Interactive Visualization of Large Remote AMR Data

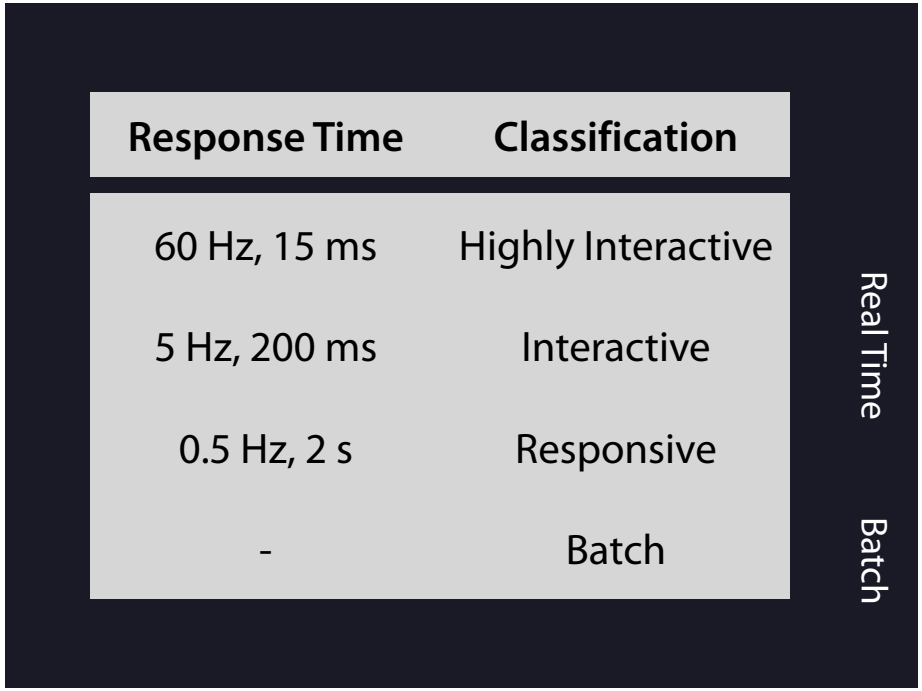
Steffen Prohaska
Obergurgl, April 2005

I'm briefly present a system we built recently. It allows to access remotely stored AMR data, transfer parts of them to a local machine, which is responsible for rendering the data.

Before I talk in more detail about our system, I'd like to briefly talk about some basic ideas and problems with large and remotely stored data.

Interactive Visualization

Before that I'd like to briefly characterize interactive visualization.



I put some respons times and some classifications in this table. Depending on the use case you may require higher or lower, or even now interactivity.
Some examples: immersive environments track the eye position of the user and it is crucial that the system rapidly adjusts to the head movements. Therefore, a highly interactive system is mandatory.
Normal interaction with 3D scenes is still comfortable at several frames per seconds. If it takes a couple of seconds until an application responds it becomes annoying to use it. A batch system processes all inputs and generates the output as fast as possible. But no interaction is required.

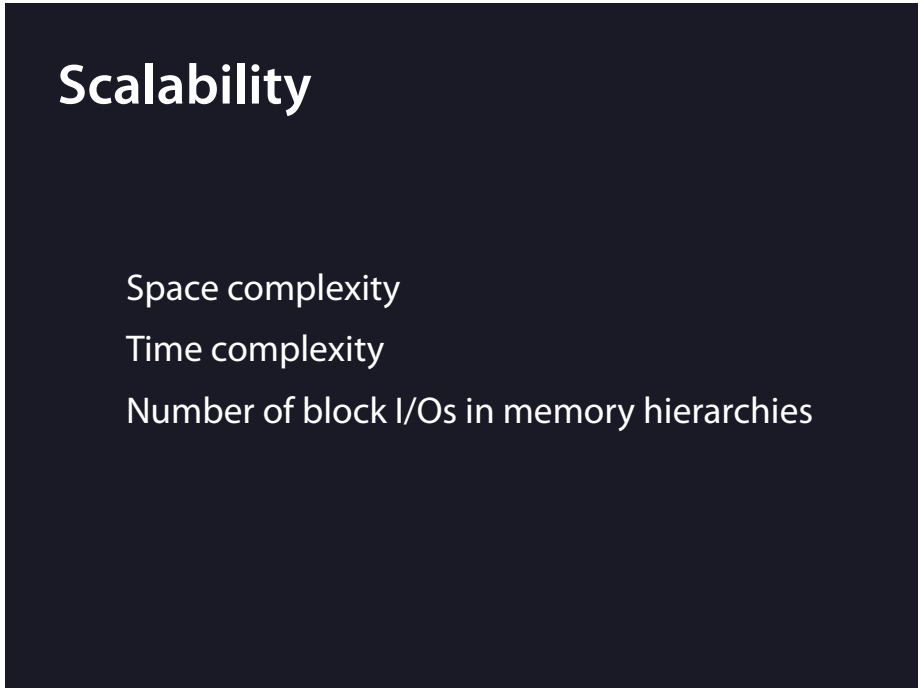
What we normally target are interactive systems that are able to render with several frames per second.

