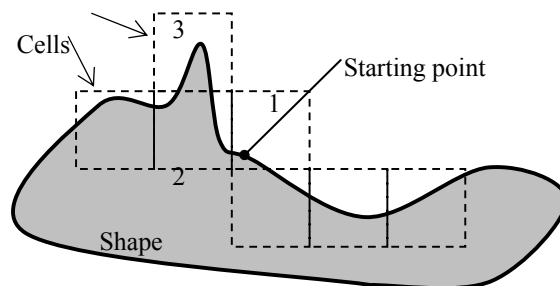


Fig 1. Surface extraction of the shape. Each processing node is assigned to the surface inside a fixed cell. When the cell is processed, the next cell is taken from the queue for processing

As soon as the node completes a part of the surface, the part is immediately sent to the client for visualization, and the next node is requested. If the function changes due to interaction,

¹ Email: chongsk@ntu.edu.sg

² Email: konstantin@ntu.edu.sg

³ Email: assourin@ntu.edu.sg

the affected cells are marked as invalid. Coarse extraction takes place to determine the new cells. The cells are submitted to available nodes for surface extraction. Cells that are found not containing the surface are marked and removed from further processing. The advantage of this visualization scheme is that only cells known to contain the surface are probed – otherwise we would have to check every point in space which would be prohibitive even for Grid.

The processing can be expressed with the producer-consumer scheme. The initial function is consumed by a node to perform coarse extraction. The node produces cells worth searching for the surface. These cells are consumed by all the participating nodes, which in turn produce surface patches and request for additional cells to consider. The additional requests are needed as not all the cells containing the surface are detected during the initial extraction.

As there are many complex types of communications in the system and due to the producer-consumer pattern, Tuple Space concept comes in naturally [2]. Tuple Space is a collection of tuples. A Tuple is an ordered collection of data which can be put and retrieved from the Tuple Space according to a filter. Tuple Space allows a node to be blocked while waiting for the particular type of the message, such as to refine a cell, or for to process a proposed cell, or to modify a function.

Tuple is a list of values, like

(*Session_ID*, “*modification*”, *Data_ID*), or
(*Session_ID*, “*new cell*”, *Coordinates*).

The communication process goes as follows:

- Modeling process is assigned an ID.
- An initial amount of computation nodes is allocated, initialized, and connected to the Tuple Space, waiting for a function with the respective ID to arrive.
- The function and the initial point are submitted to the Tuple Space.
- An initial cell is grabbed by the Rendering Coordinator and all the cells intersecting the surface are extracted. After the extraction, cells to be processed are put into the Tuple Space.
- The nodes grab the cells, return surface patches, and request additional cells if needed.
- The clients grab the available patch and render it.

The procedure repeats until no new cells can be found in the Tuple Space.

To modify a function-defined model, one has to modify the underlying function. The client submits modification parameters to the Tuple Space so that all the processing nodes could immediately receive the changes.

Then, the coordinating node computes what cells have to be updated, and put this information on the Tuple Space. The processing nodes pick the relevant cells and send updates to the client.

Each cube with the surface extracted is sent over to the client application. To make the process run in real-time, the processing power required for each cube is kept small. The models which require heavy computations will still be displayed at an interactive rate, but with a lower resolution.

Let us consider the way how the discussed structure is

organized so that the visualization and the processing parts connect in a way that is transparent to the user, secure and reliable.

The standard schedulers require client machines to be equipped with a complete suite of the Grid software, and all the communication to be carried out through the API. However, Grid software is excessively complicated for normal user to install and configure. We decided to implement the communication between the client and the Grid part independently using a custom protocol. It is designed to transfer pieces of the shape’s surface, as well as to request for partial visualization, update, and shape modifications.

It is not possible to organize a single point of contact for every node participating in the Tuple Space. Hence, we opted for a distributed Tuple Space configuration where each node knows where to direct certain tuples. Figure 2 shows one of the possible layouts of the Tuple Space. For example, suppose the incoming tuple has to be directed to the initial processing node, while the following tuples – to the processing nodes. It is quite straightforward to keep the information on where each tuple should go in the nodes of the network.

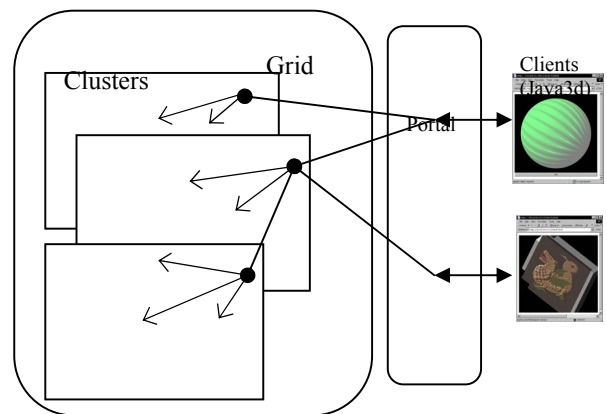


Fig 2. Information flow for the visualization process. Client requests the required computational power (nodes) through the portal. The portal forwards the requests to the clusters which form an ad-hoc network to process requests.

