University of Groningen Faculty of Mathematics and Natural Sciences

Department of Computing Science

Semi-Automatic Spine Reconstruction

R. Nijlunsing

Advisors:

dr. J.B.T.M. Roerdink

Department of Computing Science

University of Groningen

dr. ir. B. Verdonck

ICS Advanced Development

Philips Medical Systems Nederland BV

drs. D.J. Wever

Department of Orthopedics

University Hospital Groningen

September, 1997

RuG

Semi-Automatic Spine Reconstruction

R. Nijlunsing

September 1997

Abstract

The Orthopedic Department of the University Hospital of Groningen (AZG) developed a method which made it possible to use a frontal X-ray image to obtain specific quantities. These quantities were used to described the spine mathematically with help of a software program.

The University of Groningen (RUG) participated in some of these projects. Philips Medical Systems (PMS) developed a software program which combines a frontal and a lateral X-ray image to obtain a reconstruction of a spine.

A specific spine deformation exists, called "scoliosis", which is characterized by lateral deviation and axial rotation of the spine. Patients suffering from scoliosis have to be examined several times a year in a hospital. This examination should therefore be as little time consuming as possible for the medical doctor and should try to reduce the burden on the patient.

The question posed now became: how to integrate those existing methods into one software product, considering maximum flexibility and usability. The solution chosen will be described.

Contents

Preface

All good things come to an end which is also the case now that this thesis has been completed. Since I had already spent for about four years in college, the time had come; and in the end I can say I'm satisfied about what I've done. Some points were good, some points were bad, but that is always the case, so in general I can say I'm satisfied.

Of course, this work wouldn't have been completed without the help of a lot of people:

Thanks go to Bert Verdonck of Philips Medical Systems (PMS) for the patience he had assisting me and the numerous e-mails which he replied all.

Thanks to Jos Roerdink of the University of Groningen (RUG) for not letting me get too content about what I had done and keeping me going on.

And let's not forget Dirk Jan Wever of the University Hospital of Groningen (AZG) who made me understand a lot more of the medical concept of scoliosis and who provided me with medical assumptions I had to use to complete my software program.

Rutger Nijlunsing, summer of '97.

List of abbreviations

3D	Three-dimensional
AZG	University Hospital of Groningen
GUI	Graphical User Interface
PMS	Philips Medical Systems
RUG	University of Groningen
X-ray	Röntgen radiography

Chapter 1

Introduction

In the world of today, computers make up an important part. Computers are used in every thinkable field for very different purposes: from some abstract aspects like solving mathematical equations to the more practical side of administration of personnel. Most people though are not aware of the fact that some combination of the above is also possible and just as widespread. A very nice example of this is the field of **medical computer systems**. Medical computing can be considered from the points described before: from the *research* point of view, the *industrial* point of view and last but not least from the *medical* side.

1.1 Participants

The research point of view has always had strong connections with **scientific computing and imaging**. This field of science is oriented towards working with images in every imaginable way: from extracting features from images, which is called **computer vision**, to producing photo-realistic images given a representation of reality, called **computer graphics**. Like all sciences, there are also in-between fields like **image manipulation** in general. This field of science is intensively studied in the computing science department of the **University of Groningen (RUG)**.

The industrial point of view on medical systems is focused on general progress. This can also be seen as a very wide definition: general progress might be methods to simplify work for medical personnel, improve diagnostic tools or to study and improve treatment. This is what **Philips Medical Systems (PMS)** does in Best. The history of research by Philips goes a long way back to 1914 since research is important for every modern company.

The **Orthopedic Department of the University Hospital of Groningen (AZG)** on the other hand, plays a different role from the medical point of view: the department is known for its large population of patients suffering scoliosis, a disease of the spine, and wants to improve the imaging, treatment and the accuracy of measurement.

1.2 Problem definition

A spinal deformity like scoliosis can be seen on standard X-ray equipment, but precise diagnosis requires much attention by a medical doctor. In order to minimize the time and effort required, the goal is to develop a **semi-automatic scoliosis measuring system**.

Semi-automatic means that the computer is used to get as much information as possible from a digital image. This means some kind of feature extraction, with help from humans ("semi"-automatic) to narrow down the area where the features are likely to be found. In the future, automatic detection might be feasible which is currently not the case since automatic recognition is in most cases not possible.

Another goal is to choose the features to be extracted in such a way, that a reliable three-dimensional (3D) reconstruction of the spine can be made, since scoliosis is in fact a 3D deformation. In this case, it would be very convenient if the features required for such a reconstruction were the same features which could be extracted by a computer.

Two methods, which require special attention, are the method used in the AZG, in which scoliosis measurement is performed using a frontal X-ray and a method developed at PMS, in which a 3D reconstruction of the spine is made using a frontal and a lateral X-ray. I did some relevant work in merging those two methods and incorporating them into one program. I will discuss the current situation, those initial methods and what I did to integrate them.

1.3 Contents

In chapter 2, the underlying theory will be discussed. This theory has been obtained by studying a large quantity of literature. Topics touched include mathematical and anatomical definitions, a description of scoliosis, and several ways to model a spine.

Chapter 3 the current method of patient diagnosis is described, together with the developed methods of AZG and PMS. These methods are part of the problem definition, and an analysis is done to determine the characteristics of the resulting program.

Design fills chapter 4, in which the analysis from the previous chapter is refined. Also, extra attention is payed to the design of the **graphical user interface (GUI)**.

The resulting implementation is reviewed in chapter 5, in which the program is divided into smaller parts with each its own specific function.

Chapter 6 discusses the resulting program by looking at the differences between the design and the implementation and describes some improvements to be made in the future.

Chapter 2

Literature study

The human body contains a spine with vertebrae, which are studied. For projects concerning the spine, definitions are needed to have uniformity within and between projects. A uniform definition has been proposed and is also presented here.

A very convenient way to inspect the spine is by using X-rays. For example scoliosis, which is a spinal deformity, is diagnosed this way. Looking further, it becomes clear that the diagnosis can be quite sophisticated, since rotation and other 3D information can also be deducted from X-rays. Therefore, why not perform a more complete 3D spine reconstruction?

All these subjects are looked into in this chapter.

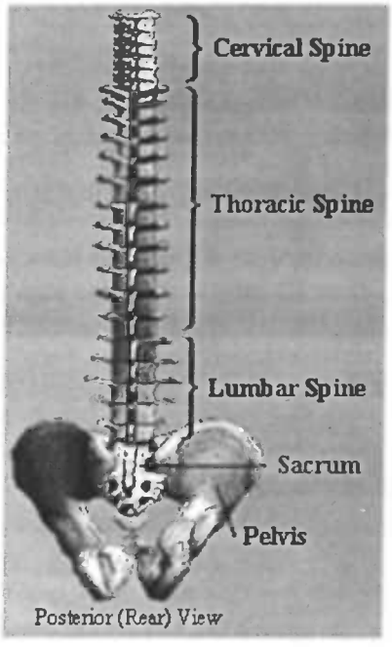
2.1 Anatomy of the human spine

2.1.1 Spine

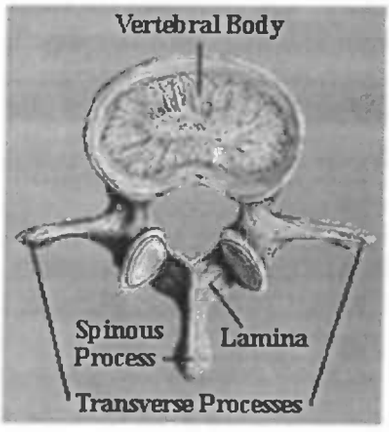
The human body contains a spine as the main structure for the body: without it, the body would be no more than a bag containing water and some left-over bones. The spine consists of vertebrae, 34 to be precise. From top to bottom they are divided into several groups, starting with 7 **cervical vertebrae**, which are part of the neck; then 12 **thoracic vertebrae**, which correspond to the chest: to this part of the spine the ribs are connected. These are called T1 to T12. The 5 **lumbar vertebrae** make up the lower part of the back and are called L1 to L5. Five **sacral** (S1 to S5) and four **coccygeal vertebrae** complete the spine. The thoracic and lumbar spine are the parts of the spine to be studied in this project and are often called the **thoracolumbar spine**. Each vertebra is connected to another vertebra by means of ligaments with an **intervertebral disc** as a separator. See Figure 2.1(a) for an overview.

2.1.2 Vertebra

Each vertebra is distinct from another vertebra in size and shape (especially when comparing two vertebrae in different parts of the spine), but they always contain the following elements (Figure 2.1(b)):



(a) Spine



(b) Vertebra

Figure 2.1: Anatomy

- The **body of the vertebra** is the main component and can quite accurately be approximated by a cylinder. The cylinder is bounded by the **upper and lower endplates**, which are approximately flat.
- The **spinous process** is attached to the body of the vertebrae, and is in a normal vertebra the mirror plane between the left and the right side.
- Two **transverse processes**. The two spaces between the spinous process and the transverse processes are called the vertebral canal.
- Two **pedicles**. These are both connections between the body of the vertebra and the transverse processes. As we will see later on, the main importance of the pedicles is the fact that they are generally clearly visible on X-ray pictures, as can be seen in Figure 2.5.

2.2 Definitions

2.2.1 Coordinate systems

The number of vertebrae is a *fait accompli* and therefore well-defined. A standard terminology on 3D spinal coordinates and deformity (Stokes 1994) has been established. The proposed standard is by now generally accepted and its use is widespread.

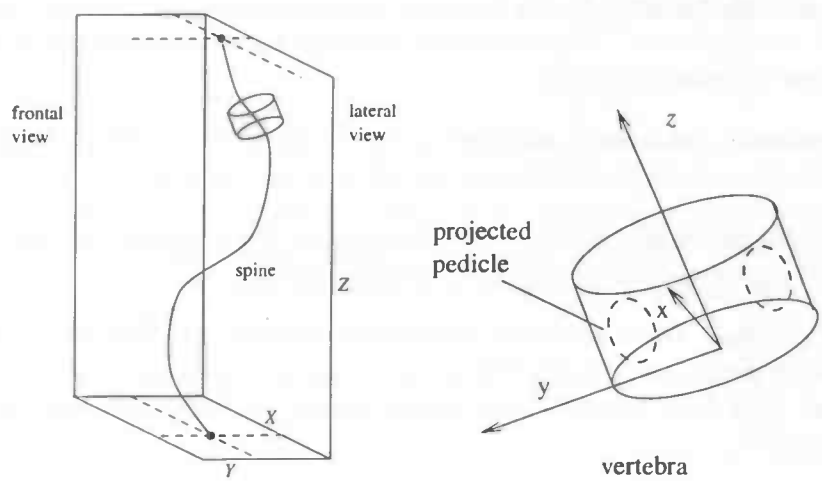
Before we can talk about locations within the spine, we should have a reference **coordinate system**. The one chosen is called the global coordinate system (see Figure 2.2(a)). The positive z-axis lies along the spine in the upward direction, the positive y-axis lies from right to left and the positive x-axis lies from posterior (back) to anterior (front). As usual, a Cartesian axis system is used so all axes are perpendicular. The origin lies at the center of the upper end-plate of S1. The line which represents the spine in the global system is called the **vertebral body line**.

On the vertebral body line, all vertebrae are located. Each vertebra has its own local (vertebral) coordinate system, see Figure 2.2(b). The direction of the axes x, y and z correspond to the rotation of the vertebra with respect to the global ones X, Y and Z. So for a normal vertebra in a normal spine, this coordinate system nearly matches the global coordinate system, except from a rotation around the y-axis caused by the natural lateral curvature. All these rotations and positions of the vertebrae combined give 3D properties of the spine, like total length, curvature and torsion.

2.2.2 Planes and views

With the coordinate systems defined, we can easily define some useful global planes, which are tied to the global coordinate system. The XZ-plane is called the **sagittal plane**, the YZ-plane is called the **frontal plane** and the XY-plane is called the **transverse plane**.

The **posterior-anterior view** is the view visible on the frontal plane, and is therefore also called the **frontal view**. The left-right view or **lateral view** is the view visible on the sagittal



(a) Global (b) Local

Figure 2.2: Coordinate systems

plane.

2.2.3 Natural curvature

A normal thoracolumbar spine has no curvature in the frontal plane, but has a double curve in the lateral one: a **thoracic kyphosis** and a **lumbar lordosis**, see Figure 2.3. Kyphosis is a curvature which is convex in the posterior direction, lordosis is convex in the anterior direction. When speaking of the angle of curvature in general, mostly the angle of curvature in the frontal plane is meant: later we will see how this quantity can be calculated.

Curvatures also determine which vertebrae are important for a physician. For that purpose, **apex** vertebrae are defined as being those vertebrae which have the greatest distance to the line through T1 and L5, see Figure 2.4. These vertebrae are most interesting since the rotation angle (see below) is at its maximum here.

2.2.4 Rotation angles

The **vertebral rotations** of a vertebra are the rotations necessary to transform the global coordinate system into the local one. A problem is the fact that it matters in which sequence the rotations are carried out, especially for angles greater than 10 degrees (Drerup 1984). Note that all those angles are angles in 3D and thus cannot be directly compared to the angles derived from a 2D projection, called **apparent angles**.

Mostly however, the apparent rotation around the z-axis is taken into account in which case vertebral rotation refers to this angle. The rotation around the x-axis is called the lateral **tilt angle**. Rotating around the y-axis is called forward and backward **inclination**. Furthermore,

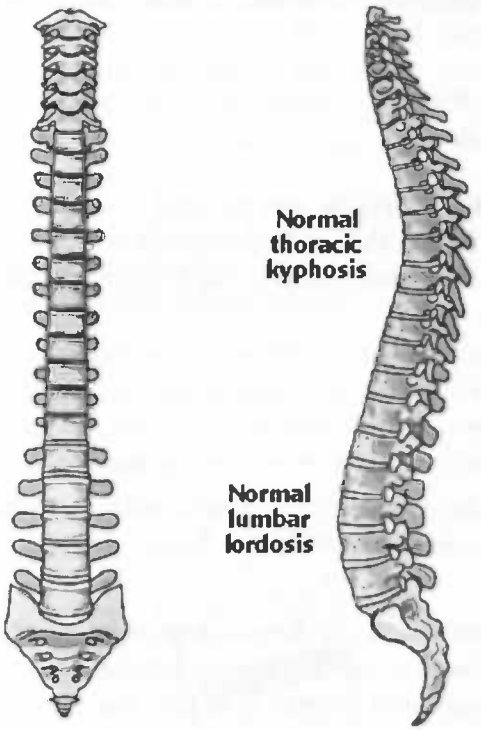


Figure 2.3: Curvature of normal spine

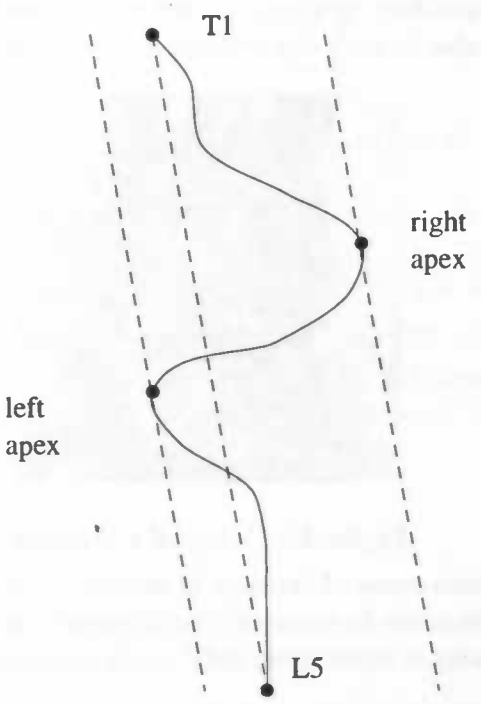


Figure 2.4: Apices

a healthy spine does not have any vertebral rotation along any of its axis with respect to the global coordinate system.

2.2.5 Wedge angle

In a normal vertebra, the endplates are parallel to each other and perpendicular to the vertebral body line; this means that the vertebra is not "wedged". The **wedge angle** is the angle between the upper and lower endplate of a vertebra, and mostly the apparent angle is meant in the frontal view.

2.3 X-rays

Now let us combine the definitions of the planes with the anatomy of the human body: for example, what exactly is visible on a frontal X-ray? For that, we have to know how X-rays work.

