AUSTRIAN GRID

A PROTOTYPE OF THE SEE-GRID PATHOLOGY FITTER

Document Identifier:	**AG-DA1c-3-2005_v1.doc (PUBLIC)**
Workpackage:	**A1c**
Partner(s):	**Research Institute for Symbolic Computation (RISC)** **Upper Austrian Research (UAR)**
Lead Partner:	**RISC**
WP Leaders:	**Wolfgang Schreiner (RISC), Michael Buchberger (UAR)**
Privacy:	**public**

Delivery Slip

	Name	Partner	Date	Signature
From	Károly Bósa	RISC	29.07.2005	
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
1.0	2005-07-28	Initial Version	See cover on page 3

A PROTOTYPE OF THE SEE-GRID PATHOLOGY FITTER

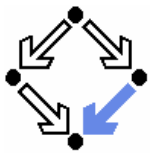
Karoly Bosa
Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz
{Karoly.Bosa, Wolfgang.Schreiner}@risc.uni-linz.ac.at

Michael Buchberger
Thomas Kaltofen

Department for Medical Informatics
Upper Austrian Research (UAR)
Thomas.Kaltofen@uar.at

August 29, 2005



Abstract

In the previous phase of the SEE-GRID project, we implemented the "SEE++ to Grid Bridge", via which normal SEE++ clients are able to access and exploit the computational power of the Austrian Grid.

This document discusses the theory and the design of a new functionality of the SEE-GRID system called pathology fitting and described its two simple parallelized versions. SEE-GRID is based on the SEE++ software for the biomechanical simulation of the human eye. SEE++ was developed in the SEE-KID project by the Upper Austrian Research and the Upper Austria University of Applied Sciences. SEE++ consists of a client component for user interaction and of a server component that runs various computations.

The pathology fitting algorithm in SEE-GRID uses the result data of the medical examination Hess-Lancastar Test as input and it is able to determine (approximately) the pathological cause of strabismus in case of a patient. This paper addresses the following issues:

- How the sequential pathology fitting algorithm works.
- How the sequential pathology fitting can be improved by using the results of the previous phase of the SEE-GRID project.
- How the pathology fitter can be parallelized.

At the end of this paper, we summarise our first experiences with the pathology fitting and make a plan for the further developments.

1 Introduction

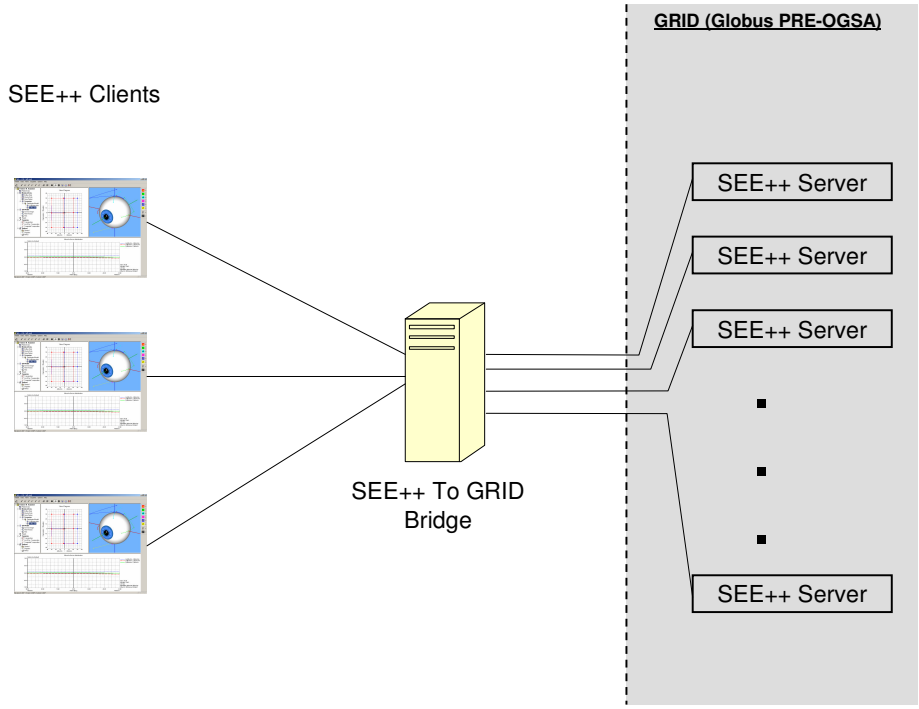


Figure 1: The Current Architecture of SEE-GRID

The design of SEE-GRID is based on the SEE++ software for the biomechanical simulation of the human eye. SEE++ was developed in the frame of the SEE-KID project by Upper Austrian Research and the Upper Austria University of Applied Sciences [SEE-KID, Buchberger 2004, Kaltofen 2002]; it consists of a client component for user interaction and of a server component that runs various computations ("currently "Hess Diagram Calculation").