All the interactive systems are sort of real time systems. They should, at least in average guarantee response times. This may limit the decision, which resources to use.

The next thing is large data. Large is always relative to the available resources.

Data may be larger than the resources on a graphics board, or larger than the available main memory.

The main problem is scalability.

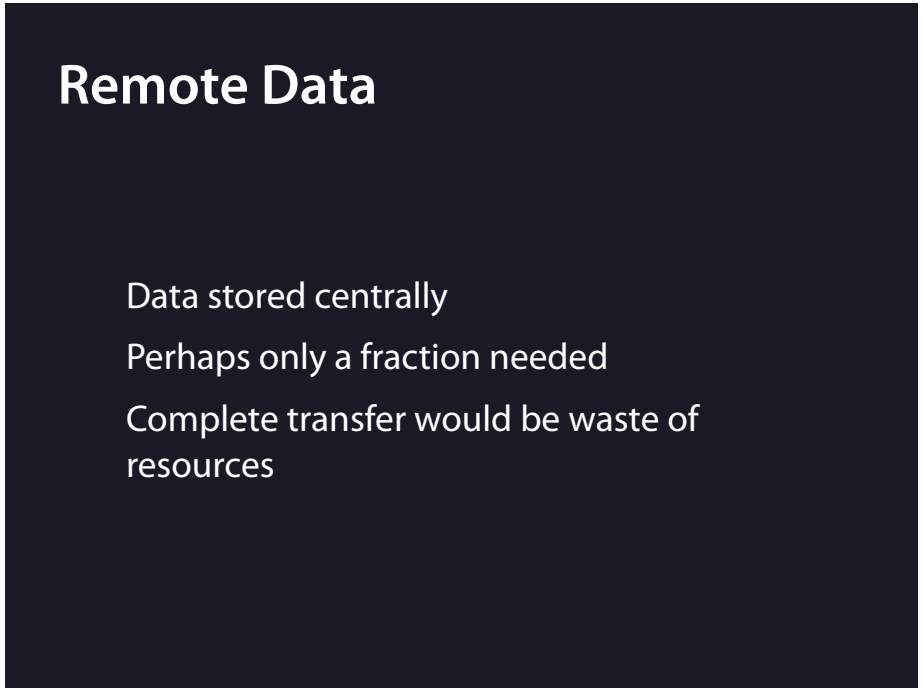


Already for algorithms that can be executed completely in main memory, it is valuable to analyze how they scale with the relevant data sizes.

For example time complexity: Marching cubes takes O(D) time to extract an isosurface, even if no triangle is generated. Changing the iso value may already be a slow operation. There exist algorithms that build a data structure in a preprocessing step, taking O(D log D) time. Afterwards the surface is extracted in O(sqrt(D) + F) time. Splitting algorithms in a batched preprocessing step to generate optimized data structures for visualization is more likely to be required.

