# Spitting behaviour and fang morphology of spitting cobras

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität

vorgelegt von

Ruben Andres Berthé

aus

Düsseldorf

Bonn, April 2011

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

| | |
|---|---|
| 1. Gutachter: | Professor Dr. Horst Bleckmann |
| 2. Gutachter: | Professor Dr. Wolfgang Böhme |
| Tag der Promotion: | 20. September 2011 |
| Erscheinungsjahr: | 2011 |

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Spitting cobras are well known, as their name suggests, for their ability to "spit" at a possible threat. In real it is no spitting like in mammals as no saliva is involved. The ejected fluid consists of the snake's venom, ejected through the fangs (Koch and Sachs, 1927; Rasmussen et al., 1995). In the regions where spitting cobras live, this ability was known for a long time. The cobra depicted on the pharaoh's crown in ancient Egypt is a spitting cobra (Murray, 1948) and the possible threat of encounters with a spitting cobra could have found its way into the myths of the mediaeval times as the basilisk or cockatrice (Alexander, 1963) which are told to kill on sight. The first mentioning of a cobras ability to spit in modern scientific literature is likely to be the naming of a snake from Java as *Naja sputatrix* which literally means "spitting *Naja*" (Boie, 1827). The first to taxonomically define a species known as spitting today (*Naja siamensis*) was Laurenti (1768) but his description does not state if this species is able to spit venom. The ability to spit venom in a directed manner seems to be unique to spitting cobras. Sometimes a similar ability is reported for the Asian viper *Protobothrops mangshanensis* (Chen, 1997) but it has not been proven yet. Few reports of other snakes having spat could be the result of venom drops hanging at the fangs of the agitated snake which were then propulsed forward when the snake lunged out.

Like all venomous snakes, the spitting cobras belong to the superfamily Colubroidea (advanced snakes). The phylogeny of this superfamily is not yet well established. Current publications tend to differ in the placement and existence of some families and subfamilies (compare Lawson et al., 2005; Castoe et al., 2007; Lee et al., 2007; Pyron et al., 2011). In the Colubroidea the cobras belong to the snake family Elapidae and the subfamily Elapinae (figure 1). Traditionally the Elapidae include all front-fanged snakes with relatively immobile maxillae (proteroglyphs) (Slowinski et al., 1997). Within the cobras the ability to "spit" venom is likely to have evolved three times independently (Wüster et al., 2007). Two times in the "true cobras" (genus *Naja*) and once within their nearest relative *Hemachatus haemachatus* which is the only species of the genus *Hemachatus* (see figure 2 for a list of species). All species of the subgenus *Afronaja* and most species of the subgenus *Naja* (except *N. kaouthia*, *N. naja*, and *N. oxiana*) are able to spit (Wüster and Thorpe, 1992; Mattison, 1995; Wüster et al., 1997; Slowinski and Wüster, 2000; Wüster et al., 2007). About half (15 out of 28) of the members of the genus *Naja* currently recognized as full species are able to spit.

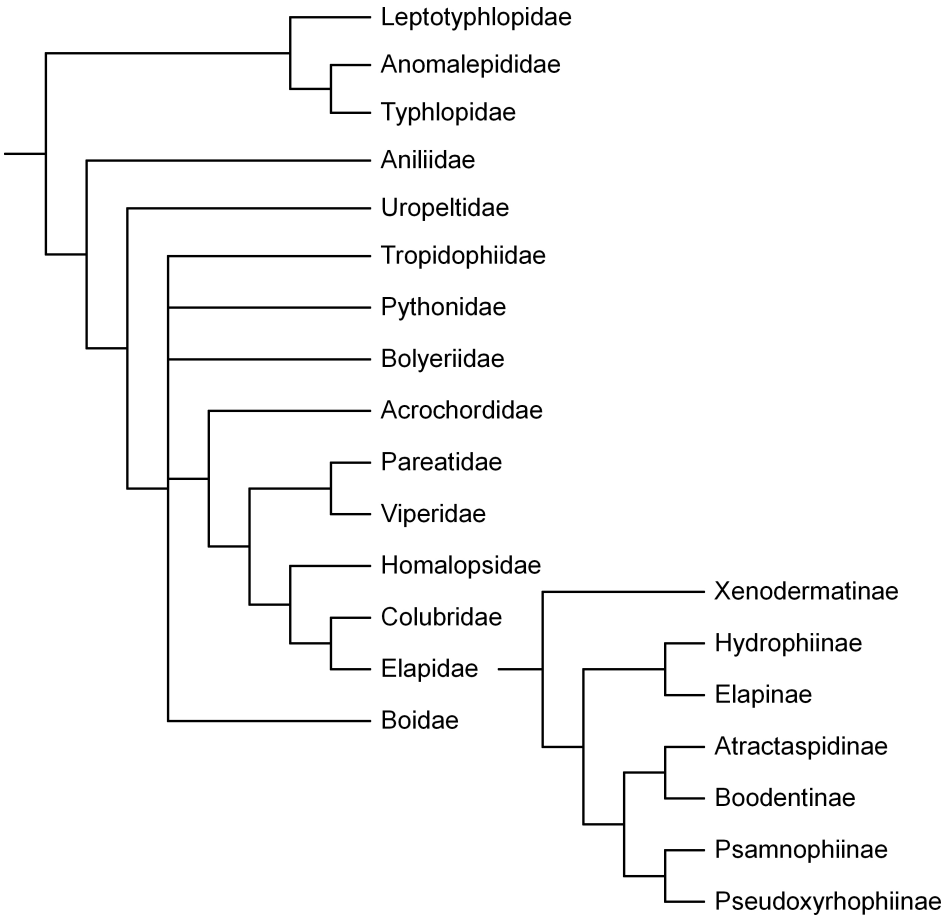


**Figure 1:** Combined phylogenetic tree of the suborder Serpentes and the family Elapidae. (combined after Lawson et al., 2005; Castoe et al., 2007; Lee et al., 2007)

The ability to spit differs between species. The African species *N. nigricollis*, *N. mossambica*, and *N. pallida* are reported to be able spitters. For adult individuals of *N. pallida* the effective spitting range is at least 2.5 m (Rasmussen et al., 1995). The Asian species tested so far and the African species *H. haemachatus* have a smaller spitting range of about 1.5 m and were more stereotypical in their behaviour (Rasmussen et al., 1995; Wüster et al., 2007). *N. pallida* is able to spit while moving its head in any direction or with its movements being restricted by a tube. *N. siamensis* and *H. haemachatus* always lung forward with a hissing sound while spitting. This hissing led some authors to assume that the cobras' venom is propulsed forward at least partly by the exhaled air (FitzSimons, 1912; Koch and Sachs, 1927; Kopstein, 1930). Meanwhile it was shown that the venom is ejected by the pressure muscles, mainly the Musculus adductor mandibulae externus superficialis, exert on the venom gland (Freyvogel and Honegger, 1965; Rasmussen et al., 1995; Young et al., 2004).

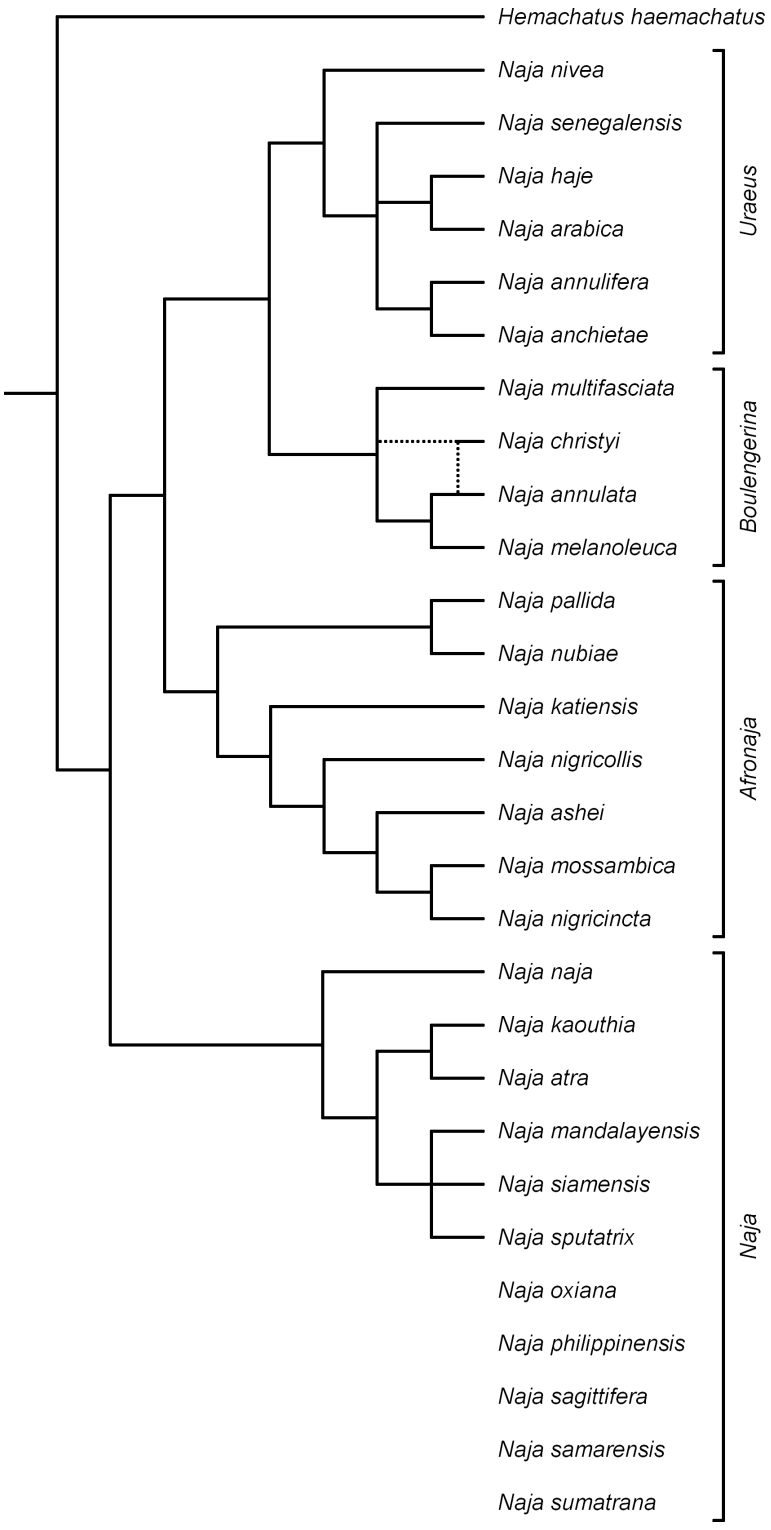


**Figure 2:** Phylogenetic tree of the genera *Naja* and *Hemachatus*. At the right side of the figure the subgenera of the genus *Naja* are named. The five species at the bottom of the figure belong to the subgenus *Naja* but have not been included in a rigorous phylogenetic analysis. (combined after Wüster et al., 2007; Wallach et al., 2009)

This spraying of venom is thought to be a solely defensive behaviour used to repel an aggressor and get enough time to flee. When the venom reaches the eye(s) it causes severe pain. If the venom is not washed out the contact could lead to permanent blindness (FitzSimons, 1912; Koch and Sachs, 1927; Warrell and David Omerod, 1976; Grüntzig et al., 1985; Ismail et al., 1993a,b). Contact with any other mucous membrane could cause inflammation (Warrell and David Omerod, 1976). Some authors describe a burning sensation if the venom reaches an eye but did not experience any permanent damage in consequence (Goring Jones, 1900; Hobley, 1911; Barbour, 1922; Kopstein, 1930). Bogert (1943) placed the venom of two elapids (*H. haemachatus*, *N. nivea*) and two viperids (*Bitis arietans*, *Crotalus atrox*) into the eyes of rats. The rats died as a result of the treatment if the venom originated from a cobra. As the amount of venom is not precisely described this is difficult to compare with possible results of envenomation from venom spitting. Ismail et al. (1993a) used about two thirds of the amount of venom spat by an adult cobra (compare Cascardi et al., 1999) and could not induce the death of the treated rats. Members of one of the two tested rat strains even recovered eyesight. As a side effect these studies clearly adduce evidence of the venom spitting being a defensive behaviour as the amount of venom needed to kill prey in this manner exceeds the amount that could enter the eyes under normal circumstances. Contact with unharmed skin does not cause any immediate discomfort (FitzSimons, 1912; Barbour, 1922, and personal observation). Therefore the venom is much more effective when directed to the eyes of an aggressor. Some authors state that the cobras are doing just that, naming either the eyes or face as the target of the snakes' spitting (FitzSimons, 1912; Barbour, 1922; Koch and Sachs, 1927; Broadley, 1959; Warrell and David Omerod, 1976; Westhoff et al., 2005). FitzSimons (1912) stated that he has made experiments that verified the targeting of faces but did not tell anything about the methods he used. Other authors report cases of humans and dogs being hit in the eyes or face by the venom, but without explicitly stating that the cobras aimed at these targets (Goring Jones, 1900; Hobley, 1911; Kopstein, 1930). Westhoff et al. (2005) and de Pury (2006) made some experiments regarding the aim of the spitting in dependence of eyes. In the first study partially manipulated photos of human faces were presented to spitting cobras. The photos were approximately real size but the distance of the eyes and their presence were altered. In the second study flesh-coloured carnival masks with complete facial features like nose, mouth and ears were presented. These could be equipped with glass eyes in a way that different eye distances

could be realized. Both experiments did not show any effect of the eyes' distance on the size of the area hit by the venom. However, the results of de Pury (2006) suggested that spitting cobras may be less prone to spit at targets lacking eyes.

Based on these experiments this study should answer the question whether cobras do recognize eyes and if they do which characteristics of the eyes they primarily use for this task. In addition to the characteristics of eyes, the features needed to discriminate faces were investigated, as Westhoff et al. (2005) showed that eyes without a face did not elicit spitting.

Cobras, like the other elapids, have hollow fangs in the front of their mouth through which they can eject their venom. The first and largest work about cobra fangs so far was conducted by Bogert (1943). He showed that the fangs of spitting cobras differ from the fangs of non-spitting cobras in one important character: The discharge orifice of spitting species is generally shorter and tear-shaped. There is no groove in the fang which continues the discharge orifice towards the fang tip (compare figure 3). This way the venom is ejected forward and not downward like in other elapids and vipers. In addition Bogert (1943) states that the discharge orifices of spitting cobras are directed to the front of the fang while the discharge orifices of non-spitting cobras are slightly shifted to the outer side of the fang (compare figure 4). Additional studies on the fangs of spitting and non-spitting cobras by Wüster and Thorpe (1992) provide measures of the lengths of the discharge orifices of nine Asiatic cobras. In that study the authors could separate non-spitting and spitting cobras by the length of the discharge orifice relative to the length of the fang. The lengths of the fangs which were adjusted to the length of the head, did not differ between the two groups. De Pury (2006) gave first measures for the venom canals of *N. nigricollis* and *N. pallida* and pointed out possible differences. Additionally the presence of "swellings" basally of the discharge orifice within the venom canal could be documented. They were more pronounced in *N. nigricollis* and *N. pallida* than in *N. siamensis*. These "swellings" were also reported by Young et al. (2009a) as "ridges". Greene (1999) mentions helical grooves within the venom canal which impart a spin to the venom, but did not provide a photography.

In this study the morphology of the fangs of spitting cobras should be more thoroughly analyzed especially in comparison to the fangs of non-spitting species.

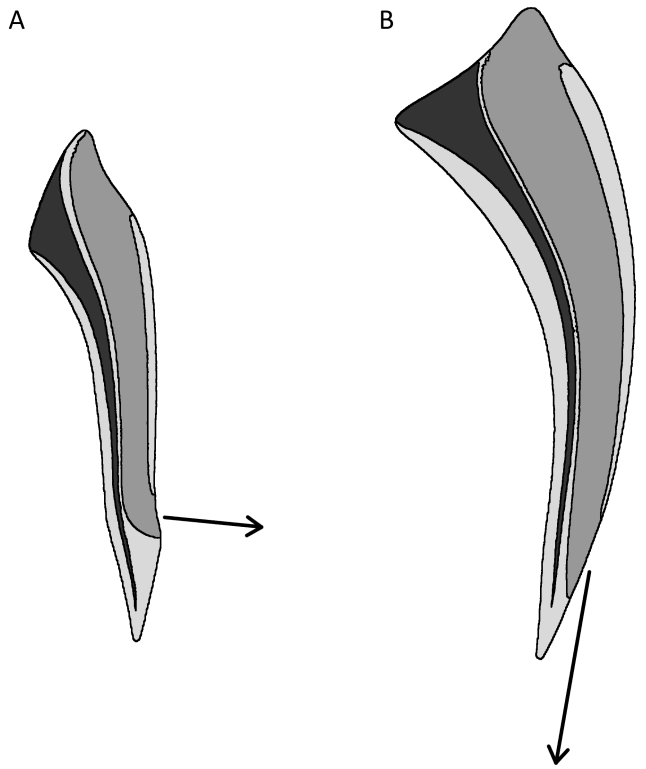


**Figure 3:** Sagittal sections of fangs from a spitting (A: *H. haemachatus*) and a non-spitting (B: *N. melanoleuca*) cobra. The venom canal is coloured medium grey, the pulp cavity is coloured dark grey. The arrows indicate the direction of venom ejection. (after Bogert, 1943)


**Figure 4:** Transversal sections of fangs from spitting (A, B) and non-spitting (C, D) cobras. The sections are positioned at the region of the discharge orifice. The small, black area in the lower half of a subfigure is the pulp cavity. A: *N. nigricollis*, B: *N. philippinensis*, C: *N. haje*, D: *N. naja*. (after Bogert, 1943)

# 2 Materials and Methods

## 2.1 Overview over distribution and habitat of the study species

The Asiatic species *Naja kaouthia* Lesson, 1831, *N. naja* (Linnaeus, 1758), and *N. siamensis* Boie, 1827 generally prefer habitats in the vicinity of water. While *N. siamensis* is mainly restricted to forests and habitats with dense vegetation the widely spread *N. kaouthia* and *N. naja* inhabit more diverse biotopes. The three species tend to cope well with man-made environments (Trutnau, 1998). *N. naja* inhabits nearly all of India and partly the bordering countries. The western border of the distribution of *N. kaouthia* is in north eastern India. Towards the east its distribution spreads to south western China, southern Vietnam, and southern Thailand. *N. siamensis* inhabits Thailand, Cambodia, southern Vietnam, and Laos (Wüster and Thorpe, 1994; Wüster, 1996; Trutnau, 1998; Chan-ard et al., 2000).

The African species *N. haje* (Linnaeus, 1758), *N. nigricollis* Reinhardt, 1843, and *N. pallida* Boulenger, 1896 prefer drier biotopes like savannas or semi-deserts. *N. nigricollis* is found in disturbed forests or plantations, too. In contrast, *N. melanoleuca* Hallowell, 1857 prefers rain forest but is also found in woodland or moist grassland in the vicinity of water (Trutnau, 1998; Spawls et al., 2002). The distribution of *N. haje* covers northern Africa and the Yemen. *N. melanoleuca* and *N. nigricollis* are found in central Africa from the Atlantic Ocean to the Indian Ocean. *N. pallida* inhabits north eastern Africa (Broadley, 1968; Trutnau, 1998; Luiselli and Angelici, 2000; Spawls et al., 2002).

*Hemachatus haemachatus* (Bonnaterre, 1789) is found in the grasslands of south eastern Africa, mainly South Africa. It is the only viviparous species among the cobras (Koch and Sachs, 1927; Trutnau, 1998; Broadley and Wüster, 2004).

*Dendroaspis angusticeps* (Smith, 1849) is the only non cobra species in this study and was included in the fang morphology analysis for comparison. It is an arboreal elapid inhabitating forests in south eastern Africa (Trutnau, 1998).

## 2.2 Behavioural experiments

The snakes were housed alone or as pairs in wooden terraria of varying size (small: 75 cm × 60 cm × 60 cm (width × height × depth); large: 200 cm × 60 cm × 100 cm) depending on the size and number of the snakes. Water was provided

ad libitum, and the snakes were fed with pre-killed mice about every third week. For the experiment the snake was placed into an experimental terrarium (75 cm × 53 cm × 37 cm). To make sure the snake would spit, it was tested whether the cobra spat at the experimenter. If a snake did not spit another cobra was tested. Cobras that were going to shed their skin were not used in the experiments. An experimental area of $1.4 \times 1.4$ m was confined by white fabric hanging from the ceiling to the floor to provide a light and uniform background without any distraction for the snake. During the experiments 2.2.1 to 2.2.3 the snake could be observed through a small window in the side-wall of the terrarium so that it was not disturbed by the presence of the experimenter. The snake's perception of the experimenter through the window was prevented by two measures. First, the area around the experimenter's side of the window was darkened. Second, a white insect screen fabric was installed at the snake's side of the window. The snake was induced to spit by moving different targets in front of the snake. The targets were attached to a white coloured rod that was operated by the experimenter. Instead of following a preassigned pattern the movement of the target was influenced by the snake's position and behaviour. Movements that agitated the snakes were preferred over movements that failed to do so.

### 2.2.1 Comparison of eyes

The front of the experimental terrarium was closed by acrylic glass. The target consisted of a roughly face-shaped plate of about 375 $cm^2$. The target was made of medium-density fibreboard painted matt black (Plaka 70 and Silky mat varnish; Pelikan; Germany) and equipped with different glass eyes of 2 cm diameter (310, 320, and 350, Lauschaer Glasaugen, Germany) glued to a magnets (nickel plated neodymium). By the use of additional magnets on the back of the plate the glass eyes could be fixed on the plate without leaving marks of their position when removed. Inter eye distance was 5 cm. The eyes were completely black or had an colourless iris (figure 5). In addition they could be shiny or matt and placed horizontally or vertically (one eye above the other) on the plate. Plates without eyes were presented as control.

Spitting frequency and reaction times (time span from the beginning of the stimulation to spitting) were monitored. To adjust for individual differences and changing readiness to spit the rection time was divided by the mean time for each test series. Each test series consisted of nine different presentations in random order. The different presentation were:

**Figure 5:** Photos of glass eyes. From left to right: black & shiny, black & matt, colourless iris & shiny, colourless iris & matt.

- completely black – matt – horizontal
- completely black – matt – vertical
- completely black – shiny – horizontal
- completely black – shiny – vertical
- colourless iris – matt – horizontal
- colourless iris – matt – vertical
- colourless iris – shiny – horizontal
- colourless iris – shiny – vertical
- without eyes

### 2.2.2 Comparison of shapes

The front of the experimental terrarium was closed with acrylic glass. The targets were the roughly face-shaped plate used for the comparison of eyes and additional ones of the same composition. At the end of each test series the experimenter presented himself to the snake to check if the snake's willingness to spit had vanished during the experiments. If the snake did not spit at the experimenter the whole test series was discarded as it would be unknowable whether the snake did not spit because of the target or because it was no longer willing to spit.

**First experiments** The additional targets differed from the face shaped plate in shape but not in size (375 $cm^2$). The additional shapes were an equilateral triangle either pointing upward or downward, a square with a horizontally aligned baseline, and a rectangle ($50 \times 7.5$ cm) with either a horizontally or vertically aligned longer edge. Each test series consisted of seven presentations, meaning that each shape and orientation was presented once per test series.

**Main experiments** The additional targets differed from the face-shaped plate in shape and/or size. The additional shapes were an equilateral triangle either pointing upward or downward and a "triangle" with rounded tips. The sizes were 185 $cm^2$ (upward pointing triangle and face shape), 375 $cm^2$ (all shapes)

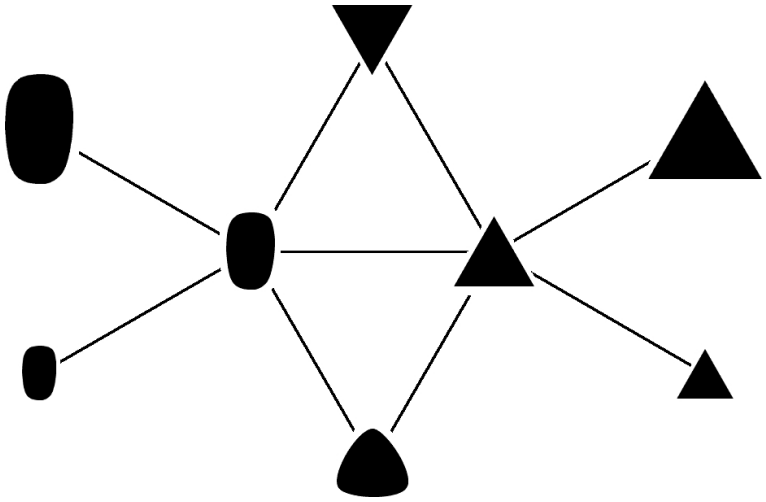

**Figure 6:** The shapes which were presented to the snakes. Shapes directly compared to each other are connected by lines. The four shapes in the middle had a size of 375 $cm^2$. The bigger ones on the left and right side were 750 $cm^2$ in size, the smaller ones 185 $cm^2$.

and 750 $cm^2$ (upward pointing triangle and face shape). Because of the cobras' changing willingness to spit, the shapes were tested pairwise (cf. figure 6) and the results of different pairs were not pooled. Each test series consisted of six presentations (both shapes three times each) in random order.

### 2.2.3 Comparison of sizes

The front of the terrarium consisted of a metal grid (mesh size 1 cm), which had been painted matt black (Dupli-Color color spray; Motip Dupli GmbH; Germany) to reduce light reflections. The target consisted of two circular disks with matt black self-adhesive plastic foil (UNI, Alkor Venilia GmbH, Germany). One disk had a diameter of 20 cm, the other disk had a diameter of 10 cm. The flanks of the two disks were attached to each other. For the experiments the following configurations were used: The smaller disk positioned above or below as well as to the left or to the right of the larger one (example in figure 7). Glass eyes normally used in taxidermy (Bird eyes 185 M27 26 mm, KL-Glasaugen, Germany) were glued to magnets and could be fixed on the disks due to a metal ring on the back of the disks. Thus they could be placed and removed without leaving marks at their positions when removed. Inter eye distance was about 5 cm. After each test the disks were photographed with a digital single-lens reflex camera (Olympus E330 with zuiko digital 14–45 mm, Olympus corporation, Japan) and cleaned from the venom. For the analysis of the spitting behaviour the area

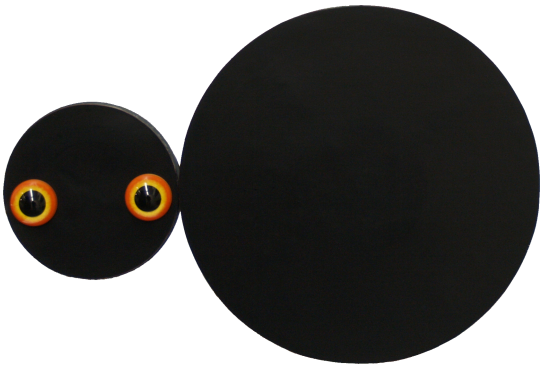

**Figure 7:** Example showing a possible arrangement of the two disks and the glass eyes. The grey area at the upper right border of the smaller disk is the side of the disk.

covered by venom on a disk was adjusted by the disk size and the sum was normalized (large disk: $x_l = \frac{\sum_{ld} v}{\sum_{ld} v + 4 \cdot \sum_{sd} v}$, small disk: $x_s = \frac{4 \cdot \sum_{sd} v}{\sum_{ld} v + 4 \cdot \sum_{sd} v}$, $ld =$ large disk, $sd =$ small disk, $v =$ venom). This was done for two reasons. First, the disks were of different sizes which would lead to a predominance of the larger disk if the amount of venom would have been used. Second, the individual cobras spat varying amounts of venom and the behaviour of a cobra which spit more venom than the others could have dominated the data.