In the previous phase of the SEE-GRID project [SEE-GRID Deliverable 2005], we implemented the "SEE++ to Grid Bridge", which is the initial component of SEE-GRID. Then we demonstrated how normal SEE++ clients are able to access via this bridge (see Figure 1) to the Austrian Grid and how a noticeable speedup can be reached in SEE++ — by applying simple data parallelism — by the exploitation of the huge computational power of the Grid.

The current phase of the project was mostly a studying period. Primarily we investigated the problem of the Pathology Fitting [SEE-GRID Design, 2004] and we implemented two initial grid based variations of it, see Section 2. Then we have got some experiences with these preliminary versions, see Section 4.

Since our original plan was to implement SEE-GRID as some kind of (web) service on the top of a Globus infrastructure [SEE-GRID Design, 2004], we started to investigate the new Web Service interfaces of Globus 4, too.

2 Pathology Fitting

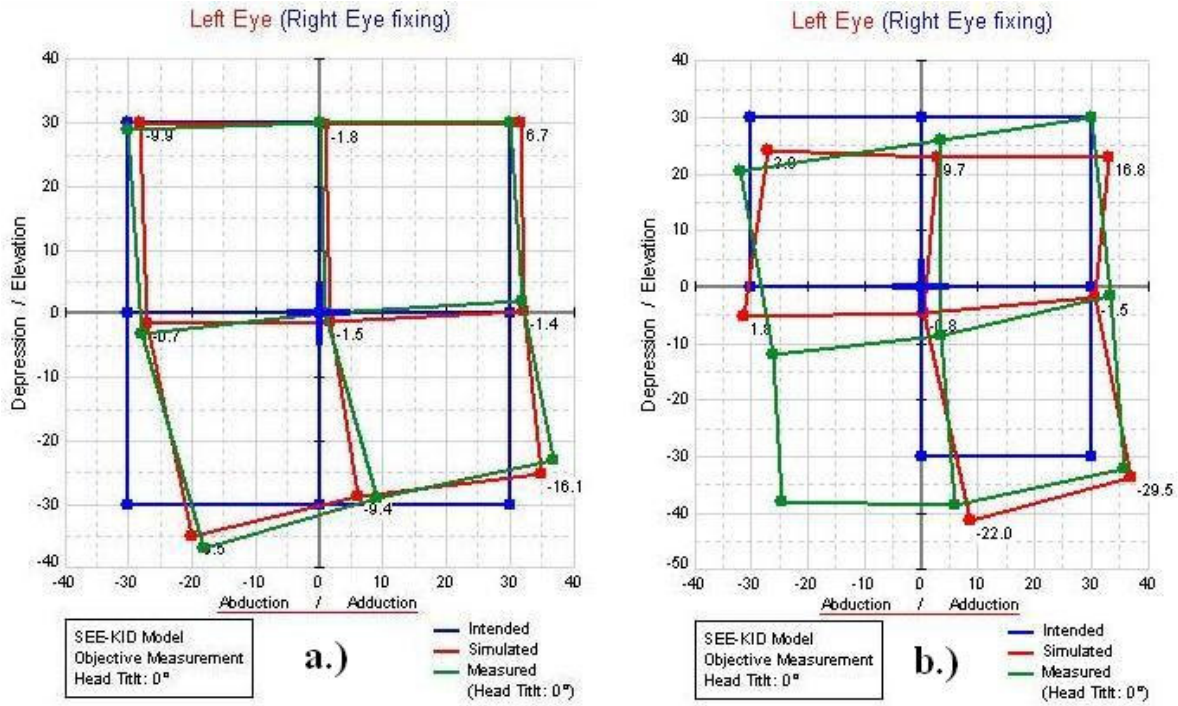


Figure 2: Gaze Patterns: Intended (blue lines), Measured (green lines) and Simulated (red lines)

SEE++ is able to simulate the result of the Hess-Lancaster test which is a test for binocular functions with separated images for both eyes [SEE-KID, 2004]. A gaze pattern (or "Hess Diagrams") which is the outcome this kind of examination is an eye pair's image of set of reference points in the plane. The most common kind of gaze patterns used by the software contains 9 points (see Figure 2). If a patient stares such a diagram with a perfectly healthy pair eyes, these points form a regular pattern (called intended gaze pattern) for her (see the blue lines on Figure 2). But if somebody has an impaired pair eyes, the seen pattern is distorted (see the green lines on Figures 2).