Consider N tuples going for distribution among the consumers. Then, for n similar nodes awaiting these tuples, the optimal queue is N/n . Every time a tuple is consumed, the length of the neighboring queues is equalized. By this way, all the processing nodes are made constantly busy, and the load is averaged. This process is illustrated in Figure 3.

In order to improve the performance, we have split the data flow during the interactive shape modeling into two parts: synchronous control through the Tuple Space and asynchronous data transfer. Putting all the data in the Tuple Space is the straightforward solution, but it is not efficient because high-bandwidth traffic will be concentrated on one machine. Bulk data, such as surface patches, which require no further processing on the Grid side, is delivered directly to the database and forwarded to the clients.

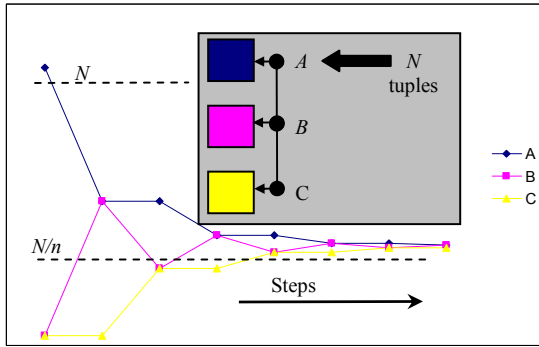


Fig 3. Tuples propagation. At each step, the neighboring nodes share the available tuples. The graph shows the number of tuples in queue at each node. The load keeps balance automatically, without central monitoring.

The Tuple Space's primary goal is to efficiently distribute work between the available nodes, propagate function updates and coordinate the flow of information back to the clients. The bulk of final data, such as surface patches generated by the nodes, do not belong to the Tuple Space. It is therefore sent directly to the clients through a dedicated daemon on the portal machine.

The configuration of this system is shown in Figure 4. As the portal is directly accessible from the processing nodes, it is possible to take advantage of all the available bandwidth by sending the final data in parallel from different nodes.

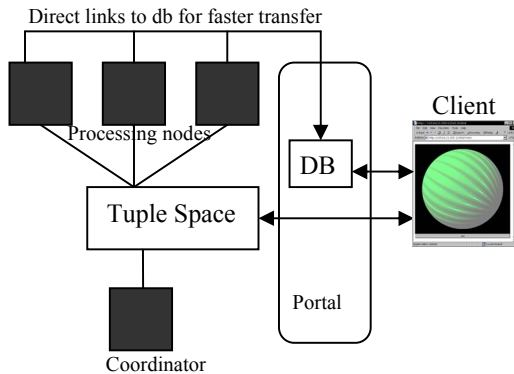


Fig 4. Data flow combining flexibility of Tuple Space and fast direct communication for fast response.

This framework has to be highly flexible as it does not use 100% of the computational resources all the time during the modeling. Therefore a universal approach was developed to deal with such highly flexible systems. A network database, which is accessed by both the clients and the resource nodes, stores all the transactions between the renderer and the clients. It allows for achieving very high granularity – a worker can immediately switch between different render tasks, as it has no direct connection to the modeler.

Both clients from outside and resource nodes running inside Grid are connecting to a meeting point accessible from outside. The clients supply requests for visualization and modification of the models. The resource nodes, waiting for such requests, perform them and return the results into the database to be retrieved by the clients who requested the

modifications.

The visualization can be performed in a similar manner. The requests to visualize a certain model in a given area are submitted to the database from where they are immediately fetched by any available worker, which returns the results back when they are ready. The visualization requests are just bounding boxes containing the area of interest. The clients submit series of such bounding boxes and gradually receive model representation in each of them. A connection-less point-based representation [3] is used to avoid seams between the boxes.

The flow of information consists of fairly small chunks – there is no need to transfer the whole model but rather only its incremental updates. As the amount of work needed for each modification is small, very smooth work distribution can be achieved.

In this paper we describe only the design and implementation of the core of the system including the Tuple Space, database, and portal integration.

The communication database serves two purposes:

1. Task coordination.
2. Intermediate data storage.

Therefore, it was implemented as a two-layer system. Control Layer is written in a high-level language and is capable of serving complex queries. Data layer is a simple hash table where clients and resource nodes can store information referenced from the Control Layer.