### 2.2.4 Venom distribution and target distance

The front of the arena consisted of a metal grid (mesh size 1 cm). The grid was painted matt black to reduce light reflections. Two cameras (ccd-2112 camera pcba, Conrad, Germany) were installed at the sides of the arena. To elicit spitting, the experimenter made erratic head movements in front of the snake at a "starting distance" not exceeding 1 m. The experimenter then slowly reduced the distance to the snake either until spitting occurred, or the head of the experimenter was within 5 cm of the grid. To protect his face, the experimenter wore a visor. A plastic sheet (30 cm × 40 cm), attached to the visor, was used to collect the venom. Preliminary tests revealed that the outer dimensions of spitting patterns did not exceed the dimensions of the plastic sheet. When the snake spat the experimenter stopped moving and a second experimenter measured the distance between the plastic sheet of the first experimenter and the front grid of the arena. To reduce the measurement error caused by the reaction time of the experimenter, fast movements towards and away from the snake were avoided. The distance between the grid and the snout of the snake was determined from video

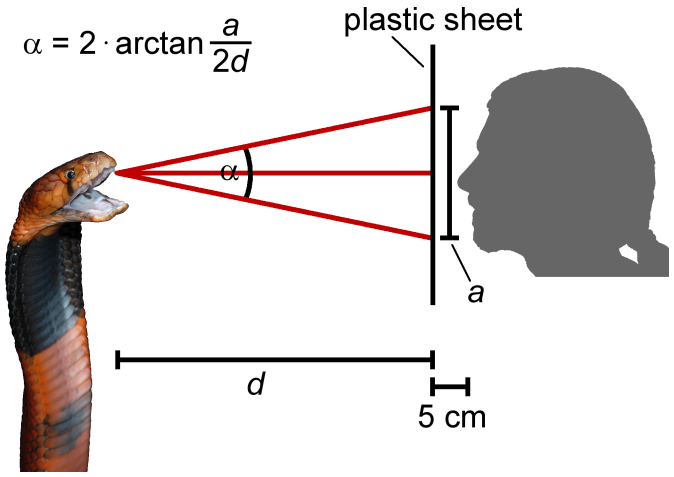


**Figure 8:** Schematic drawing to illustrate the relation of spitting angle, dimension of spitting pattern and target distance $d$. The face of the experimenter was located 5 cm behind the plastic sheet. The spitting pattern had a height of $a_{vertical}$. The spitting angle $\alpha_{vertical}$ was calculated with the formula shown in the figure. The spitting angle $\alpha_{horizontal}$ was calculated in the same way.

images. Subsequently the distance from the snake to the sheet was calculated. The plastic sheet with the venom was exchanged and dried after each spitting trial. The height and width of a spitting pattern were defined as the height and width of the smallest possible rectangle with a horizontal baseline, containing the outermost venom drops that exceeded 1 mm in diameter (figure 9). From the height and width and the measured distance between cobra and sheet, the vertical and horizontal angles of venom ejection were calculated (figure 8).

Spitting pattern values of individual and pooled snakes were correlated (Spearman test) with distance to target. A Kolmogorov-Smirnov test was used to determine if there was a difference in height or width of the spitting pattern as a function of distance, between the two species of snakes. The test was repeated for each 10 cm step in distance. All tests were performed using SPSS (SPSS Statistics 17.0, SPSS Inc., USA).

### 2.2.5 Additional experiments

**Videos** At the beginning of the study it was tried to arouse the cobras by playing back videos of moving humans or stuffed birds that were moved. The experimenters that had been filmed, moved as if they tried to agitate a spitting cobra positioned at the location of the camera. Birds used for this were a Common Buzzard (*Buteo buteo*), a Northern Goshawk (*Accipiter gentilis*), and a Carrion Crow (*Corvus corone*). Additional videos were randomly moving, abstract faces

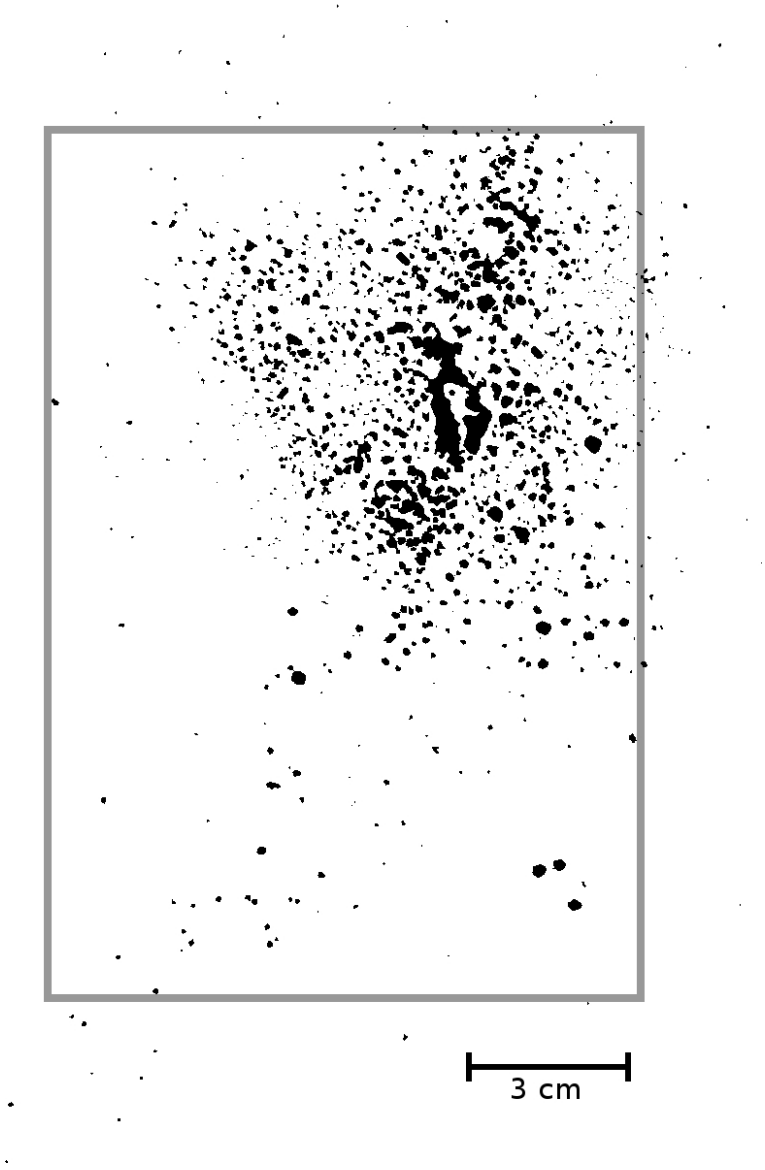


**Figure 9:** A sample spitting pattern of *N. nigricollis*. The gray rectangle borders the spitting area (for definition see page 20). Scale bar 3 cm.

generated with the 3D graphics application Blender (Blender 2.43, Blender Foundation, Netherlands). All videos were either projected on a rear-projection screen or presented to the snakes on an 21 inches cathode ray tube monitor.

**Mounted birds** At times, before or after a test series of the other behavioural experiments the cobra was provoked by moving a mounted bird in front of the experimental terrarium. Birds used for this were a Common Buzzard (*Buteo buteo*), a Northern Goshawk (*Accipiter gentilis*), and a Carrion Crow (*Corvus corone*).

## 2.3 Analysis of venom fangs

### 2.3.1 Morphology data

Altogether twenty six fangs of nine different snake species were scanned with a microtomograph (SkyScan 1072, SkyScan, Belgium). The species and the numbers of fangs per species are listed as follow.

- *Dendroaspis angusticeps*: 2 fangs (ang1, ang2)
- *Hemachatus haemachatus*: 2 fangs (hae1, hae2)
- *Naja haje*: 1 fang (haj1)
- *N. kaouthia*: 3 fangs (kao1, kao2, kao3)
- *N. melanoleuca*: 2 fangs (mel1, mel2)
- *N. naja*: 3 fangs (naj1, naj2, naj3)
- *N. nigricollis*: 5 fangs (nig1, nig2, nig3, nig4, nig5)
- *N. pallida*: 4 fangs (pal1, pal2, pal3, pal4(e))
- *N. siamensis*: 4 fangs (sia1(e), sia2(e), sia3, sia4)

The pictures of the slices of the fangs were created from the X-ray images by the software NRecon (NRecon version 1.5.1.4, SkyScan, Belgium). These slices were then processed by a self written program (see pages 97 to 201 for the source code). The processed slices were used for a 3D-reconstruction of the fangs with the software Avizo (Avizo 6.0.0, Mercury Computer Systems Inc., USA). The self written program was used to measure the venom canal of the fangs. For each

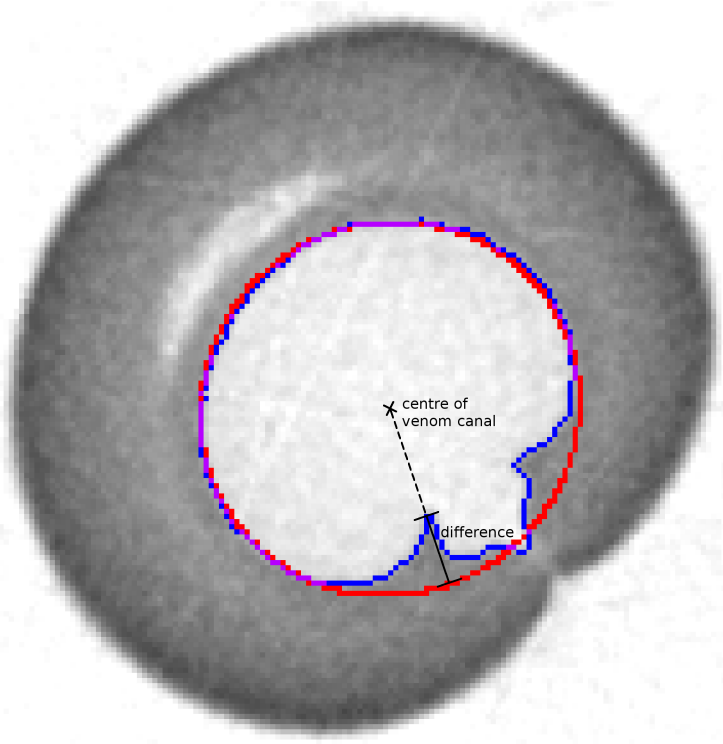


**Figure 10:** Example showing the results of the program used to find the border of the venom canal and the ellipse fitted to it. The border of the venom canal is marked in blue and violet. The red and violet pixels are intersected by the ellipse generated by the program. One pixel equals 6.283031 µm in both x- and y-direction.

slice the fang and the venom canal, if present, were identified and marked by the program. If the venom canal was present its circumference and area ($A_C$) were measured. In addition an ellipse was fitted to the canal and the distances between the pixels of the border of the venom canal and the ellipse were calculated (cf. figure 10). To account for the curvature of some fangs and the fact that some fangs were scanned aslant, additional slices were calculated and measured as described above. These slices were based on planes that used the connections between the centres of two adjacent slices (averaged over eleven slices – for example the means of slices 31 to 41 and slices 32 to 42) as their surface normals.

For each fang the length ($l_F$) was measured as the straight-line distance from the distal end of the entrance lumen to the tip (distal end) of the fang as by Bogert (1943). Additionally, the length of the discharge orifice ($l_O$) and the length of the venom canal ($l_{Ct}$ and $l_{Cc}$) and the length of the fang tip ($l_T$) were measured. These distances were measured as straight-line distances, too. In the case of the discharge orifice the points spanning the line were the basal and distal ends of

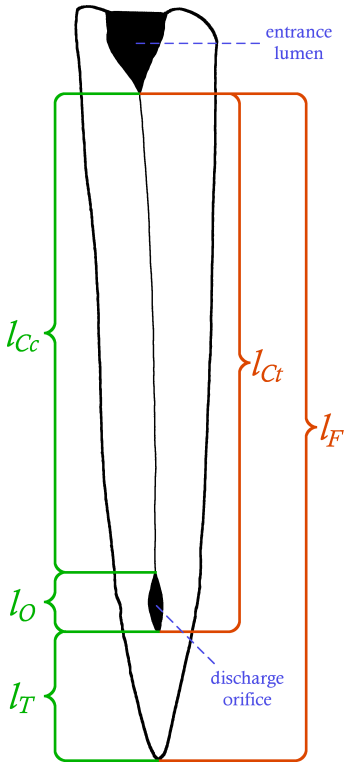


**Figure 11:** Illustration of the fang measures $l_{Cc}$, $l_O$, $l_T$, $l_{Ct}$, and $l_F$.

the discharge orifice. For the length of the venom canal two measures were taken. One is the total length including the discharge orifice ($l_{Ct}$), the other is restricted to the closed section ($l_{Cc}$) of the venom canal. In both cases the basal end of the measurement was defined by the distal end of the entrance lumen as used for the measurement of the complete fang. The length of the fang tip was defined as the distance from the distal end of the fang to the distal end of the discharge orifice (illustration in figure 11). As an additional parameter the area of the discharge orifice ($A_O$) was measured.

Within the venom canal the length of the distal ridges, if present, were measured for the left and right side of the canal. The length of the ridges was measured for both the total extent of the venom canal ($l_{Rlt}$ and $l_{Rrt}$) and the closed section of the venom canal ($l_{Rlc}$ and $l_{Rrc}$).

### 2.3.2 Spraying analysis

Two fangs each of the species *N. nigricollis* (nig4 and nig5) and *N. pallida* (pal3 and pal4) were used in an experiment of "artificial spitting". The fangs were glued to a polyvinyl chloride tube (1.5 mm inner diameter, 2.1 mm outer diameter, approx. 4 cm length; cf. figure 12) by a plastic glue (UHU allplast, UHU GmbH

& Co KG, Germany). Care was taken not to bring the glue in contact with the orifices of the fangs. These tubes were connected to larger tubes (4 mm inner diameter, 7 mm outer diameter, approx. 20 cm length) by polypropylene tubing reducers (Rotilabo mini tubing reducer 1.6 – 4.0 mm, Carl Roth GmbH & Co KG, Germany). The other ends of the larger tubes were equipped with Luer tubing connectors (Rotilabo Leur tubing connector 4.0 mm female, Carl Roth GmbH & Co KG, Germany) to provide connection to a pressure sprayer (hobby universal, Gloria Haus- und Gartengeräte GmbH, Germany). The shower tube of the pressure sprayer was cut off after approximately 30 cm and a Luer tubing connector was glued to the end of the tube (Rotilabo Leur tubing connector 4.0 mm male, Carl Roth GmbH & Co KG, Germany; UHU plus schnellfest, UHU GmbH & Co KG, Germany). The pressure sprayer had been cleaned from grease before usage.

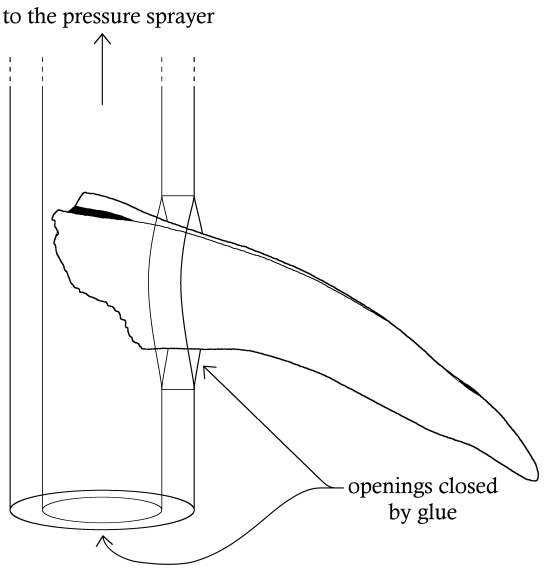


**Figure 12:** Schematic drawing to illustrate how the fangs were connected to the tubes.

The experiments were conducted with two aqueous solutions of glycerol at room temperature ($20 \pm 1\,°C$) with each of the four fangs. The solutions were 75 and 85 weight percent of glycerol which results in a dynamic viscosity of about 36 and 109 Pa s (compare Segur and Oberstar, 1951). These viscosities are within the range of data measured for venoms of *N. pallida* and *N. nigricollis* (data provided by Alexander Balmert). The pressure within the pressure sprayer was regulated that the maximum spraying range of fang nig4 in combination with the 85 % solution of glycerol was about 2.5 m. The associated pressure was 2.5 bar. During

the experiments the liquid stream leaving the fang was oriented horizontally and directed perpendicular at a black polyvinyl chloride plate (Forex). The distance between the fang and the plate was 60 cm. The duration of the liquid stream was defined by a plastic plate falling through the stream. The stream was blocked except for a 5.5 cm high window cut into the plate. This resulted in a duration of $21.3 \pm 0.6$ ms (measured by high speed recordings with 3000 frames per second; Fastcam ultima APX, Photron, USA) during which the liquid stream was able to reach the black target plate. The droplets on the plate were photographed with a digital single-lens reflex camera (Nikon D90 with AF-S DX NIKKOR 18-105 mm 1:3,5-5,6G ED VR, Nikon corporation, Japan).

# 3 Results

## 3.1 Behavioural experiments

### 3.1.1 Comparison of eyes

In the main experiments a total of 324 valid tests (36 test series) were performed with nine snakes (four *N. nigricollis*, four *N. pallida*, one *N. siamensis*). The number of test series varied between one and nine per snake. In 87.7 % of the cases a snake spat at the presented target. The spitting frequencies for the different attributes of the eyes (black, colourless, shiny, matt, horizontally, vertically, present, absent) varied between 80.6 % and 90.3 % ($87.5 \pm 3.2$ %; figure 13 and table 1). The reaction times (time span from the beginning of the stimulation to spitting) as well as the spitting frequencies differed between individuals. The spitting frequencies ranged from 55.6 % to 100.0 % for the nine individuals. The cobra with the lowest reaction time spat after $1.9 \pm 0.5$ seconds (spitting frequency 100.0 %) while the one that had the longest reaction time spat after $19.5 \pm 18.0$ seconds (spitting frequency 93.3 %). No correlation was found between the spitting frequency and the reaction time of individual cobras (Spearman-Rho-test, $N = 9$, $r_s = -0.458$, $p = 0.215$). The distribution of the reaction times is shown in figure 15. There were no differences for the different eyes used, neither in the frequency of spitting, nor in the reaction time (table 2). The same was true for the presence or absence of eyes (for tests for every combination see table 14). Altogether the tested individuals of *Naja pallida* (92.6 %; four individuals) were more incline to spit than the individuals of *N. nigricollis* (80.8 %; four individuals) and *N. siamensis* (80.6 %; one individual). In general no difference between *N. nigricollis* and *N. pallida* was found concerning the spitting frequency at the different constellations of glass eyes. The only exception was the target without eyes (test results in table 3): the individuals of *N. nigricollis* spat less at this target than those of *N. pallida*. The species *N. siamensis* was not compared to the others due to the low number of individuals.

### 3.1.2 Comparison of shapes

In the first experiments 42 tests distributed over seven test series were performed with four snakes. The cobras spat three times at the roughly face-shaped target but at none of the other five targets. The additional shapes used in the first experiments were the a square, two rectangles, and two triangles.

**Table 1:** Comparison of the different glass eye characteristics. Reaction time is given in seconds. For the standardized time the reaction time of each test was divided by the mean reaction time of the corresponding test series.

| Eye characteristic | Tests | Spitting frequency | Reaction time | Standardized time |
|---|---|---|---|---|
| black | 144 | 86.8 % | 11.2 ± 10.8 | 0.90 ± 0.65 |
| colourless iris | 144 | 90.3 % | 15.0 ± 16.3 | 1.09 ± 0.95 |
| shiny | 144 | 86.8 % | 12.4 ± 12.7 | 0.95 ± 0.68 |
| matt | 144 | 90.3 % | 13.8 ± 15.2 | 1.05 ± 0.94 |
| horizontal | 144 | 89.6 % | 13.6 ± 15.0 | 1.05 ± 0.94 |
| vertical | 144 | 87.5 % | 12.7 ± 13.0 | 0.95 ± 0.67 |
| present | 288 | 88.5 % | 13.1 ± 14.0 | 1.00 ± 0.82 |
| absent | 36 | 80.6 % | 12.9 ± 11.5 | 1.00 ± 0.77 |

**Table 2:** Tests for differences in the snakes' reaction towards the different glass eye characteristics. For the standardized time the tests reaction time was divided by the mean reaction time of the corresponding test series. $N_1$ and $N_2$ are the numbers of the spitting events for the eye characteristics. The statistical tests used were G-tests for spitting frequency and Kolmogorov-Smirnov tests for reaction time and standardized time. The test statistic is given in the column "Value".

| Tested pair | Parameter | $N_1$ | $N_2$ | Value | $p$ |
|---|---|---|---|---|---|
| black & colourless | spitting frequency | 144 | 144 | 0.859 | 0.354 |
| black & colourless | time | 125 | 130 | 1.201 | 0.112 |
| black & colourless | standardized time | 125 | 130 | 1.078 | 0.195 |
| shiny & matt | spitting frequency | 144 | 144 | 0.859 | 0.354 |
| shiny & matt | time | 125 | 130 | 0.639 | 0.809 |
| shiny & matt | standardized time | 125 | 130 | 0.545 | 0.927 |
| horizontal & vertical | spitting frequency | 144 | 144 | 0.308 | 0.579 |
| horizontal & vertical | time | 129 | 126 | 0.526 | 0.945 |
| horizontal & vertical | standardized time | 129 | 126 | 0.642 | 0.804 |
| present & absent | spitting frequency | 288 | 36 | 1.675 | 0.196 |
| present & absent | time | 255 | 29 | 0.958 | 0.318 |
| present & absent | standardized time | 255 | 29 | 0.929 | 0.354 |

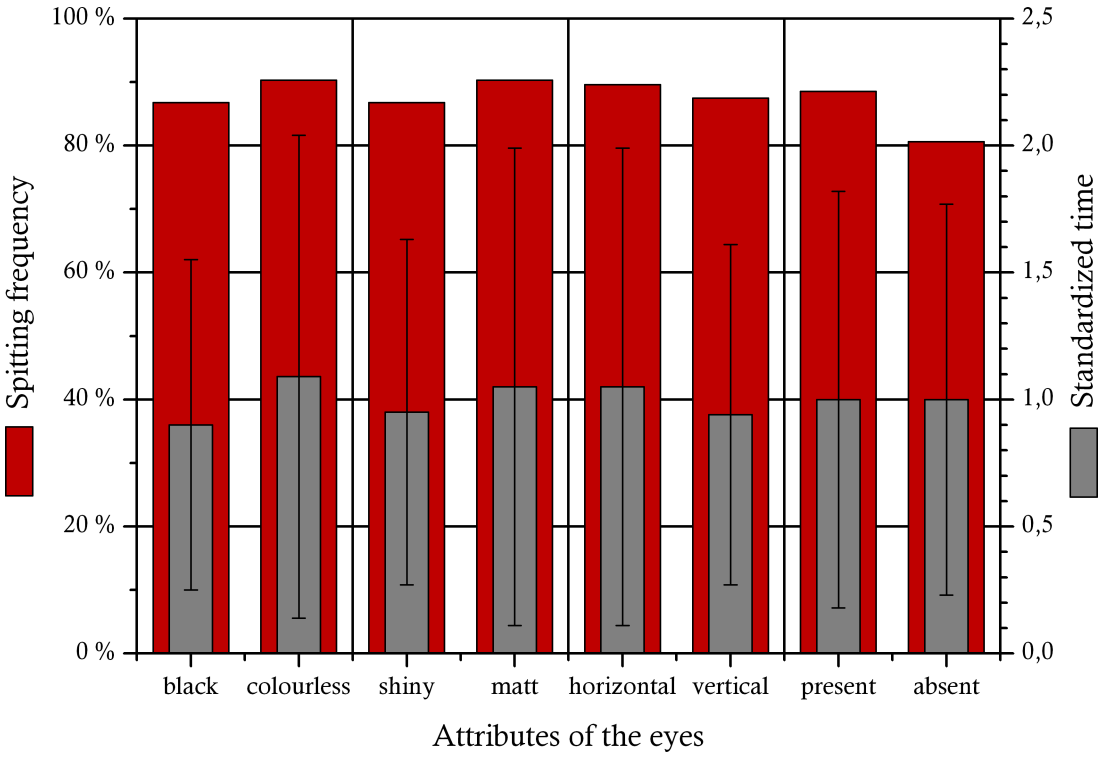


**Figure 13:** Comparison of characteristics of glass eyes concerning spitting frequency and reaction time. All tests that shared the attributes given on the x-axis were pooled. The standardized time of a single test is the reaction time divided by the mean reaction time of the corresponding animal and test series, respectively. Red = spitting frequency (left y-axis), grey = mean standardized time (right y-axis).

In total 750 valid tests with 22 individuals (seven *N. nigricollis*, eight *N. pallida*, six *N. siamensis*, one *H. haemachatus*) were performed during the main experiments. The number of test series per individual ranged from 1 to 28. The presented targets were roughly face-shaped plates and upward pointing triangles of different sizes as well as an downward pointing and a rounded triangle. No differences were found between the two species with the largest number of valid test series *N. nigricollis* and *N. pallida* (table 5). The tests showed a difference in the spitting behaviour regarding the shape of the presented targets (figure 14). The cobras spat at the face shaped target in 80.6 % of the trials and at the triangular shaped target in 22.2 % (all results are listed in table 4). The size of the target had no effect on the spitting behaviour. Neither by doubling nor halving the area of the target altered the spitting frequency. Turning the triangle upside down (pointing downward) did not affect the test results compared to the upward pointing triangle. When the triangle was rounded the cobras spat more often at

**Table 3:** Comparison of the two species *Naja nigricollis* and and *N. pallida* regarding the spitting at different constellations of glass eyes. The characteristics of the glass eyes are abbreviated as: c = colourless iris, b = black, m = matt, s = shiny, h = horizontal, and v = vertical. $N$ gives the number of cases for the two species (marked as $_{nig}$ and $_{pal}$) either spitting ($_{spit}$) or not spitting ($_{not}$) at the target. The tests performed were Fisher exact tests.

| Eyes | $N_{nig,spit}$ | $N_{nig,not}$ | $N_{pal,spit}$ | $N_{pal,not}$ | $p$ |
|---|---|---|---|---|---|
| cmh | 10 | 1 | 19 | 2 | > 0.999 |
| cmv | 11 | 0 | 20 | 1 | > 0.999 |
| csh | 9 | 2 | 19 | 2 | 0.593 |
| csv | 10 | 1 | 20 | 1 | > 0.999 |
| bmh | 9 | 2 | 21 | 0 | 0.111 |
| bmv | 8 | 3 | 18 | 3 | 0.390 |
| bsh | 10 | 1 | 20 | 1 | > 0.999 |
| bsv | 7 | 4 | 19 | 2 | 0.148 |
| absent | 6 | 5 | 19 | 2 | 0.032 |

this triangle than at the normal triangle. Also the cobras spat more often at the face-shaped target than at the rounded "triangle", but the difference was not significant. The readiness of the cobras to spit varied greatly between individuals. The reaction times ranged from $1.6 \pm 1.6$ seconds to $29.7 \pm 32.8$ seconds and the spitting frequency from 0 % to 100 %. For additional tests the results were grouped by snake and presented target. There was a negative correlation between spitting frequency and reaction time (Speraman-Rho-test, $N$ = 71, $r_s$ = −0.540, $p < 0.001$). This correlation completely disappeared when the cases with a spitting frequency of 100 % were omitted (Spearman-Rho-test, $N$ = 34, $r_s$ = 0.014, $p$ = 0.938). The cases with a spitting frequency of 100 % had a significantly lower reaction time than the rest ($t_{100\%}$ = 7.85 s ± 7.88 s, $t_{<100\%}$ = 20.67 s ± 11.30 s, Kolmogorov-Smirnov test, $N$ = 71, $Z$ = 2.546, $p < 0.001$). A histogramm of the reaction times is shown in figure 15.