The intensity of a point on the X-ray film is dependent on the tissue the beam passes. The emitted X-ray is being absorbed by the tissue, depending on the radiographic density of it. This makes that the accumulated density is in fact what you see on the film.

This makes it clear that on a frontal X-ray the body of the vertebrae, the spinous process and the pedicles are visible, since the human eye is very sensitive to contours. Note that it is not quite correct to state that the pedicles itself are shown: pedicle shadows would be more accurate. The intervertebral discs however, are not visible, see Figure 2.5. On a lateral image the vertebral bodies can also be seen easily when not obstructed by the lungs or shoulders.

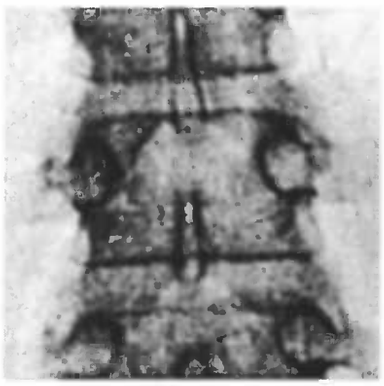


Figure 2.5: X-ray of a vertebra

2.4 Scoliosis

Scoliosis (Richardson 1994) is a *deformity of the spine that is characterized by both lateral curvature and vertebral rotation*, or expressed in definitions presented earlier: *Scoliosis is*

an habitual lateral displacement of the vertebral body line of the spine from its normally symmetric alignment in the mid sagittal plane. In the frontal plane, the vertebral body in the area of the deformity rotates towards the convex side of the curve, and the spinous process rotates towards the **concave** side of the curve (Wever D.J. 1997). An exterior manifestation are shoulders and/or hips not being on the same height. Note that this last is different from what is stated by Richardson (1994).

This has also consequences for the position of the ribs: on the convex side, the rib is pushed posteriorly so the thoracic cage is narrowed and on the concave side the rib is pushed laterally and anteriorly. This is the so-called **rib-hump deformity**.

Another effect of the disease is the deformed vertebral body and intervertebral disc. A typical frontal view can be seen in Figure 2.6: the endplates are not parallel to each other any more, and the forces which are acting on the intervertebral disc are also unevenly distributed over the surface. Not visible are the transverse processes, which are less deformed.

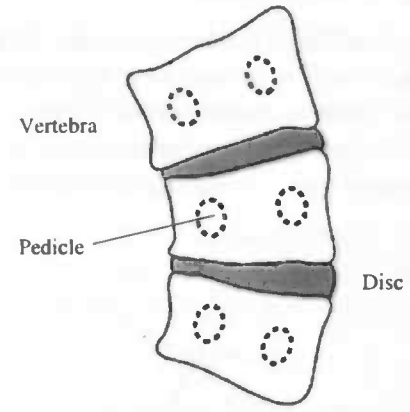


Figure 2.6: Part of scoliotic spine (frontal view)

Scoliosis has many causes and just as many classifications. It can be classified into **non structural**, **transient structural** or **structural** scoliosis. There are as many causes as types: muscle imbalance, tumors and neurological disorder can all cause scoliosis.

Structural Idiopathic scoliosis accounts for more than 75% of all cases, which develops for no known reason. Idiopathic structural scoliosis can be subclassified into infantile, juvenile and adolescent type (most common), which is just determined by the age of onset.

2.4.1 Treatment

Treatment of scoliosis consists of the use of **special braces**, **electrical stimulation** and/or **surgery**. The method chosen depends on curvature of the spine and, just as importantly, the progression of the curvature in time. This progression is measured approximately two to three times a year.

With a curvature of 10 to 20 degrees, doing physical exercises like swimming is sufficient as a treatment (Veldhuizen 1997). A curvature of 20 to 40 degrees is treated with a **brace** which

tries to correct the angle by giving a lot of support during the period of growth. As a general rule it can be said that after skeletal maturity curvature below 30 degrees does not progress.

A curvature more severe than 40 degrees is treated operatively by implanting a spinal implementation. An example of such an implant is the **Cotrel-Dubousset Instrumentation** (Labelle, Dansereau, Bellefeur, Poitras, Rivard, Stokes & de Guise 1995), which can effectively improve the thoracic spine by reducing the spinal screw.

2.4.2 Cobb's angle

Several methods of measuring the angle of curvature were developed: **Cobb's angle, Ferguson method** and the **constrained curvature method**. Of these, only Cobb's angle is the widely approved method. It would not be more than logical that when newer image techniques become more common and other planes of projection are used, Cobb's angle no longer necessarily is the most suitable choice.

Cobb's angle is determined by making a postero-anterior (PA) X-ray and then for each curvature, find and calculate the maximal angle between the upper endplate of the upper vertebra and the lower endplate of the lower vertebra as in Figure 2.7. This instantly shows us the main reason why the use of Cobb's angle is so widespread: it is an easy and fast measurement method and it is easy to compare one measurement with another.

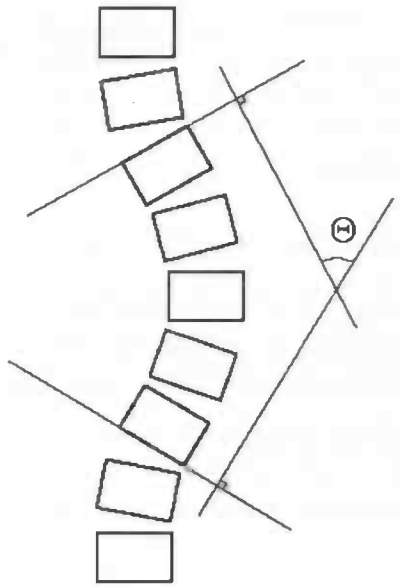


Figure 2.7: Measuring Cobb's angle θ

Despite its advantages, it also has some serious drawbacks:

- It measures the angle in a 2D plane instead of the curvature in 3D space. It would be much more logical (whenever it becomes clinically possible of course) not to use just the (arbitrary) frontal plane, but to use the plane which gives the maximal Cobb's angle.

The main problem is that this plane cannot be determined a priori, so it is presently not possible.

- Instead of only looking at the curvature, it also takes the rotation of the vertebra in the frontal plane into account since the upper and lower endplates of the two vertebrae are used. The angle between the perpendiculars to the vertebral body line would be more accurate.

2.4.3 Obtaining the vertebral rotation

Since Cobb's angle is highly inadequate for gaining a better understanding of the scoliotic curve, several methods have been developed to get more information out of a single, frontal X-ray. This is especially important since there are different shape variations of the spine which give the same Cobb's angle (Drerup 1985). Last but not least it gives a well-needed method to evaluate the effectiveness of existing methods.

For this reason a physician generally estimates the apparent vertebral rotation around the local z-axis (which is a characteristic of scoliosis). This is done by looking at the relative position of the pedicles (whenever visible) with respect to the position of the mid line, see Figure 2.8, again on a frontal X-ray. Several methods have been proposed for measurement (Drerup 1984), which will be discussed here.

Other methods preceded these methods: Cobb, for example, proposed a method based on the spinous process. But since severe scoliosis can lead to asymmetric vertebrae, and because of the fact that the spinous process does not always point to the local x-axis, this method was abandoned because of the low accuracy.

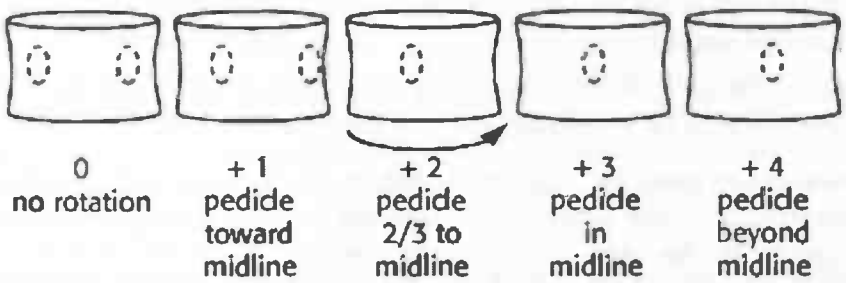


Figure 2.8: Rotation estimation

Nash and Moe

The first method proposed was of Nash and Moe (1969), and is based on the currently most common method of measuring the rotation. The vertebra is described by a simplified model and taking this model as a reference, the (apparent) rotation can be calculated.

The model consists of a cylinder for the body of the vertebra. The endplates are now parallel to each other, so the wedge angle is zero. The only parameter specified is d, the diameter of the cylinder. Both pedicles, P_1 and P_2, are now defined to be at distance r from the middle

of the cylinder, at the angle κ (sign matters) from the symmetry line S. All this can be seen in Figure 2.9 for a pedicle P; note that the symmetry line equals the local x-axis. An assumption made with this model, is that both pedicles have the same angle κ.

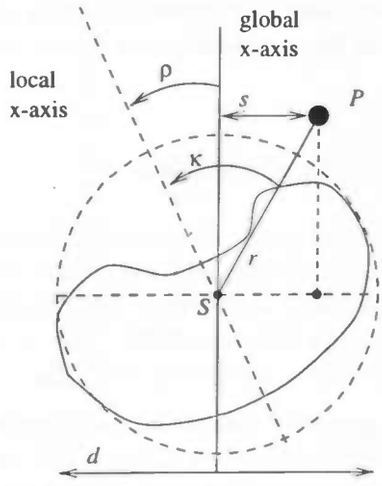


Figure 2.9: Vertebra model

A cylinder is a good approximation, since the three deviations which are seen mostly on scoliotic vertebrae do not affect the parameters much: When the wedge angle is unequal to zero, the mean angle of the two endplates is taken. When the sidewalls are concave instead of straight, the diameter is taken at the waist of the vertebra. The most severe deviation however is the fact that the endplates are elliptical instead of circular which gives a different diameter d depending on the vertebra rotation. This effect can be minimized by only considering rotation angles less than 40 degrees.

To make the model independent of scale, it can easily be seen that the ratio r/d together with the angle κ suffice for an unambiguous description of the model.

But now, what do we measure? This is variable s, the distance of the projected midpoint to the projected pedicle on the X-ray. Since a pedicle is relatively large, a points on the pedicle has to be chosen: in this case, the middle is selected. As we see, s is dependent of ρ, the vertebral rotation angle. The vertebral rotation is estimated for both pedicles, and are called ρ_1 and ρ_2. This yields:

$$\rho_1 = \kappa - \arcsin(s_1/r) \tag{2.1}$$

$$\rho_2 = -(\kappa + \arcsin(s_2/r)) \tag{2.2}$$

so as a good estimation for ρ:

$$\rho = \frac{\rho_1 + \rho_2}{2} \tag{2.3}$$

when both pedicles are visible, ρ_1 or ρ_2 when only one is visible.

Nash and Moe then made some assumptions to fill in the unspecified parameters: when $s = 0$ is measured, so when the pedicle is projected exactly in the middle of the vertebra, they estimated the rotation to be $\rho = 50°$, so they took $\kappa = 50°$. When a pedicle is projected on the cylinder border so $s = d/2$, $\rho = 0°$ was estimated. This gives the Nash and Moe equation:

$$\rho = 50° - 100° \frac{s}{d} \tag{2.4}$$

The absence of the *arcsin* stems from the fact that Nash and Moe measured the percentage displacement of the pedicle, and not the absolute one. This becomes, when when more accurately following the model:

$$\rho = 50° - arcsin\left(\frac{s/d}{r/d}\right) \text{ with } d/r = 2\ sin(50°) \tag{2.5}$$

Drerup investigated this further and calculated with a test set of vertebrae that $\rho = 50°$ was too high and got better results with his "Nash-Moe $-10°$" method: this only differs from the conventional method in assuming $\rho = 40°$.

Actually, a lot of different methods were developed which only differ in the parameters. That is why research was done not only on estimating the parameters by fitting the measured rotations onto the real rotation, but by looking at the real, physical data (Drerup 1985). Drerup found that for **a healthy** vertebra $r/d = 0.59 \pm 0.07$ holds.

Stokes method

Stokes, Bigalow & Moreland (1986) noted that there are more factors determining the projection of a vertebra and thus the position of the pedicle: the axial rotation, the shape of the vertebral body, the distance of the vertebra to the X-ray film and its location. Now instead of just using a fixed value for the depth-to-width ratio, it is more logical to use a value dependent on the anatomic location of the vertebra within the spine, since the thoracic vertebrae have completely different shapes from the lumbar ones.

They did their study on vertebrae T4 to L4 of 99 patients attending a scoliotic clinic. From stereo X-rays, the mean and standard deviation of the width-to-depth ratio was calculated. The values published however, were a factor two off and were therefore corrected by Stokes (Stokes 1991), see Figure 2.10(a).

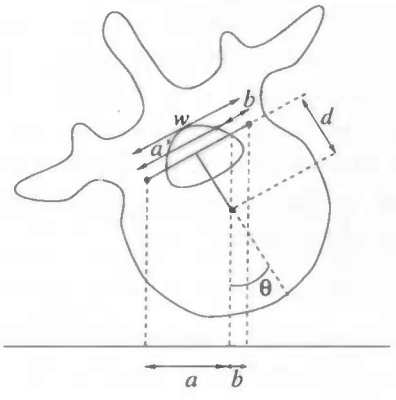
The method used can now be found by looking at Figure 2.10(b), and realising that:

$$tan(\theta) = \frac{(a' - b')/2}{d} \text{ and } \frac{a' - b'}{a' + b'} = \frac{a - b}{a + b} = \frac{a' - b'}{w} \tag{2.6}$$

$$\text{so: } tan(\theta) = \left(\frac{a - b}{a + b}\right) \times \frac{w}{2\ d} \tag{2.7}$$

Vertebra	w/d
T4	3.00
T5	2.75
T6	2.32
T7	2.08
T8	1.84
T9	1.90
T10	1.90
T11	1.92
T12	2.00
L1	1.94
L2	1.84
L3	2.08
L4	2.50

(a) width/depth ratios



(b) Method

Figure 2.10: Stokes' method

Comparison and improvement

It is not easy to compare these methods since not only the vertebrae matter, but also the local coordinate system with respect to the global one: some method might be good at small angles, while another one might be good at translated vertebrae. That is why as an indication for the accuracy three numbers are used (Drerup 1985): the difference Δb between the angle found by s_1 and s_2; the standard deviation σ_{abs} of the absolute rotation measurement and last but not least the standard deviation σ_{rel} of relative rotation measurement. σ_{rel} is actually more important than σ_{abs}, since often is looked at the change in rotation instead of the absolute values.

This gives (Drerup 1985, Stokes, Bigalow & Moreland 1986):

Method	Δb	$\pm$	$\sigma_{\Delta b}$	σ_{abs}	σ_{rel}
Nash-Moe	$-18.7°$	$\pm$	$4.5°$	$12.0°$	$4.7°$
Nash-Moe $-10°$	$-1.3°$	$\pm$	$4.5°$	$5.6°$	$4.7°$
Drerup 1985	$-2.5°$	$\pm$	$3.5°$	$5.0°$	$4.8°$
Stokes 1986				$3.6°$	?

In case of the method of Stokes there is only one measurement for both pedicles, so Δb is not applicable. The σ_{rel} is, though, but has not been given.

A problem with these methods is that it does not take left-right translations into account: when a vertebra is translated horizontally, the projected pedicles move and so the measured rotation does change. This is because of the projection not being orthogonal, as we will see in section 2.5.4.

Stokes also did some research in error analysis, which shows us that it is next to impossible to improve this accuracy. The apparent error introduced by misselecting a pedicle by only 1mm, was 2.2° and the apparent error by misselecting the axis of the center was 4.4°, which is better than the current accuracy. He also noted that it is impossible to point out "the best method", since each method has different error characteristics made up of a random error and a systematic error component.

2.5 3D spine reconstruction

If it would be possible to obtain a 3D reconstruction of the spine, it would help the orthopedic surgeons in getting a better overview of the complete spine and at the same time be able to get the same information as they get now, like Cobb's angle and the rotation of each vertebra.

There are several ways in which a reconstruction can be done, but methods using two X-rays called **biplanar radiography** do not have too much disadvantages. For example, a CT scan would give all the information needed (...and more), but would burden the patient too much on both the time required for examination as for the radiation received. This would also introduce another source of error: movement of the patient during examination.

Another way of reconstruction was the first used method: taking a known spinal model as example and trying to "fit" the acquired X-ray onto the known model by means of scaling, rotation and bending. This has not been used widely though.

2.5.1 Radiography using two X-rays

Reconstruction of a 3D density space, using more than one 2D images can be done by computerized tomography when the number of input images is large. However, that is not the case here.