Basically, SEE++ deals with the calculation of a gaze pattern from an eye model (the "forward problem"). In case of the pathology fitting, we need to calculate an eye model from a gaze pattern (the "inverse problem"). As usual, the inverse problem is the mathematically more difficult and the computationally more time-consuming one.

2.1 Theory and Design

Pathology Fitting takes an initial eye model of one eye and gradually "improves" it (by modifying individual parameters or combinations of parameters) until the gaze diagram calculated from the eye model "nearly" matches the measured diagram of a patient. A model of one eye consists of the data of the eye globe (globe radius, cornea radius, etc.) and characteristic of the 6 eye muscles (see Figure 3).

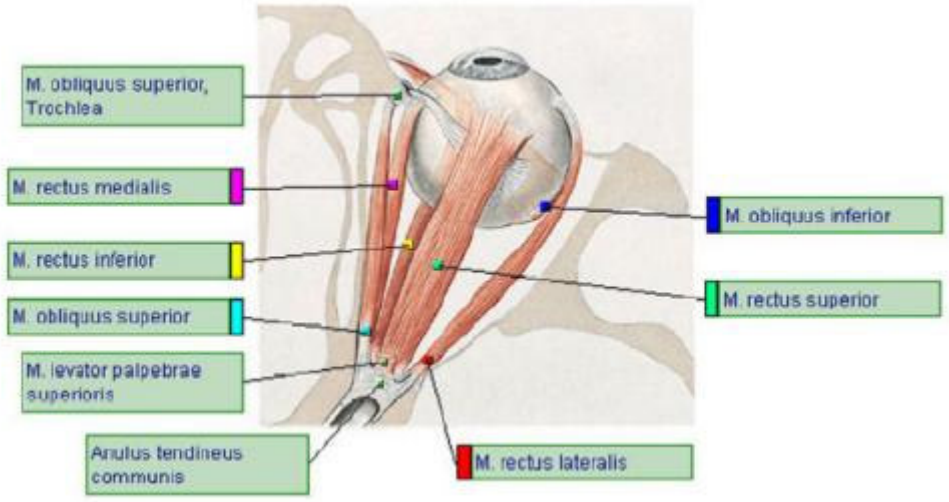


Figure 3: Top View of the Right Eye

The main effect (the direction of pull) of each muscle can be derived from its designation [Buchberger, 2004]:

- Superior rectus (upper straight eye muscle): upward,
- Inferior rectus (lower straight eye muscle): downward,
- Lateral rectus (outside straight eye muscle): sideways outward,
- Medial rectus (internal straight eye muscle): sideways inward,
- Superior oblique (upper diagonal eye muscle): downward and outside,
- Inferior oblique (lower diagonal eye muscle): upward and inside.

In the SEE++ system, all six eye muscles are simulated with a force model [Buchberger, 2004] and they are characterized with the following parameters:

- **Coordinates of the origin**. The origin is the point where the one of the ends of a muscle connects to some bones on back side of the orbita.

- **Coordinates of the insertion**. The insertion is the point where the other end of a muscle connects to the skin of the eye globe.

- **Coordinates of the pulley**. The pulley is the point where the string of a muscle elastically connects to a side of the orbita near to the insertion point. The main function of the pulley is to stabilize the pull direction of the muscle.

- The **Muscle length** represents only the contractile part of the muscle without its tendon.

- The **Tendon length** defines the summarized length of the tendons on the both ends of a muscle.

- **Innervation.** All eye muscles are controlled by some nerves. These contain the cell body of neurons which innervate the muscles and thus serve as a common path through which all eye movement control must be accomplished. In normal case, each eye muscle is innervated by approximately 1000 neurons and the

single neurons branch out in the eye muscle and innervate approximately 4 to 40 muscle fibers respectively. Roughly, the innervation is a number which shows how the muscle is supplied with nerves.

- The **Active strength** of a muscle results from activation (innervation) of a muscle initiated by the brain.

- The **Passive strength** of a muscle represents the flexible stretch characteristics of a muscle, which works opposite to the active strength.

The Pathology Fitter receives these muscle characteristics for all six muscles as an input parameter and starts to modify them in a specific order (except the origin and the pulley). The SEE-KID software essentially reduces by proper encoding the pathology fitting problem to a numerical optimization problem to which standard solutions can be applied. Unfortunately, there are several problems, which make the things more complicated:

- While an eye model uniquely determines a gaze pattern, the inverse does not hold, i.e. many eye models yield the same gaze pattern; furthermore, most of these eye models do actually *not* match the patient's real physiology.