### 3.1.3 Comparison of sizes

In 85.7 % of the tests relatively more venom was found on the larger disk of the pair of disks presented. When the glass eyes were positioned on the smaller disk at least one eye was hit in 30.8 % of the cases. When the eyes were positioned on the larger disk the result was about the reverse (in 71.4 % of the cases one or both

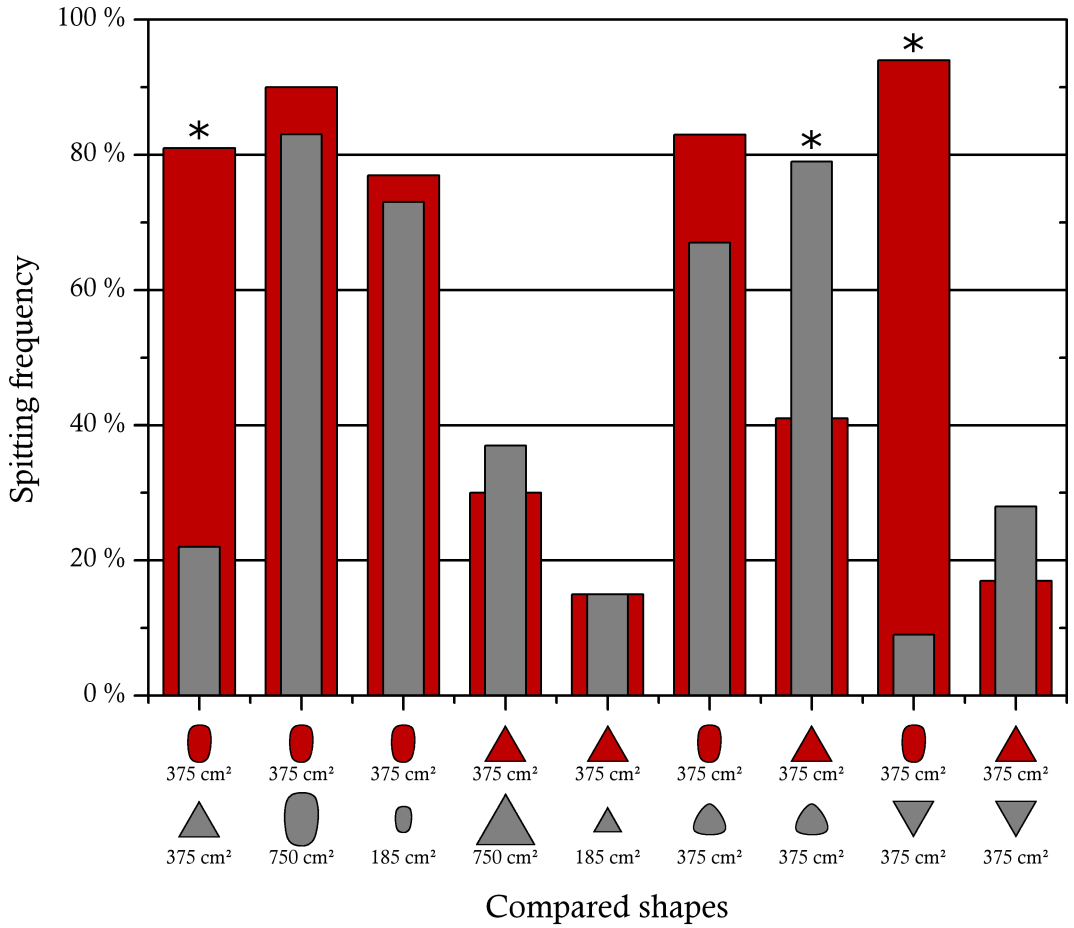


**Figure 14:** Comparison of differently shaped targets concerning spitting frequency. Pairs of bars which differ significantly are marked by an asterisk (cf. table 4). The symbols at the bottom indicate which bar belongs to which shape (shapes are to scale).

eyes were hit). No dependence of the aiming of the snakes on the position of the eyes was found (G-test, $df = 1$, $G = 1.053$, $p = 0.305$; the snake was supposed to have aimed at the disk with the higher venom density). Summarized distributions of the venom on the disks can be seen in figures 16 to 18. The venom density on the larger disk was higher than on the smaller disk (Wilcoxon test, $N = 42$, $Z = -4.283$, $p < 0.001$). For the adjusted venom amounts in 32 of 36 cases at least 90 % of the venom covered the larger disk, in two cases at least 90 % of the venom covered the smaller disk, and in the remaining two cases none on the disks were covered by that amount of venom (cf. table 6). For the tests with the higher venom density on the larger disk, the venom density on its inner half (centre) was higher than on its outer half (surrounding) (Wilcoxon test, $N = 36$, $Z = -4.619$, $p < 0.001$). There was no differences between the upper and lower

**Table 4:** Comparison of differently shaped targets concerning spitting frequency. The first two columns specify the shapes which were compared. $N$ = number of test. A subscript number specifies the shape. The tests performed were G-tests and the degrees of freedom were 1 for all tests.

| First shape | Second shape | $N_{1,spit}$ | $N_{1,not}$ | $N_{2,spit}$ | $N_{2,not}$ | $G$ | $p$ |
|---|---|---|---|---|---|---|---|
|  375 cm² |  375 cm² | 58 | 14 | 16 | 56 | 52.303 | < 0.001 |
|  375 cm² |  750 cm² | 38 | 4 | 35 | 7 | 0.852 | 0.329 |
|  375 cm² |  185 cm² | 51 | 15 | 48 | 18 | 0.364 | 0.546 |
|  375 cm² |  750 cm² | 9 | 21 | 11 | 19 | 0.300 | 0.584 |
|  375 cm² |  185 cm² | 5 | 28 | 5 | 28 | 0.000 | > 0.999 |
|  375 cm² |  375 cm² | 35 | 7 | 28 | 14 | 3.158 | 0.076 |
|  375 cm² |  375 cm² | 16 | 23 | 31 | 8 | 12.444 | < 0.001 |
|  375 cm² |  375 cm² | 31 | 2 | 3 | 30 | 56.239 | < 0.001 |
|  375 cm² |  375 cm² | 3 | 15 | 5 | 13 | 0.648 | 0.421 |

half (Wilcoxon test, $N = 36$, $Z = -0,456$, $p = 0,649$) or between the left and right half (Wilcoxon test, $N = 36$, $Z = -1,791$, $p = 0,073$). Due to the low number of trials with the higher venom density on the smaller disk, the corresponding tests for that disk were omitted.

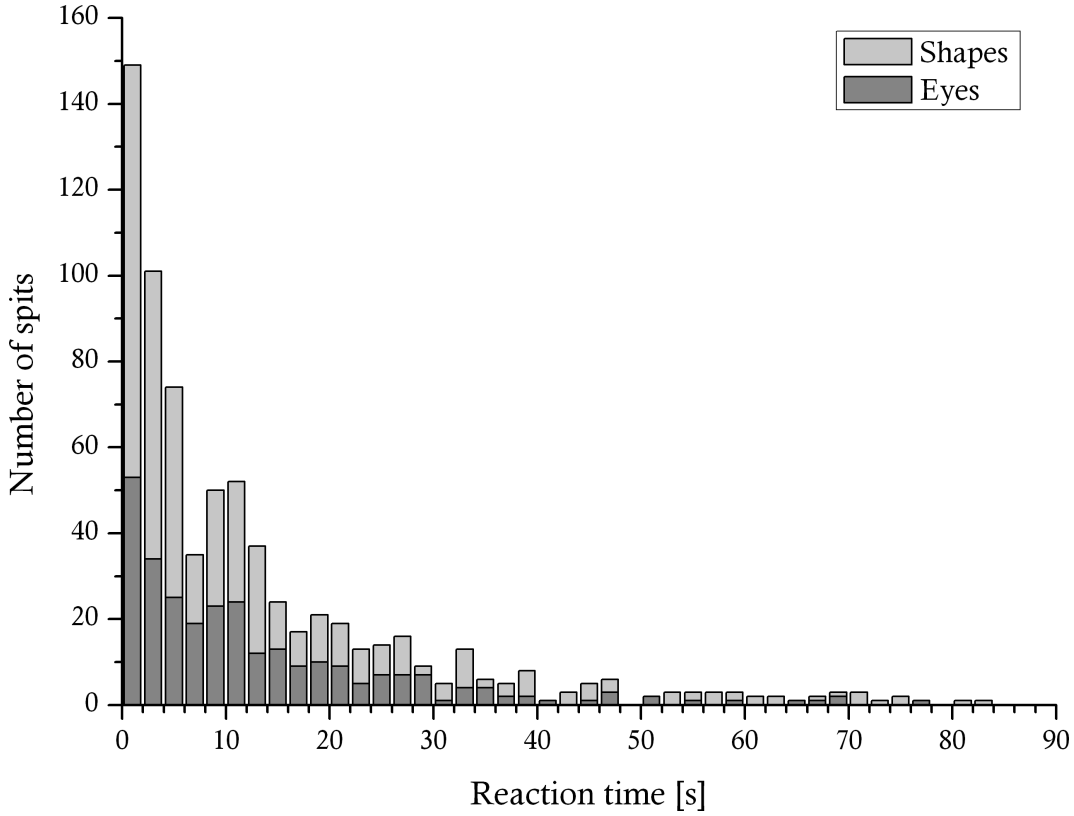


**Figure 15:** Distribution of reaction times. The light grey bars show the number of spits for the experiments with the differently shaped targets (section 3.1.2), the medium grey bars show the numbers for the comparison of eyes (section 3.1.1). Since the number of tests differed between the experimental series the light grey bars are generally larger.

**Table 5:** Comparison of the two species *Naja nigricollis* and *N. pallida* regarding the spitting at different targets. The additional target specifies the test series as the targets were presented pairwise (cf. sections 2.2.2 and 3.1.2). $N$ gives the number of cases for the two species (marked as $_{nig}$ and $_{pal}$) either spitting ($_{spit}$) or not spitting ($_{not}$) at the target. Fisher exact tests were used.

| Tested target | Additional target | $N_{nig,spit}$ | $N_{nig,not}$ | $N_{pal,spit}$ | $N_{pal,not}$ | $p$ |
|---|---|---|---|---|---|---|
| 375 cm² | 375 cm² | 8 | 4 | 44 | 10 | 0.264 |
| 375 cm² | 375 cm² | 1 | 11 | 10 | 44 | 0.673 |
| 375 cm² | 750 cm² | 5 | 1 | 30 | 3 | 0.502 |
| 750 cm² | 375 cm² | 3 | 3 | 29 | 4 | 0.059 |
| 375 cm² | 185 cm² | 12 | 6 | 39 | 9 | 0.322 |
| 185 cm² | 375 cm² | 14 | 4 | 34 | 14 | 0.759 |
| 375 cm² | 750 cm² | 3 | 6 | 6 | 15 | > 0.999 |
| 750 cm² | 375 cm² | 3 | 6 | 8 | 13 | > 0.999 |
| 375 cm² | 185 cm² | 0 | 6 | 4 | 20 | 0.557 |
| 185 cm² | 375 cm² | 0 | 6 | 3 | 21 | > 0.999 |
| 375 cm² | 375 cm² | 4 | 2 | 25 | 5 | 0.573 |
| 375 cm² | 375 cm² | 4 | 2 | 18 | 12 | > 0.999 |
| 375 cm² | 375 cm² | 3 | 3 | 10 | 17 | 0.659 |
| 375 cm² | 375 cm² | 5 | 1 | 22 | 5 | > 0.999 |
| 375 cm² | 375 cm² | 6 | 0 | 18 | 0 | *1 |
| 375 cm² | 375 cm² | 0 | 6 | 2 | 16 | > 0.999 |
| 375 cm² | 375 cm² | 0 | 6 | 0 | 9 | *2 |
| 375 cm² | 375 cm² | 0 | 6 | 2 | 7 | 0.486 |
| 375 cm² | all | 35 | 13 | 156 | 27 | 0.054 |
| 750 cm² | all | 3 | 3 | 29 | 4 | 0.059 |
| 185 cm² | all | 14 | 4 | 34 | 14 | 0.759 |
| 375 cm² | all | 7 | 32 | 30 | 105 | 0.661 |
| 750 cm² | all | 3 | 6 | 8 | 13 | > 0.999 |
| 185 cm² | all | 0 | 6 | 3 | 21 | > 0.999 |
| 375 cm² | all | 9 | 3 | 40 | 17 | > 0.999 |
| 375 cm² | all | 0 | 12 | 4 | 23 | 0.292 |

*1 test not possible because the snakes allways spat at the target

*2 test not possible because the snakes never spat at the target

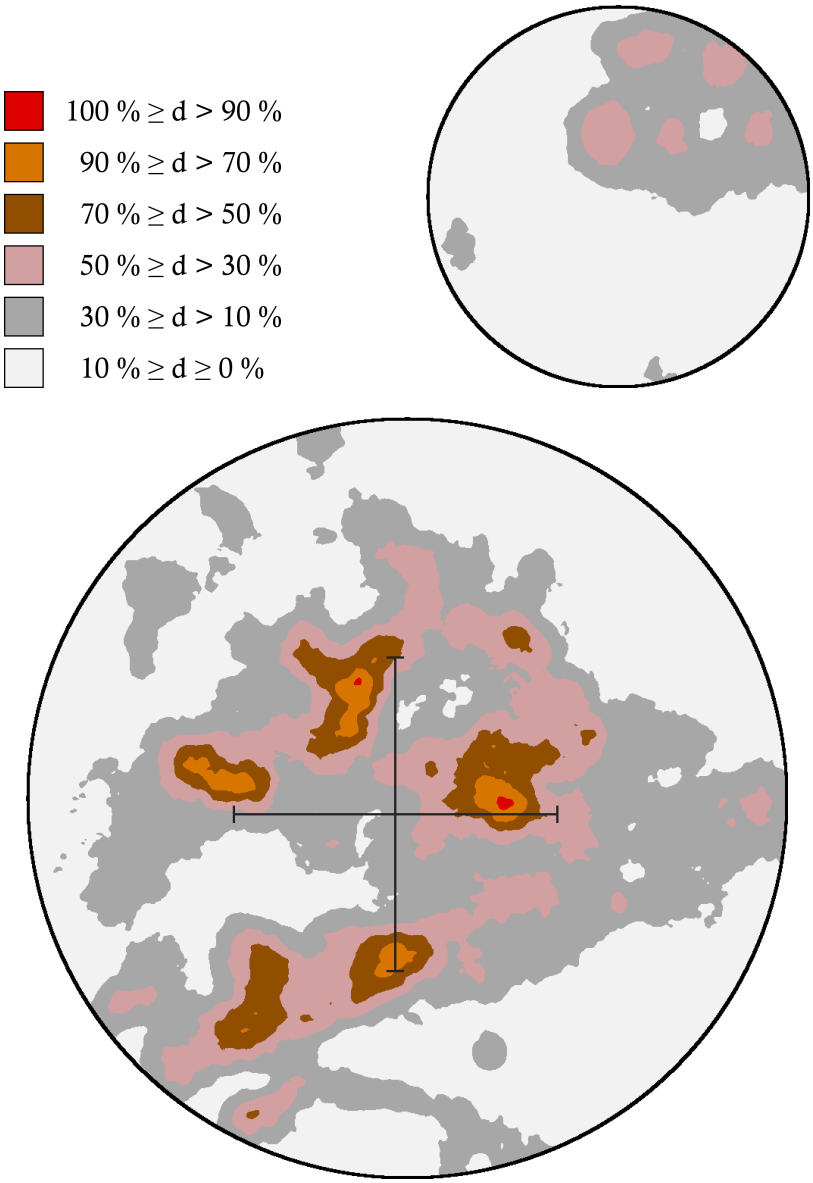


**Figure 16:** Colour coded distribution of venom on the two differently sized disks. No eyes were attached. The cross shows the centre of the venom distribution and its horizontal and vertical standard deviation. 100 % = maximal venom density. $N = 15$.

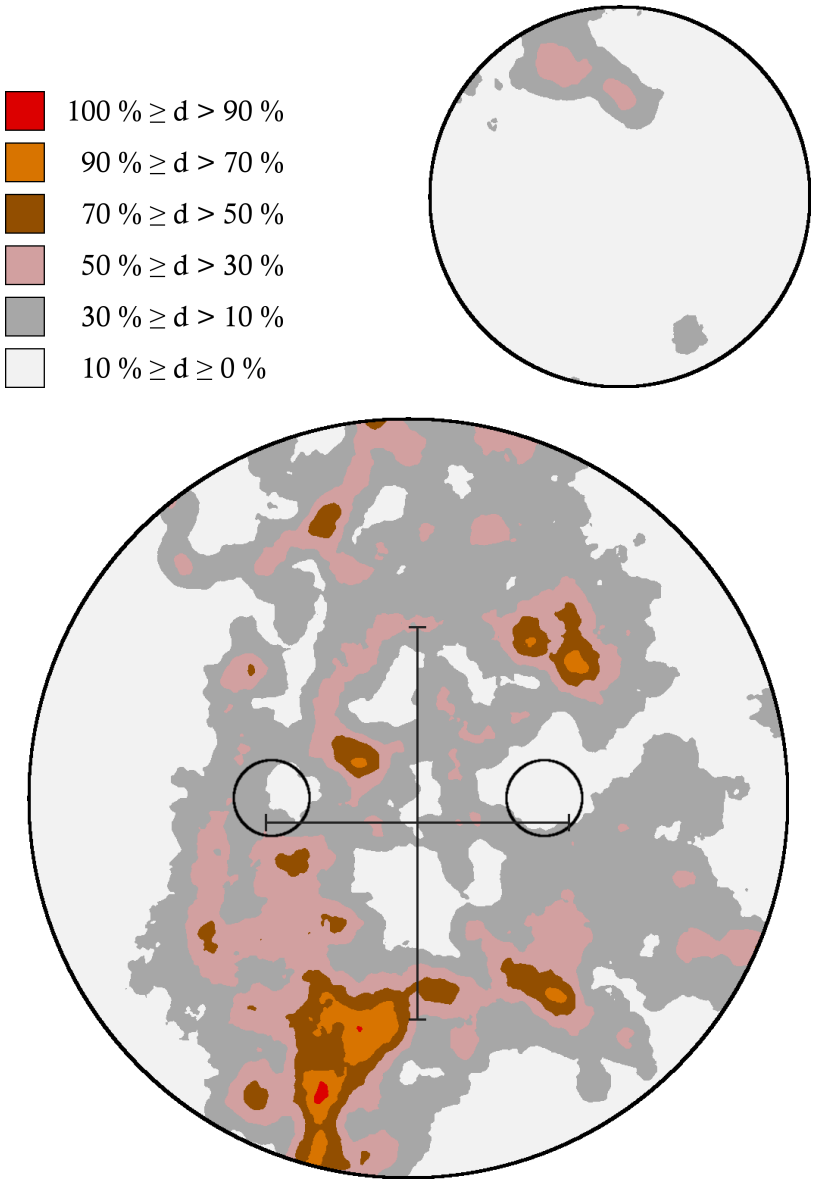


**Figure 17:** Colour coded distribution of venom on the two differently sized disks. Mock eyes were attached to the larger disk. The cross shows the centre of the venom distribution and its horizontal and vertical standard deviation. 100 % = maximal venom density. $N = 14$.

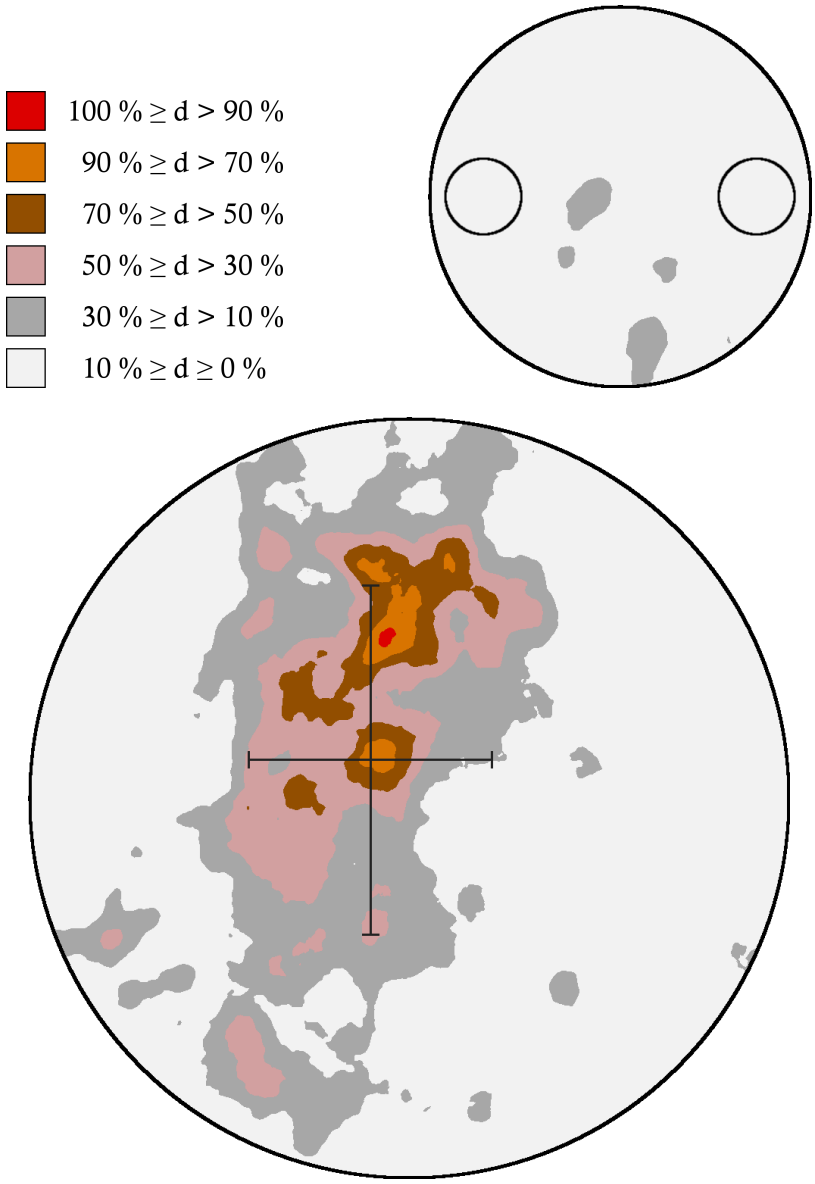


**Figure 18:** Colour coded distribution of venom on the two differently sized disks. Mock eyes were attached to the smaller disk. The cross shows the centre of the venom distribution and its horizontal and vertical standard deviation. 100 % = maximal venom density. $N = 13$.

**Table 6:** Comparison between large and small disk concerning the proportion of venom on either disk. The columns "large disk" and "small disk" give the number of occurrences of at least a certain percentage of venom, given in the column "venom", having hit the corresponding disk.

| Position of eyes | $N$ | Venom | Large disk | Small disk |
|---|---|---|---|---|
| no eyes | 15 | 100 % | 3 | 0 |
| no eyes | 15 | 90 % | 12 | 1 |
| no eyes | 15 | 75 % | 13 | 1 |
| no eyes | 15 | 50 % | 13 | 2 |
| large disk | 14 | 100 % | 2 | 0 |
| large disk | 14 | 90 % | 10 | 1 |
| large disk | 14 | 75 % | 11 | 2 |
| large disk | 14 | 50 % | 11 | 3 |
| small disk | 13 | 100 % | 3 | 0 |
| small disk | 13 | 90 % | 10 | 0 |
| small disk | 13 | 75 % | 11 | 0 |
| small disk | 13 | 50 % | 12 | 1 |

### 3.1.4 Venom distribution and target distance

A total of 178 spitting patterns (for an example see figure 9) were collected (143 from *Naja pallida* and 35 from *N. nigricollis*) to investigate whether the spraying of venom is stereotypical or could be adapted. The total number of patterns collected from one individual ranged from 3 to 63 (*N. pallida*, seven individuals) and from 6 to 12 (*N. nigricollis*, four individuals), respectively. Spitting distances varied between 6.0 cm and 88.1 cm.

The height of all spitting patterns ranged from 4.4 to 40.3 cm (*N. pallida*) and 4.8 to 36.4 cm (*N. nigricollis*). Spitting pattern width varied between 3.4 to 33.9 cm (*N. pallida*) and 5.8 to 22.4 cm (*N. nigricollis*) (figure 19). No differences were found between *N. pallida* and *N. nigricollis* (data grouped for each distance interval of 10 cm; Kolmogorov-Smirnov tests: $N_1 = 35$, $N_2 = 143$, $p > 0.05$). Plotting the dimensions of the spitting patterns of the four snakes from which 15 or more spitting patterns could be collected, as function of target distance revealed positive correlations in three cases (regression slopes 0.14, 0.22, and 0.30), negative correlations in two cases (regression slopes −0.54, −0.87) and no correlation in three cases (for p-values see table 7). Regardless of whether all individuals of both species or only the individuals of one species were pooled, no correlations between target distance and pattern dimensions were found (table 7).

Spitting angles of *N. pallida* and *N. nigricollis* were calculated according to the formula given in figure 8. Spitting angles varied between 7.2° and 89.8° (vertical angles) and 6.9° and 82.9° (horizontal angles) (figure 20). In the abovementioned four snakes $\alpha_{horizontal}$ correlated negatively with target distance; i. e. the horizontal spitting angles decreased with increasing target distance (for p-values see table 7). In three of the four snakes $\alpha_{vertical}$ was also negatively correlated with target distance. In the fourth snake the correlation coefficient was also negative, but this was not significant. Regardless of whether all individuals of both species or only the individuals of one species were pooled, a negative correlation between target distance and spitting angles was found (table 7).

Using the function $2 \cdot arctan\left(\frac{a}{2 \cdot (d+5)}\right)$ (cf. figure 8) we calculated best fitting curves of the data. Regression analyses yielded the constants $a_{vertical} = 23.74$ and $a_{horizontal} = 15.94$. Irrespective of the functions used (e.g. exponential or logarithmical curves) there was a significant decrease in spitting angles with target distance (table 8).

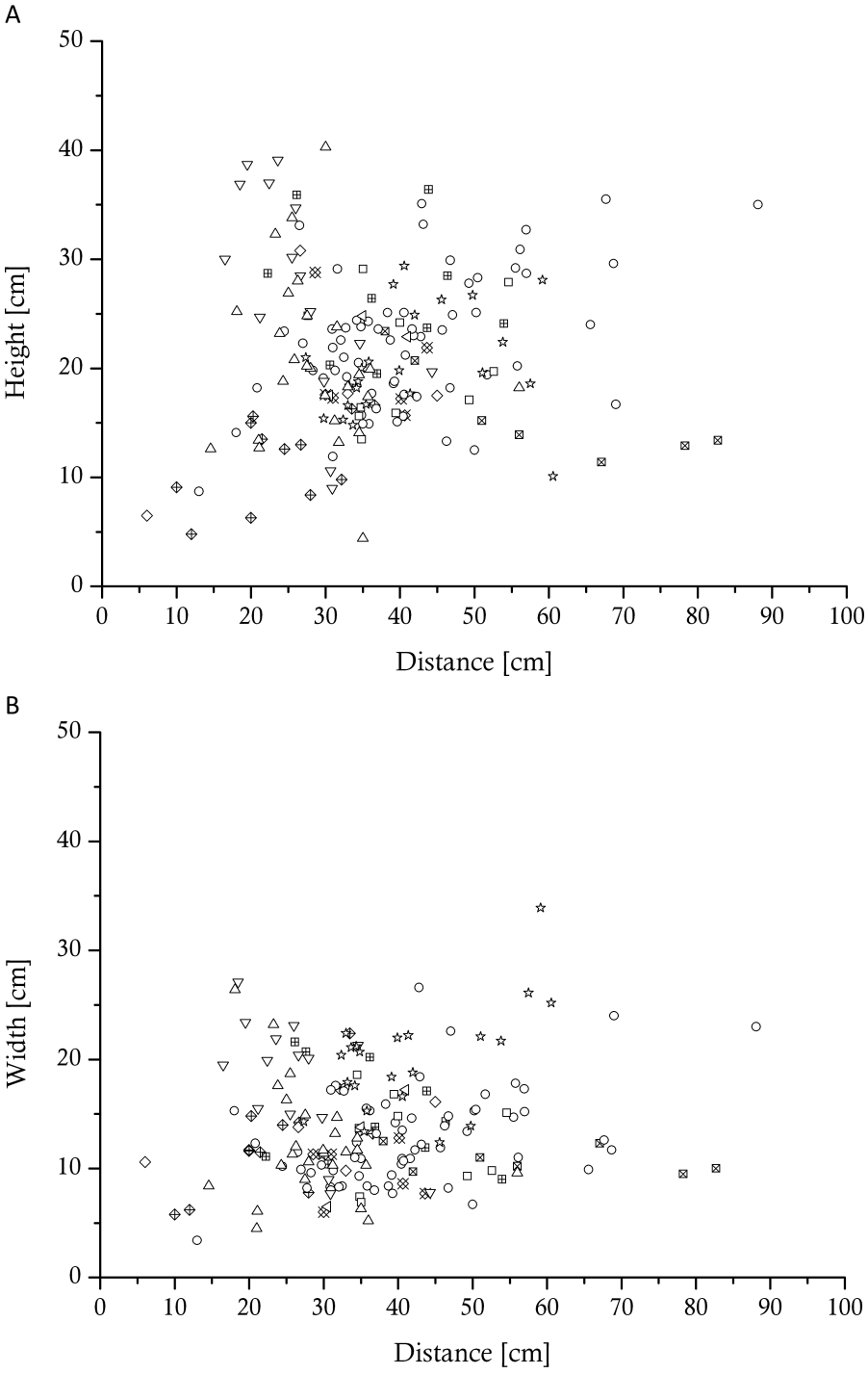


**Figure 19:** Height (A) and width (B) of the spitting patterns as function of target distance. Different symbols indicate different individuals.