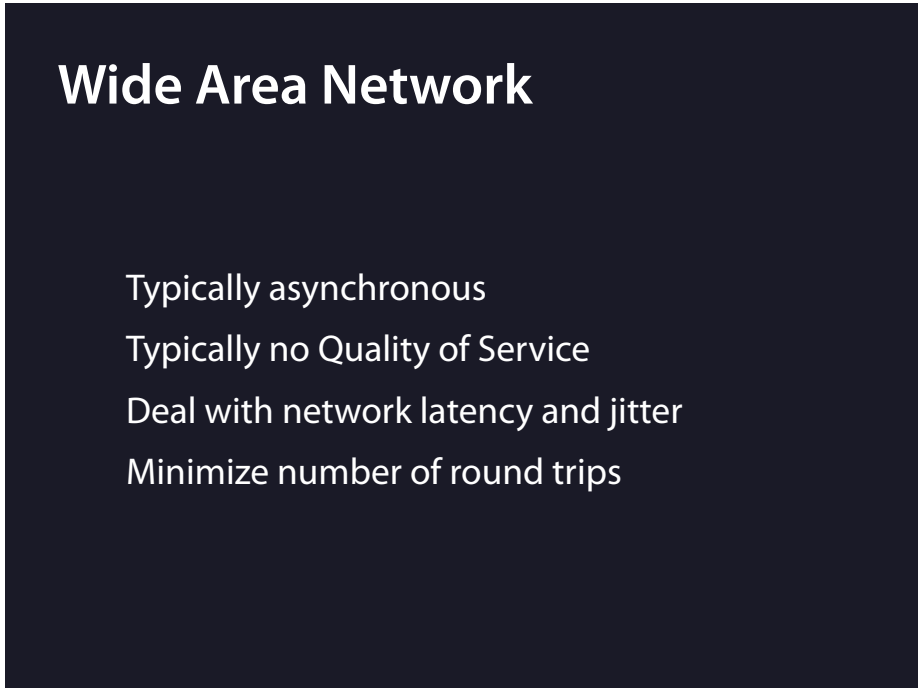
If data are larger than the available main memory, they must be loaded from disk to memory if required. As disk I/O is performed in blocks, it is important to store data that reduced the number of block I/Os. This is a general concept in the whole memory hierarchy (Disk, RAM, and CPU caches). Storing a volume e.g. as x-y slices allows fast access of these slices. But y-z slices may be extremely slow, as only one entry of a block is used. If data is stored as time steps, a single time step may be quickly accessed, but moving through many of them in a small sub volume may be unexpectedly slow.

If you achieved to handle all the large data if they are stored locally, you can think about accessing remotely stored data.



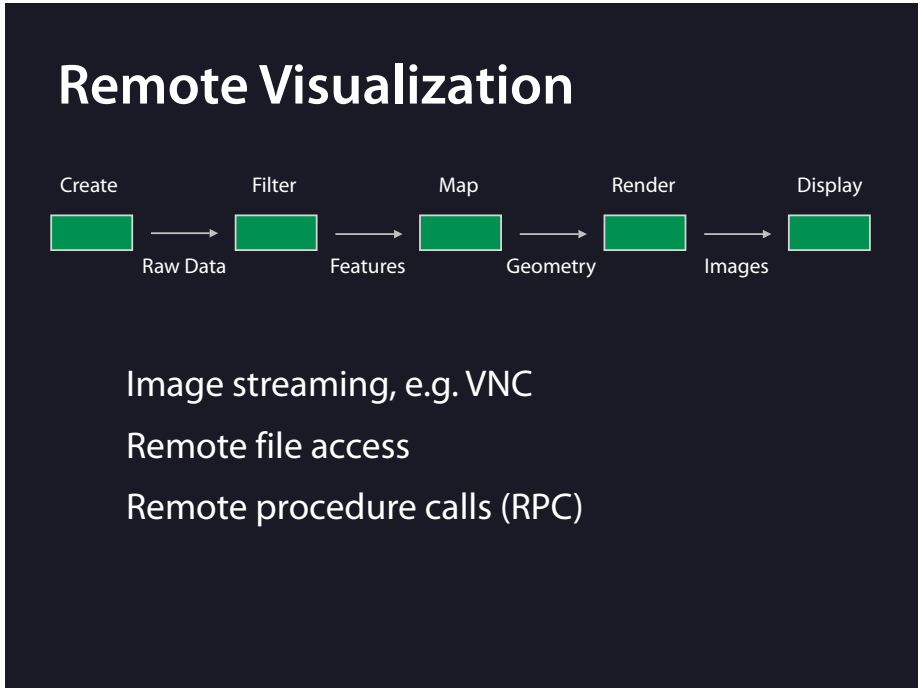
I put some reasons to do so on this slide.