- At the moment, it is very hard to define precisely in a formal way which eye model configurations are pathologically not possible (except some special cases, e.g.: the insertion must not be located on the cornea, etc.).

- There is no general rule yet that can define in which order the muscle parameters have to be modified (fitting strategy). In different situation, different strategies can be applied that may results different eye models. Therefore, in the current versions, the user always denotes by a "parameter selection" S those parameters in a specific order that may be modified. For instance such a possible strategy could be the following: 1. Innervation, 2. Muscle length, 3. Tendon length, 4. Horizontal insertion and 5. Vertical insertion.

The pathology fitting algorithm always receives the models of fixing eye, following (or pathological) eye and reference eye [SEE-GRID Design, 2004], a matrix M with the coordinates of the points of the measured gaze pattern and the fitting strategy S chosen by the user. The rough sketch of the algorithm is the following:

pathologyFitting($E_{following}$, E_{fixing}, $E_{reference}$, M, S)

```
E1 := E_following
C1 := gazePattern(E1, E_fixing, E_reference)
if C1 matches M return E1

loop
        p := nextParameterVariation(S)
        if p is equal to NULL return E1
        E2 := apply(p, E1)
        C2 := gazePattern(E2, E_fixing, E_reference)
        if C2 fits M better than C1
            E1 := E2
            C1 := C2
            if C1 matches M return E1
```

Of course, a measured gaze pattern and a simulation one almost never match exactly. There are two important consequences of this:

- The algorithm does not terminate most of the case before it tries to fit all the possible kinds of muscle parameters given in the fitting strategy.
- If the algorithm returns an eye model, it cannot be decided from the comparison of the measured and the simulated gaze patterns, whether a proper fitting strategy was applied.

3 Pathology Fitting with Parallel Gaze Pattern Calculation

As it can be seen from the sketched algorithm above, the gaze pattern calculation is called several times during a pathology fitting process. Hence, we intended to combine the existing sequential version of the pathology fitting algorithm and the parallel gaze pattern calculation (developed in the previous phase of the project) and to test it.

The implementation of the algorithm did not require the modification of the existing communication protocol. We simply reused the existing SOAP messages described in [SEE-GRID Deliverable 2005]. This means the followings:

- When a gaze pattern calculation is triggered by the fitting process, a new thread *T* is created. While the new gaze pattern is calculated the pathology fitter is blocked (because the last calculated gaze pattern of the last created eye model is always used for the creation of a next/new eye model).
- *T* sends a message **Calculate_Binocular_Test(*E1, E2, E, I*)** [SEE-GRID Design, 2004] to the "SEE++ to Grid Bridge" where *E1, E2, E* define an eye model (fixing eye, following eye, reference eye) and *I* is a matrix which determines the points of the gaze pattern used on the client. The "SEE++ to Grid Bridge" splits *I* and distributes the subsets of *I* to SEE++ servers running some Grid site.

- *T* uses the *session_Id* received as an answer for Calculate_Binocular_Test message and sends a message **Poll_Status(*session_Id***) [SEE-GRID Design, 2004] to the "SEE++ to Grid Bridge" from time to time. The bridge collects the status information of the gaze pattern calculation from the corresponding SEE++ servers then it summarises and returns them to the client.

- If *T* receives an aswer "calculation_terminated" for the last Poll_Status message, it sends a message **Poll_Result(*Allocation_id***) [SEE-GRID Design, 2004] to the "SEE++ to Grid Bridge". The bridge collects the calculated parts of the gaze pattern from the SEE++ servers and assembles them and returns with the complete calculated gaze pattern.

- If *T* receives the gaze pattern, it sends a signal to the blocked pathology fitter and terminates. The fitter takes the gaze pattern and resumes its execution (with the comparison of the calculated and measured gaze patterns).

4 Parallel Pathology Fitting

We also wanted to develop a simple parallelized version of the pathology fitting algorithm independently from the previously mentioned implementation. For achieving this, we separated the pathology fitting algorithm from the SEE++ client and extended the communication protocol with some new SOAP messages (see Section 4.1).

For parallelizing the algorithm, we exploited that the pathology fitting algorithm always checks before the calculation of a new eye model, whether the values of muscle parameters of the last calculated eye model are located within some domains (that are typical for the kinds of the corresponding muscle paramters). By this, the pathology fitter filters out the obviously pathological impossible muscle parameter values. If such a value is outside the corresponding domain, then the value is set to the closest bound value of the domain.