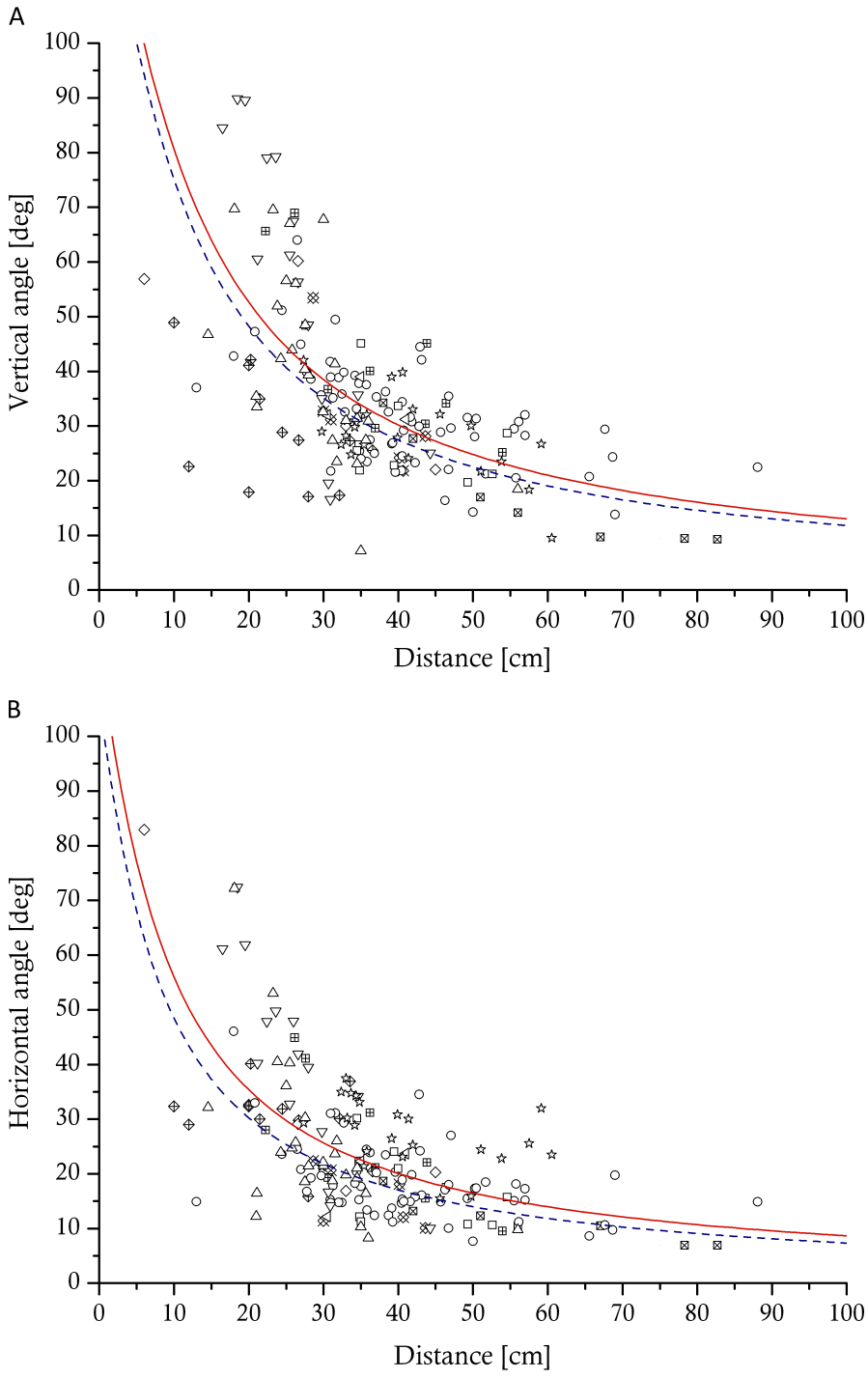


**Figure 20:** Vertical (A) and horizontal (B) spitting angles as function of target distance. The continuous lines are the best fit curves of the form $2 \cdot arctan\left(\frac{a}{2 \cdot (d+5)}\right)$ (see Fig. 8). The dashed line in each panel represents the theoretical angle needed to cover the face/head of the experimenter (21.5 cm high and 13.5 cm wide). Different symbols indicate different individuals.

**Table 7:** Correlation between snake to target distance and attributes of the spitting pattern. $N$ = number of spitting patterns, $r_s$ = test statistic of the Spearman-Rho test. The significance level was 0.05.

| Tested group or individual | Tested correlation with distance | $N$ | $r_s$ | $P$ |
|---|---|---|---|---|
| All individuals | Vertical angle | 178 | −0.640 | < 0.001 |
| All individuals | Horizontal angle | 178 | −0.592 | < 0.001 |
| All individuals | Pattern height | 178 | 0.094 | 0.211 |
| All individuals | Pattern width | 178 | 0.077 | 0.310 |
| *N. pallida* | Vertical angle | 143 | −0.695 | < 0.001 |
| *N. pallida* | Horizontal angle | 143 | −0.517 | < 0.001 |
| *N. pallida* | Pattern height | 143 | 0.060 | 0.480 |
| *N. pallida* | Pattern width | 143 | 0.106 | 0.206 |
| *N. nigricollis* | Vertical angle | 35 | −0.522 | 0.001 |
| *N. nigricollis* | Horizontal angle | 35 | −0.803 | < 0.001 |
| *N. nigricollis* | Pattern height | 35 | 0.225 | 0.194 |
| *N. nigricollis* | Pattern width | 35 | −0.078 | 0.654 |
| pj5 | Vertical angle | 63 | −0.641 | < 0.001 |
| pj5 | Horizontal angle | 63 | −0.400 | 0.001 |
| pj5 | Pattern height | 63 | 0.377 | 0.002 |
| pj5 | Pattern width | 63 | 0.384 | 0.002 |
| p12 | Vertical angle | 15 | −0.907 | < 0.001 |
| p12 | Horizontal angle | 15 | −0.864 | < 0.001 |
| p12 | Pattern height | 15 | −0.736 | 0.002 |
| p12 | Pattern width | 15 | −0.532 | 0.041 |
| pj2 | Vertical angle | 23 | −0.384 | 0.070 |
| pj2 | Horizontal angle | 23 | −0.427 | 0.042 |
| pj2 | Pattern height | 23 | 0.932 | 0.064 |
| pj2 | Pattern width | 23 | 0.443 | 0.034 |
| pj4 | Vertical angle | 26 | −0.907 | < 0.001 |
| pj4 | Horizontal angle | 26 | −0.864 | 0.001 |
| pj4 | Pattern height | 26 | −0.736 | 0.271 |
| pj4 | Pattern width | 26 | −0.532 | 0.337 |

**Table 8:** Nonlinear regression for the spitting angles. For each function the top values are for vertical angles and the bottom values for horizontal angles. The first function is the one the spitting angles should follow theoretically (cf. figure 8).

| Function | $a$ | $b$ | $R^2$ | $F$ | $P$ |
|---|---|---|---|---|---|
| $2 \cdot arctan\left(\frac{a}{2 \cdot (d+5)}\right)$ | 23.738 | — | 0.332 | 1451.13 | $< 0.001$ |
| | 15.938 | — | 0.419 | 1241.82 | $< 0.001$ |
| $a + arctan\left(\frac{1}{b \cdot (d+5)}\right)$ | −0.173 | 0.025 | 0.387 | 794.23 | $< 0.001$ |
| | −0.083 | 0.047 | 0.415 | 612.49 | $< 0.001$ |
| $a + b \cdot ln\,(d+5)$ | 2.502 | −0.515 | 0.379 | 782.29 | $< 0.001$ |
| | 1.987 | −0.427 | 0.399 | 593.85 | $< 0.001$ |
| $a \cdot e^{b \cdot (d+5)}$ | 1.527 | −0.023 | 0.390 | 798.44 | $< 0.001$ |
| | 1.346 | −0.030 | 0.404 | 599.29 | $< 0.001$ |

### 3.1.5 General observations

The willingness to spit varied greatly between individuals. Some snakes never spat during the whole study, while others could be provoked to spit nearly every time. Additionally, the willingness to spit varied over time resulting in the exclusion of a given snake from the experiments for a certain time. Weeks or months could pass until the snake spat again reliably and thus could again be used for the experiments. Although the movements of the targets presented were not recorded, the movements inducing spitting differed between individuals. One snake could easily be provoked to spit by very slow movements while other snakes spat only after rapid movements of the target. Some snakes were more agitated when the target approached the snake from above while other snakes were agitated by frontal approaches. Repeated presentation of a target that typically failed to elicit spitting caused most snake to ignore any target after a while. In these cases they generally stayed motionless until the test series was aborted and the snake removed. Sometimes the snake suddenly snapped out of the rigour and reacted to the target again. Snakes of the species *N. siamensis* showed the tendency to present the backside of their hood to the annoyance. Both *H. haemachatus* and *N. siamensis* lunged forward while spitting and struck the front of the terrarium. Therefore they could not be used for experiments 2.2.3 and 2.2.4 as the contact could alter the spitting direction.

### 3.1.6 Additional experiments

**Videos** Except for one *N. pallida* which spat once at a video showing a stuffed crow on the crt monitor, the cobras did not spit at the videos presented to them. Sometimes they stayed in an erect posture and followed the movements on the screen but often did not react at all to the videos. When the video was projected on a screen the shadow of the experimenter moving through the beam of the projector typically led to a higher arousal of the snake than the video itself.

**Mounted birds** Few snakes reared and spread their hood but none spat at the birds moved in front of them.

## 3.2 Analysis of venom fangs

### 3.2.1 Morphology data

List of abbreviations (cf. figure 11)

$l_F$ length of fang
→ distal end of the entrance lumen to distal end (tip) of the fang

$l_O$ length of discharge orifice
→ basal end of the discharge orifice to distal end of the discharge orifice

$l_{Ct}$ length of venom canal
→ distal end of the entrance lumen to distal end of the discharge orifice

$l_{Cc}$ length of closed part of the venom canal
→ distal end of the entrance lumen to basal end of the discharge orifice

$l_T$ length of fang tip
→ distal end of the discharge orifice to distal end (tip) of the fang

$A_O$ area of discharge orifice

$A_C$ area of venom canal
→ of a certain cross section

$e$ numerical eccentricity
→ of an ellipse fitted to a certain cross section

Exemplary pictures of the 3D reconstructions of the fangs are shown in figures 21 to 25. Reconstructions of all fangs are given in the appendix, figures 53 to 78. The measures of the individual fangs are listed in table 9. There was no difference between the overall length of the fangs of spitting and non-spitting cobras (results

of this and the following tests are shown in table 10). Generally spitting cobras have shorter and smaller (concerning area) discharge orifices than non-spitting cobras. These differences were more distinct if the measures were normalized to fang length $\left(\frac{l_O}{l_F}\right.$ and $\left.\frac{A_O}{l_F \cdot l_F}\right)$. The normalized length of the fang tip (part of the fang distally to the discharge orifice) was greater in spitting than in non-spitting cobras, while there was no difference in the normalized length of the closed part of the venom canal. Additionally the venom canal of spitting cobras is relatively wider $\left(\frac{A_C}{l_F \cdot l_F}\right)$ than the venom canal of non-spitting cobras (table 10; data for each fang are given in figures 79 to 86).

It was possible to discern spitting from non-spitting snakes by sorting the fangs by the quotient of tip length and fang length (figure 26). In addition this caused the fangs of the spitting snakes to be grouped by the species they belonged to, without overlap between species. The tips of the fangs pal4, sia1, and sia2 were broken (cf. figure 74 with 24 and figures 25 and 76 with figure 77) and therefore the fangs were shifted towards the right side of the diagram. Estimations for the complete fangs are given as pal4e, sia1e, and sia2e.

The venom canals of fangs of *Naja nigricollis* and *N. pallida* had distinctive ridges basal to the discharge orifice (figures 23 and 24) while the ridges of the fangs of *N. siamensis* and *Hemachatus haemachatus* were smaller and less distinctive (figures 25 and 21). These ridges had a length of up to 47.4 % of the length of the venom canal ($28.2 \pm 11.1$ %; table 11). There could be more than one ridge per side of the fang along the venom canal. There were no ridges in the non-spitting species (figure 22). In the spitters of the genus *Naja* the distal ridges were by far the highest, generally more than twice the size of the following ones (figures 31 to 43). In *H. haemachatus*, the ridges of one fang were about the same size (figures 29 to 30). The distal ridges of the genus *Naja* and some of the ridges of *H. haemachatus* were not parallel to the suture. They were furthest apart at their basal ends and got nearer to each other towards the discharge orifice (subfigures A of figures 29, 30, and 36 to 43). The other more basal ridges (if present) were approximately parallel to the suture. For non-spitting cobras the maximal differences between the fitted ellipses and borders of the venom canal are depicted in figures 44 to 47. The depth of the suture was larger in African than in Asian cobras (table 12). There was no difference between Asian spitting and non-spitting cobras from. For African cobras this could not be tested because two fangs were broken. *Dendroaspis angusticeps* had a very shallow suture (cf. figures 53 and 54).

The angles of the fitted ellipses together with the eccentricity (figures 48 to 50) describe the base shape (without ridges) of the cross section of the venom canal. Exemplary ellipses with numerical eccentricities from 0.0 to 0.9 are depicted in figure 87. In most species (*D. angusticeps*, *N. kaouthia*, *N. melanoleuca*, *N. naja*, and *N. pallida*) the ellipses' major axes were approximately parallel to the sagittal plane. In *H. haemachatus* and *N. nigricollis* the ellipses' major axes were mostly perpendicular to the sagittal plane, while in *N. siamensis* there was a tendency to the major axes being more or less perpendicular to the sagittal plane at the fang's distal end and being more or less parallel to it towards its basal end. The part scanned of the fang of *N. haje* showed only a relatively short segment with a closed venom canal. Within that segment the major axes were approximately perpendicular to the sagittal plane.

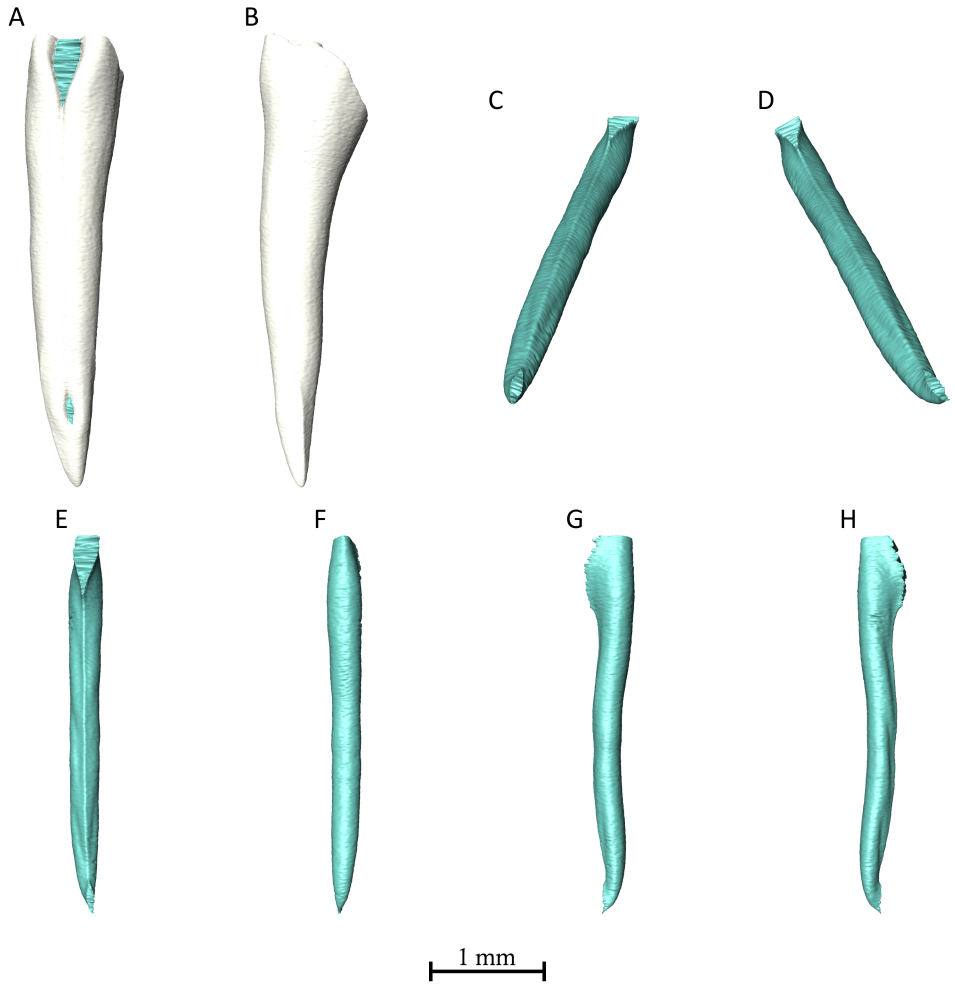


**Figure 21:** (*Hemachatus haemachatus*) 3D reconstruction of fang hae1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed.

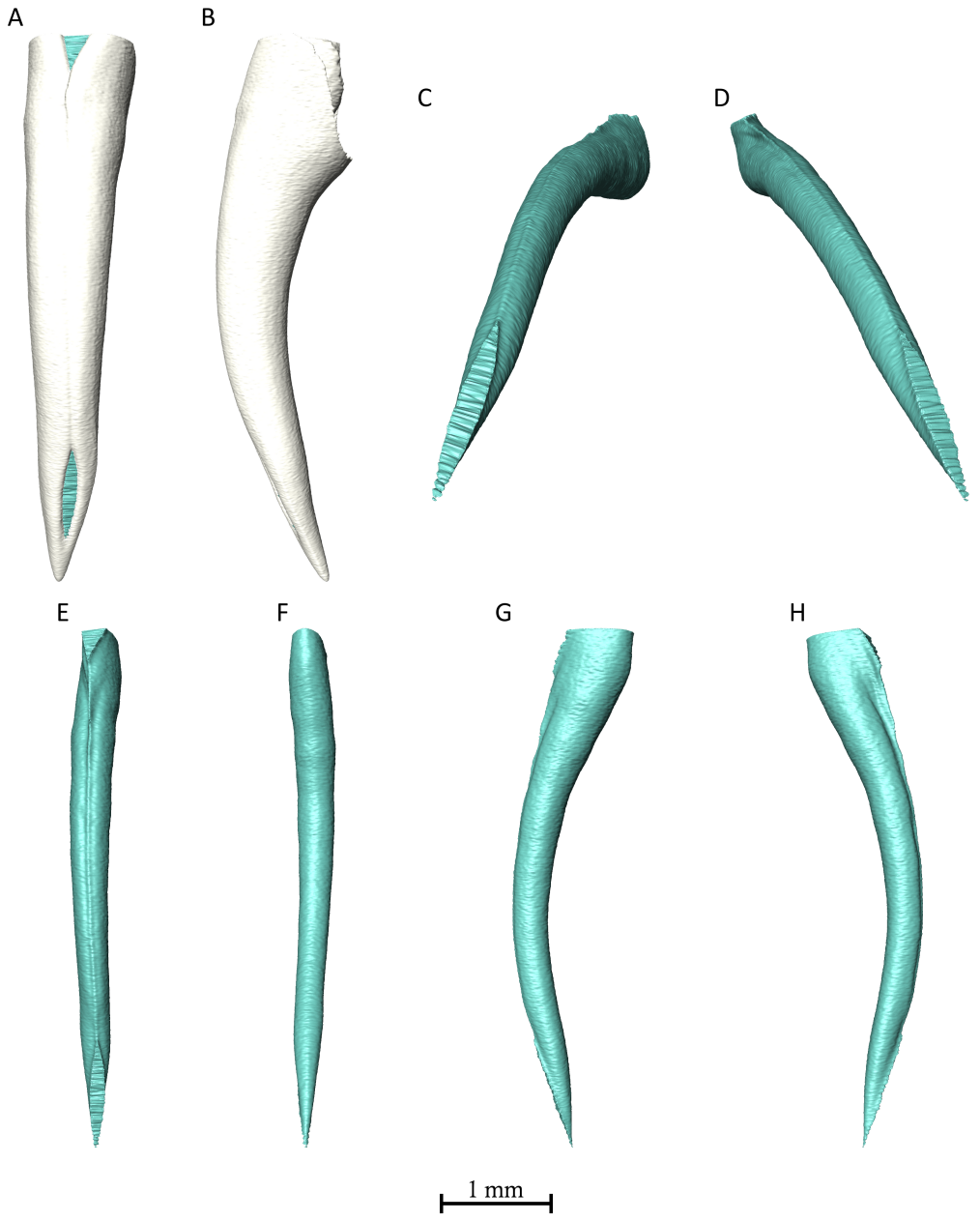


**Figure 22:** (*Naja naja* – non-spitting cobra) 3D reconstruction of fang naj3. Cyan = venom canal. For additional information see figure 21.

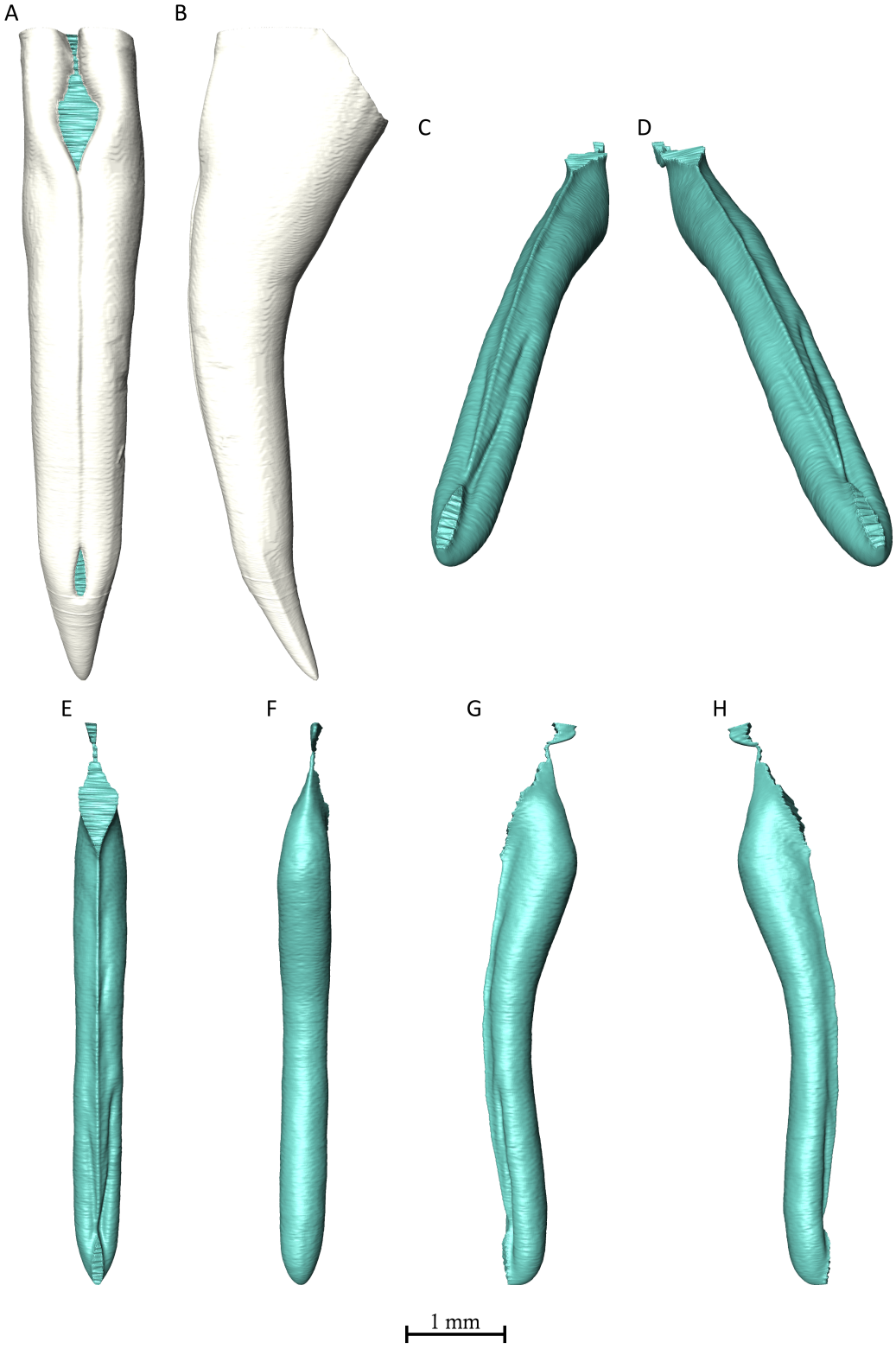


**Figure 23:** (*Naja nigricollis*) 3D reconstruction of fang nig1. Cyan = venom canal. For additional information see figure 21.