The idea is that a 3D model of a spine can be estimated from only a limited number of 3D points on the spine. These points are called anatomic **features** or **landmarks** and are determined by the chosen model for the spine. The reconstruction of a point consists of approximating the 3D coordinates as good as possible given the (limited) information. The word "model" is used here for a mathematical and/or geometric construct which *represents* a real object in 3D.

The information given here are two X-rays of a patient, both X-rays taken from another angle (Brown, Burstein, Nash & Schock 1976). A possibility might be to use an angle of 90° for simplicity of reconstruction. A (much) smaller angle has the disadvantage of lower accuracy, but greatly simplifies finding corresponding feature points. Another advantage is the ability to use it in stereo vision by humans. This has not been done very often since it is difficult to get a good visual image since the human does not expect to be able to see through objects (Stokes 1985).

The 90° method however has some technical problems which make it difficult if not impossible for daily use at a hospital. For example, the size of the room required is much larger in comparison with small angles. Secondly the installed X-ray source might not be flexible

enough for a 90° movement. These problems can be solved by using *digital* X-ray imaging techniques and equipment from Philips.

Another problem is movement of the patient during the period in which the X-rays are taken, which is especially a problem when the patient actually has to move before the second X-ray can be taken. This inaccuracy can be minimized by constraints and positioning devices, or just as important: by minimizing the time needed to take the X-ray pictures so the patient is more able to maintain its position. An example of a constraining device was a special reference helmet which was connected to the head of the patient by means of an inflatable airbag.

Besides the features chosen, a calibration has to take place to identify the exact positions of the X-ray sources so it is known which position on the 2D film corresponds to which beamline in 3D. All these coordinates must be expressed in the global coordinate system.

A point visible on both 2D images would give a 4D vector (x_1, y_1, x_2, y_2). This vector can be projected to another 4D vector (x, y, z, ϵ), by a linear transformation, with (x, y, z) the 3D coordinates and ϵ an indication of the error.

This idea introduces a lot of parameters and choices to be made. It also gives some specific sources of error which are dependent on the method chosen.

X-ray projection

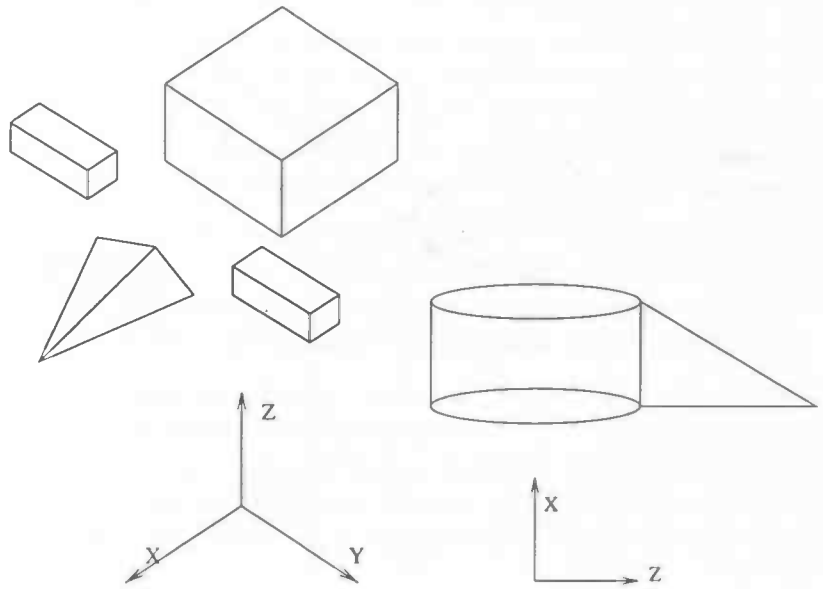
A normal X-ray tube works by emitting radiation from a single, non-moving source which is projected onto a 2D X-ray film-cassette. This gives perspective projection because the X-rays are not cast parallel, which projects any object not coplanar with the X-ray film plane as an nonlinear image (Brown et al. 1976). This leads to variable magnification and image distortion.

2.5.2 Models

To choose a model to represent the vertebra is a delicate matter: a too simplistic model gives too little information to be valuable, but a too complicated one can not be deducted from the restricted information provided by the X-rays, and when it can be deducted it will be unstable.

A model proposed is one by using standard objects that represent the shape of a vertebra (Vandegreind, Hill, Raso, Durdle & Zhang 1995) by using three hexahedrons and a tetrahedron, see Figure 2.11(a). It is a good compromise between model complexity and realism, but it lacks the possibility for modeling misformed vertebrae.

Another widely used model for a vertebra is based on an ellipse for each endplate with a geometrical object attached to give an impression of the vertebral rotation (Figure 2.11(b)).



(a) Hexa- and tetrahedron (b) Ellipses

Figure 2.11: Vertebra models and their local coordinate system

2.5.3 Features

Features are points which have to be identified on the projections made of the patient. These features are needed as parameters for the model and therefore should be visible on both X-rays. This last constraint can be relaxed a little when feature points can be interpolated like points on the border of the endplates. Because of the visibility, the best features are accurate locateable and symmetric in all three planes: a cube or a very small sphere would be nice. Alas, these do not exist.

Even a simple feature like "the pedicle" gives a rather large area to choose a point from, and is therefore not strict enough. In case of the pedicles, there are some obvious choices: the upper, lower, inner, outer edge or the middle point.

The midpoint is used since it is the average of the other four points, so the selection of this point is more accurate, to be exact: always the same point will be calculated to be the midpoint. The biggest disadvantage however, is that it is not clear what exactly is "the middle of a pedicle" and where it is located in 3D, which makes it less accurate.

Another point used is the inner edge point. It is much more accurate to estimate the location in 3D, although it only relies on one point. Drerup (1985) showed this: when the angle of projection changes, the inner edge point projection moves far less than for example the outer edge pedicle point; see Figure 2.12

This does not only apply to pedicle points of course: when the angle between the X-rays taken is small, corresponding points will be easier to find in general. For example, when selecting features on the endplates, the center or a border can be used: the border point can not easily

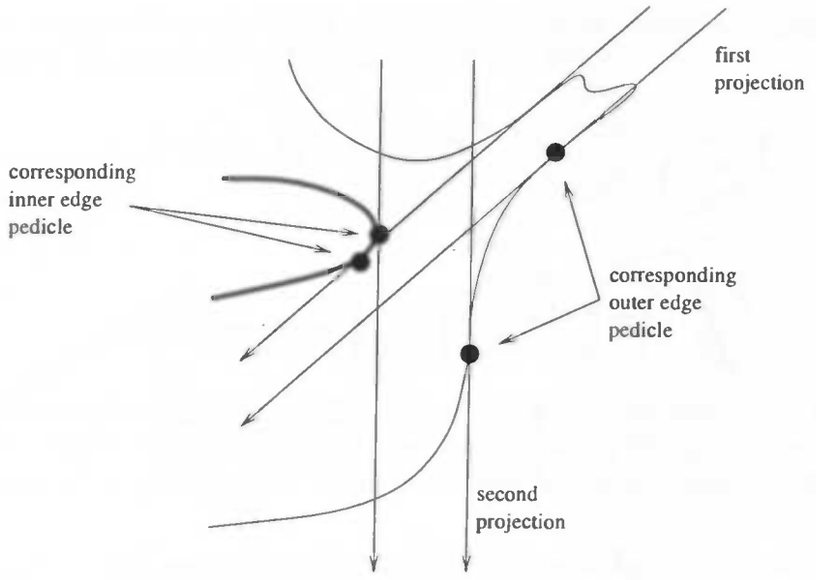


Figure 2.12: Inner edge point is more accurate than outer edge point

be identified now on both X-ray images.

Special care has also to be taken to insure that the features chosen are still visible, despite the scoliotic deformities. Also, the features have to be visible on a wide range of lumbaric and thoracic vertebrae.

The accuracy obtained by measuring the points is of course also dependent on the type of the point. For example, to use the spinous process as a feature point is much less accurate than using a pedicle point (André, Dansereau & Labelle 1994). As a good alternative, feature points on the endplates can be used.

An alternative to finding *existing* features is to use artificial ones, like implanted metal markers which are easily seen on radiographs, but this is in most cases not possible nor desirable.

2.5.4 Projections

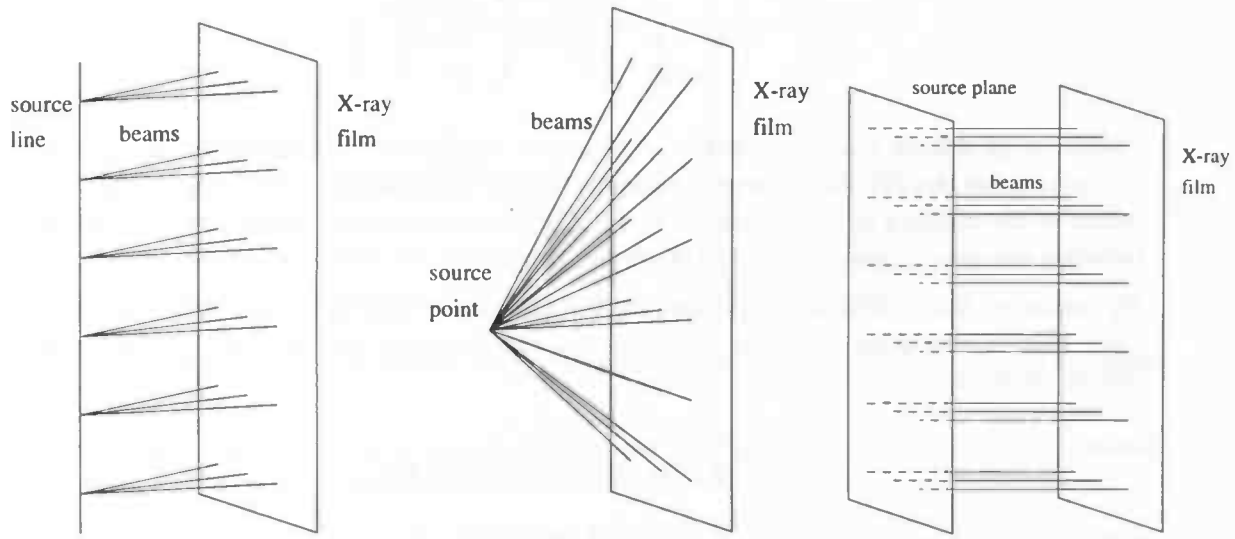
Before explaining the working of the projections, first we have to introduce not only a global and local coordinate systems, but also one for the X-ray films (or, in this case, plane-of-projection). The most logical choice is of course just the normal x and y-axis referring to the horizontal and vertical axis respectively. The notation (x_p, y_p) will be used for the projected points on this plane. Note that this plane actually is definable in global coordinates quite easily in case of a lateral or frontal projection.

The idea of a projection is that a beam passing point (x_g, y_g, z_g) is projected on the film on location (x_p, y_p), according to the type of projection and the location of the source. All projections here have in common that the projection line is the line through an object point and its corresponding projection point, to a point on the source.

Orthogonal projection is the easiest one. Each point (x_g, y_g, z_g) in the global coordinate system is mapped to the local coordinate system $(x_p, y_p) = (x_g, z_g)$ (when neglecting translations) for a lateral image and $(x_p, y_p) = (y_g, z_g)$ for a frontal image. This is very convenient since images are not distorted in any way since all beams are parallel. However, orthogonal projection often is not a good approximation, so applications are limited. An example which comes close is the sun casting a shadow of an object on a flat surface.

Perspective projection is the effect of a point source casting rays in every possible direction. In practice, this means that objects are scaled according to the distance of the object to the source. In case of a vertebra, this deforms the projection since the vertebra has depth. Perspective projection is what you get from conventional X-ray equipment.

Cylindrical projection works in-between: since the source is a 1D line instead of a 0D point or 2D plane, the object is perspectively projected into one dimension and orthogonally into the other. The Philips X-ray method is not completely cylindrical, but an approximation, since the source is not a line but a discrete number of points at a regular distance from each other. The effect is that the global z-axis is projected orthogonally and the x and y-axis are projected in perspective.



(a) Cylindrical (b) Perspective (c) Orthogonal

Figure 2.13: Projections and their projection lines

2.5.5 The DLT method

A widespread model for the projection is the so-called **DLT method**. DLT stands for Direct Linear Transformation and has the properties of being relatively simple and accurate, since it consists of only a linear transformation between the 3D object space and the 2D image planes.

DLT assumes a perspective projection of the 3D coordinate in object space (X, Y, Z) to the measured 2D plane coordinate (x^*, y^*). This gives (Pivet 1996):

$$x^* + \delta x + \Delta x = \frac{L_1 X + L_2 Y + L_3 Z + L_4}{L_9 X + L_{10} Y + L_{11} Z + 1} \tag{2.8}$$

and

$$y^* + \delta y + \Delta y = \frac{L_5 X + L_6 Y + L_7 Z + L_8}{L_9 X + L_{10} Y + L_{11} Z + 1} \tag{2.9}$$

where δx and δy are non-systematic errors and Δx and Δy are random errors which are all ignored. $L_1 ... L_{11}$ are the 11 parameters which are determined by the location and orientation of the source and the projection plane. These parameters are proportional to the scale factor of the camera.

This simple expression can be rewritten to the basic photogrammetric relation:

$$\begin{pmatrix} x - x_0 \\ y - y_0 \\ -\mu c \end{pmatrix} = \lambda A \begin{pmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{pmatrix} \tag{2.10}$$

where (x, y) equals the 2D plane coordinate of the projected coordinate in object space, (x_0, y_0) equals the 2D plane coordinate of the source orthogonally projected, λ equals the scale factor of the camera and A denotes the 3×3 orientation matrix which describes the relation between the coordinate system of the object space and the one of the plane of projection.

By assuming that the error introduced has a symmetrical and an asymmetrical lens distortion part, and by assuming these can be defined as a polynomial (Hatze 1988), $x - x_0$ and $y - y_0$ can be expressed as:

$$x - x_0 = \lambda_x (x^* + \Delta x - x_0) \tag{2.11}$$

$$y - y_0 = \lambda_y (y^* + \Delta y - y_0) \tag{2.12}$$

where λ_x and λ_y are the scale factors of the respective axes.

By substitution of equation 2.10 in 2.8 and 2.9, and by rearranging terms, it can be shown that since A is orthogonal, an extra relation of the form

$$L_n - h(L_1, ..., L_{n-1}, L_{n+1}, ..., L_{11}) = 0 \tag{2.13}$$

holds, which proves that only 10 independent parameters are needed to determine the 11 DLT parameters which gives the linear MDLT (Modified DLT) method.

Using this method, called the nonlinear MDLT since it compensated for non-linear errors, the magnitude of error was reduced to about 1%.

2.5.6 Calibration

The parameters needed for the projection are calculated by means of **calibration**. Calibration is the process of minimizing the error of the backward projection. Calibration is done by measuring the location of some pre-determined **control points** which are most of the times artificial features like rulers or implants.

The idea behind calibration is to minimize the distances between corresponding curve pairs. These curves are the projection lines. When several projections are used, like in stereo X-ray techniques, each projection of a feature gives a projection line. Now the parameters for the projection are estimated in such a way that the resulting corresponding projection lines have a minimal distance to each other as in Figure 2.14.

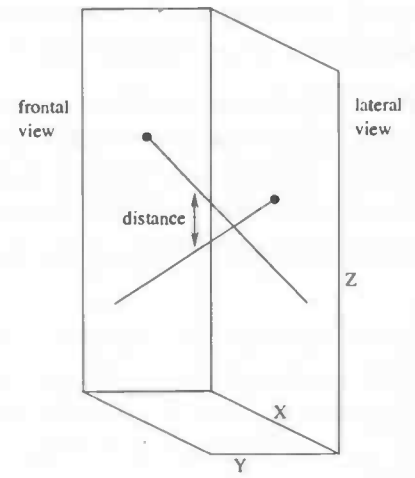


Figure 2.14: Distances between projection lines

The accuracy depends on the chosen control points (Chen, Armstrong & Raftepulous 1994):

- The control points must be evenly distributed so that the accuracy is nearly constant in the object space to be monitored.
- The number of control points must be large, but going beyond 16 control points with the DLT method does not improve accuracy any further. A large number of points however, makes it possible to negate to some extent the effect of the nonlinear distortions. This is especially important when the **field of vision** (FOV) is large, since lenses generally have a larger distortion at the edges of the lens (André et al. 1994).
- The distribution of the points is more important than the number of control points.

