Software Engineering Project “Bugfind”
Software Requirements Specification

Project group
Perttu Haimi, Petteri Hintsanen, Philipp Masche, Toni Ruokolainen and Osku Salerma

Instructor
Peter Lamberg

Helsinki 9th May 2001
Software Requirements Specification
UNIVERSITY OF HELSINKI
Department of Computer Science

Version history

- v0.1, 28.1.2001 (first draft)
- v0.2, 31.1.2001 (more explicit version)
- v0.3, 5.2.2001 (more details)
- v0.35 7.2.2001 (cleaning up, adding stuff)
- v0.5 14.2.2001 (hopefully almost done)
- v0.6 15.2.2001 (fixed errors found in the meeting)
- v0.9 20.2.2001 (proposal version for freezing)
- v1.0 21.2.2001 (frozen)
- v1.1 11.3.2001 (minor corrections after client review; also includes some requirements found during the design phase)

Contents

List of Figures

1 Preface

Viruses are the simplest lifeforms, so simple that it is often argued whether they really are "alive" in the general sense of the word. Viruses cannot replicate on their own, without the metabolic machinery of their host organism. But because of their (relative) simplicity, they are good objects for a biological study.

In biology many important questions concern mechanisms of various phenomena, for instance how DNA replication (copying) is done in the cell. Also, if more of the mechanism of a disease is known, it is often possible to design more efficient treatments to the disorder. One way to approach the problem is to examine the structure of the pieces of the puzzle. The structure may give useful clues of the functions and mechanisms.

Because of their small size, making a 3D model of a virus is not easy. They are not visible in conventional microscope, since they are several times smaller than the wavelenght of visible light. On the other hand, they are very big when compared to proteins and other macromolecules, whose structure can often be studied with X-ray chrystallography. One method to study virus structure is electron microscopy (EM).

The virus sample is mixed with water, and then frozen very quickly to prevent the formation of ice chrystals, which would disturb the natural structure of the particles. In electron microscope, the sample is then bombarded with an electron beam.

Since EM produces only 2D projections of the virus particles, the 3D model must be calculated from these projections. This is routinely done in tomography, but then the angles of the projections are known in advance. Virus particles, on the other hand are (more or less) randomly orientated in the sample. Other complication is the fact that the pictures are very noisy. The ice which surrounds the particles is almost as permeable to electrons as the viruses, which makes the contrast of the images poor.

Reconstruction process is computationally intensive task. For good resolution, hundreds or thousands of 2D projections (EM images of viruses) are needed [L+98]. Image files are typically large, in the order of tens of megabytes, and contain several hundreds to thousends of particles.

2 Description of the current system

The current system uses *SPADE* [K+00] and *ETHAN* [Kiv00] programs for automatic detection of spherical particles from electron micrographs (see Figure 1). SPADE is an interactive graphical program which uses ETHAN to do the actual detection of particles. Both programs currently read micrographs in *mrc* and *pgm* format and write detected virus images in mrc format. The programs are result of the colloboration between The Bamford Laboratory and Department of Computer Science at the University of Helsinki. The very first version of SPADE was made as a software engineering exercise by T. Kaituri, E. Kallioniemi, J. Paaso, M. Teräs and J. Virtanen. The current version is mostly written by Anatoly Verkhovsky. ETHAN is written by Teemu Kivioja.