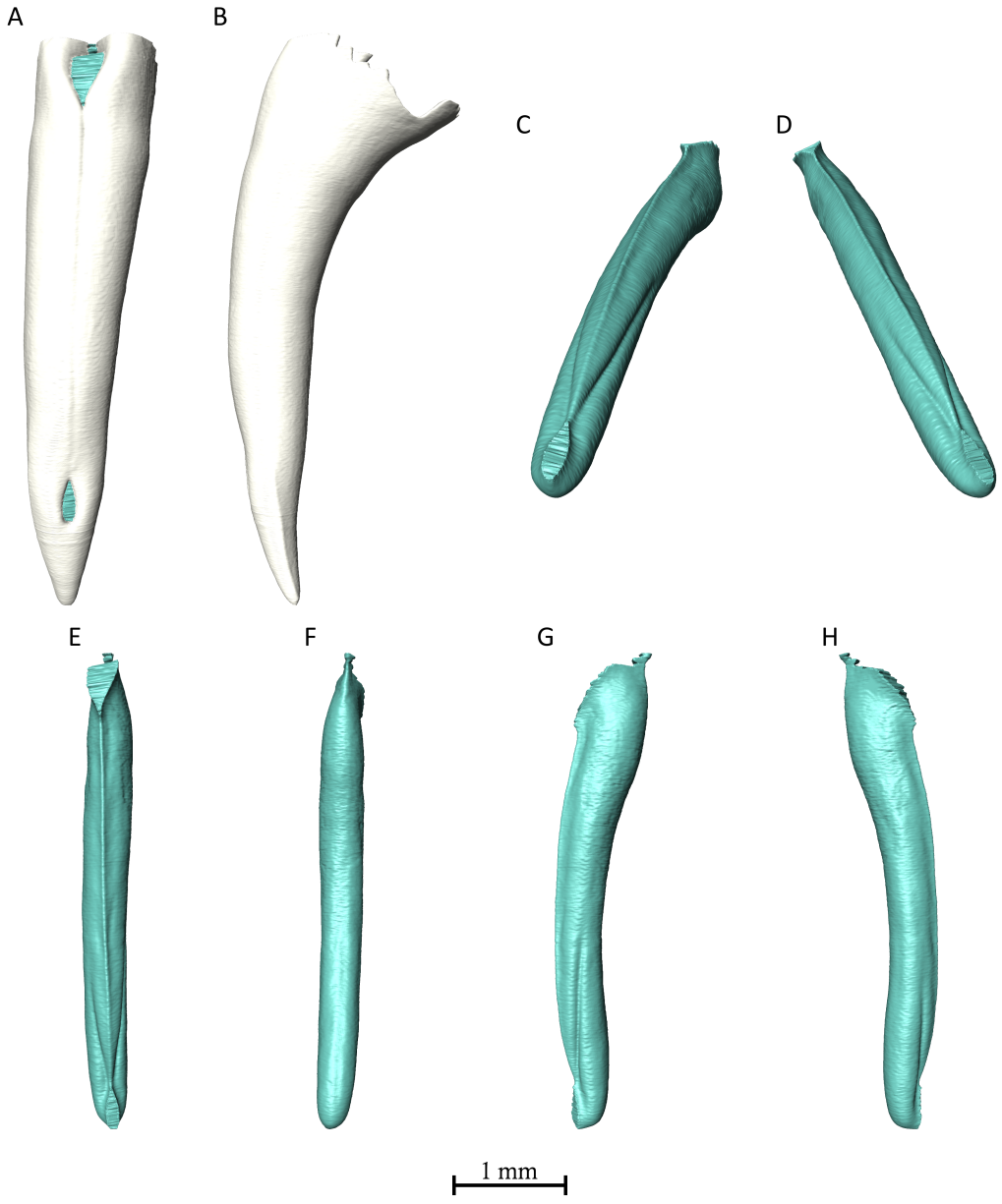


**Figure 24:** (*Naja pallida*) 3D reconstruction of fang pal1. Cyan = venom canal. For additional information see figure 21.

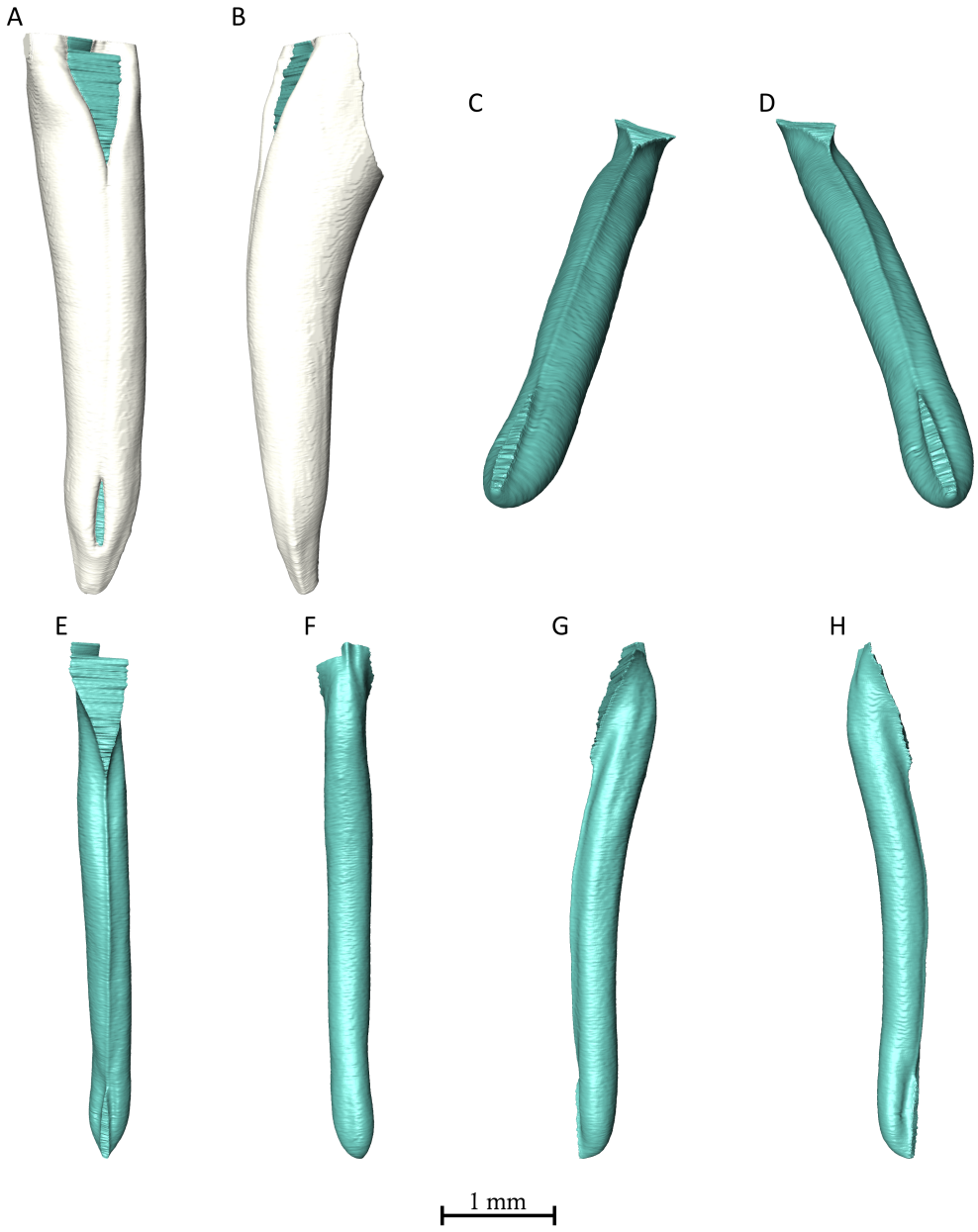


**Figure 25:** (*Naja siamensis*) 3D reconstruction of fang sia1. Cyan = venom canal. The fang tip is broken. For additional information see figure 21.

**Table 9:** Measurement of the fangs (cf. figure 11). $l_F$ = length of the fang, measured from the distal end of the entrance lumen to the fang tip (Bogert, 1943). $l_O$ = length of the discharge orifice of the venom canal, $l_{Ct}$ = total length of the venom canal from the distal end of the entrance lumen to the distal end of the discharge orifice, $l_{Cc}$ = length of the section of the venom canal between the two orifices, $l_T$ = distance from the fang tip to the distal end of the discharge orifice, $A_O$ = area of the discharge orifice. All measurements are given in millimetres or square millimetres, respectively.

| Fang | $l_F$ | $l_O$ | $l_{Ct}$ | $l_{Cc}$ | $l_T$ | $A_O$ | |
|---|---|---|---|---|---|---|---|
| ang1 | | 1.078 | | | 0.346 | 0.106 | *1 |
| ang2 | 6.120 | 1.388 | 5.709 | 4.438 | 0.467 | 0.154 | |
| hae1 | 3.505 | 0.340 | 2.860 | 2.521 | 0.675 | 0.018 | |
| hae2 | 3.595 | 0.285 | 2.883 | 2.598 | 0.728 | 0.015 | |
| haj1 | | 1.038 | | | 0.519 | 0.075 | *1 |
| kao1 | 4.079 | 0.877 | 3.663 | 2.800 | 0.444 | 0.059 | |
| kao2 | 5.566 | 1.172 | 4.533 | 3.374 | 0.571 | 0.092 | |
| kao3 | 4.523 | 0.877 | 4.020 | 3.163 | 0.563 | 0.071 | |
| mel1 | | 0.690 | | | 0.614 | 0.044 | *1 |
| mel2 | 2.866 | 0.534 | 2.562 | 2.042 | 0.325 | 0.026 | |
| naj1 | 4.477 | 1.256 | 3.978 | 2.773 | 0.555 | 0.104 | |
| naj2 | 5.234 | 1.339 | 4.860 | 3.583 | 0.407 | 0.105 | |
| naj3 | 5.183 | 1.275 | 4.649 | 3.450 | 0.587 | 0.121 | |
| nig1 | 5.883 | 0.639 | 4.495 | 3.855 | 1.530 | 0.060 | |
| nig2 | 5.820 | 0.693 | 4.290 | 3.596 | 1.624 | 0.049 | |
| nig3 | 5.813 | 0.627 | 4.358 | 3.732 | 1.545 | 0.058 | |
| nig4 | 5.316 | 0.511 | 4.021 | 3.510 | 1.377 | 0.045 | |
| nig5 | 5.577 | 0.691 | 4.275 | 3.584 | 1.389 | 0.069 | |
| pal1 | 4.960 | 0.516 | 3.888 | 3.374 | 1.157 | 0.045 | |
| pal2 | 4.756 | 0.488 | 3.724 | 3.236 | 1.126 | 0.042 | |
| pal3 | 4.407 | 0.489 | 3.458 | 2.970 | 1.035 | 0.043 | |
| pal4 | 4.729 | 0.574 | 3.747 | 3.176 | 1.076 | 0.056 | *2 |
| pal4e | 4.835 | 0.574 | 3.747 | 3.176 | 1.165 | 0.056 | *3 |
| sia1 | 4.240 | 0.799 | 3.630 | 2.833 | 0.664 | 0.055 | *2 |
| sia1e | 4.378 | 0.799 | 3.630 | 2.833 | 0.796 | 0.055 | *3 |
| sia2 | 4.162 | 0.744 | 3.706 | 2.962 | 0.506 | 0.046 | *2 |
| sia2e | 4.413 | 0.744 | 3.706 | 2.962 | 0.744 | 0.046 | *3 |
| sia3 | 3.472 | 0.492 | 2.889 | 2.401 | 0.647 | 0.031 | |
| sia4 | 4.718 | 0.739 | 3.932 | 3.196 | 0.839 | 0.061 | |

*1 fang broken – basal part not scanned

*2 fang tip broken → $l_F$ and $l_T$ too short

*3 fang tip estimated

**Table 10:** Test results of Kolmogorov-Smirnov tests for differences between fangs of spitting and non-spitting cobras. A subscript $d$ marks that only the distal half of the venom canal was considered. Parentheses after a $\bar{x}$ or a $\sigma_x$ in the $X$-column specify the means or standard deviation tested. $N$ is the number of fangs, $\bar{x}$ is the mean value, and $\sigma_x$ is the standard deviation. Values of spitting cobras are marked with a subscript $s$ while values of non-spitting cobras are marked with a subscript $ns$. Lengths are given in mm and areas are given in mm$^2$. The numerical eccentricity $e$ is dimensionless. $Z$ is the test statistic of the Kolmogorov-Smirnov test. The significance level was set to 0.05.

| $X$ | $N_s$ | $N_{ns}$ | $\bar{x}_s \pm \sigma_{x_s}$ | $\bar{x}_{ns} \pm \sigma_{x_{ns}}$ | $Z$ | $p$ |
|---|---|---|---|---|---|---|
| $l_F$ | 13 | 7 | $4.948 \pm 0.713$ | $4.490 \pm 0.834$ | 0.820 | 0.511 |
| $l_O$ | 13 | 9 | $0.616 \pm 0.111$ | $1.006 \pm 0.281$ | 1.794 | 0.003 |
| $\frac{l_O}{l_F}$ | 13 | 7 | $0.126 \pm 0.028$ | $0.230 \pm 0.034$ | 2.133 | $< 0.001$ |
| $\frac{l_T}{l_F}$ | 13 | 7 | $0.228 \pm 0.037$ | $0.111 \pm 0.016$ | 2.133 | $< 0.001$ |
| $\frac{l_{Cc}}{l_F}$ | 13 | 7 | $0.662 \pm 0.020$ | $0.676 \pm 0.030$ | 1.055 | 0.216 |
| $A_O$ | 13 | 9 | $0.0508 \pm 0.0101$ | $0.0774 \pm 0.0312$ | 1.537 | 0.018 |
| $\frac{A_O}{l_F \cdot l_F}$ | 13 | 7 | $0.00213 \pm 0.00047$ | $0.00390 \pm 0.00071$ | 2.133 | $< 0.001$ |
| $\bar{x}\left(\frac{A_C}{l_F \cdot l_F}\right)$ | 13 | 7 | $0.00565 \pm 0.00082$ | $0.00353 \pm 0.00078$ | 2.133 | $< 0.001$ |
| $\bar{x}_d\left(\frac{A_C}{l_F \cdot l_F}\right)$ | 13 | 7 | $0.00493 \pm 0.00076$ | $0.00276 \pm 0.00054$ | 2.133 | $< 0.001$ |
| $\sigma_x\left(\frac{A_C}{l_F \cdot l_F}\right)$ | 13 | 7 | $0.00105 \pm 0.00012$ | $0.00105 \pm 0.00040$ | 0.609 | 0.852 |
| $\sigma_{x_d}\left(\frac{A_C}{l_F \cdot l_F}\right)$ | 13 | 7 | $0.00046 \pm 0.00019$ | $0.00026 \pm 0.00011$ | 1.219 | 0.102 |
| $e$ | 13 | 7 | $0.233 \pm 0.056$ | $0.319 \pm 0.087$ | 1.149 | 0.143 |
| $e_d$ | 13 | 7 | $0.232 \pm 0.075$ | $0.280 \pm 0.114$ | 0.609 | 0.852 |

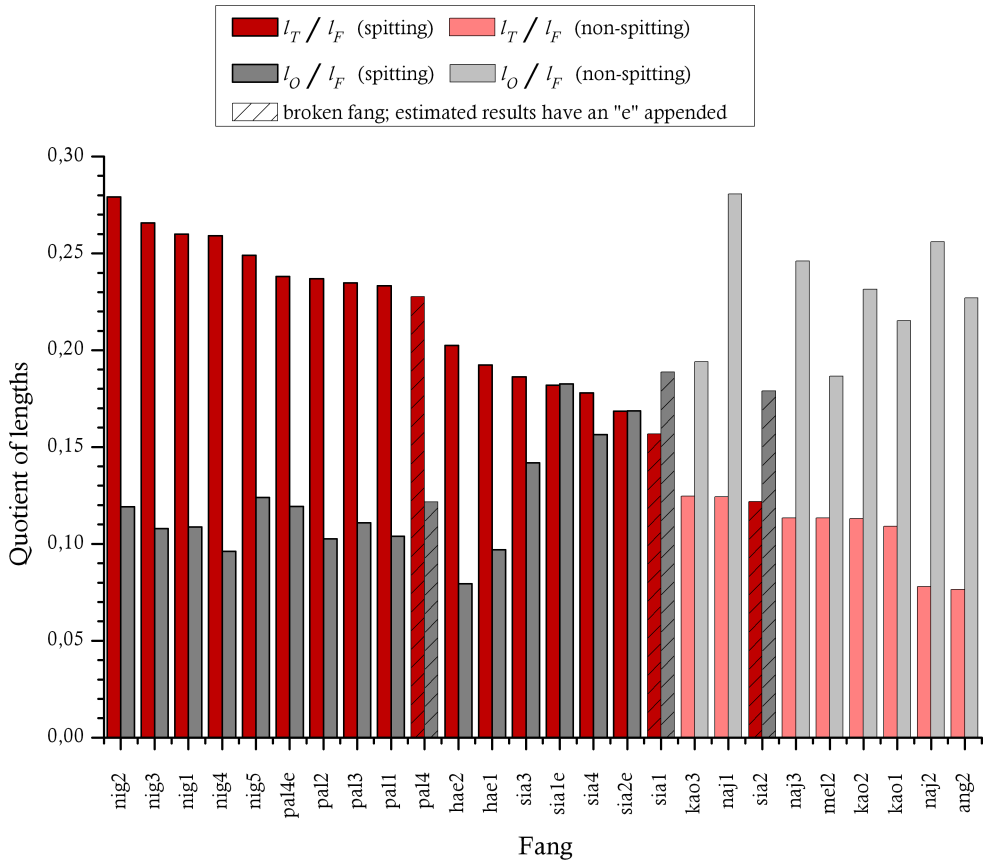


**Figure 26:** Relative length of the discharge orifices and the fang tips of the measured fengs. Given are the length of the fang tip ($l_T$) divided by the length of the fang ($l_F$) and the length of the discharge orifice ($l_O$) divided by the length of the fang ($l_F$). (The values of $l_F$, $l_T$, and $l_O$ are listed in table 9.)

**Table 11:** Length of the distal ridges in the venom canal. $l_{Rlc}$ is the length of the ridge on the left side within the closed part of the venom canal, $l_{Rlt}$ is the length of the ridge on the left side of the venom canal including the discharge orifice. $l_{Rrc}$ and $l_{Rrt}$ are the corresponding valules for the right side of the canal. $l_{Cc}$ and $l_{Ct}$ are the lengths of the closed part of the venom canal and the venom canal including the discharge orifice.

| Fang | $l_{Rlc}$ | $l_{Rrc}$ | $l_{Rlt}$ | $l_{Rrt}$ | $\frac{l_{Rlc}}{l_{Cc}}$ | $\frac{l_{Rrc}}{l_{Cc}}$ | $\frac{l_{Rlt}}{l_{Ct}}$ | $\frac{l_{Rrt}}{l_{Ct}}$ |
|---|---|---|---|---|---|---|---|---|
| nig1 | 1.600 | 1.337 | 1.913 | 1.320 | 0.415 | 0.347 | 0.426 | 0.294 |
| nig2 | 1.262 | 1.062 | 1.593 | 1.422 | 0.351 | 0.295 | 0.371 | 0.331 |
| nig3 | 1.148 | 1.158 | 1.255 | 1.412 | 0.308 | 0.310 | 0.288 | 0.324 |
| nig4 | 1.006 | 1.422 | 1.162 | 1.448 | 0.287 | 0.405 | 0.289 | 0.360 |
| nig5 | 1.043 | 0.564 | 1.317 | 0.577 | 0.291 | 0.157 | 0.308 | 0.135 |
| pal1 | 1.557 | 1.188 | 1.608 | 1.296 | 0.461 | 0.352 | 0.413 | 0.333 |
| pal2 | 0.606 | 0.607 | 0.657 | 0.664 | 0.187 | 0.178 | 0.176 | 0.178 |
| pal3 | 0.679 | 0.765 | 0.752 | 0.835 | 0.229 | 0.257 | 0.217 | 0.241 |
| pal4 | 1.504 | 0.911 | 1.556 | 1.021 | 0.474 | 0.287 | 0.415 | 0.273 |
| sia1 | 0.503 | — | 0.924 | 0.422 | 0.178 | — | 0.255 | 0.116 |
| sia2 | 0.120 | 0.896 | 0.448 | 1.110 | 0.041 | 0.302 | 0.121 | 0.300 |
| sia4 | — | 0.298 | 0.198 | 0.699 | — | 0.093 | 0.050 | 0.178 |

**Table 12:** Comparison of the dimension of the suture. For each fang the mean depth of the suture of the distal half of the fang was adjusted by the length of the fang. African species are abbriviated as $Afr$ and Asian species as $Asi$. Spitting and non-spitting species are abbreviated as *spit* and *non* respectively. $N$ is the number of fangs, $\bar{x}$ is the mean value, and $\sigma_x$ is the standard deviation. $Z$ is the test statistic of the Kolmogorov-Smirnov test. The significance level was set to 0.05. Values for *D. angusticeps* and *H. haemachatus* are given for comparison.

| Tested pair | $N_1$ | $N_2$ | $\bar{x}_1 \pm \sigma_{x_1}$ | $\bar{x}_2 \pm \sigma_{x_2}$ | $Z$ | $p$ |
|---|---|---|---|---|---|---|
| $\frac{Afr}{all}$ — $\frac{Asi}{all}$ | 10 | 10 | $-0,00610 \pm 0,00290$ | $-0,00160 \pm 0,00073$ | 2.012 | 0.001 |
| $\frac{Afr}{spit}$ — $\frac{Asi}{spit}$ | 9 | 4 | $-0,00612 \pm 0,00307$ | $-0,00189 \pm 0,00071$ | 1.479 | 0.025 |
| $\frac{Afr}{non}$ — $\frac{Asi}{non}$ | 1 | 6 | $-0,00596$ | $-0,00142 \pm 0,00074$ | test not possible | |
| $\frac{Afr}{spit}$ — $\frac{Afr}{non}$ | 9 | 1 | $-0,00612 \pm 0,00307$ | $-0,00596$ | test not possible | |
| $\frac{Asi}{spit}$ — $\frac{Asi}{non}$ | 4 | 6 | $-0,00189 \pm 0,00071$ | $-0,00142 \pm 0,00074$ | 0.645 | 0.799 |
| $\frac{all}{spit}$ — $\frac{all}{non}$ | 13 | 7 | $-0,00482 \pm 0,00325$ | $-0,00206 \pm 0,00184$ | 1.172 | 0.128 |
| *D. angusticeps* | 1 | | $-0,00028$ | | for comparison | |
| *H. haemachatus* | 2 | | $-0,00394 \pm 0,00113$ | | for comparison | |

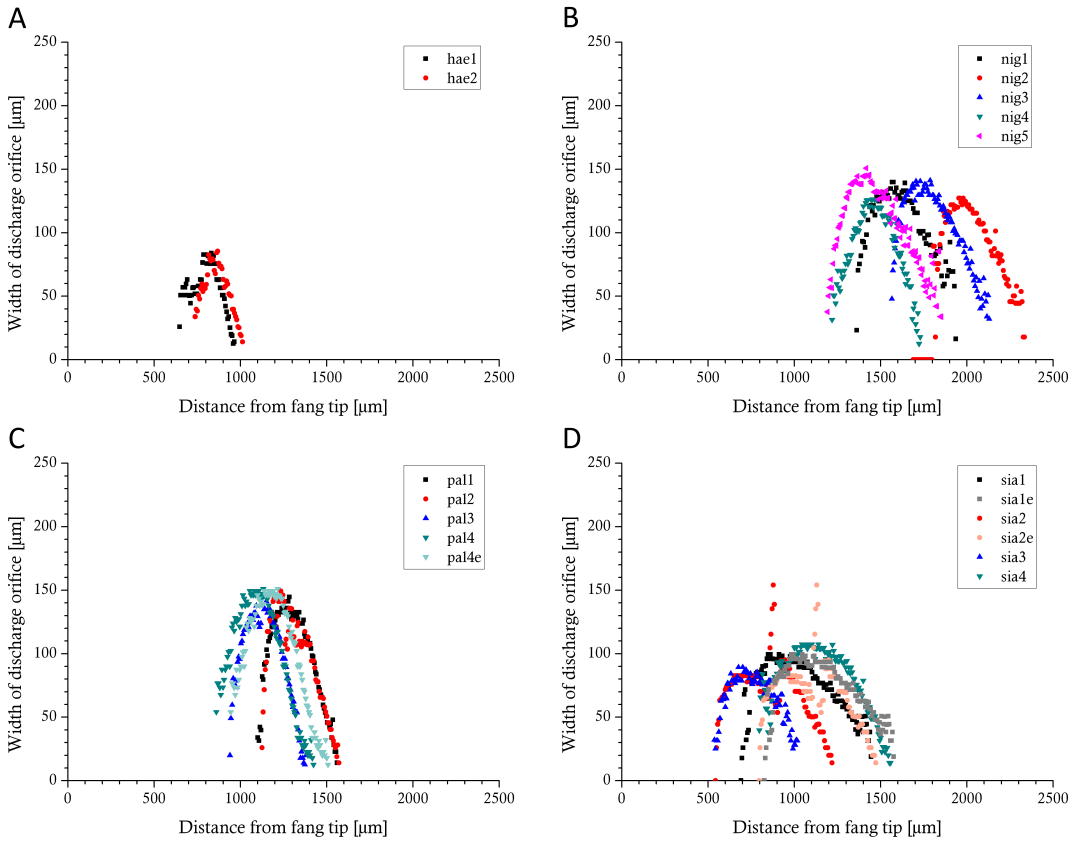


**Figure 27:** Width of the discharge orifices of spitting cobras. A: *H. haemachatus*, B: *N. nigricollis*, C: *N. pallida*, D: *N. siamensis*

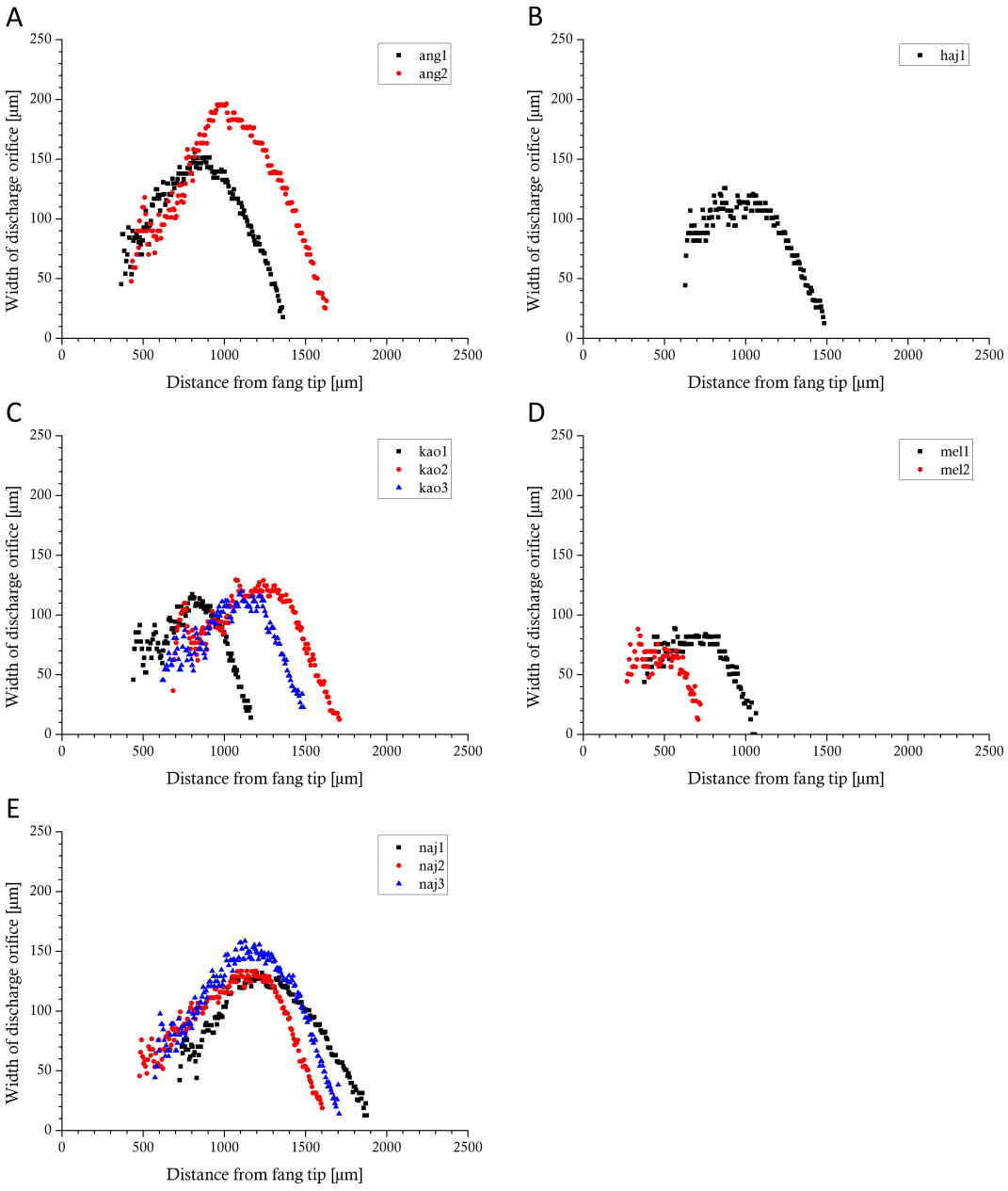


**Figure 28:** Width of the discharge orifice of non-spitting snakes A: *D. angusticeps*, B: *N. haje*, C: *N. kaouthia*, D: *N. melanoleuca*, E: *N. naja*