In the parallel version, an arbitrary muscle from the six eye muscles is chosen first (in the current version, this muscle is the "Lateral Rectus"). Before the algorithm starts the fitting procedure, it takes the first element of the current fitting strategy (which is a kind of muscle parameter e.g. muscle length, active strength, etc.) and divide the domain of this kind parameter of the previously chosen muscle to distinct subsets. Each subdomain is assigned to a SEE++ server. By this, we managed to distribute the search space between the servers, since each server can take values for this parameter of the choosen eye muscle only from the subdomain assigned to the server.

4.1 New Messages in the SOAP Protocol

For establishing the Grid based version of the pathology fitter, we added five new messages to the SOAP communication Protocol (every two-way/blocking message is executed in an independent thread) :

- **Two-Way Message**

  Request: **PathologyFitter (E, M, fixingMode, S)**
  Answer: **sessionID**

  For triggering the pathology fitting, the SEE++ client issues this message, where *E* is the eye model, *M* is the measured gaze pattern, *fixingMode* specify which eye the fixing eye and which is the following eye and at last *S* is the fitting strategy.

  If a SEE++ server receives such a message it starts the pathology fitting algorithm in a different thread and returns an identifier *sessionID* which identifies the computation on the server.

  If the "SEE++ to Grid Bridge" receives such a message, it allocates some SEE++ servers which are running on some grid sites (similarly to the case of the parallelized gaze pattern calculation [SEE-GRID Deliverable 2005]). It sends a message ParallelPathologyFitter (E, M, fixingMode, S, n, fitterID) to each allocated server. The "SEE++ to Grid Bridge" returns an identifier *sessionID* which identifies the computation.

- **Two-Way Message**

  Request: **ParallelPathologyFitter (E, M, fixingMode, S, n, fitterID)**
  Answer: **sessionID**

  In the parameter list, *E,M, FixingMode and S* are the same as in the parameter list of PathologyFitter message. *n* is the number of the SEE++ servers allocated for this pathology fitting calculation (this value is given by the user as a command line argument of the "SEE++ to Grid Bridge", see Section 4.2.1). *fitterID* is an integer number which is greater than or equal to 0 and less than *n* and which is uniquely identifies the server within the current pathology fitting calculation.

  If a SEE++ server receives such a message, it can easily determine from *n* and *fitterID* those subdomain of the corresponding muscle parameter of the "Lateral Rectus" which is assigned to this server. Namely, it divides the corrensponding domain to *n* subdomain and it uses only the subdomain determined by the *fitterID*.

  Then the server starts the pathology fitting algorithm in a different thread and returns an identifier *sessionID* which identifies the computation on the server.

- **Two-Way Message**

  Request: **FitterPollStatus (sessionID)**
  Answer: **terminated, goodness**

  After a PathologyFitter message is sent, the SEE++ client issues this message from time to time with *sessionID* received as an answer for PathologyFitter message.

If the "SEE++ to Grid Bridge" receives such a message, it forwards it to the corresponding SEE+ servers.

If a SEE++ server receives such a message, but the corresponding fitting process is not terminated, then it returns a false *terminated* value and an undefined *goodness* value. If the fitting process is done, then the value of the *terminated* is true and value of the *goodness* is a number which shows the difference between the measured gaze pattern and the last simulated one on this server.

The "SEE++ to Grid Bridge" collects the *goodness* values from the server and compares them. If it find the smallest one, it sends a message GetModifiedSimulation (*sessionID)* to the server from where smallest value arrived. It also send RemoveModifiedSimulation(*sessionID)* to all the other servers allocated for the current pathology fitting calculation.

- **Two-Way Message**

  Request: **GetModifiedSimulation (sessionID)**
  Answer: **E'**

  If a SEE++ client receives true *terminated* value as an aswer for the last sent FitterPollStatus message, then it send such a message to the "SEE++ to Grid Bridge".

  If the "SEE++ to Grid Bridge" receives such a message, it sends back the calculated eye model which is received from the server from where smallest *goodness* value arrived.

  If a SEE++ server receives such a message it returns with the corresponding calculated eye model and then removes it.

- **One-Way Message**

  Request: **RemoveModifiedSimulation (sessionID)**

  When a SEE++ server receives such a message, it simply removes the corresponding calculated eye model

4.2 User Manual

4.2.1 The "SEE++ to Grid Bridge"

This manual is only an extension of the one that is described in [SEE-GRID Deliverable 2005]. The user is able to start the executable called "seepp2grid" with some parameters. The only parameter that has to be given compulsory is the fully qualified domain name of the grid site on which "seepp2grid" program will start a/some SEE++ server(s):

```
seepp2grid [-help] [options…] GRIDSITE
```

The following new parameter can be used for the "seepp2grid":

`-dist_fitter` or `-f` : To number of the parallel branches of the search tree.