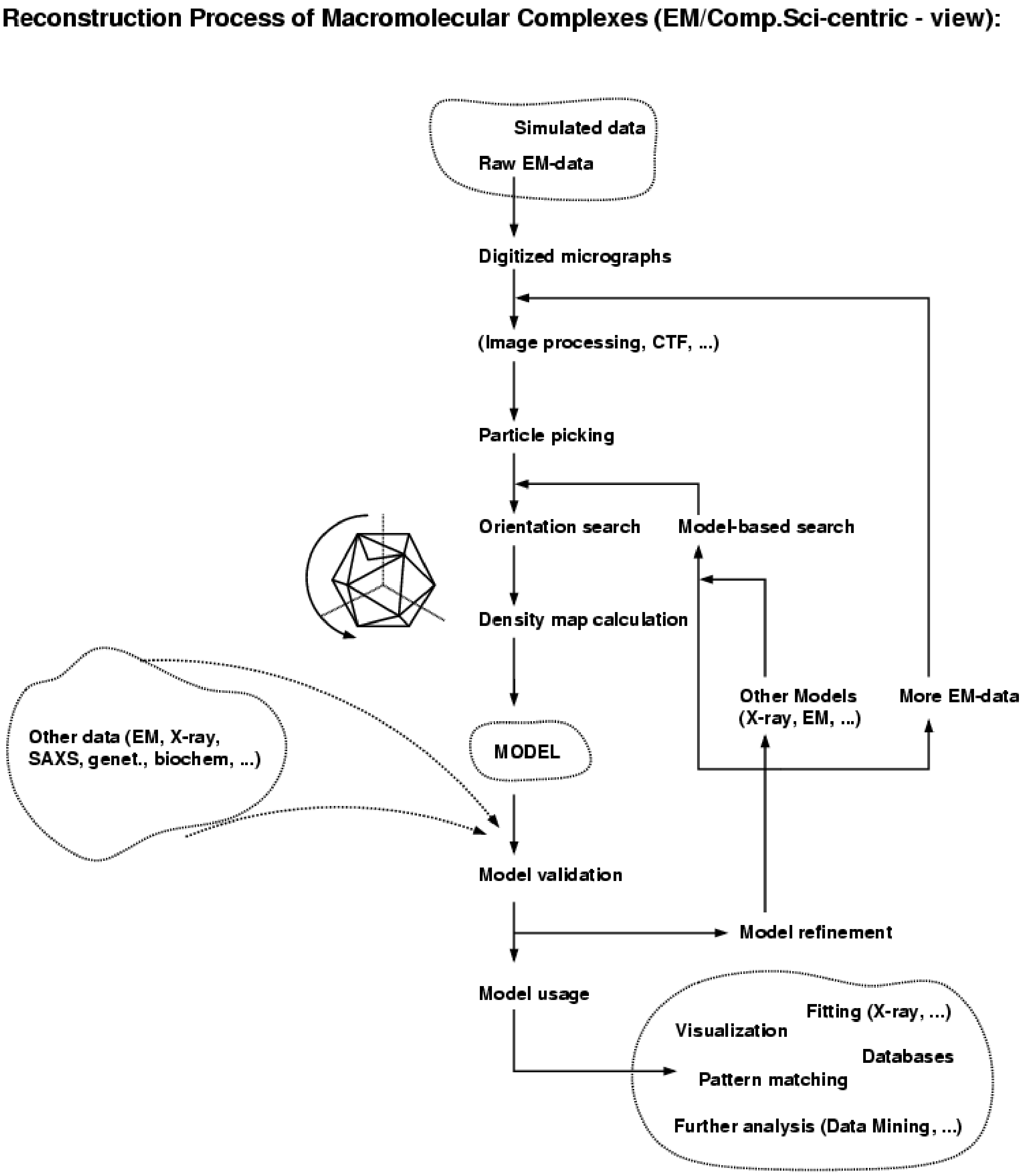

Figure 1: Reconstruction of macromolecular complexes

3 Overall Description

3.1 Software functionality

The task of software engineering project "Bugfind" is to make a program capable of recognizing asymmetric particles from the cryoelectron microscopy images. Program will use a "draft" model, from which it makes projections and compares them to the EM-images.

The intention of reconstruction process is to make a 3D model of a specific macromolecular complex (virus, ribosome, protein complex, etc.) from 2D projections taken from electron micrographs. The reconstruction process involves very demanding tasks computationally and thus parallel computing will be used.