2.5.7 Automation

Automatic spine reconstruction is a process which needs (almost) no human intervention to make a reconstruction of a spine based on X-ray images, so the computer itself has to derive all reconstruction parameters from the images.

Automatic is in this case still a bridge too far, but semi-automatic feature detection is becoming feasible. Semi-automatic is the process in which some amount of human intervention time is needed, and so the computer is guided into the right direction by using a *priori* knowledge. The knowledge needed is dependent on the algorithms used.

Reconstruction can be divided into several smaller problems, and can even be solved in a completely other way. This is very beneficial, since other problems might be solved automatically, and so different methods exist which all solve a different part. For example, a lot of methods try to find the contour of the vertebrae, since some features like the endplates can be deducted from them, although not in a straightforward way.

A **center and radius** for each vertebra was used to find the contour of a vertebra (Pluim & Westenberg 1996). This was done by the use of **snakes**, which are deformable contour models: a sort of a rubber band which contracts until it fits around the object. Although encouraging results were obtained, the image quality was too low for practical use.

A method which comes very close to automatic contour recognition and therefore Cobb's angle calculation, is the **gradient polygons image processing algorithm** (Moreno, Stefano & Piero 1995). It is a local technique which works by very sharp edge detection and a gradient information encoding. But like the previous method, the encoding is different for each image and so complete automation is still not possible.

Chapter 3

Problem definition

In this chapter I will examine the current methods of patient examination and define the problem according to these methods.

3.1 Current method of patient examination

Prior to anything else, it is important to know how the current clinical method works. The method I describe here is in use in most hospitals and can thus be called "common practice". Each method consists of actions taken by a medical doctor and a prescription which shows us *how* to derive and *what* to derive from that analysis.

Nowadays, people suffering from scoliosis are examined approximately two to three times a year by taking a frontal X-ray. This is done by letting the patient stand up against an iron frame with the arms holding the frame and trying to stand in a natural position.

In severe scoliosis cases this position might give a problem since the spine is so much deformed that the patient cannot stand straight up, so a natural position is out of the question.

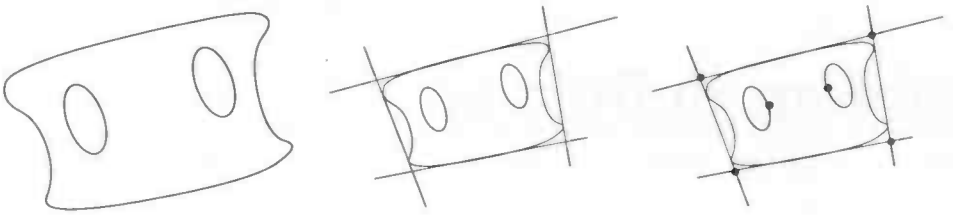
Only one X-ray is taken (in contrast to a biplanar or using a reduced FOV method). This radiologist uses his expert knowledge to estimate the upper vertebra plate of the vertebra most tilted and the lower vertebra plate of the most counter-tilted vertebra. Those plates are marked by a line, from which two perpendicular lines are drawn. The angle between those last lines is Cobb's angle is determined by means of a geometric ruler. See also Figure 2.7.

This angle is the only information you get, but has the advantage that it can be determined within a minute.

3.2 Wever's method

The current method leads to a wish to get more information from the X-ray taken at the expense of some time. So D.J. Wever from the AZG developed a method which he uses to extract more information from a frontal X-ray.

His method uses the same principle of taking the X-ray, resulting in an analog picture which is difficult to manipulate digitally by computer. Instead of only indicating the vertebra plates causing the Cobb's angle, all visible vertebrae are visited. At each vertebra, both plates and both sides are approximated by lines with a pencil. The four crossings of those lines and the inner points of the pedicles are selected as landmarks and thus marked with white paint, see Figure 3.1.



(a) Original (b) Lines (c) The landmarks

Figure 3.1: Wever's method for acquiring vertebra data

The main advantage of white paint is that it is easy recognizable since the white color is clearly distinguishable against the dark X-ray. The contrast is that high that it is reasonably easy to automatically extract the location of those landmarks after the image has been scanned.

As we see, the corner points do not have to be situated on the vertebra itself; this stems from the fact that the crossings of lines are used and not the edge of the vertebra. Should the latter be done this would pose a problem: the corner of a vertebra is not always clearly defined since the corners aren't rectangular but round.

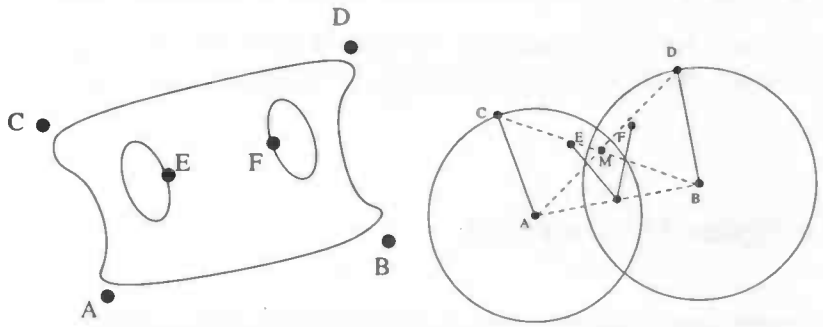
Now that those points have been identified (sometimes an origin is also indicated on the X-ray) and have been converted digitally, a spreadsheet called EXCEL is used to extract useful features from the landmarks by a macro. The minimal amount of landmarks needed are the corner-points, the pedicle points are optional. The corner-points are named A, B, C and D and the pedicle-points (when available) are named E and F as in Figure 3.2(a).

The macro starts at the bottom by finding the lowest two points. The left one is classified as an A landmark and the right one as a B landmark. If there are two pedicles, the two points which are closest to the midpoint between A and B are classified as E/F. In case of only one pedicle, only one point is classified as E/F. Note that the number of pedicles has been entered manually and so simplifies the matter further. The time required to complete the process is however increased.

Now of the remaining points, the landmark closest to A is classified C and the landmark closest to B is classified D. All this can be seen in Figure 3.2(b). This process is repeated until all indicated points have been classified.

The average of the four corner-points A, B, C and D gives a **midpoint** M for each vertebra. These points are plotted in a graph for a general overview of the spine. This makes it quite easy to see whether the spine is straight or has some structural deviation from the y-axis.

The average of the length of the line AC and the line BD is used as the **height** of the vertebra.

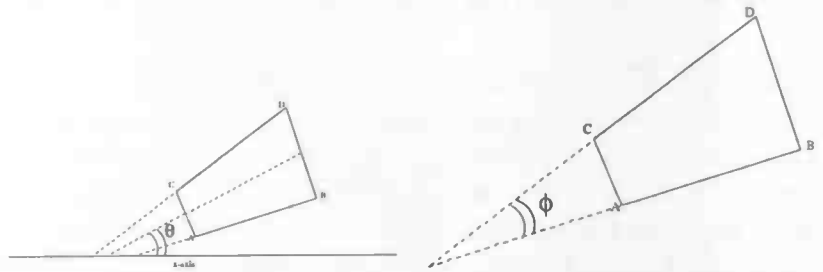


(a) Landmark names (b) Scanned points

Figure 3.2: Wever's algorithm

All heights are summed to get to total length of the spine and to get an idea how rapidly the patient is still growing. The height of the lumbar spine is separately calculated from the height of the thoracic spine.

From the angle line AB makes with the x-axis averaged with the angle line CD makes, the **tilt angle** is calculated, see Figure 3.3(a). This is effectively the rotation in the frontal plane. The angle between the lines AB and CD is the **wedge angle**, which is an indication of the deformity of the vertebra, see Figure 3.3(b). In the same way the angles of the intervertebral discs are determined (tilt and wedge).



(a) Tilt angle (b) Wedge angle

Figure 3.3: Angles

The list of tilt angles is used to determine the **Cobb's angles**. It starts at the bottom and calculates the most tilted vertebra found now. When the rotation changes from clockwise to counterclockwise or vice versa, which can be deduced from change in the sign of the tilt angles, this kept vertebra is recorded to make up a part of a Cobb's angle pair. In the end, all Cobb's angles are reported.

The pedicles which have not been used now give information about the vertebral rotation around their own vertical axis. The method Wever uses is the Stokes method (Stokes et al. 1986) with correction applied as described (Stokes 1991). Stokes uses the position of both

pedicles and a table of empirically determined values of width-to-depth ratios.

This method gives a lot of information, but has as main disadvantage that it takes a lot of human intervention. Each picture takes about 10 minutes to get processed.

3.3 The Spine3D method

A totally different approach is taken in the Spine3D (Pivet 1996) project. This project was the result of a cooperation between Philips Medical Systems (PMS) and Laboratoires d'Electronique Philips (LEP). I will discuss here the choices made and the implications it has on the results.

The Spine3D project aims at **analyzing the shape of the spine in three dimensions**. Before we will go into this any further, note that this is by no means a trivial task: a lot of choices have to be made.

3.3.1 Digital X-ray imaging

Since the accuracy of a film decreases at increased distance from the center of the lens, reducing the Field of Vision (FOV) would produce images with less systematic errors. Therefore, PMS developed a novel method for digital X-ray imaging. By taking a lot of reduced-FOV pictures automatic reconstruction of the whole X-ray (van Eeuwijk, Lobregt & Gerritsen 1997) is possible.

Digital X-ray has the additional advantage that it completely discards the need for scanning an X-ray, a process which introduces additional noise and other errors in the picture. Another way in which the picture quality is improved, is the fact that a reduced FOV can lead to local dose adjustment. This means that instead of one global selected dose level, the exposure control of the X-rays is adjusted in real-time to the tissue the X-ray passes at that moment. This is a big advantage since the human body tissue has a large range of radiographic density, so under-exposure and over-exposure are handled automatically.

Since an image intensifier is used together with methods to reduce the X-ray dose, the actual dose is lower than when using the conventional X-ray method. Reducing the dose is also done by shuttering in the direction of the translation, so in fact an almost rectangular area is the result of the projection.

The complete reconstructed image has a size of about 2000 times 512 pixels. Each image obtained by the reduced FOV method is between 40 and 80 pixels high, so the process yields 30 to 40 partly overlapping images, which are merged to images like in Figure 3.4.

This X-ray equipment is directly connected to an **EasyVision workstation**, which is a stable and easy expandable working environment with many graphics tools available. A GUI builder called `xUiEdit` is also installed for rapid prototype development and to ease the implementation of the front-end.

(a) Frontal view (b) Lateral view

Figure 3.4: X-ray images obtained with reduced FOV method

3.3.2 Projection

With the FOV reduced and a complete image which has been reconstructed from it, the effective projection is also changed since there is not a unique point source but multiple ones. The kind of projection it produces is something like a cylindrical projection instead of perspective projection.

It is not a perfect cylindrical projection though but an approximation of it since in a cylindrical projection the source is a line and not a number of points on a line, as is the case here.

3.3.3 Model

The model chosen consists of two completely independent ellipses representing the end-plates, and a tetrahedron representing the spinous process. Only the direction of the tetrahedron is of importance: size and form are determined by the ellipses. Each 3D ellipse has 8 degrees of freedom: a 3D midpoint, 2 rotation angles, 1 intrinsic rotation and the length of both (2) axes. The tetrahedron adds 1 to arrive at a total of 17 parameters to be calculated. See Figure 3.5.

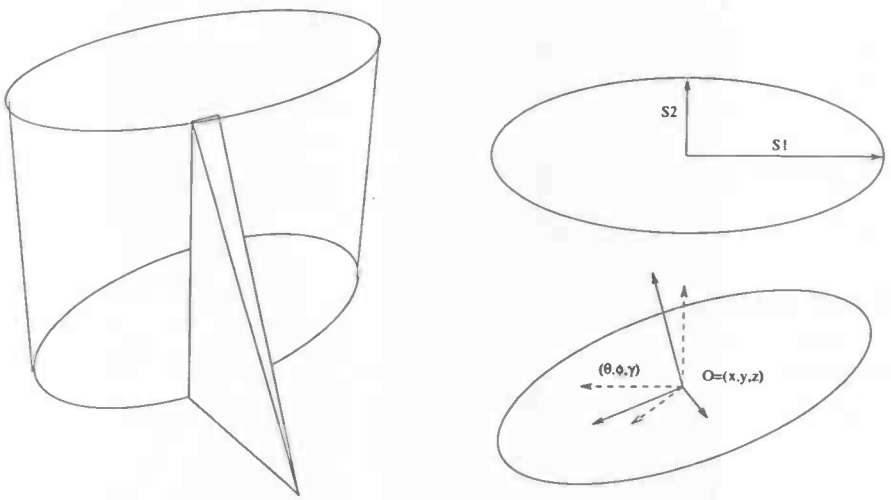


Figure 3.5: Spine3D model parameters

The advantage of this model is that it allows us to incorporate deformed vertebrae into the model, what is especially important with scoliosis.

3.3.4 Angle between X-rays

For an exact 3D reconstruction one X-ray image does not suffice. Two X-rays do, but this poses the next question: from which angles should the X-rays be taken? Instead of choosing a small angle between the two X-rays, an angle of 90 degrees was taken. This makes reconstruction of a point from two projections a lot easier, but has a practical problem: hospitals do not have the possibility to take orthogonal X-rays without the patient moving since the

angle is not small enough, so this introduces an error.

3.3.5 Landmarks

The chosen landmarks can be seen in Figure 3.6: on the frontal image the four corner-points and the right pedicle. On the lateral image the four corner-points suffice. From the relative pedicle location and the "Nash-Moe $-10°$" method, the rotation is determined.

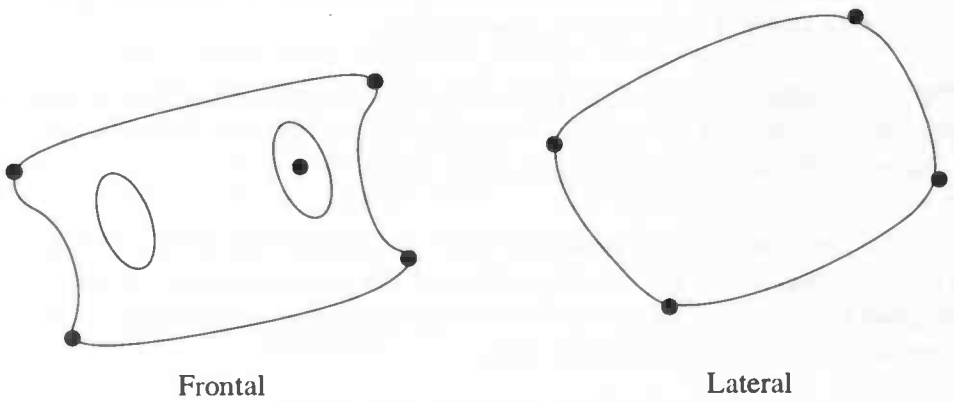


Figure 3.6: Landmarks of Spine3D

3.3.6 Remarks

Although Spine3D has a strong theoretical basis, like every program the GUI can be improved on. The integration of a looking glass is high on the priority list, since it is tedious and time consuming to use the ad-hoc looking glass available in the EasyVision environment.

Spine3D uses the spinous process for visualizing the rotation of the vertebra by pointing into the direction of rotation. This is very misleading since the spinous process is always rotated towards the center. A more correct representation would be to copy the real behavior of the spinous process or to visualize the two transverse processes.

Instead of just using one pedicle for vertebral rotation estimation, Stokes' method using two would be more accurate. This is not difficult to implement.

3.4 The problem

3.4.1 Goals

The main goal is to design and develop a prototype which combines Wever's method with the Spine3D method. This goal was divided into subgoals on which further development was based. Note that those subgoals are not at all independent of each other.

- The functionality should include Wever's method so users of the current manual method can continue using this method. This means that at least the program should output the information the EXCEL macro did using the same information.

- The functionality should also include the existing functionality of the Spine3D modeling environment. This means a 3D reconstruction of the spine according to Pivet's method.

- The functionality should not be restricted to those methods, in fact the program should be as flexible as possible to allow newer methods easily to be implemented, or to update the current method according to new studies.

- The interaction time required for a complete analysis should be as short as possible. The accuracy should be as high as possible. These goals cannot be reached both at the same time of course, a balance should be found.