The control layer is capable of blocking the request until the relevant data comes in. The database notifies the resource nodes when there is a task to do, and tells the clients how to do it when their request is ready.

To optimize the database and bandwidth usage, the data layer is distributed among several nodes and even among several different clusters. Usually, each TCP connection is limited to a certain speed unless we use UDP streaming, as in [4].

In our case, we have as many connections as the number of databases. Therefore we are using a physical limit of the line instead of the single-connection limit.

Tuple Space is a very flexible way of communication between a various amount of worker nodes and several client nodes. While it is possible to use the same space for many concurrent modeling sessions, it is not optimal performance-wise. Many different clients keep connecting to the same ports and prevent the system from scaling.

For our case, the efficiency can be improved by separating tasks into permanently connected clusters, while each of them is acting as a separate Tuple Space. The process runs as follows:

- The clients request for computing resources needed for the session.
- The topmost Tuple Space receives the requests.
- The nodes in the cluster consume the request and spawn the connected Tuple Space for this session
- The Tuple Space is given a separate port in the portal machine for clients to connect.
- The topmost Tuple Space returns the connection

point to the client.

The data sent to the clients is stored in the database to avoid the unnecessary calculations for additional requests.

This two-level architecture allows the application to dynamically add new nodes to the topmost level, as well as to connect additional nodes to the dedicated Tuple Spaces at the runtime.

III. DISCUSSION AND RESULTS

Let us highlight the main advantages of our design. The Tuple Space system allowed an arbitrary number of participating parties to be linked dynamically. It enabled adding and removing resources at run-time as needed. This is quite important for applications with rapidly changing load, such as shape modeling.

Splitting the model to independent cells resulted in fast detection of the modification area, as well as in small turn-around times. An average modification takes a few seconds to complete depending on the network condition.

Efficient bandwidth utilization becomes possible due to the multiple simultaneous connections opened from the processing nodes to the meeting point. Usually, each TCP connection is limited to a certain speed. In our case, we have as many connections as the number of processing nodes, therefore we are using the physical limit of the line instead of the single-connection limit.

A dedicated node connected to the Tuple Space deals with the scheduling of surface extraction and rendering. If one node is not sufficient, it is straightforward to add more nodes as needed.

TABLE 1:
EXPERIMENTAL RESULTS FOR 3D DRAGON RENDERING:
SINGLE FRAME OF 500 X 500 RESOLUTIONS.

Hardware involved (Type and number of CPUs)	Rendering Time for the Dragon. (seconds)
20 nodes of Xeon 2.6 GHz	40
One node of Pentium IV 3GHz	720

Table 1 shows the experimental results for the shape shown in Figure 5. The minimum possible time for the reference model is determined by the bandwidth between the client and the portal. Currently, no compression is implemented, so the rendering time can be improved in future by using compressed geometry.

The proposed method of Grid-based shape modeling has certain advantages over traditional shape modeling. We are not limited by the function complexity anymore, as the system is dynamic and can use more resources transparently as needed.

IV. CONCLUSION

To cope with ever increasing complexity of geometric models, we have proposed an algorithm and a system layout to

leverage the power of Grid and provide the users with a high-precision shape modeling environment. The algorithm was implemented in a framework for free-form function-based shape modeling on Grid, which enables user to offload most of the computations required during the shape modeling process.

We proposed and implemented the core system performing the client-Grid data exchange and visualization. The proposed protocol enables users to work with models that are too complex for handling on a single PC. The proposed framework is flexible and has very low requirements on the user side (Java and Java3D). This makes the developed shape-modeling tool an attractive low-cost application which can be accessed from any Internet-connected computer.



Fig 5. A complex function-defined 3D shape rendered on Grid. The shape is defined by 8,000 individual interactive operations

V. ACKNOWLEDGMENTS

This project is a part of the R&D Collaboration Adaptive Enterprise @ Singapore (AE@SG).

REFERENCES

- [1] K. Levinski and A. Sourin. Interactive Function-Based Shape Modeling for Cyberworlds, *Proc 2004 International Conference on Cyberworlds*, Tokyo, 18-20 November, 2004, 54-61.
- [2] D. E. Bakken, R. D. Schlichting. Supporting fault-tolerant parallel programming in Linda. *IEEE Transactions on Parallel and Distributed Systems*, 6:3, 1995, 287-302.
- [3] J. P. Grossman and W. J. Dally: Point sample rendering. *In Proc. of the 9th Eurographics Workshop on Rendering*, 1998, 181-192
- [4] E. W. Bethel and J. Shalf "Grid-Distributed Visualizations Using Connectionless Protocols", *IEEE Computer Graphics and Applications*, 23:March, 2003, 51-59.