The reconstruction process can be divided to subtasks (see Figure 1):

- Image preprocessing. This includes Gaussian filtering to achieve better signal-to-noise–ratio and maybe some additional computation.
- Particle picking. EM slides may contain several thousands of particles. Hence it is necessary to have an automatic method for choosing the best candidate areas for further analysis.
- Determining of particle orientations.
- Calculating the 3D model.

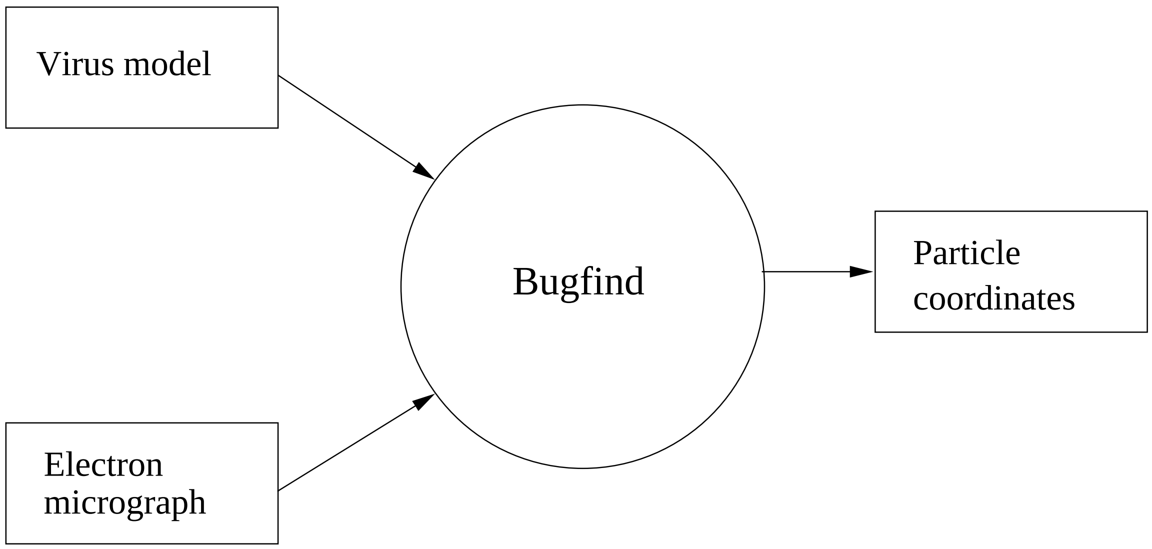

Figure 2: High level abstraction of the system

The software produced by this project will do the particle picking and give some information about their orientation. The current system can find projections of symmetrical viruses; the goal for this project is to design and implement a software that is capable of recognizing asymmetrical macromolecular complexes. This software uses a 3D model of a complex which is then projected to two dimensions in different angles. These two-dimensional projections are used to find the macromolecular complexes from the electron micrograph. The general view of the program can be seen in Figure 2.

3.2 Software platform

The software will be implemented in *ANSI-C* using the *MPI*-standard [MPIa] the for distribution implementation. We will use the *MPICH*-library [MPIb]. The *GCC* [GCC] and *MPICC* compilers will be used. The software will run in Beowulf [Beo] cluster with 11 Linux machines.

3.3 Software users

The customers and users for this project are Janne Ravantti and the Bamford lab at the Viikki Biocenter. The software will be distributed under the GNU General Public Licence.

4 Information Description

4.1 Information content description

Information content in this project will consist of *2D micrographs*, *3D virus models* and user-provided *configuration information*. Both the 2D and 3D files are huge in size: the micrographs approximately 170 MB/file and the 3D models about 50 MB/file.

The input data is in the form of unsigned char. Data structure for both the model and the micrograph will be a large array. The data type of the elements may be either unsigned char, float or real (as defined in the fourier transform library). The actual choice depends on performance, space requirements, algorithms and ease of implementation issues. The 3D model and the micrographs are in mrc format and their data is provided in unsigned char format.