- See how these manual and semi-automatic techniques can be mapped upon each other. This mapping is the key to integration of the two methods: without it, the user is required to first input all points for the first method and completely independent of this input all the points for the other one.

3.4.2 Starting points

During this development, I had to take into account the following starting points which together with the goals define the problem:

- The development environment will be the general PMS environment like the one used with the Spine3D project: EasyVision. This makes integrating the end product a lot easier and makes it also possible to obey the general standard on deliverables. Deliverables make it possible to exchange working programs and the knowledge required to understand those programs on the EasyVision platform.

- The X-ray techniques used are the new reduced-FOV method. This makes the use of those images a lot easier, since they are a part of the EasyVision platform.

3.4.3 Steps to be taken

To handle the problem in a structured way, the next step is to analyze the problem. There are several options to do this, but the most intuitive one is by comparing the two existing methods to determine the similarities and the differences.

In the next chapter I will go into the design issues, discuss the assumptions taken and determine the guidelines. Since a large part of the program consists of the interface, the GUI will get more attention than in an average program (as far as one can speak about "the average program").

3.5 Analysis

Since humans tend to think in "objects" instead of data structures, object oriented analysis (OOA) is a lot more attractive than the conventional methods like structured programming. Another advantage of using objects is the greater re-usability of the resulting program since not the current functionality of the object is the main issue but the object itself; the object in general does not change, but the required functionality in general does. This means that when the required functionality of the object changes or increases, existing code can be used and expanded more easily.

However, the resulting language in which the problem has to be programmed is not object oriented, so when using OOA at least the last step involves converting to a structural-only language like C. Another problem area in which OOA is good at defining relations between objects and to have objects derived from each other like sub- and superclasses, which is also not very appropriate for the current problem. This leads to the conventional structural approach with some OOA influence.

3.5.1 Both methods compared

It is not straightforward to compare two methods this different from each other, but it is useful nevertheless since it provides us with the basis of the analysis.

As the name Spine3D indicates, a 3D reconstructor and the EXCEL macro is a 2D one. Since flexibility is a key issue, the program should accept at least 3D data. 3D data is almost always acquired by two X-rays, so there should be a possibility to work with *two X-rays at the same time*. Since Spine3D works with lateral and frontal images, let's call them just that. There should however be a *possibility to work with only one* X-ray: the conventional frontal one.

The biggest difference between both methods besides the dimensions they work in is interactivity: the EXCEL macro takes the input, processes it and produces an output. On the other hand, Spine3D takes the input, produces a 3D reconstruction and expects the user to interact and to change the location of the *reconstructed* points to correct the 3D reconstruction. This means that although the introduced landmarks are almost the same, Spine3D translates it into a model which differs greatly from the inputted points.

Because of the aim that interaction time should be as short as possible and the functionality of the macro should be preserved, I chose a way in-between: *there will only be one phase* in which the medical doctor inputs the landmarks and then has the calculated data output. The interaction consists of self-correcting the point given in and *not* correcting the derived model. This is also better for the accuracy since the accuracy of the model is unknown in real-life situations. Whenever the accuracy will be known and is sufficient, there should be an extra phase introduced.

This gives us the possibility to look only at the used landmarks, and to select the best of them. Both methods use the corners of the vertebrae in the frontal plane, so I will use them too. In fact, the only difference are the pedicles: Spine3D only takes the location of one pedicle, EXCEL uses them both. Stokes et al. (1986) showed that the positions of both pedicles are important to distinguish different rotations which have the same projection, so *both pedicles*

are needed.

Because of the fact that Spine3D is a 3D reconstruction instead of just 2D, Spine3D also *uses the corners on the lateral plane.* This is like the points on the frontal plane, so adding them poses no great problem.

3.5.2 Objects and structures

Now we can start identifying the existing objects and underlying structures.

First, we observe the fact that since we reconstruct a spine, we need a model of a spine. A spine is a collection of 17 vertebrae. It is needed when information is requested regarding the whole spine instead of one vertebra, like Cobb's angle. Note that in this case the spine is the object (taken from the real world) and calculating Cobb's angle is a part of the functionality, which is likely to change with new 3D angles measurement methods.

The vertebra is the most important structure: it should at least contain the inputted information like the landmarks and contain the reconstruction. So this structure should be expandable most easily. A distinction is made between entered points and calculated points: calculated points are derived from the entered points.

And of course the images itself are objects. There should at least exist two images and their information structure only contains the 2D images itself. Since a medical doctor can identify the vertebrae quite easily, the images do not have to contain all the entered points like the EXCEL macro. Instead, the points are part of the vertebra now. Identifying is done by the knowledge that the last rib is connected to vertebra T12.

This last fact is also very important since it means that part of the algorithm should categorize all entered points.

Chapter 4

Design

Designing a program from the analysis can be done by looking at different angles at the problem, since each problem has a lot of views. In this chapter the design is discussed, by looking at the user's points of view and at the programmer's. The user is greatly helped by a good GUI, since that is his interface to the program, while the programmer is more interested in a clear and flexible program. These views give the requirements of the implementation.

4.1 Interaction

We have now identified the objectives of the program, but can still derive more design issues from the goals and add some own goals to it. For example, a short interaction time without sacrificing the accuracy is possible by a good GUI design and implementation. Since this is a main goal and it can be achieved without too much trouble, I will discuss it below. Another aspect is the balance between accuracy acquired versus interaction time.

4.1.1 GUI requirements

As I made clear by now, the GUI is a very important part of the program since small interaction time is a main goal. The GUI has the following requirements:

- Simple to learn. A more complicated term for it might be: "intuitive".
- Has a consistent look-and-feel throughout the program. When a program always has a "Confirm" and a "Abort" button, these should not be called "OK" and "Cancel" further on. The same holds for the order of the buttons, since it only confuses the user otherwise.
- Has an adequate response time to events.
- All needed functionality is integrated into one program with its windows. A user should not be required to run external programs to post-process the output.

- Is robust. Random keystrokes, mouse actions and input should not upset the program and abort its action prematurely.
- Is well-organized. All window elements are grouped together according to their function or to the class they operate upon.
- Gives enough feedback. The user should be informed whether or not an action taken has caused any changes when the changes are not directly visible. And in case of an error, this should be noted too.
- Has the ability to do all needed functions both with mouse and keyboard. A user starts learning a program with a GUI by using the mouse and use buttons for some events. But when the user gets more familiar with the program, it is time reducing to provide some short-cuts, key strokes with the same functionality as the buttons. This is the best of both worlds, since it reduces learning time and working time.
- Executes actions best fitted for keyboard with the keyboard and actions best fitted for mouse with the mouse. Position dependent actions like selecting a landmark in a window is much faster by mouse, but position independent actions like most buttons are done much faster by a keystroke.
- Has no need to unnecessary change from mouse to keyboard or vice versa. A change of input-device costs much time and is highly inconvenient. Note that this does not rule out the use of both input-devices at the same time: a mouse and a keyboard can both be controlled by one hand. Only typing takes two hands, in contrast to delivering keystrokes.
- Has one main window. A user is easily distracted by too many open windows at once, it should be clear what to do next.
- Has some intelligence on its own. For example, it is much easier to have a default action binded to a mouse-button than having to first select the action and afterwards execute it with a button press.
- Makes effective use of the precious screen space. Especially in case of programs which work with images, a large part dedicated to the image makes selecting a point on that image more accurate and reduces the need to scroll.

Note that "well-looking" is not one of the characteristics and should not be an aim on itself, for example a button with 3D appearances has no extra functionality to offer above a flat looking one. This is a commonly made mistake though.

4.1.2 Accuracy versus interaction time

It gets more complicated when it comes to the balance between interaction time and accuracy. I chose in favor of accuracy, but when new methods are developed which maintain the accuracy but decrease the interaction time, like accurate interpolation, it should be easy to insert it into the program.

Another aspect of this balance should be to be able to use the knowledge of the medical doctor to have him decide how much time to spend inputting those points: this leads to the possibility of *skipping unimportant vertebrae*. Or, even more general in the spirit of flexibility: the possibility to input incomplete vertebrae.

4.2 Algorithms

The only algorithm which needs to be described is the point classification, since it differs from the way it is done in the manual method. The rest of the program consists mainly of handling all GUI requests, and the translation of the those GUI input events to the underlying structure. This is however straightforward and no complex structures are needed.

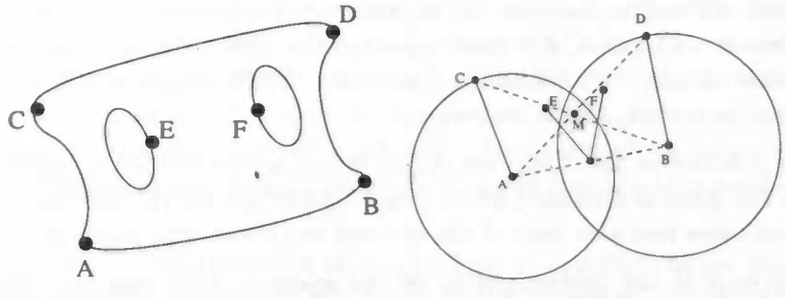
4.2.1 Algorithm requirements

Since simplicity of the resulting program is more important than selecting the most optimized algorithm, simple algorithms are favored over complicated ones. Computers of today are fast enough to handle such non-optimized implementations of algorithms without noticeable delay.

This will also mean that a lot of actions taken during the execution of the program are in fact superfluous: for example screen updates which update more than the part of the screen which was changed during the operation.

4.2.2 Point classification

The part which does the point classification gets as input the entered points in no particular order and has to determine the type of those points. The possible point types are the four corners of the vertebra and the left and right pedicle center. The base of the algorithm used is the EXCEL macro used in the manual method described in section 3.2. Little modifications were made to the macro as described below, but the names of the landmarks were kept the same as can be seen in Figure 4.1(a).



(a) Landmark names (b) Scanned points

Figure 4.1: Point classification

The implicit assumption made is that the tilt angle of a vertebra is always less than 90 degrees and that the four corner points describe a convex polygon. Now the lower two points can be surely classified as types A and B according to the macro naming convention. To be precise, the left one is classified A and the right one is classified B.

Since it is clear in advance how many points are part of the vertebra, the number of entered pedicles is also known: for example, five entered points give four corner points which are always available and one remaining pedicle. Next those pedicles are found. Pedicles are those unclassified points which are closest to the middle of the line from A to B. An improvement to the EXCEL macro is made to make a needed distinction between pedicles E and F. The pedicle closest to A is said to be pedicle E and the other one is pedicle F, as can be seen in Figure 4.1(b).

The two remaining points are sorted along the global x-axis, and the left one is called C and the remaining one D.

4.3 Output

Since the output is in this case not interactive, it is separated from the interface part. It is important though since it comprises about all needed calculations. At a second look at the EXCEL macro it became clear that it outputs more information than needed. After inspection, the next items are needed:

- A graph of the coordinates of the midpoints for a general overview.
- The spinal length information, divided into a thoracic and a lumbaric part. Needed to see if the patient is still in its growing phase.
- The magnitude of Cobb's angles and the vertebrae which are responsible for them.
- Detailed information about apex vertebrae and their neighbors. This information consists of the rotation and wedge angles.

This is a subset of the output of the EXCEL macro, with some modification: the intervertebral disc length between L1 and T12 is made a part of the thoracic length instead of the lumbaric. This is a matter of taste and definition. Since only the change in length in time is important and the definition is void, this is no issue.

As a general practice in the AZG, the origin is put at the base of vertebra L4. This is not according to the general standard which places the origin at S1, but since L4 is much more visible in most cases and also part of the entered vertebrae, this point is easily computable.

Another difference is the determination of the apexes. This was not done in the EXCEL macro and is also somewhat different from the general standard. Since a medical doctor does in general not enter all vertebrae, L5 is not available, which is needed to determine the apexes. In this case, the most downward entered vertebra is used instead of L5. When T1 is not entered, the uppermost entered vertebra is used instead.

4.4 Execution flow and functional analysis

The execution flow is the way in which the program is normally executed. It gives us an overview of the actions to be implemented. In this case, the *default* executing flow is quite simple since only a few different states are available: the program starts at the beginning and after completing the default steps the program is terminated. These default steps are:

1. Reading the images within the EasyVision environment.
2. Start the program.
3. Adjust the image.
4. Calibrate the images with some points.
5. Initialize the spine with some points.
6. Add the landmarks of the vertebrae.
7. Perform Wever analysis on it.
8. Save the images with all the information gathered.
9. Exit the program.

When looking at it more closely, it becomes clear that it would be convenient to be able to recalibrate, adjust and perform Wever analysis on it should always be enabled: that is why a *mode* is introduced. Possible modes are *Image adjust*, *Calibration*, *Initialization* and *Vertebra*. Each mode now has its own mode-specific actions which are possible in only that mode and some general actions which are always possible, like saving. This last part is called the functional analysis, and both can be seen in Figure 4.2.

4.5 Modules

Module definition is the last part of the design. A designer is free to choose the way in which the functionality is divided into the modules. The reason for modules is a better comprehension of the program which improves maintainability. Furthermore, data flow is now strictly defined, which makes it easy to change the program by only changing a module instead of finding all dependencies the hard way. This principle is called **separation of concern**. Every GUI program should contain at least the functionality mentioned below:

- A calculus module. This is in fact the part which all calculations are done and which is mostly why a program exists. In the design of this program, all output is calculated in this module.
- The interface module. A very big part of this program, since interaction is second most important after the calculus module.

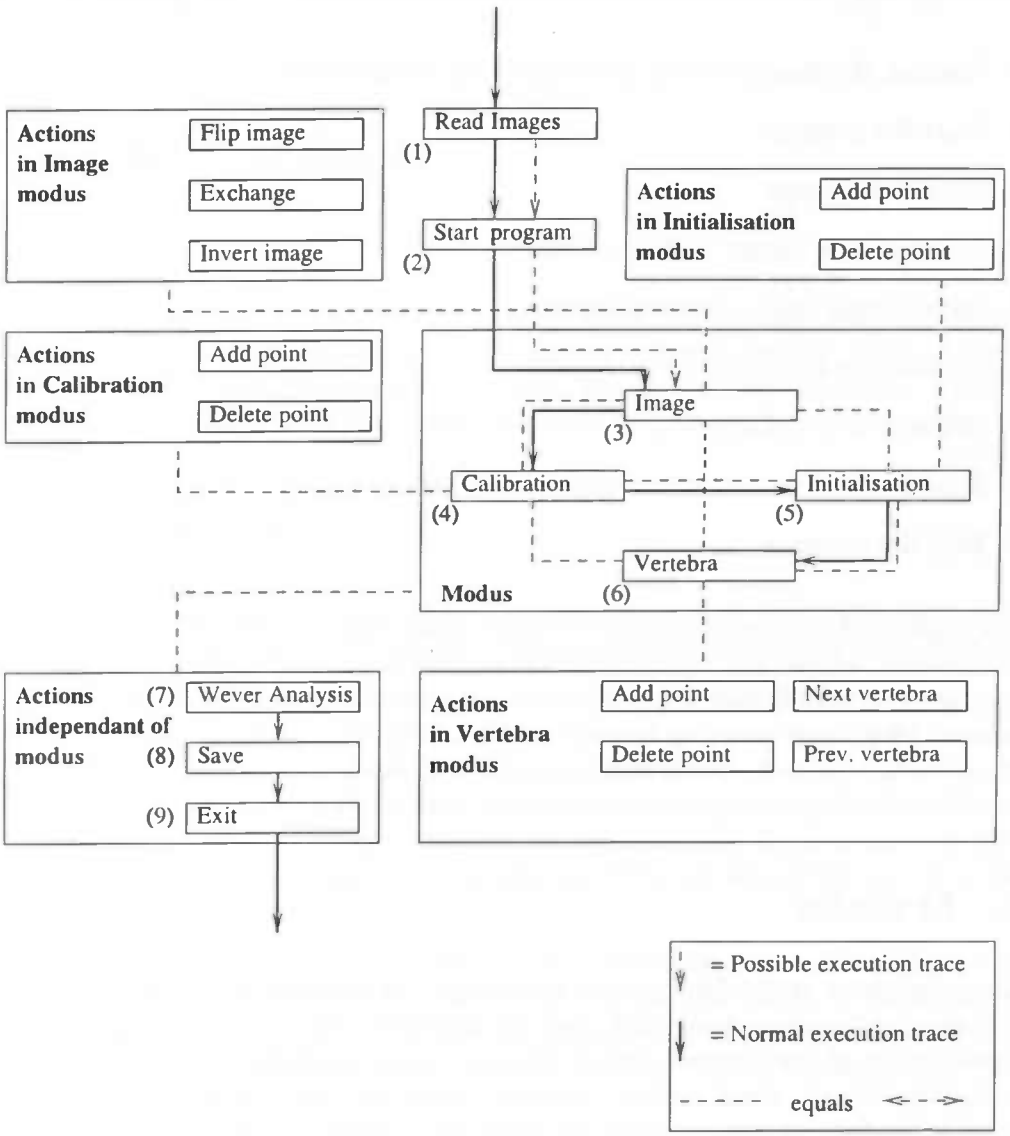


Figure 4.2: Execution flow and functional analysis

- An output module. The module which represents the output from the calculus module.
- A linear algebra module for all default operations on the somewhat more sophisticated structures.