Simulations are often run on central resources, and thus, the resulting data is also stored centrally. But for many tasks only a subset of the data is required and a complete transfer would be a waste of resources.



When I talk about remote access, I generally mean access through a wide area network, for example the Internet. Such a network typically does not provide Quality of Service. So you must deal with network latency and jitter.

A major design goal is to minimize the number of network round trips.



In principle various options are available for remote visualization.

You can split the visualization pipeline at different points.

Some examples are listed on the slide.

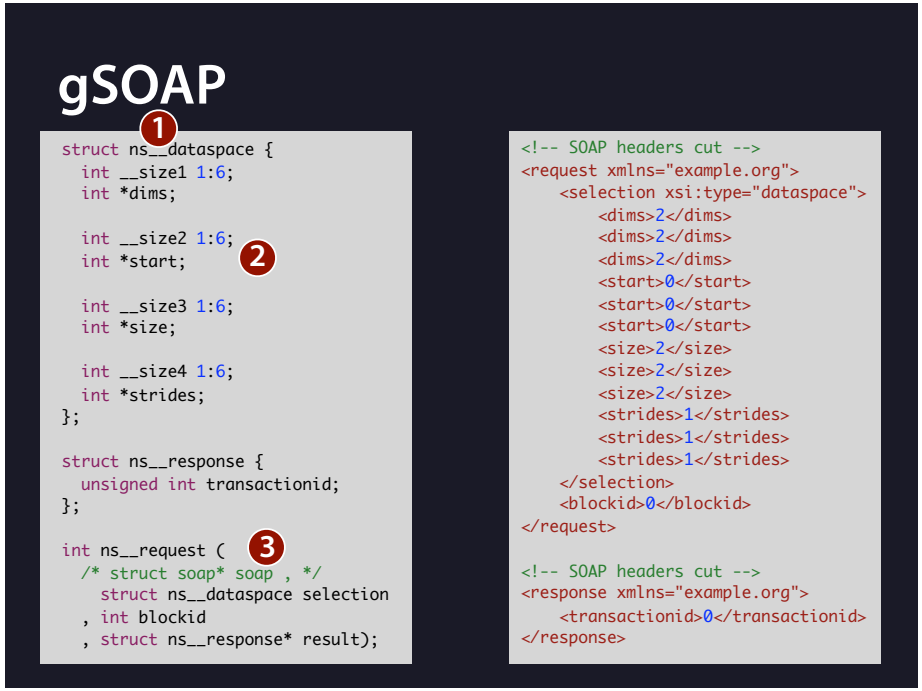
A Solution based on RPC

We use gSOAP.