4.2 Information flow description

4.2.1 Data flow

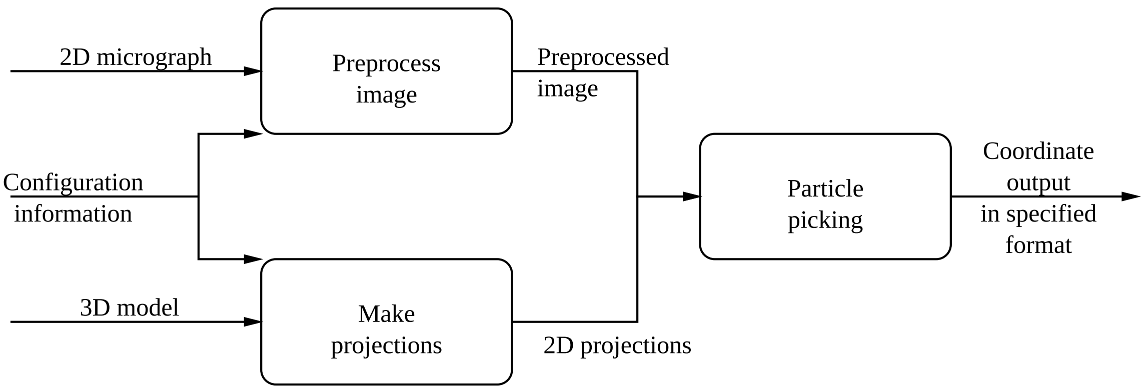

Figure 3: Data flow diagram

See Figure 3.

4.2.2 Control flow

See Figure 4.