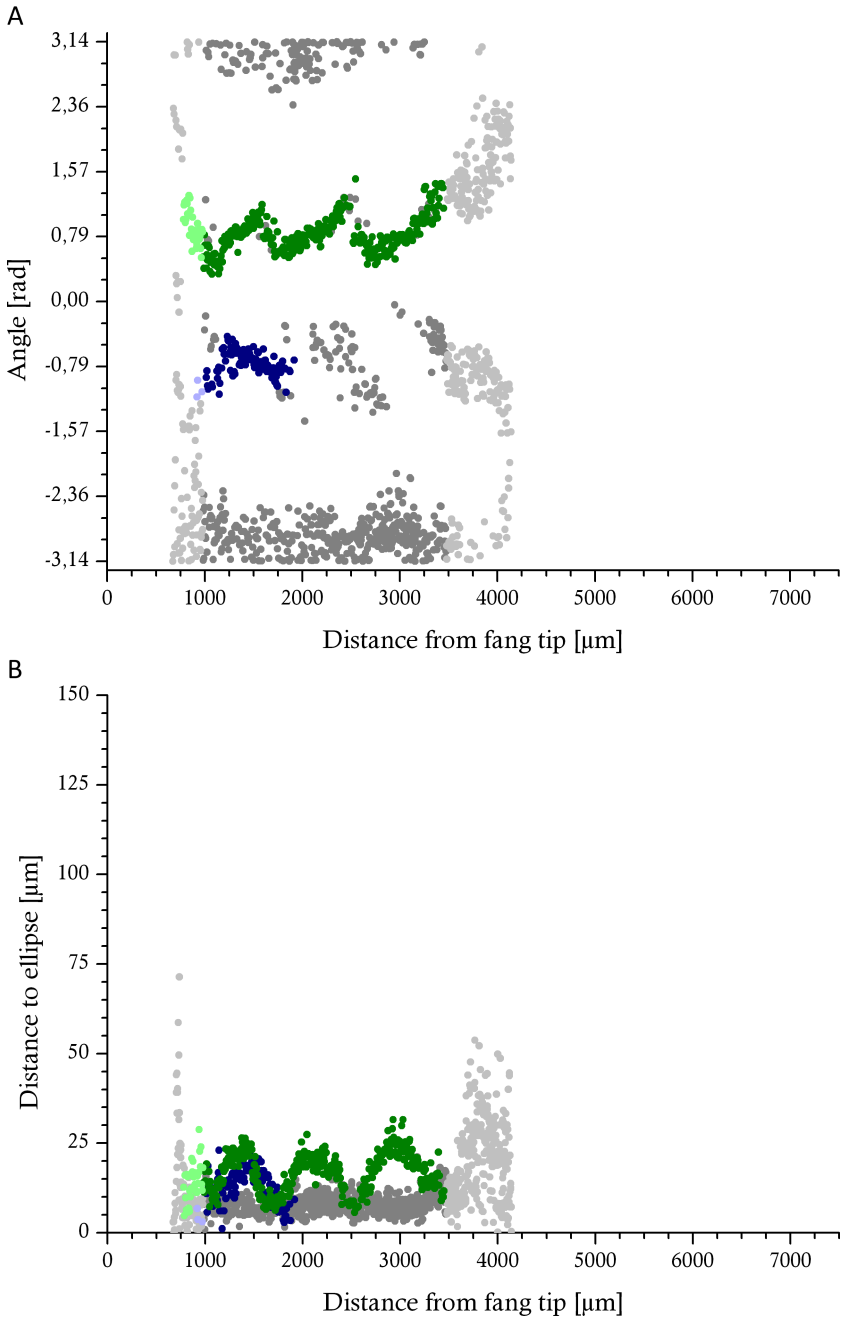


**Figure 29:** (*Hemachatus haemachatus*) Maximal differences between the border of the venom canal and the fitted ellipse of fang hae1 (cf. figure 10). In this figure and figures 30 to 47 the three largest relative maxima (concerning the difference between canal border and ellipse) per slice are given (A). The negative angles correspond to the left, the positive angles to the right side of the fang, respectively (B). The fang's suture is around zero rad for the distal end of the venom canal. Some fangs were slightly twisted so that the position of the suture will divert from zero towards the basal and of the fang. High values of differences from the fitted ellipse towards the basal end of the fang are artefacts resulting from the widening of the venom canal at that point. Light colours represent slices with an open venom canal, dark colours slices with a closed venom canal.

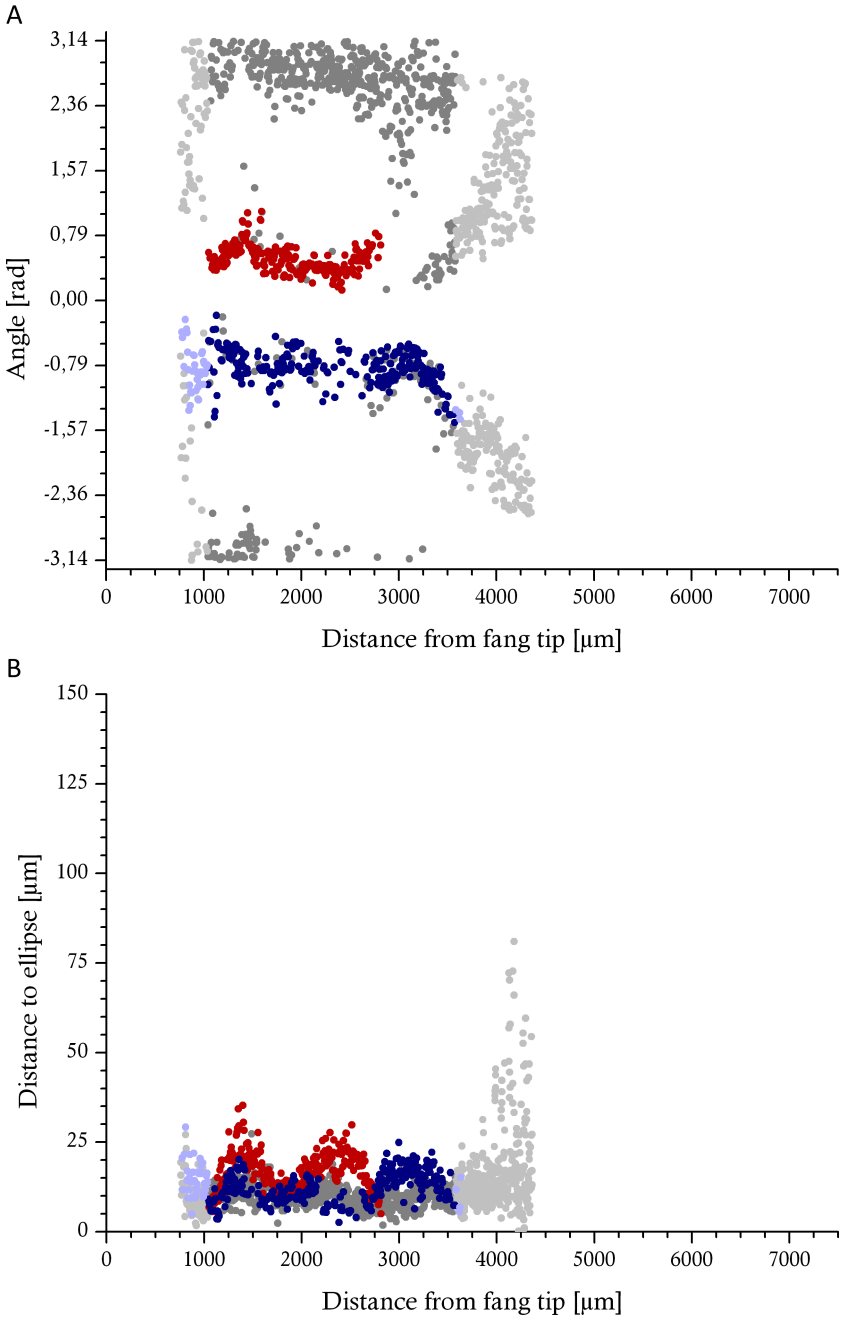


**Figure 30:** (*Hemachatus haemachatus*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang hae2. For additional information see figure 29.

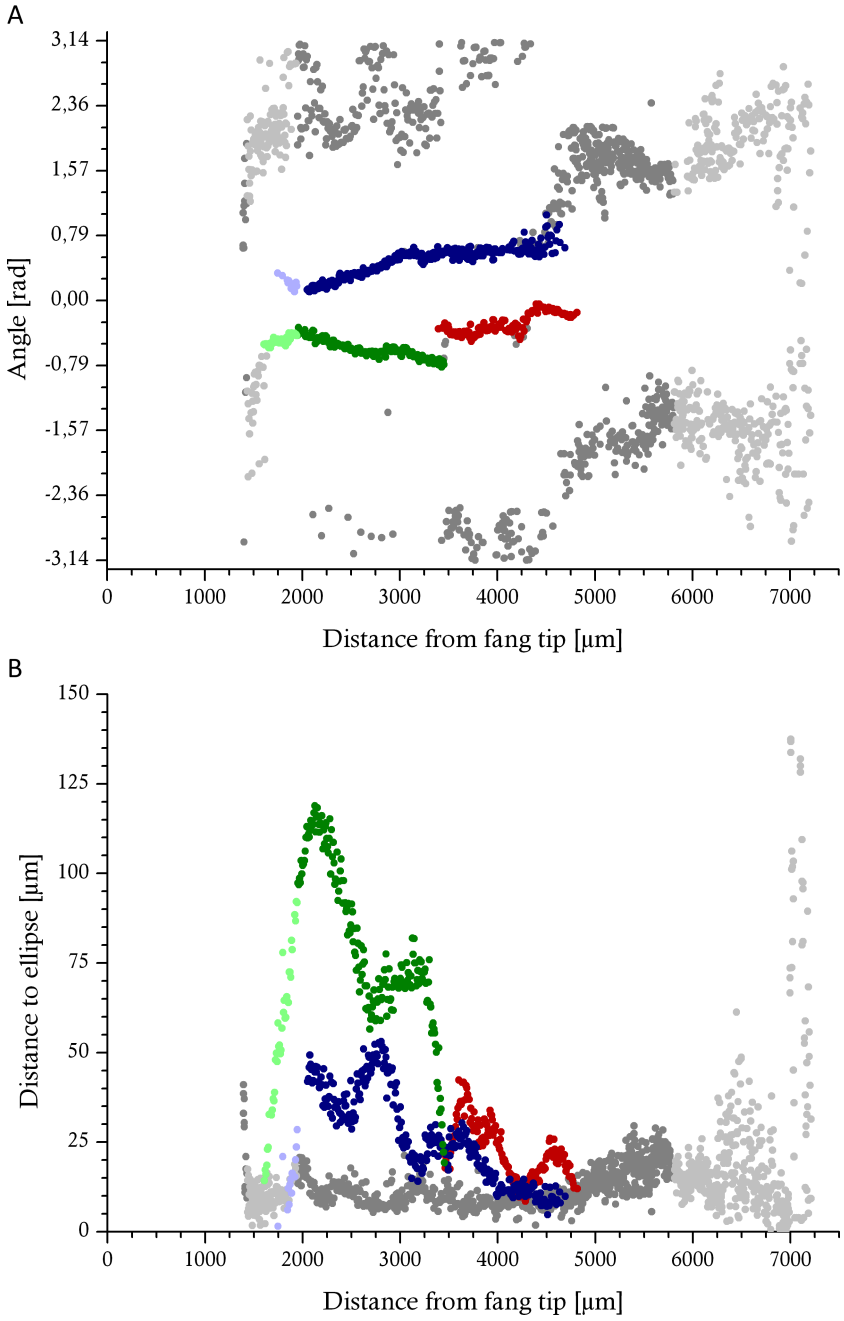


**Figure 31:** (*Naja nigricollis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang nig1. For additional information see figure 29.

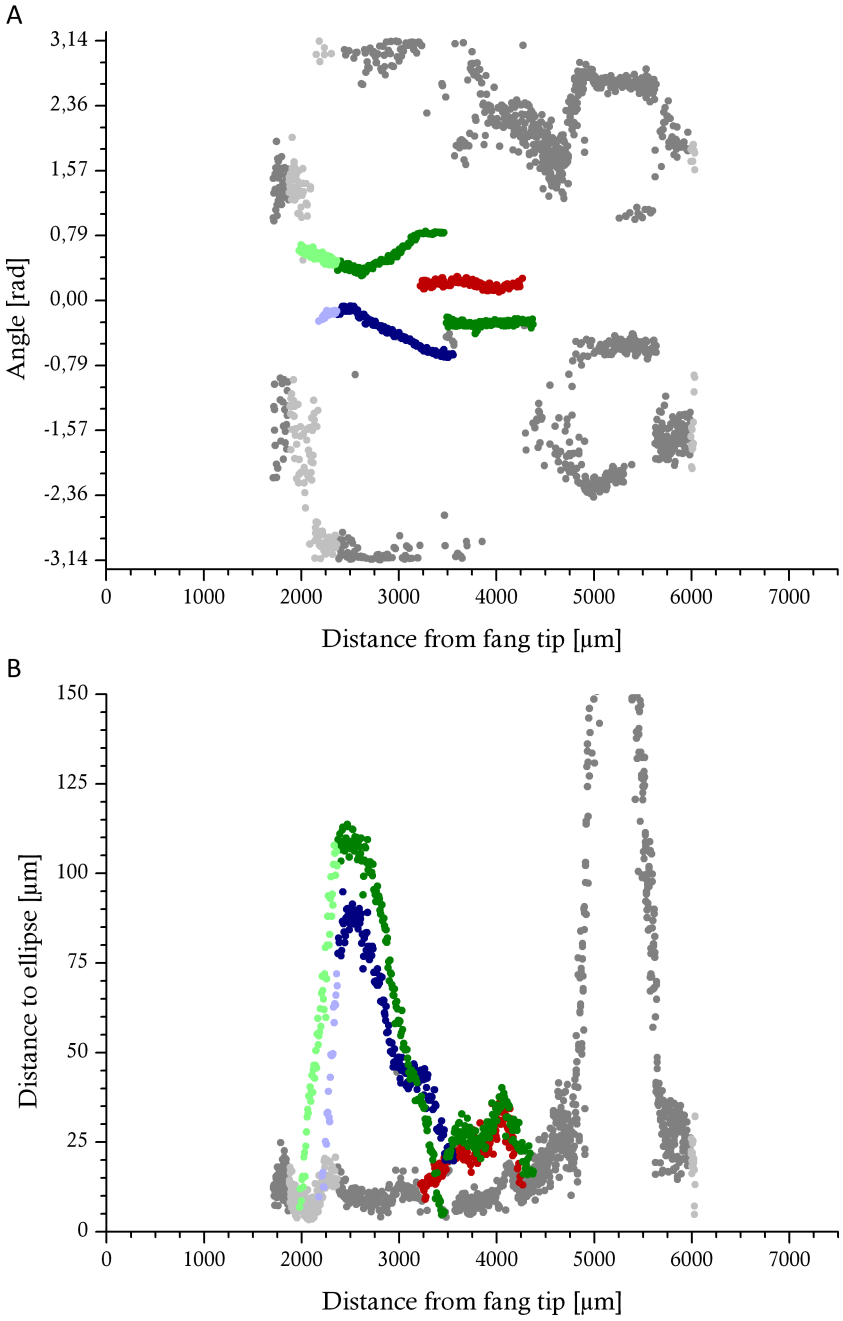


**Figure 32:** (*Naja nigricollis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang nig2. For additional information see figure 29.

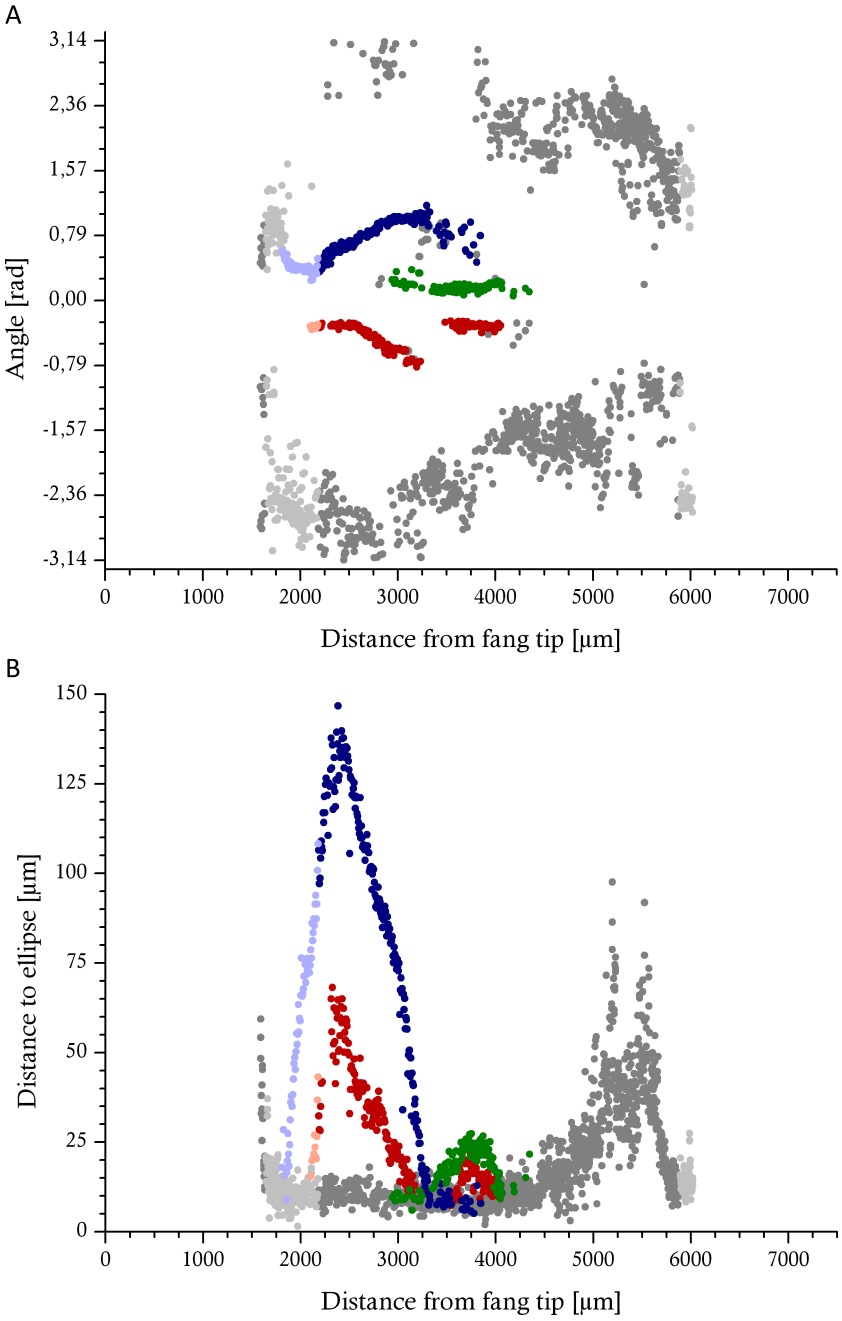


**Figure 33:** (*Naja nigricollis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang nig3. For additional information see figure 29.

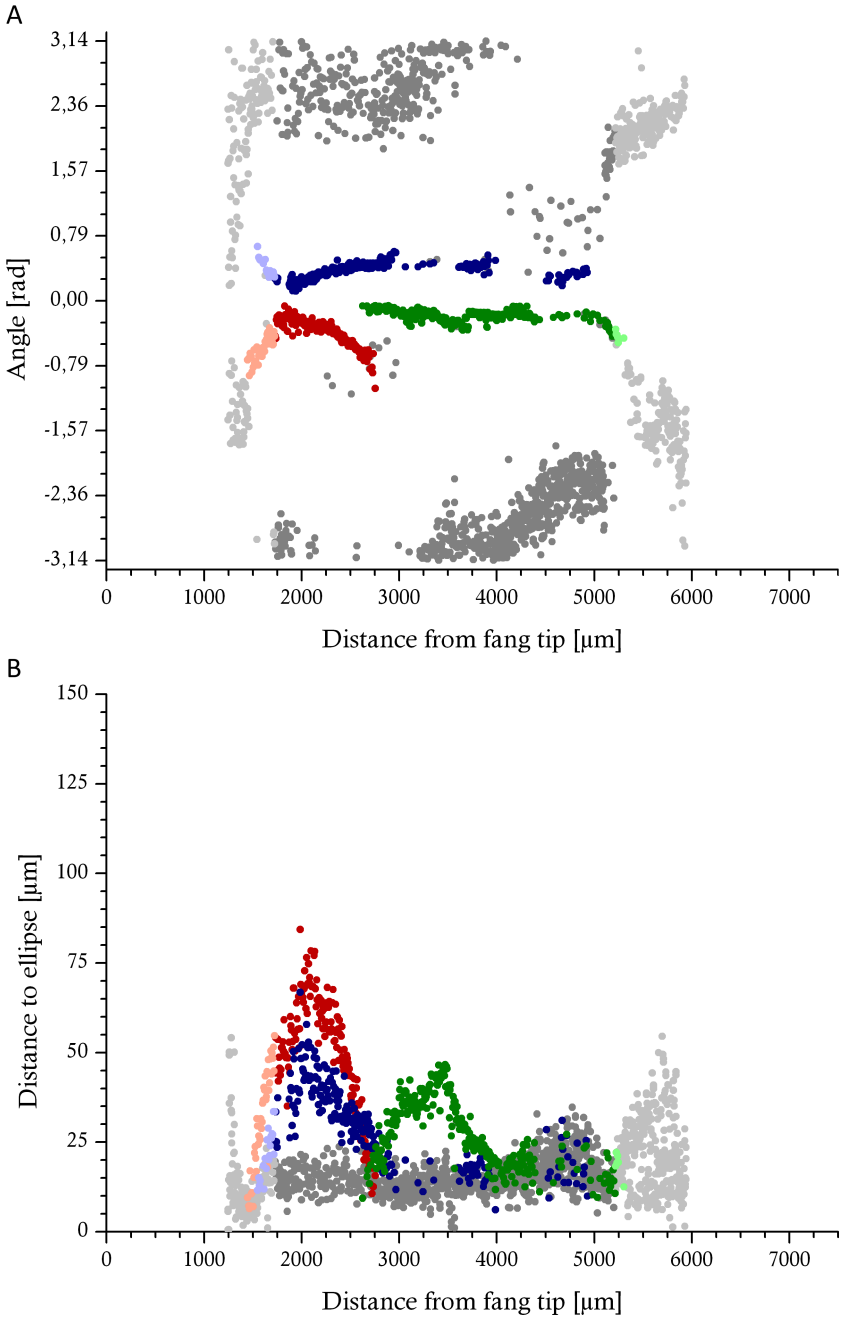


**Figure 34:** (*Naja nigricollis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang nig4. For additional information see figure 29.

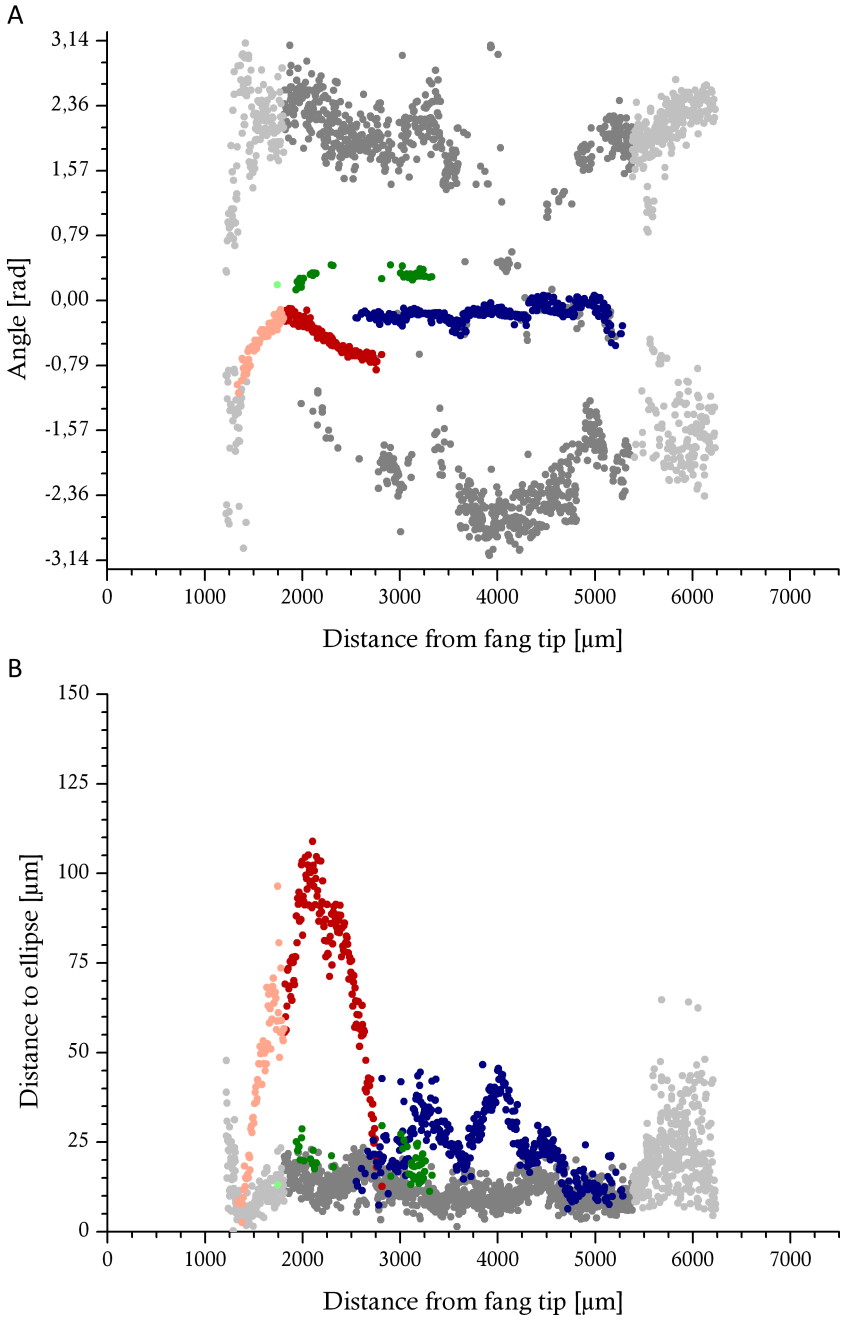


**Figure 35:** (*Naja nigricollis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang nig5. For additional information see figure 29.