For instance, the command

```
seepp2grid -n 9 -g 1 -f 5 -path /home/local/agrid/ag10022
altix1.jku.austriangrid.at
```

starts 9 processes of SEE++ server located in the directory "/home/local/agrid/ag10022"on the Austrian Grid site "altix1.jku.austriangrid.at". Moreover, if a Pathology Fitting is triggered on a SEE++ client, the bridge splits the calculation to 5 parallel branches and sends further 5 server processes.

4.2.2 The SEE++ client

Currently, the Pathology Fitter is not included in the latest official version of SEE++. Therefore, we use only a temporary GUI interface (a dialog box) for testing purpose that can be reached from the menu "Help" in SEE++. This dialog box contains two buttons one for fitting the left eye and one for fitting the right eye.

5 Implementation Status and Benchmarks

5.1 Changing to Globus Web Service Interface

In the original design document [SEE-GRID Design, 2004], we proposed to use the Web Service (WS) interface of the latest stable release of Globus [Globus, 2004] (Globus 3.2) in order to connect SEE++ to the Austrian Grid. Unfortunately, the WS part of Globus was not integrated into the software specification of Austrian Grid at the very beginning. Therefore, we had to modify our conception and to change our design to the pre-WS interface of Globus.

Since the new software specification document of the Austrian Grid [AGRID New S. Spec., 2004] has already involved the complete Globus Toolkit 4 (GT4) (with the implementation of a reworked Web Service interface, too), we started to investigate Web Service part of GT4.

The Globus Toolkit 4 includes three more or less complete implementations (e.g.: C implementation does not contain notification management yet) of the Web Service Resource Framework (WSRF) specification: on Java, on C and on Python implementations. It also contians thee kinds of runtime environment (WS container) for the WS services which are implemented on three mentioned programming language. WSRF was developed by [OASIS] and specifies roughy an extension of Web Services with useful standardised features (e.g.: stateful services, resources, notification, lifetime managements, etc).

It was suprising for us that the GT4 still does not support simple users to deploy (grid) applications as web services. If somebody intends to do this, she needs to have special rights on each grid site for writing to the directory '$GLOBUS_LOCATION' and (in case of the C WS Core) compiling the source code of the application into the globus code. This is even true in the case of the java command line tool 'globus-deploy-gar'.

Most of the new Globus services were implemented in Java *on top of WSRF* implementation "Java WS Core" (the toolkit still includes the pre-*WS components*). Our plan is to extend the "SEE++ to Grid Bridge" (which is implemented in C) in the near future such that it will be able to submit a grid job vi the WS-GRAM and to use the features of this new kind of infrastructure of Globus 4.

Unfortunatelly, we found that the currently avialable documentation about WS-GRAM is not complete yet and not enough for implemeting a client in C (the only complete and available example written in C is the source code of the command line tool 'globusrun-ws' — so, it should be debugged…).

5.2 Benchmarks of Pathology Fitting with Parallel Gaze Pattern Calculation

For testing the speedup of the pathology fitter extended with parallel gaze patter calculation, we took two real pathological gaze patterns and compared the execution time of the fitting procedure in different circumstances. We applied the default gaze pattern in both test cases (with 9 points, see Figure 2). We chose the left eye as the pathological eye (following eye) and we always tried to fit the eye model parameters in the same order (fitting strategy):

1. Active strength of the eye muscles,
2. Passive strength of the eye muscles,
3. Length of the eye muscles,
4. Length of the eye muscle tendons,
5. Horizontal insertion of the eye muscles on the eye globe,
6. Vertical insertion of the eye muscles on the eye globe.

We performed two kinds of tests: local tests within our own network and grid-based tests across the Internet. In the local tests, we used an AMD Dual Opteron 1.6Ghz. The grid-based tests were executed on the Austrian Grid site altix1.jku.austriangrid.at, which contains 64 pieces of Ithanium processors. In the second case, we investigated the effectiveness of the parallelism in different situations where 1, 3 or 9 processes of the SEE++ server are started and the maximum number of the gaze pattern points which are sent together to one process (*granularity value*) is not limited, 3, 2 or 1. Each value located in the following tables is the median execution time of 5-7 executions.