As we see, it is now relatively easy to change the output to a completely other format without changing the design and with little change in the implementation: only the output module is affected.

Alas, this is not that simple for the interface: the interface module is in this case one big module which is quite strictly tied to the underlying windowing system. To overcome this problem, a separate module is needed which interfaces between the windowing system and the interface module of the program. However, since one of the starting points was the EasyVision environment and implementation a complete new module would require a lot of time, this is not part of the design.

How all these modules interact can be seen in Figure 4.3.

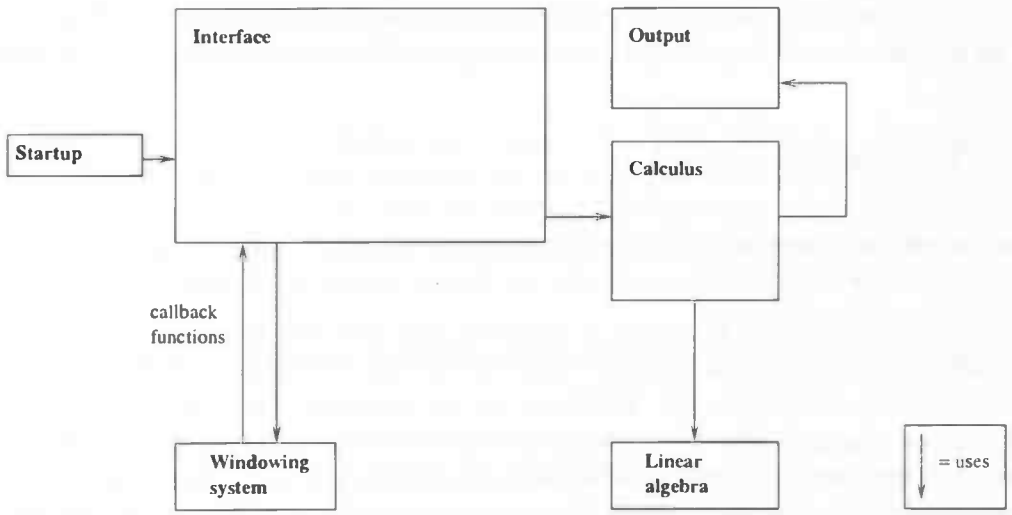


Figure 4.3: The modules of the program

This is still a very rough division into modules, which will be divided into more submodules as described in the implementation chapter.

Chapter 5

Implementation

The design already contained some modules. The implementation now decides how all these modules are subdivided into submodules, which are all part of a module. Since the programming language used is C, a submodule corresponds in this case to just one .c file. A submodule will be generated by categorizing and grouping functions with similar functionality.

5.1 Reference model

What do we take as a reference program? Since there are two programs, there are three options: take Spine3D as reference, take the EXCEL macro as reference or start from scratch.

The Spine3D program as reference to base the new program upon has the advantages that the developing environment is good (EasyVision), but the program itself lacks an easy way to change the underlying model. Furthermore, the graphical user-interface (GUI) is not very intuitive and assumes a lot of underlying EasyVision knowledge to work with. This is not surprising, the Spine3D program was meant as a *prototype* to show the possibility of 3D spinal reconstruction, but undesired since the program should be developed for use by a medical doctor.

The EXCEL macro as a reference model is worse: the environment is not right and the language in which the macro has been written is not portable at all. The macro is quite good usable though by its GUI which is integrated into the spreadsheet, but is somewhat slow. These considerations have led to the conclusion that a design from scratch was needed.

5.2 Submodules

We now take the design as the starting point, and specify the modules and its submodules more precisely *with* the connections. At first, **libraries** are part of the software which can be called from any part of the program independent of other modules. Libraries can however depend on other libraries, but not without layering.

Layering means that a module can be dependent on other modules, but no circularities may

exist. In this case there is always a bottom layer on which other layers can be built. In this case, the EasyScil environment can be seen as the bottom layer since no functionality is needed below this layer.

In short, EasyScil is a combination between the windowing system and development environment EasyVision, and the image manipulation package developed at UVA and TNO called ScilImage. It is layered as in Figure 5.1.

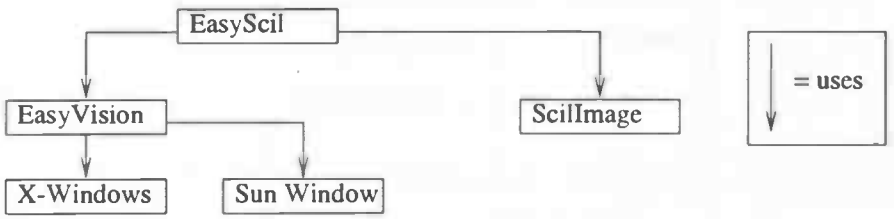


Figure 5.1: EasyScil dependencies

Another library we need is a module with geometrical functions and some linear algebra. Since such a module already existed, an expanded version of `verStruct.c` from Spine3D was used.

Each file which contains one submodule has a name of the form `spn`*Name*`.c`, where spn stands for "spine" and *Name* for the basic functions included in that file. `verStruct.c` is in this case an exception because of its origin. Now all functions contained within this file are prefixed with (an abbreviation of the) *Name*, for example functions within `spnGUI.c` all prefixed with `spnGUI`.

The modules which are not part of the library are divided into the submodules in the following way:

The `Startup` module contains `spnPanel.c` which does window management such as destroying, creating and initializing the panels. It also takes care of the startup which involves initializing the main window.

The `Output` module contains `spnGraph.c` which does all graphic output. Currently it doesn't use an EasyVision panel for output, but uses an external program called `GNUplot` for portable output.

The `Calculus` module is made up of `spnSpine.c` and `spnVertebra.c`. The latter is the base of the calculus which defines all actions on the vertebra itself, the building block of the spine. Actions include initializing, assigning, retrieving and performing calculations on it needed for Wever analysis such as calculating the tilt and wedge angle. The first, `spnSpine.c`, glues those vertebrae together to the complete spine which makes it possible to do the actual analysis, and calculate the output.

The `Interface` module is by far the largest and consists of `spnGUI.c`, `spnDetail.c`, `spnZoombox.c`, `spnGfx.c` and `spnGfxVer.c`. `spnGUI.c` contains all entry points to the callback functions which are called by EasyVision when an event occurs. Furthermore, it calls the corresponding functions in the other submodules to do the job. `spnDetail.c` handles the detail image window by selecting another image and subscribing events to this window. `spnZoombox.c`

contains all information regarding the zoombox on the frontal and lateral image and handles actions on the zoombox like moving, resizing and creating. `spnGfx.c` has all operations on complete EasyVision images and their graphics. `spnGfxVer.c` is the most important one of the submodules since it handles the vertebra graphics on the image and converts it to the internal vertebra notation using `spnSpine.c`.

How all those modules are connected to each other can be seen in Figure 5.2.

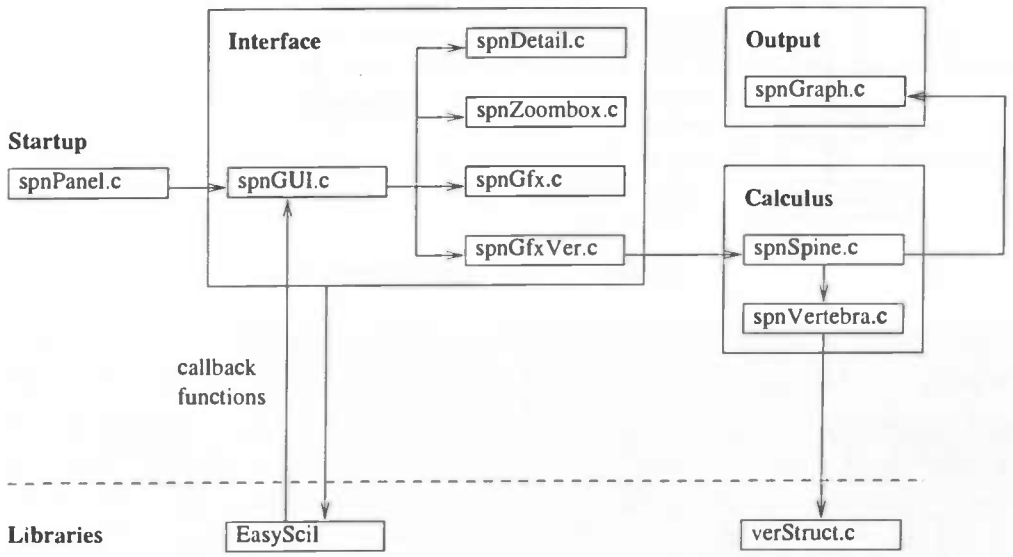


Figure 5.2: Modules and their submodules

5.3 The GUI

Just like as in the programming case, looking around and seeing what is already available speeds up the GUI development. From the existing interfaces, the GUI already available from the "limb reconstruction" module in EasyVision was selected as the base. Another possibility like Spine3D was also possible, but had too many disadvantages like a zoombox which had still to be integrated and a relatively small working area.

The GUI implemented using the GUI editor `xUiEdit` now has the areas as in Figure 5.3. It consists of some subwindows and images like the frontal image subwindow, the lateral image subwindow, the detail image, the modus subwindow, detail image control subwindow, statusline and general action subwindow.

The frontal image subwindow and the lateral image subwindow contain their corresponding X-ray images. These images are always displayed completely in those windows. A zoombox is available to quickly select another part of the image without losing the overview.

The detail image subwindow contains a zoomed part of the frontal or lateral image. This is the biggest subwindow and all actions are done here like selecting landmarks.

The detail image control subwindow "control" the view of the detail subwindow in a coarse

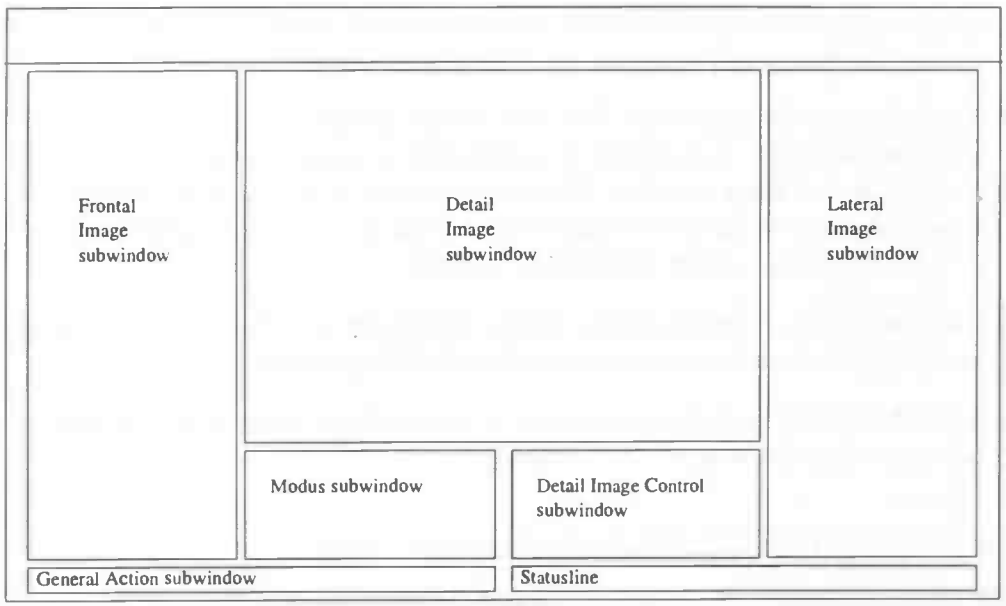


Figure 5.3: Setup of the screen

form: most global actions like zooming, applying an edge detect filter and setting the brightness and contrast are available here. These settings are kept when switching to another X-ray view.

The statusline gives an indication of what went wrong in case of an error or what the program now expects of the user. It is mainly meant to be informative and not alerting, which is why a statusbar was chosen instead of a pop-up window.

All general actions like saving, exiting, changing modus and performing Wever analysis are located in the general action subwindow. Those can always be activated.

The modus subwindow on the other hand contains all actions which are related to the selected modus. In this way modus can easily be added and removed without much hassle, since a new mode can reuse this part of the screen for its own buttons.

A lot of aspects which made the GUI look and behave like it does are based on the GUI design principles presented earlier in section 4.1.1. Some remarks about the GUI which show this are:

- Functions most used are as close to "active area" as possible; functions less used are less important and are therefore placed more remotely. The "active area" is defined as the area on the screen in which most of the mouse actions takes place and corresponds in this case to the detail image subwindow.
- The possibility to enter the points on the vertebra in a random order.
- Several working modes added. The modes are easily distinguishable and each have a clear set of functions associated with them. Implemented are the modus for general

image adjustment, initialization, calibration and vertebra manipulation. This follows closely the functional analysis.

- The zoombox is integrated. The two X-rays images are put at a side window with a zoombox attached. A zoombox is represented as a green rectangle which is just the area zoomed in the main window. This main window is as big as possible to maximize the screen use. Zooming is now simple by moving the zoombox (possibly in several ways) or by changing the zoom factor with a slider.
- Integrated edge-detection filter. Edges, which are just the features we are looking for, are better seen after an edge-detect filter has been applied.
- An esthetic one: each button is equal in size as far as possible and is aligned with other buttons either horizontally or vertically.

The resulted GUI now looks like Figure 5.4.

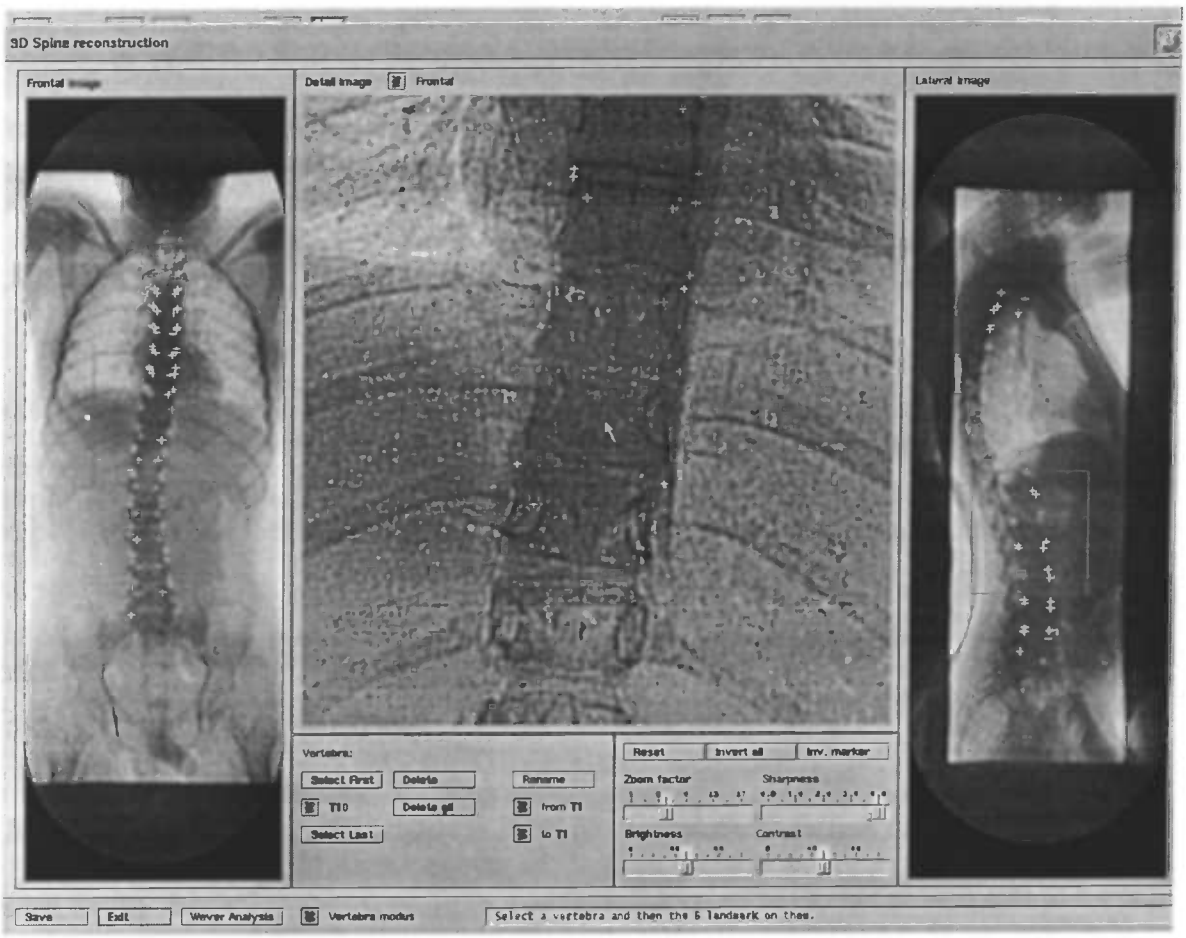


Figure 5.4: Screendump of developed program

Chapter 6

Discussion

6.1 Differences between design and implementation

As always, the design differs somewhat from the implementation. Differences consist of unimplemented features for several reasons.