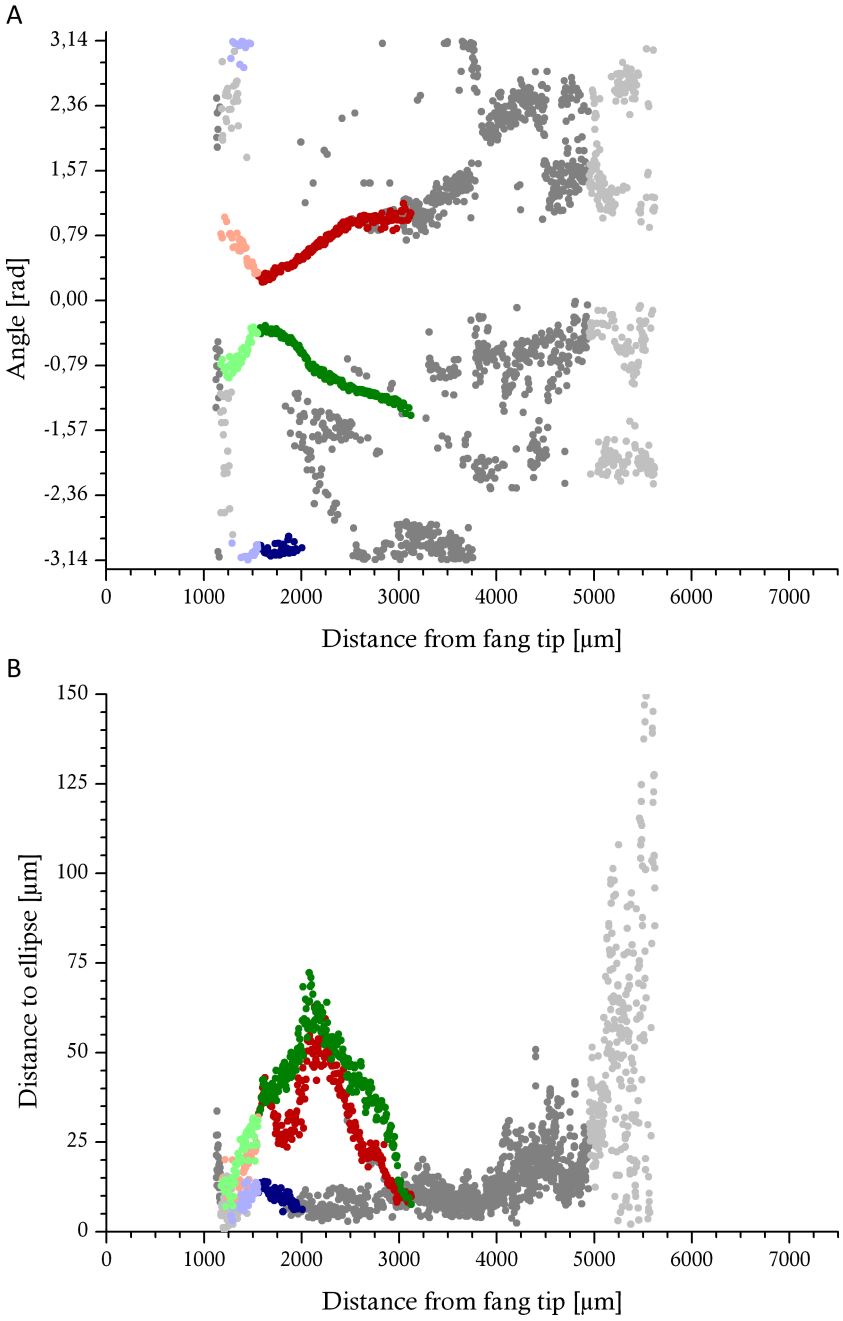


**Figure 36:** (*Naja pallida*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang pal1. For additional information see figure 29.

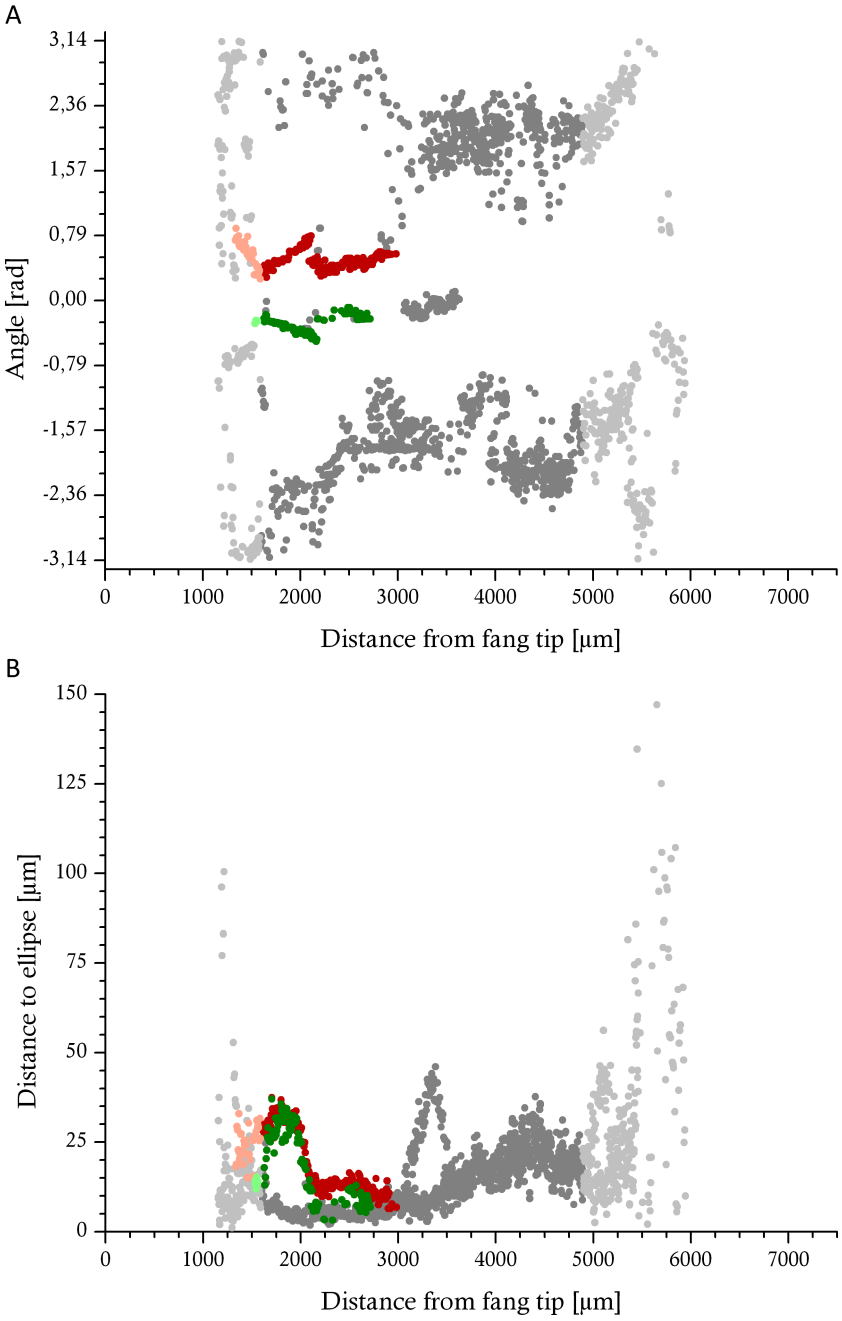


**Figure 37:** (*Naja pallida*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang pal2. For additional information see figure 29.

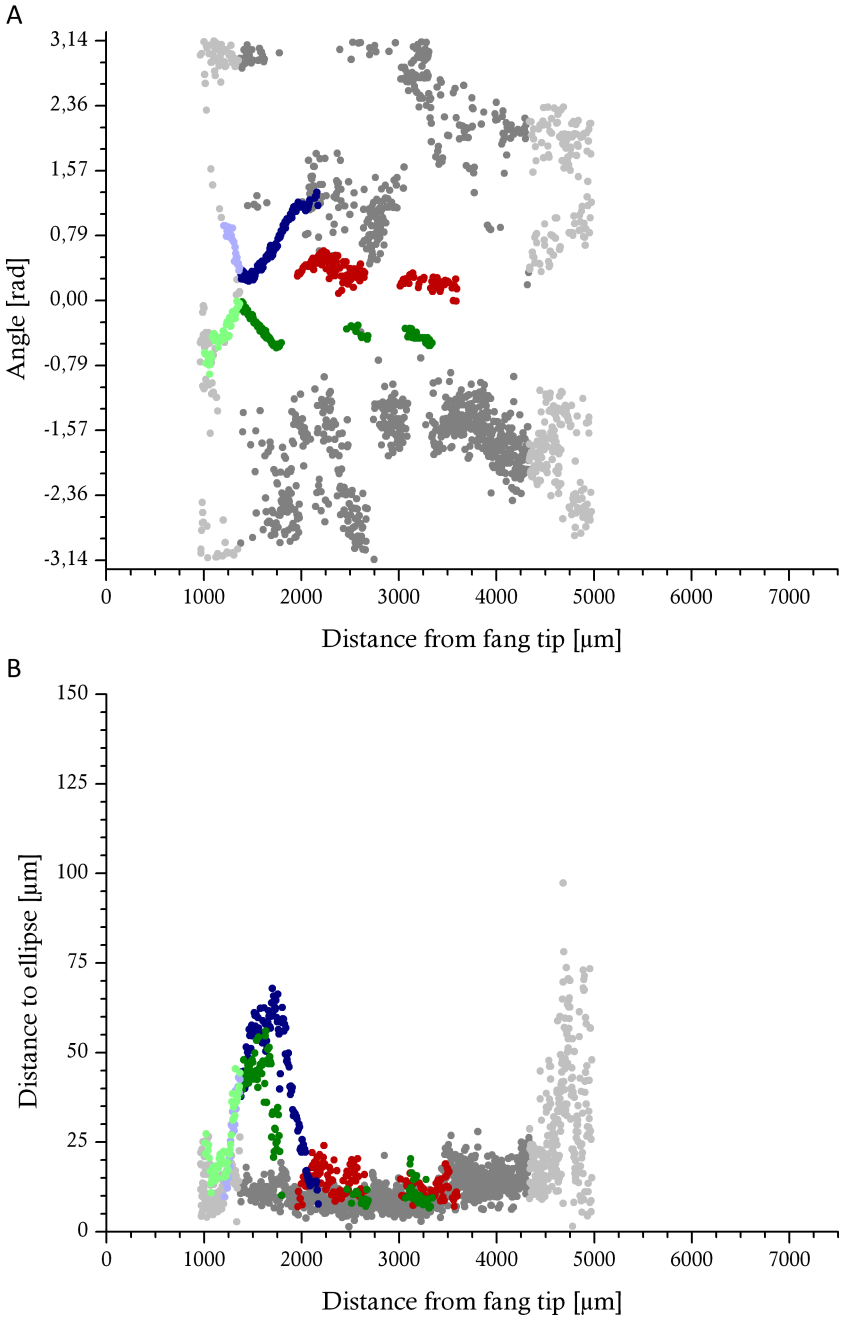


**Figure 38:** (*Naja pallida*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang pal3. For additional information see figure 29.

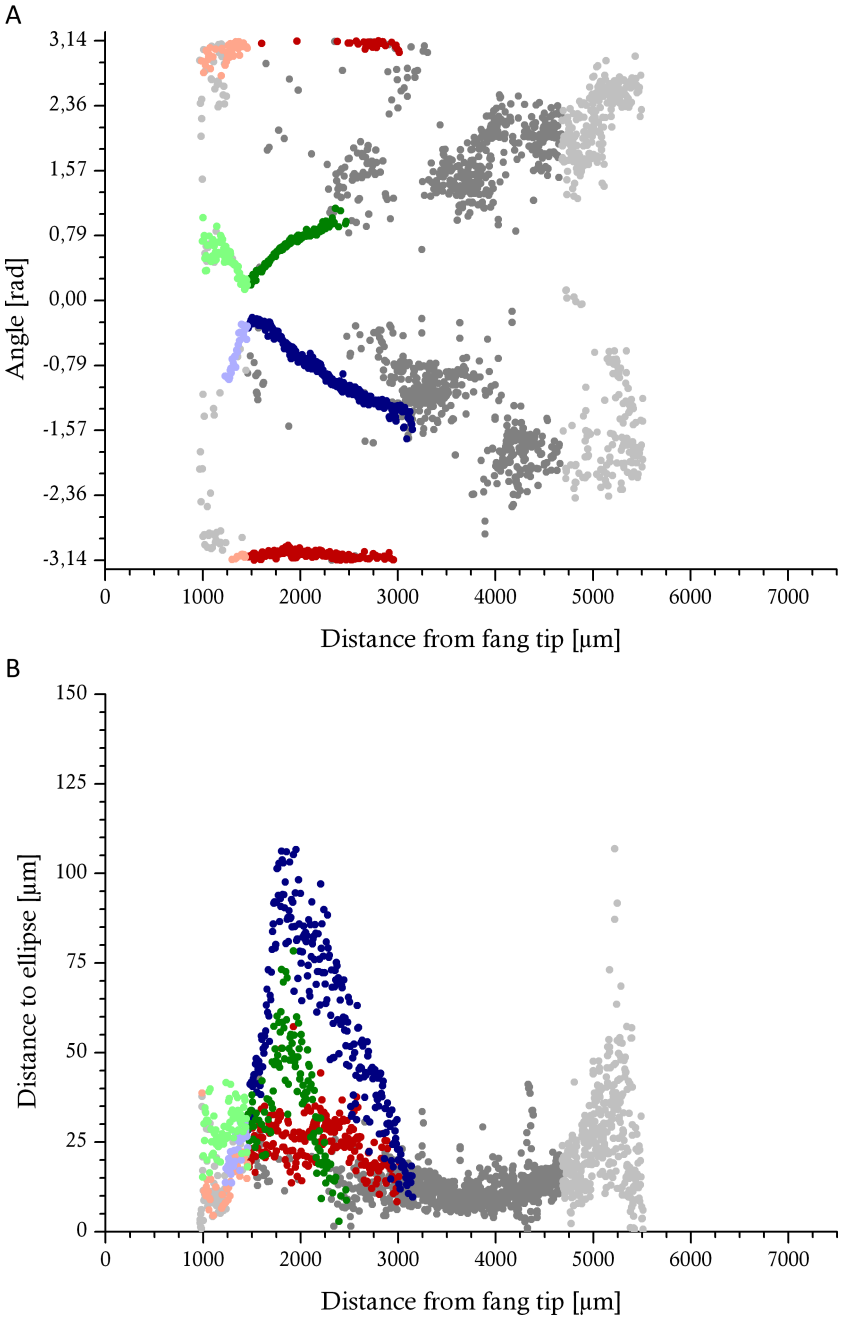


**Figure 39:** (*Naja pallida*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang pal4e. For additional information see figure 29.

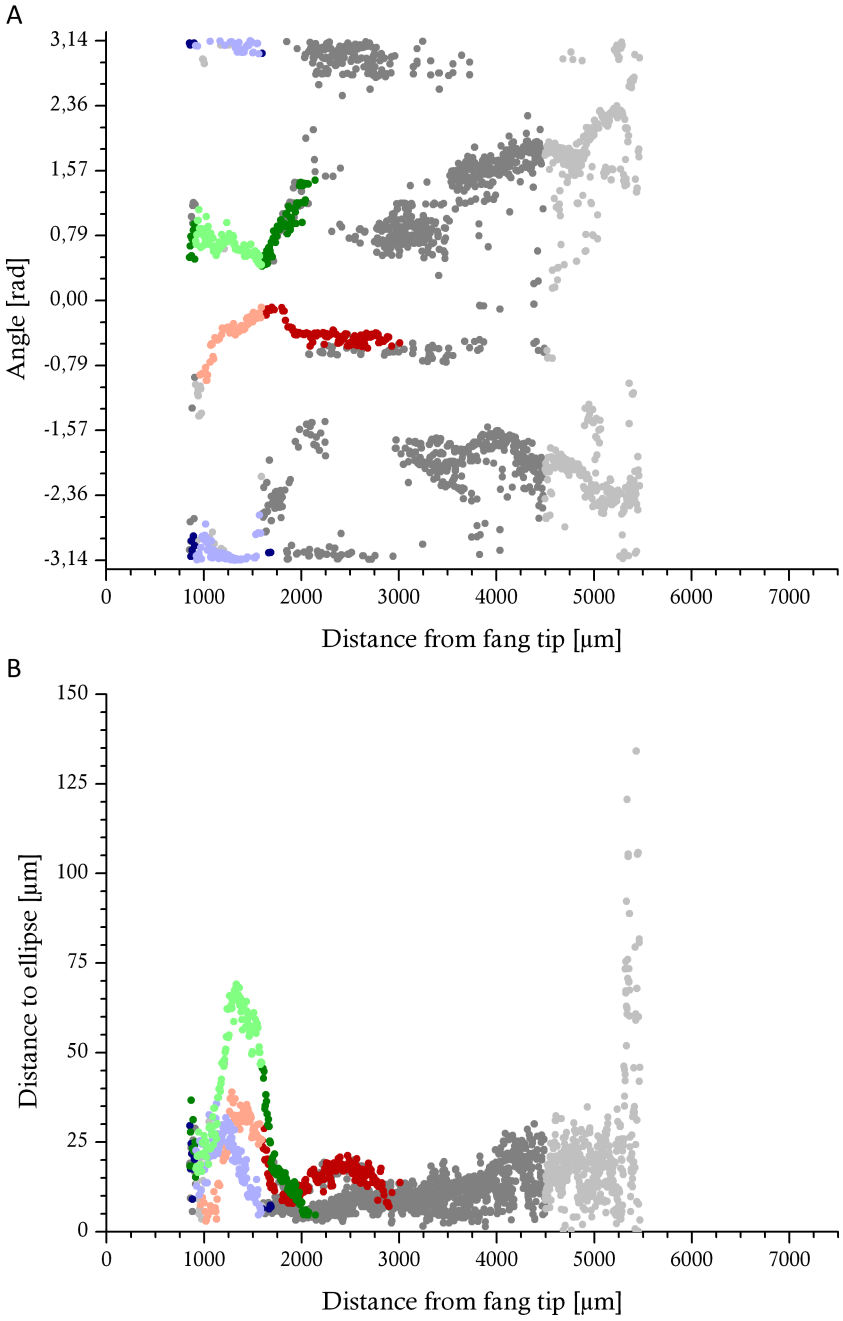


**Figure 40:** (*Naja siamensis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang sia1e. For additional information see figure 29.

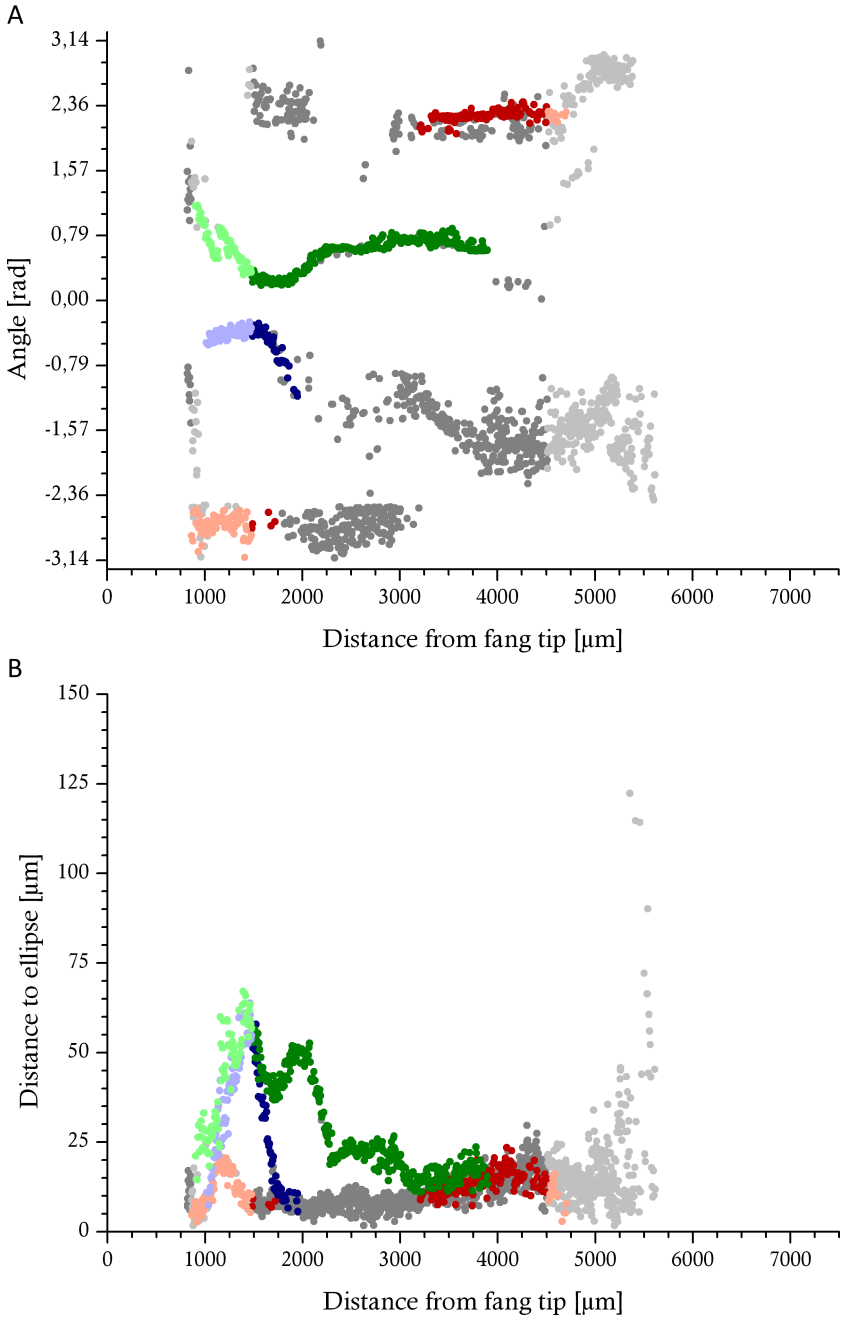


**Figure 41:** (*Naja siamensis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang sia2e. For additional information see figure 29..

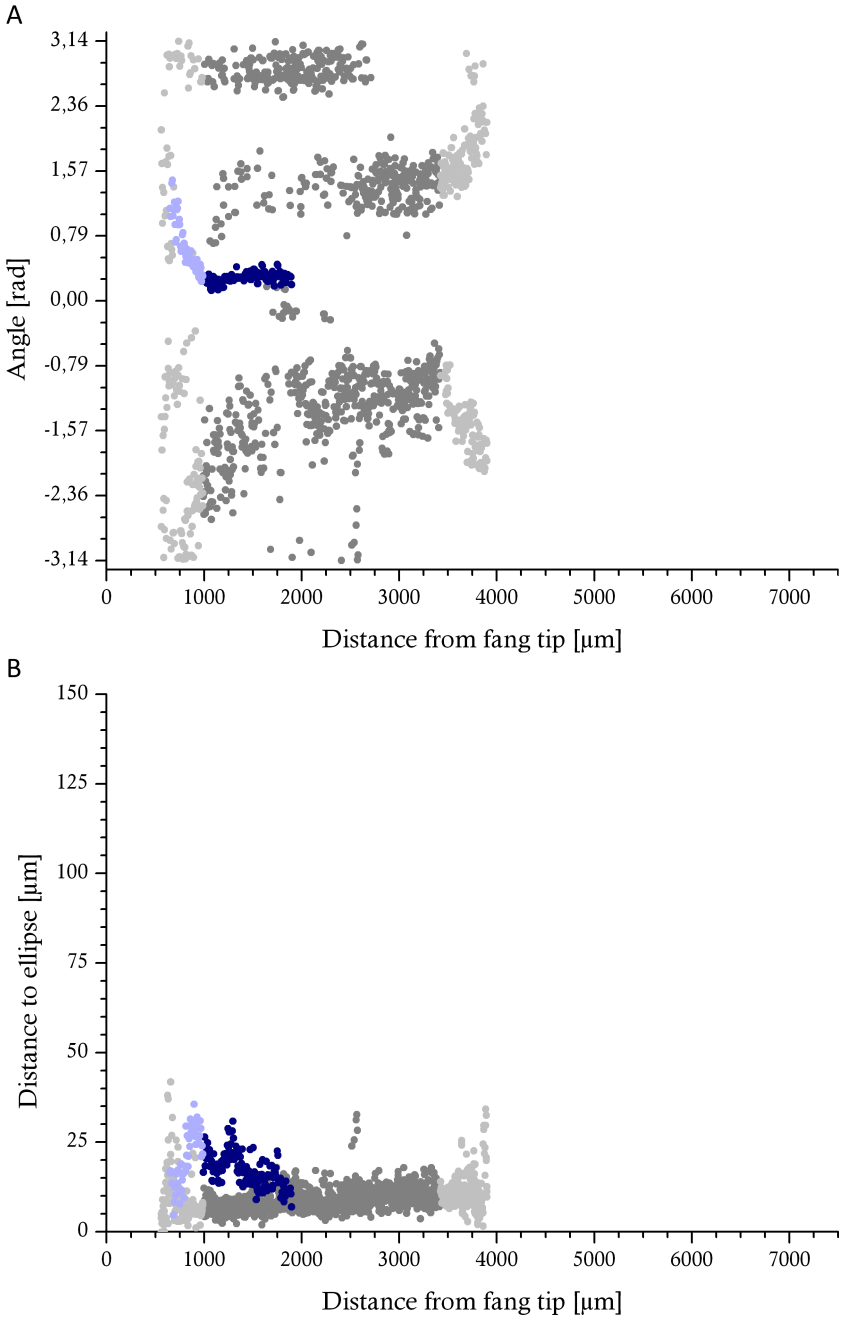


**Figure 42:** (*Naja siamensis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang sia3. For additional information see figure 29.

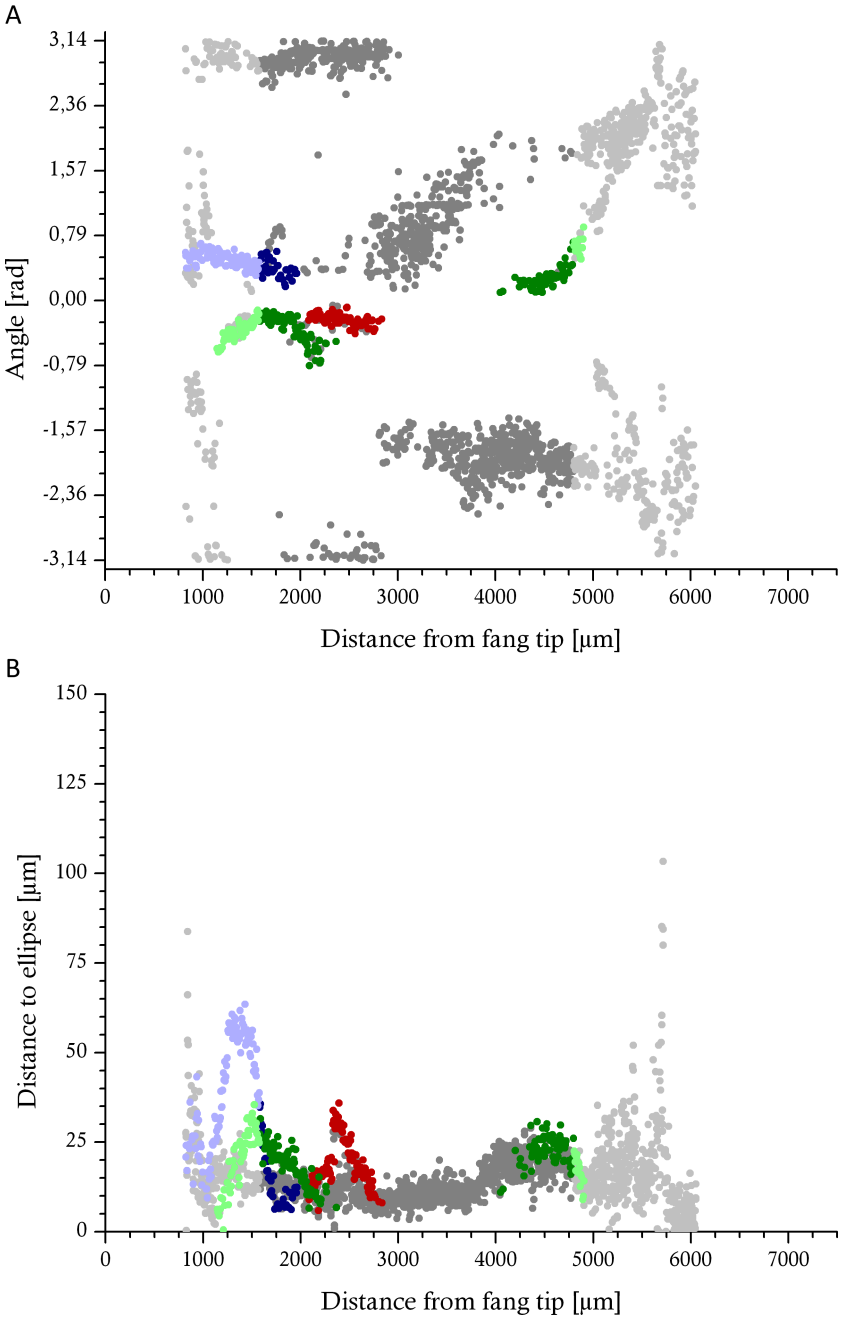


**Figure 43:** (*Naja siamensis*) Maximal differences between the border of the venom canal and the fitted ellipse of the fang sia4. For additional information see figure 29.