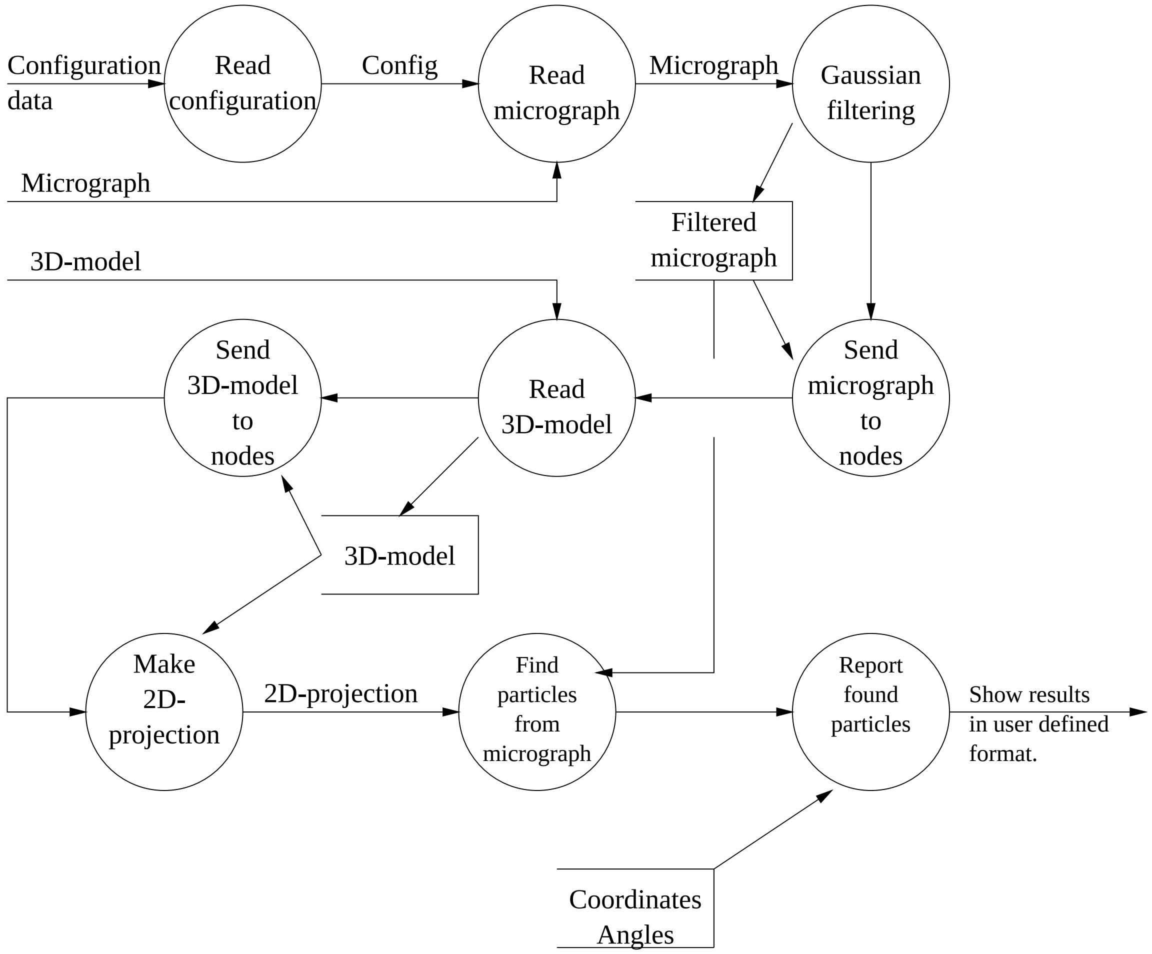

Figure 4: Control flow diagram

5 Functional Description

5.1 Introduction

We're going to use the approach suggested by Teemu Kivioja, which consists of the following steps.

1. Filter the micrograph with a Gaussian filter of size d. The idea is that 2D Gaussian is the first rough model of the particle.

2. Divide the filtered image into small squares so that there cannot be more than one particle center in each square. Add the maximum of each square to a list of candidate particle centers.

3. Compare every candidate area against a set of projections of the model. Consider also rotations and small translations (user-defined parameter) around the candidate center.

4. If the best match is worse than the user-defined limit, reject the candidate. Otherwise output the parameters of the best match and the corresponding particle area if requested by the user.

5.1.1 Handling input parametres

The only command line parameter is the name of a configuration file. Other program parameters are specified within the configuration file.

5.2 Distribution implementation

First the micrograph is preprocessed (filtered, normalized, etc...) on the main node, then it is sent to each node that's participating. After that, the main node sends the virus model to each node. Then it divides the needed projections equally between the nodes, so that if there are 500 possible projection angles and 10 nodes, each node gets sent a range consisting of 50 angles. Then the nodes do the projections themselves and search the micrograph for possible matches, and send the results back to the main node, which just sits there waiting for the results.

5.3 2D projection of 3D model

Almost all of the projection code can be adapted from the *Beklu* project, which is also a software engineering project (spring 2000), dealing with a problem similar to our project. There is a complete implementation for projecting a given 3D model in an arbitrary orientation onto a 2D plane, thus giving the desired projection.

The implementation is based on a techique where the model is bombarded from the desired orientation with an array of vectors (or rays) against the projection plane. The intensity at a given point on the projection place is the total sum of the intensities of the pixels which the corresponding ray hit during the projection.

Intuitively one can think of the whole process as taking a X-ray image from the model. The "X-ray machine" is the source of rays and is located at one side of the model (at the projection angle, that is); on the another side is the "film" (the projection plane). Now, after operating the machine, the desired projection is on the film.

We can isolate the projection code into its own module in our project without any major modifications to the original source code.

6 External interfaces

6.1 Commandline arguments

Commandline name:	`bugfind`
Synopsis:	`bugfind configuration-file`

The configuration file has to be a file of the format described in the following section.