6.1.1 Calibration

Calibration is very difficult and gives its own problems which can be divided into the practical and the theoretical part.

The practical part is the placement of the calibration object, or more general, which features to use in the calibration. A ruler has the nice characteristic of having a lot of feature points on that ruler. However, the ruler must be carefully placed in object space, since it must be visible on both X-ray images. The visibility of the ruler is not the problem however. The problem is the visibility of the feature points on them: a ruler is flat, which

The theoretical part stems from the fact that the calibration method only works for perspective projection. The images obtained from the reduced FOV method however, are approximated at best by a cylindrical projection. In fact, even this is not true, since the images are constructed with a mathematical function from all subimages.

When calibration, or another approximation to match the two images had been calculated, its use is for accurate measurement of the spinal length, and to improve the GUI. The GUI is helped since automatic placement of the zoombox is possible on an image, from information produced by the other image. Once calibration is implemented, this is easy to add.

6.1.2 Initialization

Initialization now consists of a modus with all stubs attached which do nothing. It is mainly there to demonstrate how easy it is to add more working modes.

The mode itself is meant to have some initialization points which are then used to have an

approximate location of the spine. Currently, this is not used since the only place where it has some meaning is the automatic placement of the zoombox to speedup selection of the features without having to pan the detail image. This however has been implemented by an extrapolation of the location of the center of the new vertebra, which renders this mode useless.

It might however become an important mode when semi-automatic feature extraction is introduced which could take the data from the Initialization mode as a starting point from where to look for features.

6.2 The future

The program produced is not complete. Doing more research, implementing the results and validating the resulting program are required to constantly update the program to contain the new knowledge. Some matters which currently can be and are researched are mentioned below.

6.2.1 Features which are easy to implement

Cobb's angle on both images instead of a single one is straightforward to make since it doesn't involve any other calculations than already available. This is because of the features used: the four corners are present on both the frontal and the lateral image and those are the only features to calculate Cobb's angle. A problem though is the interpretation and meaning of the Cobb's angle on the lateral image.

Improving the GUI by feedback is probably easy. After some time of use, the GUI should be evaluated to guarantee the GUI works like the user expects. Mostly this involves removing little annoyances which should not cost much time.

Using an **arbitrary angle** between the projection direction between the two X-ray images instead of 90° is quite easy to implement since it involves mainly the reconstruction, which is separated from the program. Of course, the naming should also be adapted to reflect the new situation.

6.2.2 One points rotation

The program now works with rotation based on the two pedicle points. However, there also exists methods (like Nash-Moe and its derivatives) for vertebral rotation estimation based on one pedicle. Although methods based on one pedicle are less accurate (and even plain wrong sometimes) than the methods based on two pedicles, it *does* give an estimation.

To incorporate the Nash-Moe method for one-points rotation and Stokes' method in case of two, would however not be wise without validation: it wouldn't be surprising to find that the Nash-Moe method gives completely different values for the same vertebrae. In this case, a way to validly adjust Nash-Moe to more match Stokes' method could be searched for. Another

possibility might be that it is not possible at all to merge them; in that case we should keep Stokes' method.

6.2.3 Reconstruction

Reconstruction using the current points is already possible as Spine3D shows, but has not yet been implemented. Care must be taken, however, to keep the GUI simple. Best would be to use the current detail image window to display the reconstruction. This is possible, since reconstruction is not needed during data entry, but is more comparable to an analysis which is only performed after all points have been entered.

6.2.4 Interaction

Interaction like in Spine3D makes it possible for the medical doctor to adjust the input to have the reconstruction of the spine match his idea of the actual spine. I think however that moving points based on the model is not as easy as it looks: moving the entered points as in this program is more clear since the **meaning** of those points are more clearly defined. It might be that in the future when medical doctors get to work on models instead of features, this is going to change. However, this is just not the case nowadays.

6.2.5 Interpolation

Interpolation serves several goals: to speed up the time required for the medical doctor to enter the points necessary for the required information and reconstruction, and to make a reconstruction possible even when some vertebrae are missing.

The nicest way to do interpolation within this program is to automate it: the user enters the features known, and the program automatically visualizes its idea about the unknown features. Note that this also gives implicit interaction in which the user determines the necessity of entering more points by visually inspecting the feedback. The feedback could be present by instantly showing the interpolated feature points in another color.

The problem, though, is the interpolation function to be used. This holds especially for a scoliotic spine, since these are known to have irregularities. Spine3D makes use of a spline for interpolation of the screw of the spine.

6.2.6 Automatic feature extraction

Semi-automatic feature extraction has still a long way to go. Newer techniques look promising in being able to at least assist the medical doctor, but are not yet on the level where automatic extraction is possible. To change this however, all possible improvements have to be merged: it is not merely a case of research in the field of image processing, but also better equipment ("the hardware") and the choices made (like the features used) are important.

Appendix A

Manual

```
DOCUMENTATION FOR DELIVERABLE spine
Rutger Nijlunsing <rutger@null.net>

PREREQUISITES
=============
* gnuplot pre-3.6 beta 332 installed, and existing in PATH
  (gnuplot 3.5 _won't_ work). This package contains a precompiled version
  for Solaris 2.5.1 in mybin/bin/* . When retrieving a fresh
  distribution, don't forget to add -DANSI_C to the Makefile.

* environment variable SPINEDATADIR pointing to the directory of results
  Example: export SPINEDATADIR=/usr/csg465/gnuplot
  The result directory defaults to /tmp when not specified.

* a script called 'showinfo' also in the PATH.

GENERAL USAGE
=============

Login and start xEasyScil and read in the images with the database
handling facility. At least 1 image should be read in; in case 2
images are selected, the first should be the frontal image and the
second should be the lateral one.

Now we start the spine modelling program by clicking on the magic hat
and selecting the rightmost application.

From now on, a panel is created with 3 main viewports. From the left
to right they are 1) the frontal viewport, which should contain the
frontal image; 2) the detail image viewport which contains the
selected image; and 3) the lateral viewport, which should contain the
```

lateral image.

The detail image viewport is the viewport on which all manipulation takes place. The other viewports are for the overview and for selecting the zoom area displayed in the detail image viewport. The zoombox visible can be dragged to another place with the left button of the mouse, or can be placed directly by adding the control key on the keyboard while selecting with the left button. The same holds for the detail image viewport in which in the same way the image can be recentered.

To directly manipulate the features of the detail image viewport, some sliders have been added like 1) zoom for in- and outzooming; 2) sharpness for applying an edge filter on the image; 3) brightness and 4) contrast.

Now there are 2 major modes which can be selected by using the cyclic option gadget on the status line.

* The 'image and calibrate modus', used for:
  * Image control
    * correcting the picture so that the frontal resp. lateral image from the database corresponds to the frontal resp. lateral image in the viewport. This is done with the button 'exchange'.
    * switching between negative and positive's of the X-ray. This is done with the button 'invert all'.
    * mirror the image so left resp. right corresponds to left resp. right on the viewport. This is done with the button 'flip selected'.
  * Calibration
* The 'vertebra and point modus', used for:
  * Vertebra selection: there is always one vertebra selected in time. The vertebrea are called T1 to T12 and L1 to L5. The selected vertebra can be identified by its distinct color. Vertebras can be selected by the cyclic option gadget.
  * Vertebra deletion, used for removing added vertebras. Can be found under the 'delete' and 'delete all' button.

The coordinates of the vertebrae can now be entered by first selecting the right vertebra (T1 .. L5) and then selecting in the detail image viewport the corners of the vertebra and, if visible, the inner points of the pedicles.

This process should be repeated until all vertebras for which at least the corner points are visible, have been entered.

An easy way for selecting a next or previous vertebra, is by Ctrl-Left button clicking around the center of the new vertebra, and then selecting the right vertebra by using the keys 'a' and 'z'.

The image can now be saved with the button 'save' and the Wever analysis can now be performed with the button 'Wever analysis'.

'Wever analysis' pops up some windows containing the graphs. These can be viewed by double-clicking on the windows and can be dismissed by pressing the key 'q' while the window is active. Information about the apexes and Cobb's angles is available in the directory ~/gnuplot.

MOUSE/KEYS
==========

In frontal / lateral image:

```
  Mouse: Ctrl Left button       : set zoombox
         Left button            : set zoombox

  On zoombox:
    Mouse: Left button          : move zoombox
```

Detail image:

```
  Mouse: Ctrl Left button       : set zoombox
         Left button            : move point / add point
         Middle button          : move point / delete point
         Right button           : select vertebra

  Keys:  'a'                    : Next vertebra
         'z'                    : Previous vertebra
         F5 / F6                : Change interpolation method of display
         DEL / 'd'              : delete point
         's'                    : select vertebra
```

Appendix B

Development environment

B.1 Tools

Computer and OS

Computer	:	Sun SPARCstation 5, 192Mb
Operation System	:	Solaris 2.5.1 (SunOS 5.5.1)
Server name	:	spiny.cs.rug.nl

Software

Compiler	:	Sun cc
Environment	:	EasyVision R3.1.0
Editor	:	xemacs
Shell	:	bash
Output	:	GNUplot 3.6 beta322
Typesetting	:	LaTeX2ϵ with Harvard style
Figures	:	xfig 3.1.4, xpaint 2.4.4, xwd
Shell	:	bash 1.14.7
Editor	:	xemacs, emacs 19.34
Converters	:	giftopnm, pnmtops, transfig
Miscellaneous	:	GNUmake 3.74

B.2 Development

The development took place on an EasyVision workstation. This environment which is the platform for commercial PMS products was developed by **Integrated Clinical Solutions (ICS)**, a part of PMS. On top of this, EasyScil was built. EasyScil is a software package which is specially oriented towards image processing. As the core for its image processing it uses the library ScilImage which was developed by TNO and UVA in the Netherlands.

EasyVision is particularly strong in presenting a uniform GUI which is important for the end-user which is made by the GUI builder `xUiEdit`. Another aspect of EasyVision is its global database of patients which contains images and patient information alike.

Both X-windows and the native Sun graphics can be used as the interface. This last option is useful when not the flexibility but the power is required and the only user is the end-user.

The X-ray images used are approximately 512×1600 pixels and have a depth of 8 bit greyscale which even allows some zooming on the screen without loss of quality. This high resolution is achieved by using the reconstruction algorithm.

There are several ways to use and develop with EasyScil, each with its advantages and disadvantages:

1. The quick way by using the *integrated interpreter*. This C interpreter is an expanded version found in SciIlmage itself. It allows quick testing of the possibilities of EasyScil (within limits).
2. The use of *UFO*'s which are like small programs but do not require compiling. Sort of macros.
3. *Dynamic libraries* which require compiling but have the advantage over interpreted code that it understands K&R C, is faster and does not require the user to restart the EasyScil enviroment.
4. Building a *standalone* executable. Only useful for the end-user since it requires compiling and restarting the program totally at each development phase.

Of course the dynamic libraries were used here.

For the completeness we give list of the submodules and their size in lines:

`spnDetail.c`	111
`spnGUI.c`	539
`spnGfx.c`	42
`spnGfxVer.c`	499
`spnGraph.c`	124
`spnPanel.c`	180
`spnSpine.c`	262
`spnVertebra.c`	658
`spnZoombox.c`	136
`verStruct.c`	548
`spine.h`	316
total	3415

Appendix C

Modules

Each submodule is contained in its own `.c` file. Since there is only one header (`.h`) file with all exported functions and structures this is discussed below. The other kind of "exported functions", the callbacks, are defined in the submodules itself. These are also discussed.

C.1 Exported functions and structures

These can be found in `spine.h`, the main header file.

`typedef int bool;`
The boolean type.

`typedef enum { ... } Pointtype;`
Possible point type, which include temporary ones used during calculation. Always starts before determination as `PT_UNUSED`.

`typedef enum { ... } AngleType;`
Angle type like wedge angle, tilt angle.

`typedef enum { ... } ImageType;`
The type of the image. Only frontal and lateral are implemented. `IM_3D` is for future use in the reconstruction.

`typedef enum { ... } VertebraType;`
Place of the vertebra, can be in the lumbar or the thoracic part.

`typedef struct POINT2D { double x,y; } POINT2D;`
Type of a 2D point as used in `verStruct.c`

`typedef struct POINT3D { double x,y,z; } POINT3D;`
Type of a 3D point as used in `verStruct.c`

`typedef struct EnteredPoint { ... } EnteredPoint;`
An entered point in frontal or lateral plane as used in `spnVertebra.c`.

`typedef struct CalcPoint { ... } CalcPoint;`
Calculated point from an entered point after classification. Could be extented for interpolation. See `spnVertebra.c`.

`typedef struct Vertebra { ... }`
This is the most important structure, used by several submodules. General information is handled by `spnSpine.c`, the entered points and information needed for the Wever analysis by `spnVertebra.c`

Functions from `verStruct.c`

This submodule contains classical operations on 2D and 3D points and vectors. Only the relevant functions are mentioned.

`POINT2D scal2D(double l, POINT2D A);`
Returns $l * A$ with l a scalar.

`POINT2D add2D(POINT2D A, POINT2D B);`
Returns vector $A + B$.

`POINT2D subs2D(POINT2D A, POINT2D B);`
Returns vector $A - B$.

`double dot2D(POINT2D A,POINT2D B);`
Returns $< A, B >= A_x * B_x + A_y * B_y$.

`double norm2D(POINT2D A);`
Returns the length of vector A.

`double dist2D(POINT2D A,POINT2D B);`
Returns the Euclidean distance between point A and B.

`POINT2D normalize2D(POINT2D A);`
Returns vector A scaled to length 1.

`POINT2D middle2D(POINT2D A,POINT2D B);`
Returns the middle point between A and B.

`POINT3D scal3D(double l, POINT3D A);`
Returns $l * A$ with l a scalar.

`POINT3D add3D(POINT3D A, POINT3D B);`
Returns vector $A + B$.

`POINT3D subs3D(POINT3D A, POINT3D B);`
Returns vector $A - B$.

`double dot3D(POINT3D A,POINT3D B);`
Returns $< A, B >= A_x * B_x + A_y * B_y + A_z * B_z$.

`double norm3D(POINT3D A);`
Returns the length of vector A.

`POINT3D normalize3D(POINT3D A);`
Scales vector A to length 1.

`POINT3D prod3D(POINT3D A,POINT3D B);`
Returns the outerproduct of vector A and B.

`double dist3D(POINT3D A,POINT3D B);`
Returns the Euclidian distance between point A and B.

`POINT3D middle3D(POINT3D A,POINT3D B);`
Returns the middle point between A and B.

`double angle3D(POINT3D u, POINT3D v);`
Returns the angle between vector u and v.

`bool equal(double a,double b);`
Returns TRUE iff a equals b, within an `EPSILON` tolerance.

`bool equal2D(POINT2D a,POINT2D b);`
Returns TRUE iff a equals b, within an `EPSILON` tolerance.

`double lineAngle2D(POINT2D a,POINT2D b);`
Returns the angle in radians between the x-axis and the line from a to b.