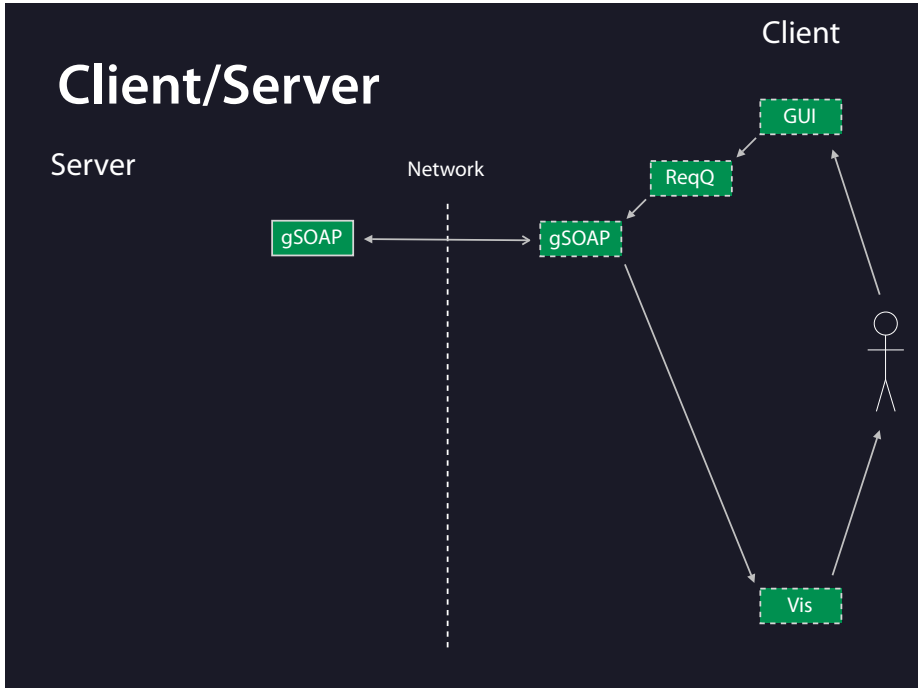
gSOAP

- Light weight SOAP server
- C-like declarations
- Preprocessor generates C-function decls and efficient SOAP parser (and XML Schema)
- Streaming, DIME, keep-alive
- www.cs.fsu.edu/~engelen/soap.html

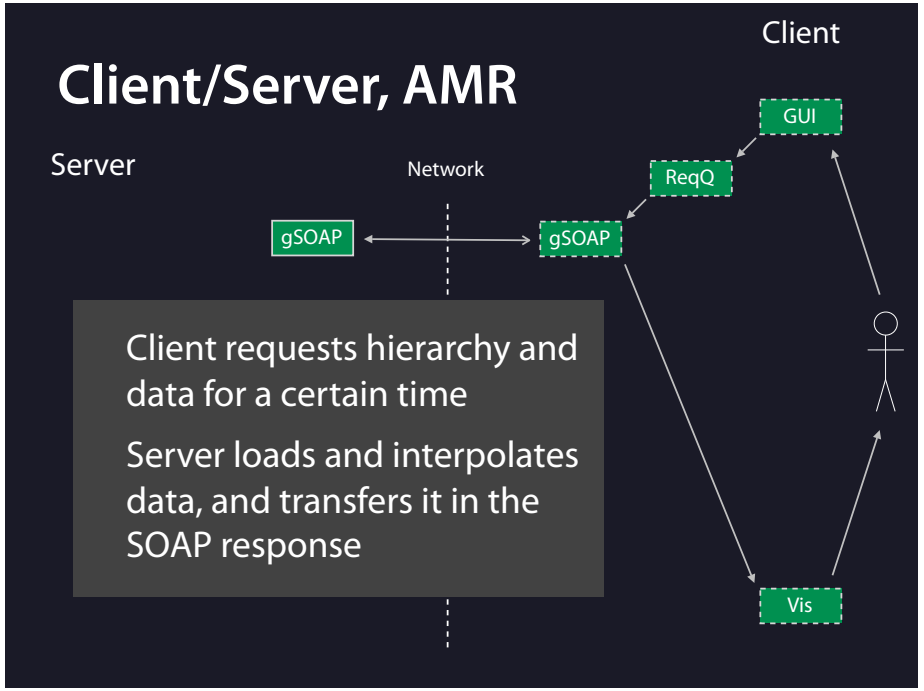
gSOAP provides a light weight SOAP server, which is automatically generated from C-like declarations. A preprocessor generates an efficient SOAP parser and C-function declarations that need to be implemented.