6.2 Configuration file

The configuration file contains in each line one parameter-assignment. The following assignments are possible:

`IMAGEFILE=`*filename*	*filename* is the name of a micrograph-file as described in the section 6.3.
`MODELFILE=`*filename*	*filename* is the name of a 3D model file as described in the section 6.3.
`OUTPUTFILE=`*filename*	*filename* is the name of the file where the output data will be stored.
`DIAMETER=`n	n is a number which describes the diameter for the gaussian-filter.
`DENSITY=`n	n is a number, which describes the projections-raster-density.
`SEARCHRADIUS=`n	n is the diameter of the circular area sought around the candidate points.
`THRESHOLD=`n	n is a threshold value used by the matching algorithm.

Further parameters may be added during design and implementation phase. Configuration file may have comments, which are marked with #-character at the first column.

6.3 Input format

Both the 3D model and EM images are mrc files. This file format has a header of at least 1024 bytes (depending on the number of optional fields). The description of header fields is in the Table 1.

6.4 Output format

The output of the program is stored in the file specified in the configuration file. The output has the following format:

`SCORE=`f	f is the quality of the correlation, a value between 0.0 and 1.0
`POSX=`n	n is the x-coordinate of the particle in the micrograph
`POSY=`n	n is the y-coordinate of the particle in the micrograph
`ANGLEALPHA=`x	x is the alpha-orientation of the particle
`ANGLEBETA=`x	x is the beta-orientation of the particle
`ANGLEGAMMA=`x	x is the gamma-orientation of the particle
`%`	record separator

7 Other characteristics of software

7.1 Performance

If the 3D model of the virus is 128^3 voxels, then it has approximately 2 million voxels. Any projection algorithm must go through most of them at least once, and assuming the algorithmic overhead per voxel is 10 instructions, we end up with one projection taking 20 million instructions. Assuming we use 500 different projections, the total cost of doing the projections is 10000 million instructions. Assuming an average node in the cluster has a MIPS-rating of 1500, it takes only 6.5 seconds to do all the projections. Even if this estimate is wrong by a factor of ten, which is entirely possible, this would seem to indicate that doing the projections won't take too much time, so it doesn't seem worth the trouble to parallelize it.

Data item	Offset (bytes)	Description
int nx	0	image size
int ny	4	
int nz	8	
int mode	12	0=uchar,1=short,2=float
int nxStart	16	unit cell offset
int nyStart	20	
int nzStart	24	
int mx	28	unit cell size in voxels
int my	32	
int mz	36	
float a	40	cell dimensions in Å
float b	44	
float c	48	
float alpha	52	cell angles in degrees
float beta	56	
float gamma	60	
int mapc	64	column axis
int mapr	68	row axis
int maps	72	section axis
float amin	76	minimum density value
float amax	80	maximum density value
float amean	84	average density value
int ispg	88	space group number
int nsymbt	92	bytes used for sym. ops. table
float extra[29]	96	user-defined info
float xOrigin	212	phase origin in pixels
float yOrigin	216	
int nlabl	220	number of labels used
char labels[800]	224	10 80-character labels
char symop[10000]	1024	Variable size symmetry operator table

Table 1: mrc header data structure

The micrograph files are about 8000x8000 pixels, which is 64 million pixels. If we tried naively matching a projection at every possible point, and the projected images were 128x128, and it would take 10 instructions per pixel, it would take a 1500-MIPS node 2 hours. Assuming no distribution overhead, 10 nodes, and 500 projections, doing the whole search for one model would take 100 hours. Which seems a little low, so assuming it's wrong by a factor of ten, it could take 41 days, which is too long.

So we can't just try to find a match at 64 million possible points, we have to somehow narrow down the possibilities first. So assuming we have an algorithm that somehow finds out probable virus-looking areas from the picture, and it takes 10 instructions per pixel and returns on the average 15000 points, that would take 640 million instructions or less than a second. But if we only had to try a match at 15000 points instead of 64 million points, the search time drops dramatically. Keeping the other assumptions same as above, it would only take 1.6 seconds for a single search, and 81 seconds for all 500 projections on 10 nodes. Now this seems ridiculously low, even upping this by 100 times would only take a little over 2 hours. So these estimates are probably all wrong.