`POINT2D make2D(double x,double y);`
Returns (x, y) as a vector or point.

`int sign(double x);`
Returns the sign of x.

`double rad2deg(double x);`
Returns x radians in degrees.

`double area2(POINT2D a, POINT2D b, POINT2D c);`
Returns half the size of the area of the triangle with corners a, b and c. See (O'Rourke 1994).

Functions from `spnPanel.c`

`char froimage[],latimage[],detimage[];`
The SciIImage names of the frontal, lateral and detail image respectively.

`IMAGE *curimage(void);`
The SciIImage image of the image currently displayed in the detail image window.

`void spnPanelProgramInit(void);`
Program initialisation.

`void spnPanelProgramDone(void);`
Program cleanup after exit.

`int spnReadBothImages(int i1,int i2);`
Read image RO_i1 and/or RO_i2 into the scilimage images. Return the number of images read (0, 1 or 2).

`void spnPanelClose(char *panelname);`
Close panel with panelname `panelname` if it exists.

`void spnPanelCloseAll(void);`
Close all openened panels belonging this program.

`void spnPanelCreate(char *panelName);`
Create panel with name `panelname`

`void spnPanelInit(char *panelname);`
Create and initialises panel with panelname `panelname`. Initialisation is panel specific.

`void spnPanelSelect(char *panelname);`
Close current panel and open another one.

Functions from spnDetail.c

`void spnDetailGetSize(int *width,int *height);`
Return the size of the detailed image window.

`void spnDetailUpdate(void);`
Display current image in detail window. Called when a frontal / lateral switch happens.

`void spnDetailSelect(Image im);`
Select image im as the current image in the detail window but only if another image has been selected.

`void spnDetailUnsubscribe(void);`
Unsubscribe all events.

`void spnDetailRacSubscribe(void);`
Subscribe events for detail image for current Read And Calibrate modus.

`void spnDetailVerSelectSubscribe(void);`
Subscribe events for detail image for vertebra mode.

Functions from spnZoombox.c

`void spnZoomboxMake(IMAGE *image);`
Make a zoombox on image image.

`void spnZoomboxDestroy(IMAGE *im);`
Destroy the zoombox on image im.

`void spnZoomboxZoom(IMAGE *image,double zoomfactor);`
Set the size of the zoombox according to the zoomfactor newZoomFac on image image.

`void spnZoomboxGetCenter(double *cx,double *cy);`
Return the center of the zoombox on the current image.

`void spnZoomboxSetCenter(IMAGE *im,double cx,double cy);`
Set the center of the zoombox on image im.

`void spnZoomboxUpdate(IMAGE *im);`
Move detailimage to the newly set zoombox position.

`void spnZoomboxSubscribe(IMAGE *image);`
Subscribe events to existing zoombox on image im.

Functions from spnGfx.c

`void spnGfxCopy(IMAGE *in,IMAGE *out);`
Copy all EasyScil graphics on image in to image out.

`void spnGfxCopyIm(IMAGE *in,IMAGE *out);`
Copy image with graphics.

`void spnGfxSave(IMAGE *im);`
Save the graphics on image im temporary for later use.

`void spnGfxRestore(IMAGE *im);`
Restore all the graphics on image im saved by spnGfxSave().

Functions from spnGfxVer.c

`void spnGfxVerMarkerColorChange(void);`
Use other palette of colors for the markers on vertebra.

`void spnGfxConvert(void);`
Read vertebra graphics from images and perform Wever output on it.

`int spnGfxCountVerPoints(IMAGE *im,char *group);`
Return the number of given points on image `im` with groupname `group`.

`void spnGfxVerDelete(IMAGE *im,char *group);`
Delete vertebra with groupname `group` on image `im`.

`void spnGfxVerDeleteAll(IMAGE *im);`
Delete all vertebra graphics on image `im`.

`void spnGfxVerTypeByNumber(VertebraType *vt,int *vnr,int number);`
Return the type and number of vertebra `number`.

`void spnGfxVerGroupnameByName(char *group,char *ver);`
Return the groupname of vertebra with name `ver`.

`void spnGfxVerGroupnameByNumber(char *group,int number);`
Return the groupname of vertebra with number `number`.

`void spnGfxVerAddPoint(char *name,double x,double y);`
Add to vertebra name point (x, y).

`void spnGfxVerColorize(void);`
Recolorize all vertebras.

`void spnGfxVerSelect(IMAGE *im,char *group,bool select,bool moveable);`
(Un)Select vertebra with groupname `group` by colorizing.

`void spnGfxVerSelectOne(IMAGE *im,char *name,bool moveable);`
Select only vertebra name by colorizing and decolorizing the rest on 1 EasyScil image.

`void spnGfxVerGetMiddle(IMAGE *im,char *group,bool *jump,POINT2D *middle);`
Return the middle of the given points of vertebra determined by im and group.

`void spnGfxVerCopy(IMAGE *in,IMAGE *out,char *group);`
Copy all vertebra points of vertebra with groupname group from image in to out and add subscriptions.

`void spnGfxVerCopyAll(IMAGE *in,IMAGE *out);`
Copy all vertebra points from image `in` to `out` and add subscriptions.

`void spnGfxVerDelPoint(char *name,double x,double y);`
Delete vertebra point nearest to (x, y) of vertebra `name`.

`void spnGfxVerPosInterpolate(bool *jump,POINT2D *middle);`
Return the midpoint of the currently selected vertebra in `middle`.

`void spnGfxVerJumpSelect(double x,double y);`
Select vertebra closest to (x, y).

`void spnGfxVerRenameAll(int shift);`
Circular shift the names of the vertebra for `shift` vertebras.

Functions from spnSpine.c

`void spnSpineInit(void);`
Call once for complete init of the spine.

`Vertebra *spnSpineGetVer(VertebraType type,int number);`
Return vertebra with type type and number number.

`void spnSpineWever(void);`
Perform Wever output on spine.

`POINT2D spnSpineOriginFr(void);`
Determine the origin of the spine in the Frontal Plane.

Functions from spnVertebra.c

`void spnVerInit(Vertebra *ver);`
Initialise vertebra ver.

`int spnVerAddPoint(Vertebra *ver, POINT2D coord);`
Add point coord to entered points in vertebra ver in current image.

`bool spnVerDeletePoint(Vertebra *ver, int handle);`
Delete entered point handle from vertebra ver in current image.

`bool spnVerCalcTypes(Vertebra *ver);`
Determine the types of the entered points using the Wever method on the current image.

`void spnVerCalcCoords(Vertebra *ver);`
Interpret entered points and extract calculated points from it on the current image.

`bool spnVerUsedFr(Vertebra *ver);`
Return TRUE if vertebra ver has enough frontal information.

`CalcPoint *spnVerGetCalc(Vertebra *ver,PointType type);`
Return the calculated point for pointtype type.

`double spnVerGetAngle(Vertebra *ver,AngleType type);`
Return the angle of type type.

`double spnVerGetHeight(Vertebra *ver);`
Return the height of the vertebrae.

`void spnVerWever(Vertebra *ver);`
Perform all Wever calculations possible on one vertebra.

`void spnImageSelect(Image im);`
Select imagetype im as the current working image for vertebras.

`Image spnImageSelected(void);`
Return current working image.

Functions from spnGraph.c

`void spnGraphInit(void);`
Initialise graph output.

`int spnGraphOpen(char *name);`
Open a graph with name name.

`int spnGraphClose(int handle);`
Close graph with handle `handle`.

`int spnGraphCloseAll(void);`
Close all graphs.

`int spnGraphPlot(int handle,int ver, double x, double y);`
Add point (x, y) of vertebra `ver` on graph with handle `handle`.

`int spnGraphPrint(int handle,char *string);`
Add string `string` to graph with handle `handle`.

`int spnGraphComment(int handle,char *string);`
Add comment to graph.

Functions from `spnGUI.c`

`void spnMsgError(char *error);`
Display error message `error`.

`void spnMsgWarning(char *warning);`
Display warning message `warning`.

`void spnMsgComment(char *comment);`
Display comment `comment`.

C.2 Callback functions

Callback functions are only found in `spnGUI.c`.
`void spnGUIdetailupdate(void)`
Frontal / lateral view update.

`void spnGUImodus(void)`
The read / calibrate / vertebra mode cycle.

`void spnGUIflip(void)`
Flip current image around x-axis.

`void spnGUIinvert(void)`
Invert current image.

`void spnGUIzoom(void)`
Zoom according to zoomslider.

`void spnGUIbrightness(void)`
Set brightness according to brightness slider.

`void spnGUIcontrast(void)`
Set contrast according to constrast slider.

`void spnGUIsharpness(void)`
Set sharpness according to sharpness slider.

`void spnGUIreset(void)`
Reset zoom, contrast, brightness and sharpness settings.

`void spnGUIexchange(void)`
Exchange frontal and lateral image.

`void spnGUIread(void)`
Read images RO_0 and RO_1 from database.

`void spnGUIzoomboxmove(IMAGE *im)`
Move detailimage to new zoombox position.

`void spnGUIzoomboxrecenter(IMAGE *im)`
Move zoombox to mouse position and update detail image.

`void spnGUIaddRacPoint(void)`
Add calibration point.

`void spnGUIaddVertPoint(void)`
Add vertebra point.

`void spnGUIdelVertPoint(void)`
Delete vertebra point nearest the cursor.

`void spnGUImoveVertPoint(id detpoint,id curpoint)`
Moves vertebra point in both images.

`void spnGUIselectfirst(void)`
Select first vertebra.

`void spnGUIselected(void)`
Select another vertebra.

`void spnGUIselectlast(void)`
Select last vertebra.

`void spnGUIselectnext(void)`
Select next vertebra.

`void spnGUIselectprev(void)`
Select previous vertebra.

`void spnGUIwever(void)`
Perform Wever output.

`void spnGUIverDelete(void)`
Delete current vertebra.

`void spnGUIverDeleteAll(void)`
Delete all vertebrae.

`void spnGUIsave(void)`
Save the current images.

`void spnGUIexit(void)`
Exit.

`void spnGUIcolorMarkerChange(void)`
Select another marker color.

`void spnGUIvertJumpSelect(void)`
Select vertebra under mousecursor.

`void spnGUIrename(void)`
Rename all vertebra by shifting their names.

Appendix D

Coding style

To make the code as readable and maintainable as possible, consistency in programming style, naming and layout is very important. The style followed in the program code is explained with examples in this appendix.

D.1 C specific

A `typedef` of a `struct` has the same name as the `struct` because C has the strange convention to have one name space *except* for `structs`. Using the same name gives a one-to-one mapping of those two name spaces. Example: the `struct Vertebra` in `spine.h`

All types start with a capital. This makes the distinction easy between variables and their type and makes it possible for the instantiation to bear the same name as its type. Example: the type of the spine is `Spine` and an instantiation is `spine`.

Define the used `#define`'s as close as possible to the place where they are used and not in a global include file. Most defines are mostly used in one or two places, which is earlier understood when the `#define` is defined at that place. Example: `GRAPH_MAX` in `spnGraph.c`.

Restrict the scope of a variable as much as possible: keep variables with a local function local to the program and the ones which have to perform a global function, global. Example: the variable `dist` in function `epFindClosestUnused()` in `spnVertebra.c`.

Use *short names for local variables* and longer ones for global variables. A variable called `i` is good for a local loop, but not sufficiently clear for a global variable. Same holds for `tempVar` which is to long and non-descriptive for a local variable. Example: `i` in `epDelete()` and `zoomboxgroup[]` in `spnZoombox.c`.

Use `assert` as much as possible to check pre- and post conditions. This makes it much simpler to debug since these pre- and post conditions are always tested during the execution of the program. False assumptions can now be found much quicker. Another advantage is that the program is prematurely terminated at the moment the error is detected. Example: `assert(collect!=nil)` in `spnZoombox.c` to ensure the zoombox is always there.

Functions have no side effect because side effects make the program difficult to follow and

debug. A function always takes some parameters and returns *one* value. When more results are needed, a procedure is used. Example: `spnDetailGetSize()` is a procedure since it returns two values; `curimage()` is a function since it returns one value.

`assert`'s *have no side effect.* A side effect would introduce errors when the debugging is turned off. Example: `assert(FALSE)`.

Prefix enumerates constants with two letters to overcome the problem of the same name space for every constant while a separate one would be much more logical. Example: `VT_LUMBAR` and `VT_THORAIC` in `spine.h` for the Vertebra Types.

No random values which are not easy changeable. Lengths of temporary strings should be easily changeable. Example: `TEMP_MAXLEN` in `spine.h`.

D.2 Comments

Enough comments which describe in human words what this part of the program tries to achieve. At least of each function defined should be described what it does and what the meaning of all parameters is in it.

No *unneeded comments* which describe the code line by line, but a descriptive one which gives the meaning of those code lines.

Mark all places in the code which do not follow the guidelines or are not intuitive with `FIXME` in a comment.

D.3 Miscellaneous

Reuse naming conventions from the existing programs `Spine3D` and the EXCEL macro as much as possible. Examples: same vertebra numbering and corner names as the EXCEL macro.

Layout the code according to the default found in `emacs`. Since emacs is widely used as a text editor and it uses a reasonable default this one is followed. Example: all the code, except for the part taken from `Spine3D`.

Give examples when a lot of functions are defined but when the relation between them is not clear. Example: start of `spnVertebra.c`.

Bibliography

André, B., Dansereau, J. & Labelle, H. (1994), 'Optimized vertical stereo bsae radiographic setup for the clinical three-dimensional reconstruction of the human spine', *J. Biomechanics* **27**(8), pp. 1023–1033.

Brown, R. H., Burstein, A. H., Nash, C. L. & Schock, C. C. (1976), 'Spinal analysis using a three-dimensional radiographic technique', pp. 355–365.

Chen, L., Armstrong, C. W. & Raftepulous, D. D. (1994), 'An investigation on the accuracy of the three-dimensional space reconstruction using the direct linear transformation technique', *J. Biomechanics* **27**(4), pp. 493–500.

Drerup, B. (1984), 'Principles of measurement ofvertebral rotation from frontal projections of the pedicles', *J. Biomechanics* **17**(12), pp. 923–935.

Drerup, B. (1985), 'Improvements in measuring vertebral rotation from the projections of the pedicles', *J. Biomechanics* **18**(5), pp. 369–378.

Hatze, H. (1988), 'High-precision three dimensional photogrammetric calibration and object space reconstruction using a modified DLT-approach', *J. Biomechanics* **21**(7), pp. 533–538.

Labelle, H., Dansereau, J., Bellefeur, C., Poitras, B., Rivard, C.-H., Stokes, I. A. & de Guise, J. (1995), 'Comparison between preoperative and postoperative three-dimensional reconstructions of idiopathic scoliosis with the Cotrel-Dubousset procedure', *SPINE* **20**(23), pp. 2487–2492.

Moreno, Stefano & Piero (1995), 'New procedure for computation and automatic classification of spinal clinical parameters by image processing of digitised radiography', *Threedimensional analysis of Spinal Deformities.*

O'Rourke, J. (1994), *Computational geomtery in C*, Cambridge.

Pivet, N. (1996), 3D-modelling of the spine from x-ray radiography, Technical report, LEP - Philips Medical Systems.

Pluim, J. & Westenberg, M. (1996), Segmentation of vertebrae, Technical report, University of Groningen. Department of Computing Science.

Richardson, M. L. (1994), Scoliosis, -.

Stokes, I. (1985), 'Biplanar radiography for measuremt of spinal shape and motion', *Automedica* **5**, pp. 37–49.

Stokes, I. (1991), '(Untitled)', *SPINE.* A letter to the editor.

Stokes, I. (1994), 'Three-dimensional terminology of spinal deformity', *SPINE* **19**(2), pp. 236–248.

Stokes, I. A., Bigalow, L. C. & Moreland, M. S. (1986), 'Measurement of axial rotation of vertebrae in scoliosis', *SPINE* **11**(3), pp. 213–218.

van Eeuwijk, A. H., Lobregt, S. & Gerritsen, F. A. (1997), 'A novel method for digital X-ray images of the complete spine', *CVRMed II and MRCAS III.*

Vandegreind, B., Hill, D., Raso, J., Durdle, N. & Zhang, Z. (1995), 'Application of computer graphics for assessment of spinal deformities', *Medical & Biological Engineering & Computing* **33**, pp. 163–166.

Veldhuizen (1997), 'Private communication'.

Wever D.J. (1997), 'Private communication'.