Machine Name	Dual Opteron	Altix 350 (altix1.jku.austriangrid.at)		
Server processes / Max. number of points sent together	1/all	1/all	3/3	9/1
1. Test Case (see Figure 2a)	6min 07.45s	3min 29.03s	2min 35.77s	2min 10.89s
2. Test Case (see Figure 2b)	5min 51.36s	3 min 17.52s	2min 19.81s	1min 58.83s

Table 1: Benchmark Results

The measured gaze pattern for the first test case can be seen on Figure 2a (green lines). In this case, the main pathological problem is that the Superior Rectus Muscle connected to the patient's left eye does not have enough (active and passive) strength. The Gaze pattern calculated from the simulated pathology, which can also be seen on Figure 2a (red lines), is nearly the same.

In the second test case (the green lines on Figure 2b), the pathological situation is completely different. The insertion and the length of Inferior Rectus and the Lateral Rectus Muscles are abnormal. As it can be seen on Figure 2b (red lines), the simulated pathology does not correspond to the real situation, because we applied a wrong fitting strategy for it.

From the measurement results it can be seen, however the current version of the pathology fitter is sequential and it is always blocked while the next gaze pattern is calculated, we could reach some speedups by applying some simple parallelizations in the gaze pattern calculation.

5.3 Implementation Status of Parallel Pathology Fitting

Since our next milestone is at the end of August 2005, the implementation of the parallel pathology is still in the debugging phase. Hence, we cannot provide test experience and benchmarks data about it.

6 Further Development Steps

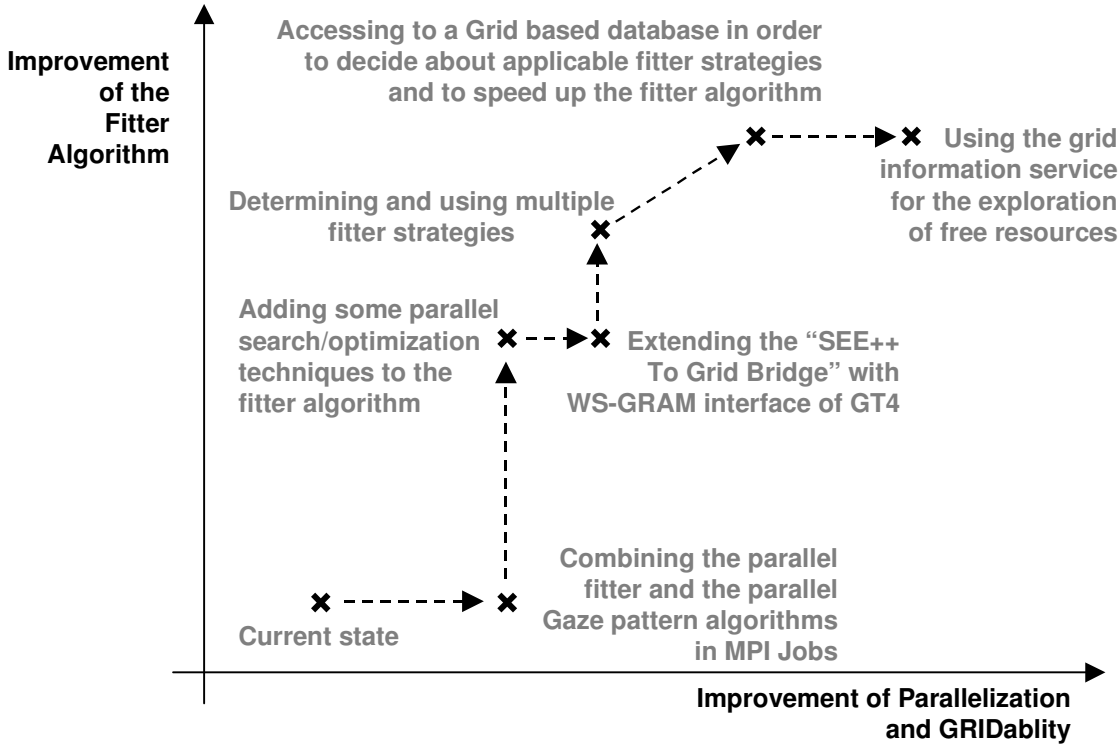


Figure 4: Directions of Further Developments

In the next steps, we intend to improve the grid based implementation of the pathology fitting (see Figure 4). First of all, we would like to combine the parallel pathology fitting and parallel gaze pattern calculation by usimg MPI.

Then, we also would like to improve the current sequential algorithm of the parameter fitting with some parallel and distributed variants of the “branch and bound” optimized for grid computing [Filho et al, 2003, Aida et al, 2003, Aida Osumi, 2005].