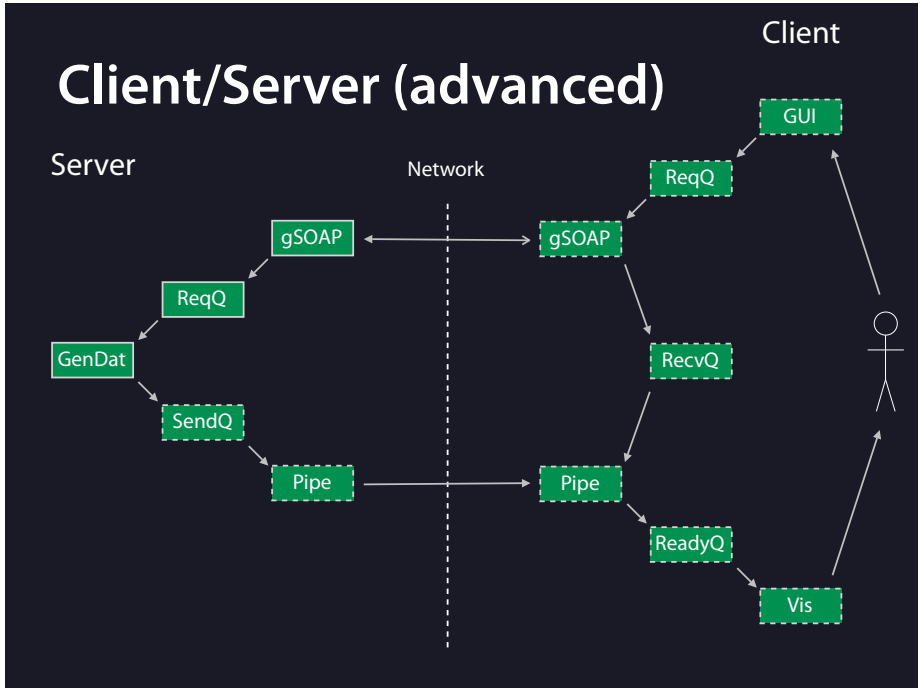


One small example to give a sense of this process.
On the left side, you can see the declaration of a SOAP call.
gSOAP handles namespaces by a simple naming convention. Two underscores in a C symbol are used to indicated a namespace, see (1). At (2), a dynamic array with a restricted size between 1 and 6 is declared. gSOAP handles all the memory allocation and bounds checking. A function for each call needs to be implemented. It is declared at (3). gSOAP adds a pointer to an internal data structure.
On the right hand side, part of the mapping to a SOAP request is shown. All these details are handled by gSOAP.

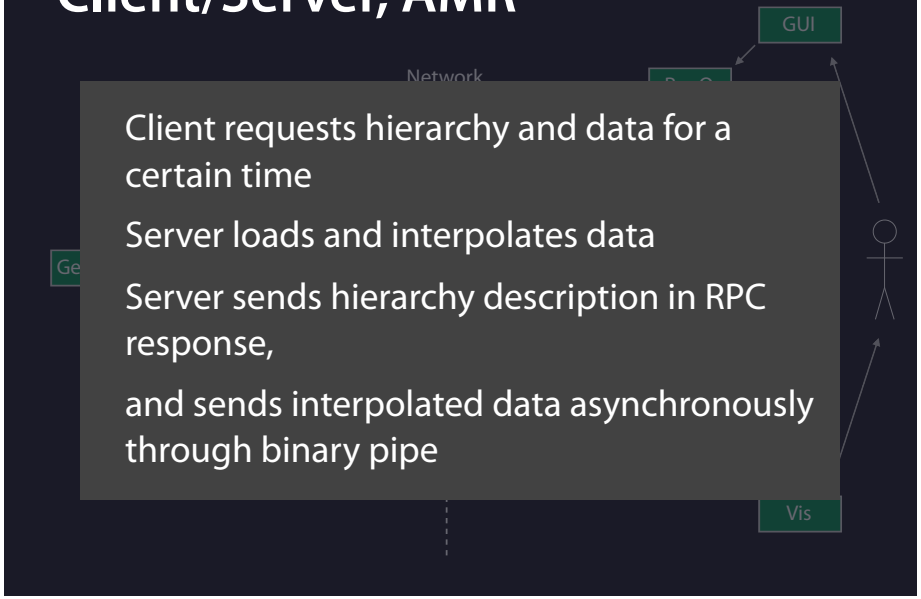

Client/Server
Client
Server
Network
GUI
ReqQ
gSOAP
gSOAP
Vis

Client/Server, AMR




Client
Client/Server (advanced)
GUI
Server
Network
ReqQ
gSOAP
gSOAP
ReqQ
GenDat
RecvQ
SendQ
Pipe
Pipe
ReadyQ
Vis

Client/Server, AMR



Client requests hierarchy and data for a certain time

Server loads and interpolates data

Server sends hierarchy description in RPC response,

and sends interpolated data asynchronously through binary pipe

Remote Procedure Call Paradigm allows easy and flexible distribution of existing solutions

Using high level operations minimizes number of network roundtrips

Separating bulk data transfer into binary stream guarantees responsiveness