7.2 Usability, error recovery, security

Ease of use is a relative term. Software like this is always going to take some sophistication to use, "normal" people wouldn't even understand what the program does before a lengthy explanation. But of course the program should be easy enough to use so that the intended users can do just that, use it. But as this program doesn't have much of an interface, this should be relatively easy to achieve.

Error recovery can mean many different things. It was decided that "normal" errors should be recovered from gracefully, for example invalid parameters, configuration files, and other user errors. In this case graceful recovery just means exit with an appropriate message so the user can fix what's wrong and try again. But errors out of our control, like hardware failures, need not be taken into consideration in the program. They would complicate the program enormously, and the benefit is questionable. If your cluster is failing somehow, the chances are you're not going to get any work done there anyway until you fix it, so might as well design the program assuming the hardware works as expected. However, if we manage to pull out the working software before the deadline (that is, we have some spare time left), we will probably think about what can be done to make the system more fault-tolerant in the case of hardware problems. These ideas will be described in the maintenance document.

Security is another one of those things that we decided to leave out of consideration. If you can't trust your cluster to be secure, well...not much our program could do about it.

The software should have some kind of adaptive qualities. But in practise that probably means only the ability for the user to specify how many nodes it should use, nothing more. We only have one cluster, so we couldn't really test much more adaptiveness than that, so doing things like "change the mode of distribution according to the speed of data transfer between nodes" would be pointless, we wouldn't know if the whole feature worked or not.

7.3 Programming style

Below is a short example of the programming style to be used in the project. Braces start on the same line as the conditional, tab is 4 spaces, there's a space on both sides of the assignment operator (an exception can be made for the initializing of simple loop variables in for-statements like below), a space after ; in for-statements, `x++` instead of `++x`, a space between the closing parenthesis and the starting brace, `int* ptr` instead of `int *ptr` and a space after if/while/for.

```
/* inverts an image */
void rgb_invert(image_info* img)
{
    int x,y;
    guint32 c,r,g,b;

    for (y=0; y < img->user_height; y++) {
        for (x=0; x < img->user_width; x++) {
            c = img->rgb_data[y*img->user_width + x];
            /* ... */
        }
    }

    /* if-else-then formatting */
    if (foo) {
        /* ... */
    } else if (blaa) {
        /* ... */
    } else {
        /* ... */
    }
}
```

8 Testing plan

Testing will be done during the programming phase as far as possible. After the programming work is finished, bottom-up-module-testing will take place. A test suite which is able to perform automated tests with given test cases will be used. One main aim of the testing besides a running program will probably be to test the efficiency of the particle picking. Test pictures with clear contrasts and recognizable particles but also test pictures with different degrees of noise and unrecognizable particles will be created and used as test cases.

References

[Beo] The Beowulf Project. WWW hypertext, http://www.beowulf.org.

[GCC] GCC: GNU Compiler Collection. WWW hypertext, http://gcc.gnu.org.

[K+00] T. Kaituri et al. *SPADE: A frontend for ETHAN.* University of Helsinki, Department of Computer Science, 1999–2000.

[Kiv00] T. Kivioja. *ETHAN: A program for automatic detection of spherical particles from electron micrographs.* University of Helsinki, Department of Computer Science, 2000.

[L+98] Lanczycki et al. Parallel computing strategies for determining viral capsid structure by cryoelectron microscopy. *IEEE Computational Science & Engineering*, pages 76–91, April 1998.

[MPIa] MPI: Message Passing Interface. WWW hypertext, http://www.mpi-forum.org.

[MPIb] MPICH: A portable implementation of MPI. WWW hypertext, http://www-unix.mcs.anl.gov/mpi/mpich.