We also want to finish and test the extension of the "SEE++ to Grid Bridge" with the WS-GRAM interface.

We plan to generalize the pathology fitting algorithm such that it could determine the proper fitting strategy automatically in every case. For this, either we may introduce some kind of precalculation phase into algorithm or we will use a large representative grid-based database for storing and searching for the pairs of measured gaze pattern and already simulated pathologies.

In the future, we may also deal with the development of the automatic exploration of available/free “SEE++ grid services” on the whole grid infrastructure. For achieving this, we intend to use the information provided by the information service of the Austrian Grid.

References

[Aida et al, 2003] Kento Aida, Wataru Natsume, Yoshiaki Futakata. *Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm*, 3rd International Symposium on Cluster Computing and the Grid, May 12 - 15, 2003, Tokyo, Japan. http://csdl.computer.org/comp/proceedings/ccgrid/2003/1919/00/ 19190156abs.htm

[Aida Osumi, 2005] Kento Aida, Tomotaka Osumi, "A Case Study in Running a Parallel Branch and Bound Application on the Grid," Proc. IEEE/IPSJ The 2005 Symposium on Applications & the Internet (SAINT2005), pp.164-173, Feb. 2005.
http://www.alab.ip.titech.ac.jp/papers/saint2005.pdf

[AGRID New S. Spec., 2004] *Austrian Grid Software Specification,*
http://www.austriangrid.at/austriangrid/internal/deliverables/docs/WP_I-1/2005/AG-D-I1-2-v0.5.doc

[Buchberger, 2004] Michael Buchberger. *Biomechanical Modelling of the Human Eye*.
Ph.D. thesis, Johannes Kepler University, Linz, Austria, March 2004.
http://www.see-kid.at/download/Dissertation_MB.pdf

[Ethereal, 2004] Ethereal Network Protocol Analyzer. http://www.ethereal.com

[Filho et al, 2003] J. Viterbo Filho, L. M. A. Drummond, E. Uchoa e M. C. S. Castro. *Towards a Grid Enabled Branch-and-Bound Algorithm*. Report RT-05/03, Department of Computer Science -- Fluminense Federal University, 2003.
http://www.ic.uff.br/PosGrad/RelatTec/reltec03.html

[Globus, 2004] The Globus Tookit. http://www.globus.org/toolkit/

[glogin, 2004] Herbert Rosmanith and Jens Volkert "glogin - Interactive Connectivity for the Grid" in: Z. Juhasz, P. Kacsuk, D. Kranzlmüller, "Distributed and Parallel Systems - Cluster and Grid Computing", Proc. of DAPSYS 2004, 5th Austrian-Hungarian Workshop on Distributed and Parallel Systems, Kluwer Academic Publishers, Budapest, Hungary, pp. 3-11 (Sept. 2004). http://www.gup.uni-linz.ac.at/glogin/

[gSOAP, 2005] gSOAP 2.7.0 User Guide, 2005. http://www.cs.fsu.edu/~engelen/soap.html

[Kaltofen, 2002] Thomas Kaltofen. *Design and Implementation of a Mathematical Pulley Model for Biomechanical Eye Surgery*. Diploma thesis, Upper Austria University of Applied Sciences, Hagenberg, June 2002. http://www.see-kid.at/download/Pulley_Model_Thesis.pdf

[OASIS] OASIS, http://www.oasis-open.org

[SEE-GRID Design, 2004] Karoly Bosa, Wolfgang Schreiner, Rebhi Baraka, Michael Buchberger, Thomas Kaltofen, Daniel Mitterdorfer. *SEE-GRID Design Overview*. Austrian Grid Deliverable A1c-1, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, November 2004.

[SEE-GRID Deliverable 2005] Károly Bósa, Wolfgang Schreiner, Michael Buchberger, Thomas Kaltofen. *The Initial Version of SEE-GRID*. Austrian Grid Deliverable A1c-1-2005, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, March 2005.

[SEE-KID, 2004] SEE-KID home page, 2004. http://www.see-kid.at

[SOAP, 2003] SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624

[WSRF Comparison, 2005] M. Humphrey, G. Wasson, K. Jackson, J. Boverhof, M. Rodriguez, J. Gawor, S. Lang, J. Bester, I. Foster, S. Meder, S. Pickles, and M. McKeown. *State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations. 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14),* Research Triangle Park, NC, 24-27 July 2005. http://www.cs.virginia.edu/~humphrey/papers/WSRFComparison2005.pdf