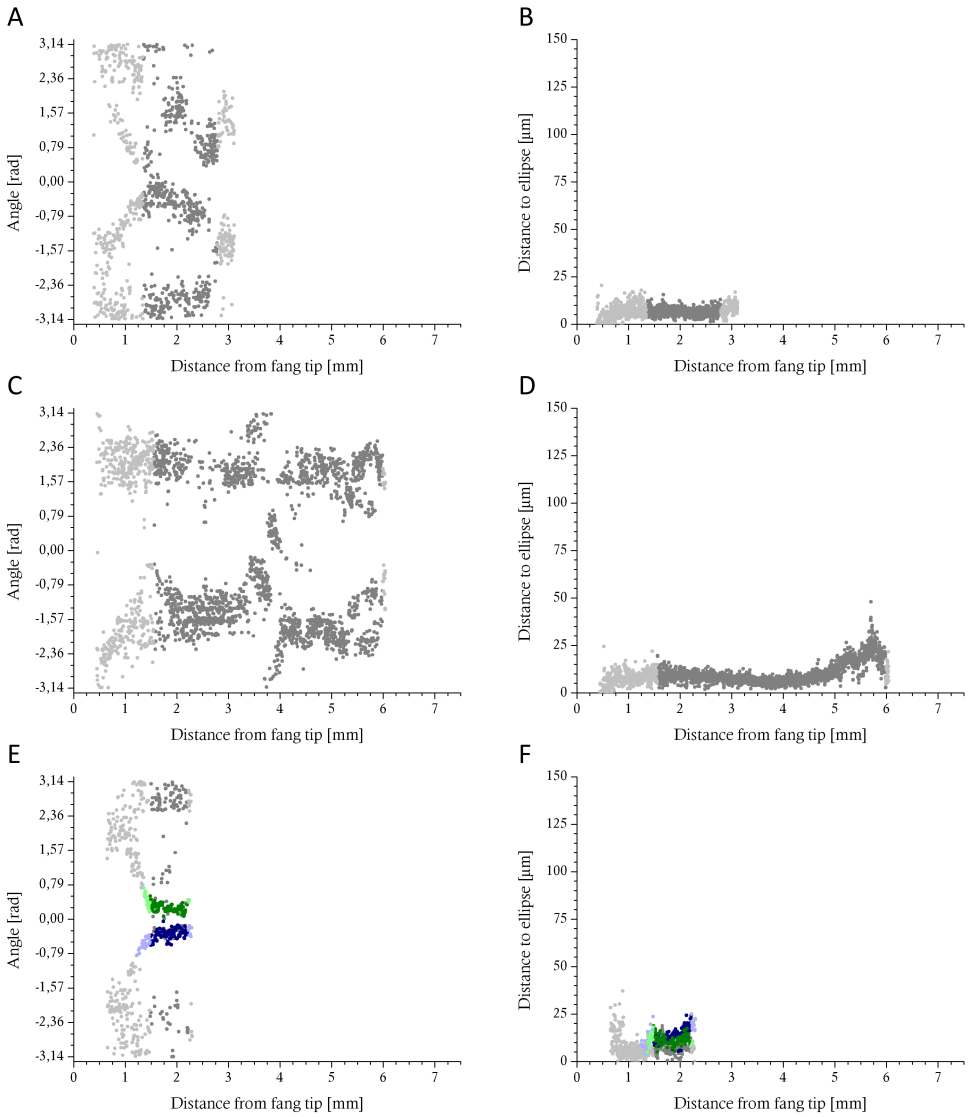


**Figure 44:** (*Dendroaspis angusticeps* (A–D) and *Naja haje*(E, F)) Maximal differences between the border of the venom canal and the fitted ellipse of the fangs from non-spitting species. A, B: ang1, C, D: ang2, E, F: haj1. For additional information see figure 29.

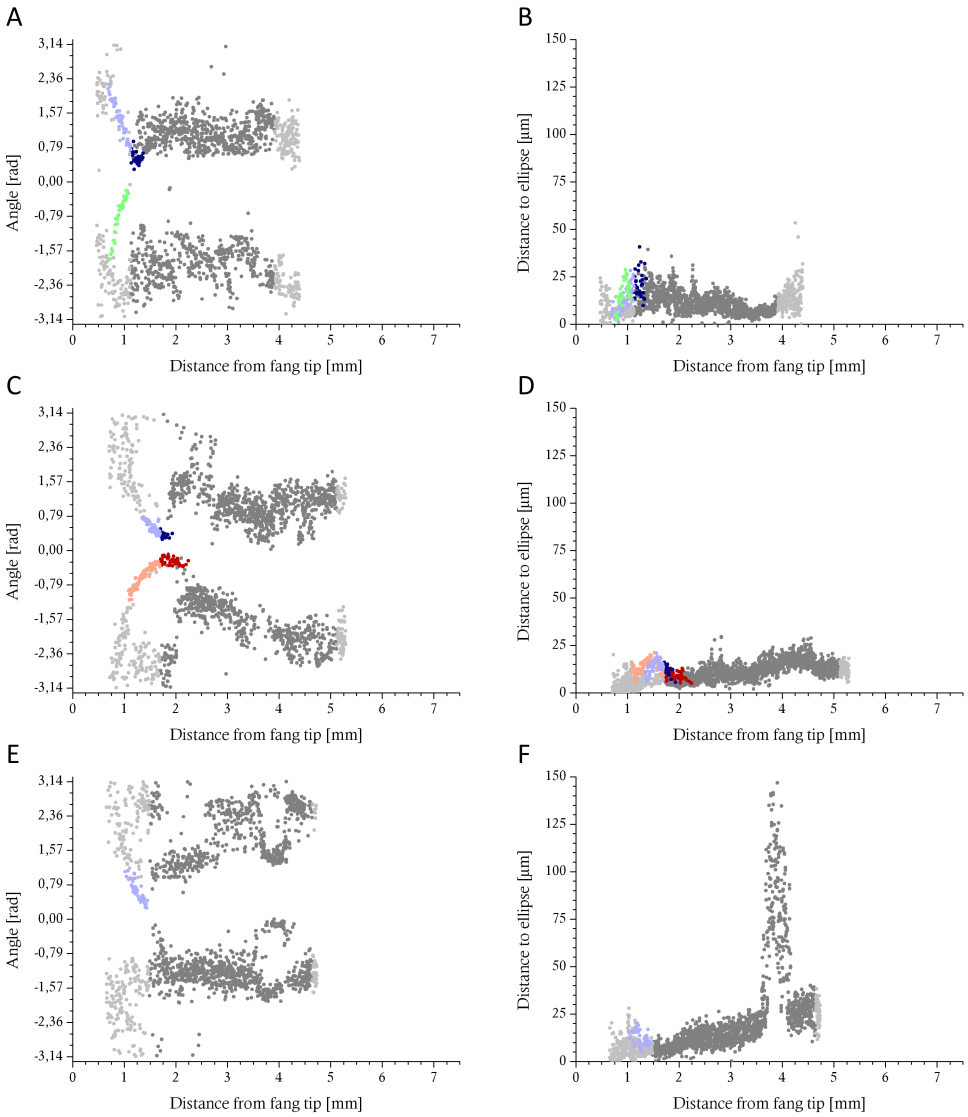


**Figure 45:** (*Naja kaouthia*) Maximal differences between the border of the venom canal and the fitted ellipse of the fangs from non-spitting species. A, B: kao1, C, D: kao2, E, F: kao3. For additional information see figure 29.

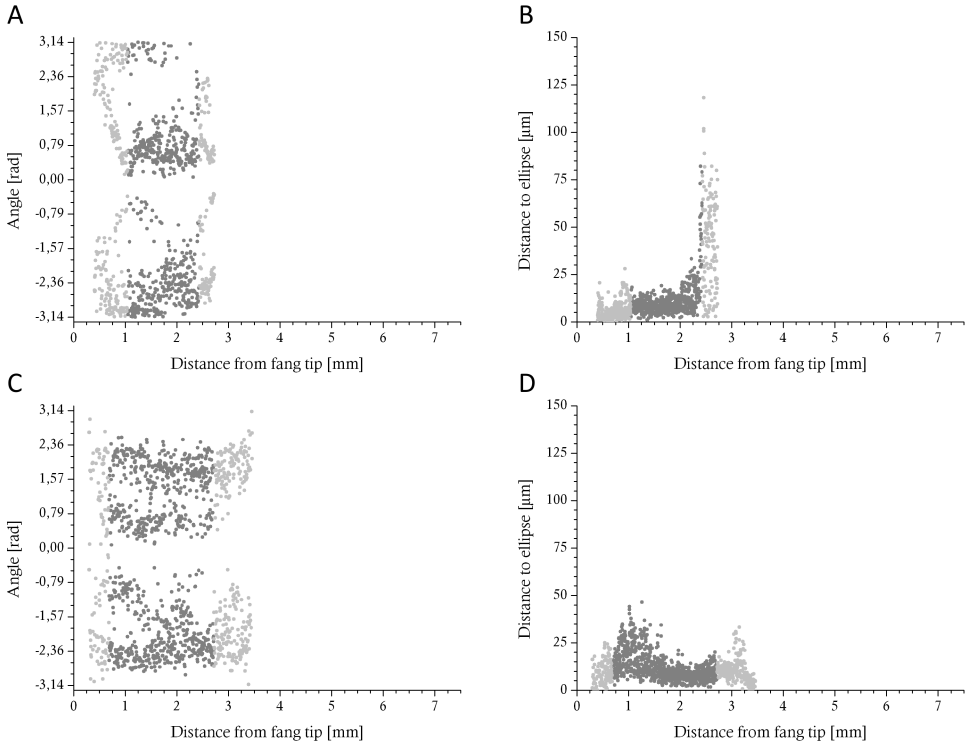


**Figure 46:** (*Naja melanoleuca*) Maximal differences between the border of the venom canal and the fitted ellipse of the fangs from non-spitting species. A, B: mel1, C, D: mel2. For additional information see figure 29.

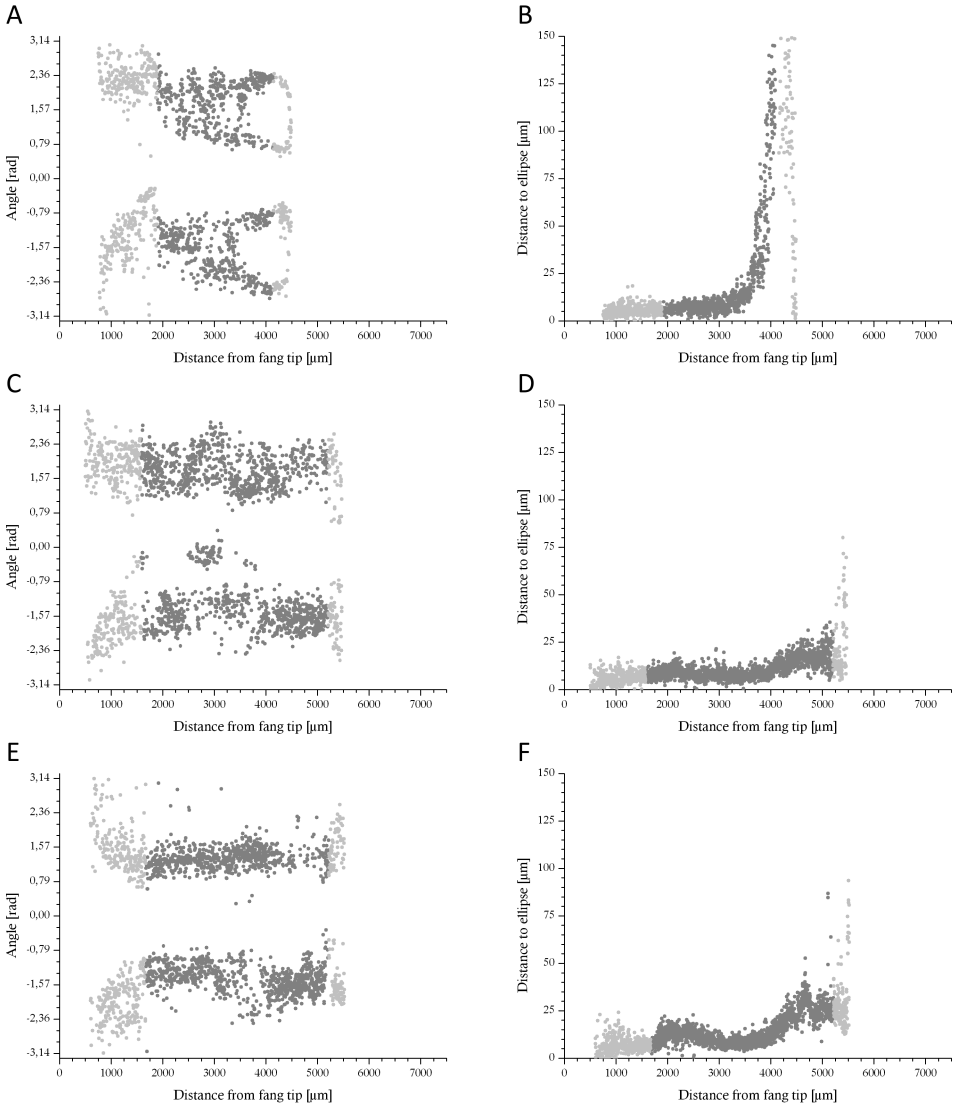


**Figure 47:** (*Naja naja*) Maximal differences between the border of the venom canal and the fitted ellipse of the fangs from non-spitting species. A, B: naj1, C, D: naj2, E, F: naj3. For additional information see figure 29.

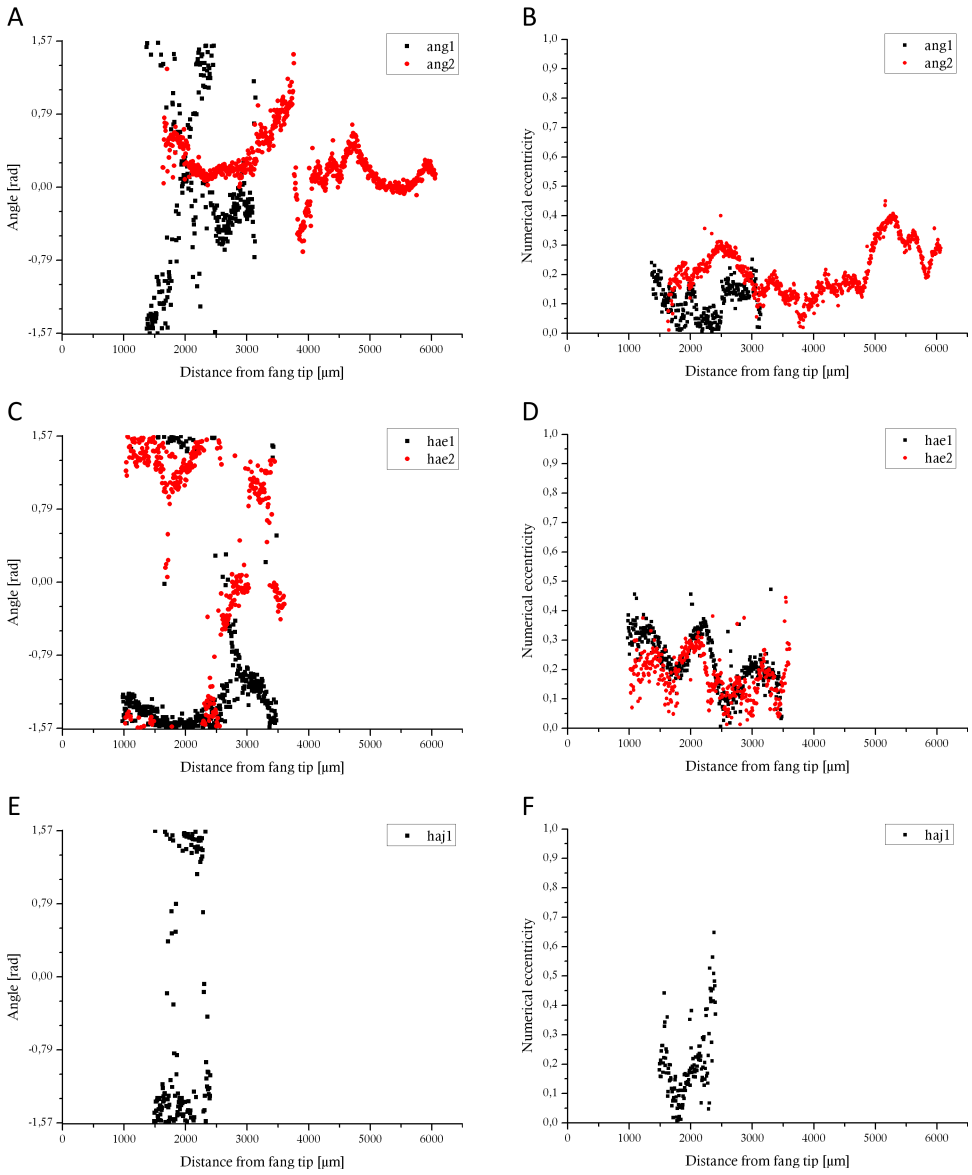


**Figure 48:** The angles (left side) and the numerical eccentricities (right side) of the ellipses fitted to slices of the fangs of *D. angusticeps* (A, B), *H. haemachatus* (C, D), *N. haje* (E, F). The negative angles correspond to the left side of the fang and the positive angles correspond to the right side of the fang respectively. The fang's suture in the subfigures on the left side is around zero rad.

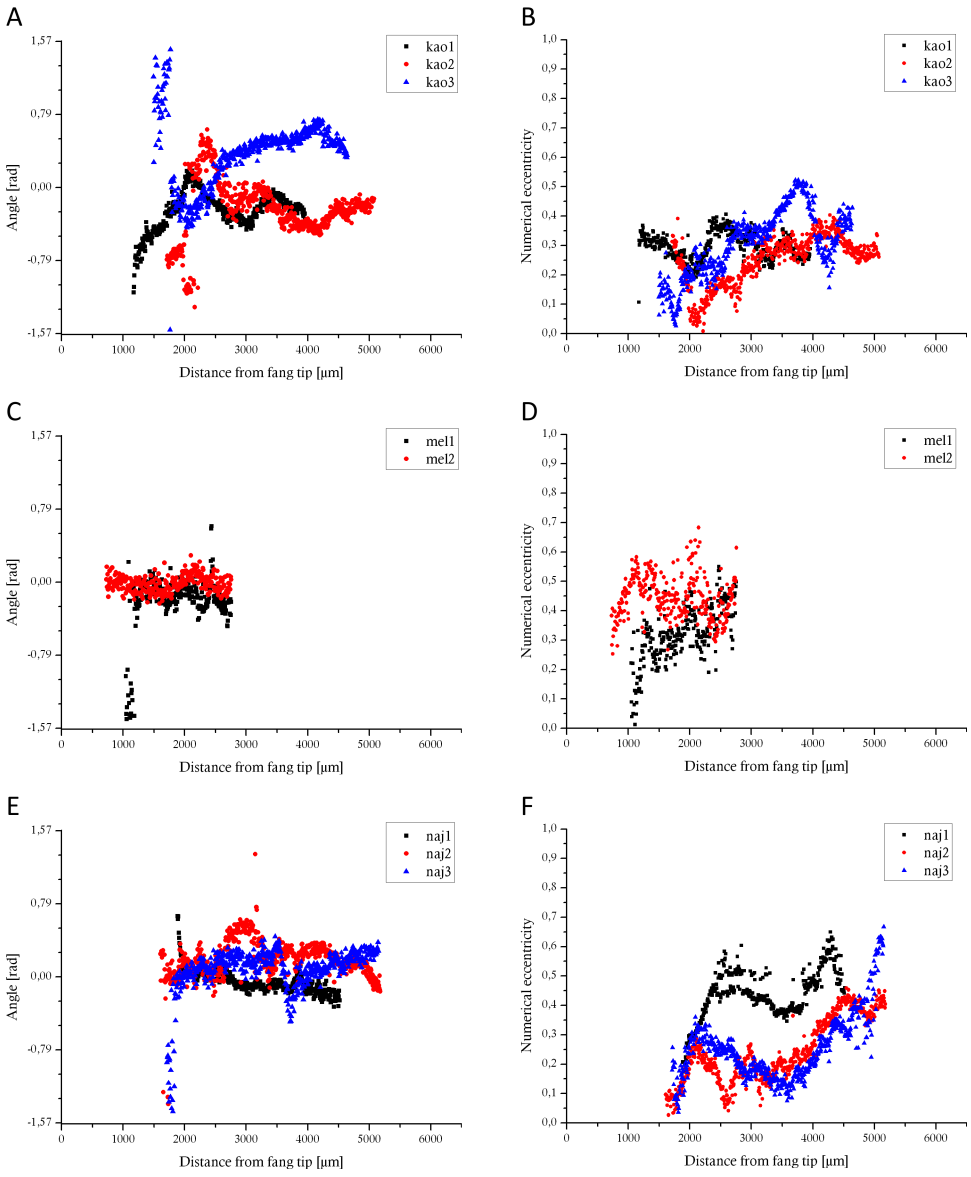


**Figure 49:** The angles (left side) and the numerical eccentricities (right side) of the ellipses fitted to slices of the fangs of *N. kaouthia* (A, B), *N. melanoleuca* (C, D), *N. naja* (E, F). The negative angles correspond to the left side of the fang and the positive angles correspond to the right side of the fang respectively. The fang's suture in the subfigures on the left side is around zero rad.

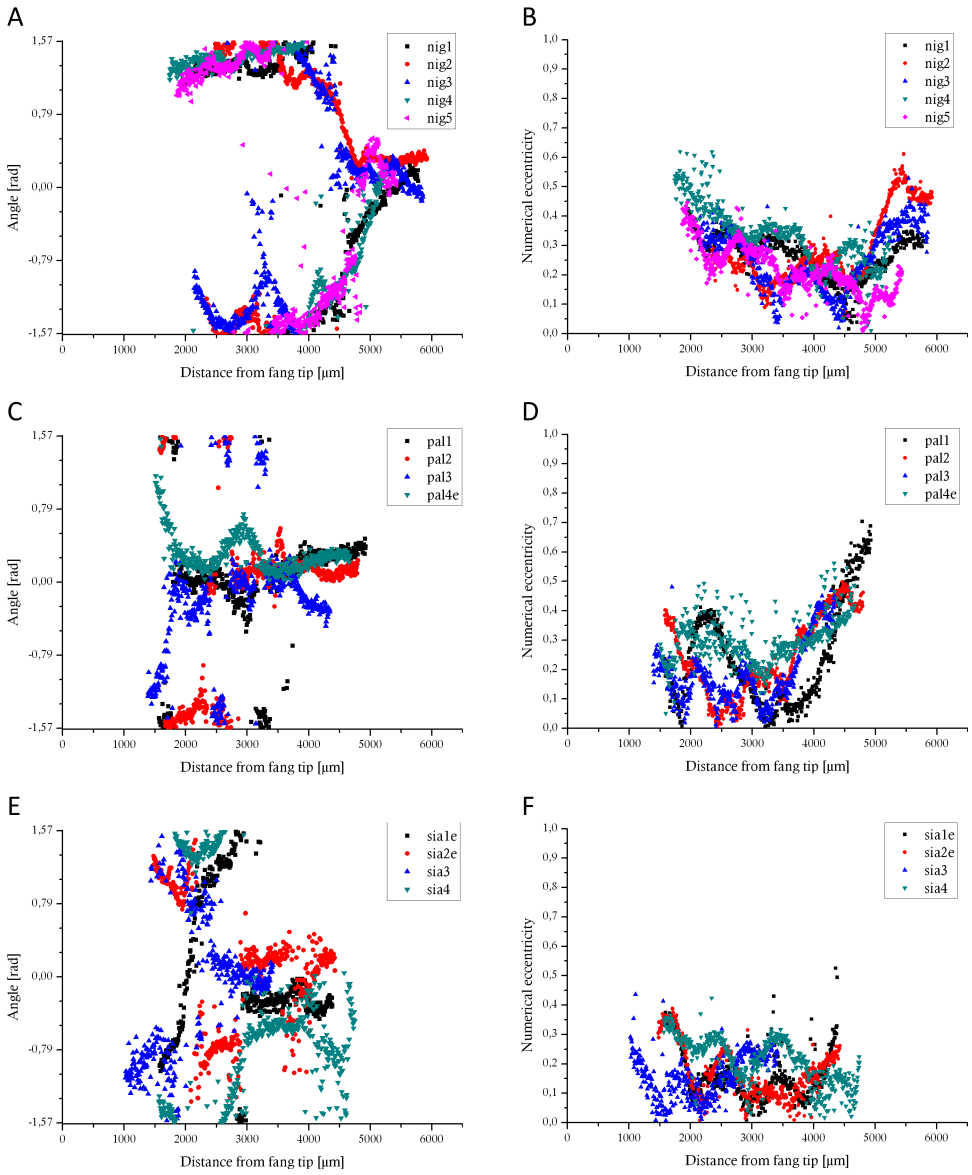


**Figure 50:** The angles (left side) and the numerical eccentricities (right side) of the ellipses fitted to slices of the fangs of *N. nigricollis* (A, B), *N. pallida* (C, D), *N. siamensis* (E, F). The negative angles correspond to the left side of the fang and the positive angles correspond to the right side of the fang respectively. The fang's suture in the subfigures on the left side is around zero rad.

### 3.2.2 Spraying analysis

To evaluate whether the general differences in the spitting patterns of *N. nigricollis* (sprayed pattern) and *N. pallida* (two distinct lines) arise from differences in the fangs or from the differences in venom viscosity, fluids were sprayed through extracted fangs. The differences in the liquids' viscosities had an effect on the distribution of droplets on the plate (compare figures 51 and 52). Independent of the fangs both the number of droplets and the amount of liquid that reached the plate was larger for the lower viscosity liquid (75 % glycerol; table 13). However, the mean size of droplets was smaller for the 75 % liquid. Both fangs of the species *N. pallida* produced a large drop of liquid surrounded by much smaller droplets. For the two fangs of *N. nigricollis* the results were more diverse. Fang nig4 produced a pattern similar to those of *N. pallida*. Differences to *N. pallida* were the smaller size of the largest drop and the increased mean size (median) of the other droplets. Fang nig5 produced a different pattern. While the size of the droplets in the centre of the pattern was highest and their size decreased towards the edge of the pattern, there was no single drop that was much larger than all the others. The number of droplets as well as the overall amount of liquid were higher in fang nig5 than in the other fangs (table 13).

**Table 13:** Measures of the droplet patterns depicted in figures 51 to 52 on pages 81–82. Width 75 % and height 75 % give the width and height of the inner 75 % of the patterns.

| Parameter | 75 % glycerol | | | | 85 % glycerol | | | |
|---|---|---|---|---|---|---|---|---|
| | nig4 | nig5 | pal3 | pal4 | nig4 | nig5 | pal3 | pal4 |
| Drops | 184 | 1528 | 520 | 229 | 23 | 578 | 8 | 72 |
| Largest drop | 36.00 | 8.06 | 46.80 | 43.44 | 11.96 | 10.41 | 19.29 | 13.50 |
| Mean size | 0.68 | 0.35 | 0.27 | 0.39 | 1.82 | 0.64 | 3.01 | 0.39 |
| Median size | 0.34 | 0.15 | 0.07 | 0.07 | 1.36 | 0.42 | 0.97 | 0.10 |
| Sum | 125.86 | 535.70 | 139.30 | 88.31 | 41.90 | 371.70 | 24.09 | 27.92 |
| Width 75 % | 216.6 | 495.8 | 166.8 | 101.5 | 145.9 | 420.5 | 46.2 | 48.0 |
| Height 75 % | 215.7 | 483.1 | 160.4 | 100.6 | 295.5 | 426.9 | 47.1 | 55.3 |

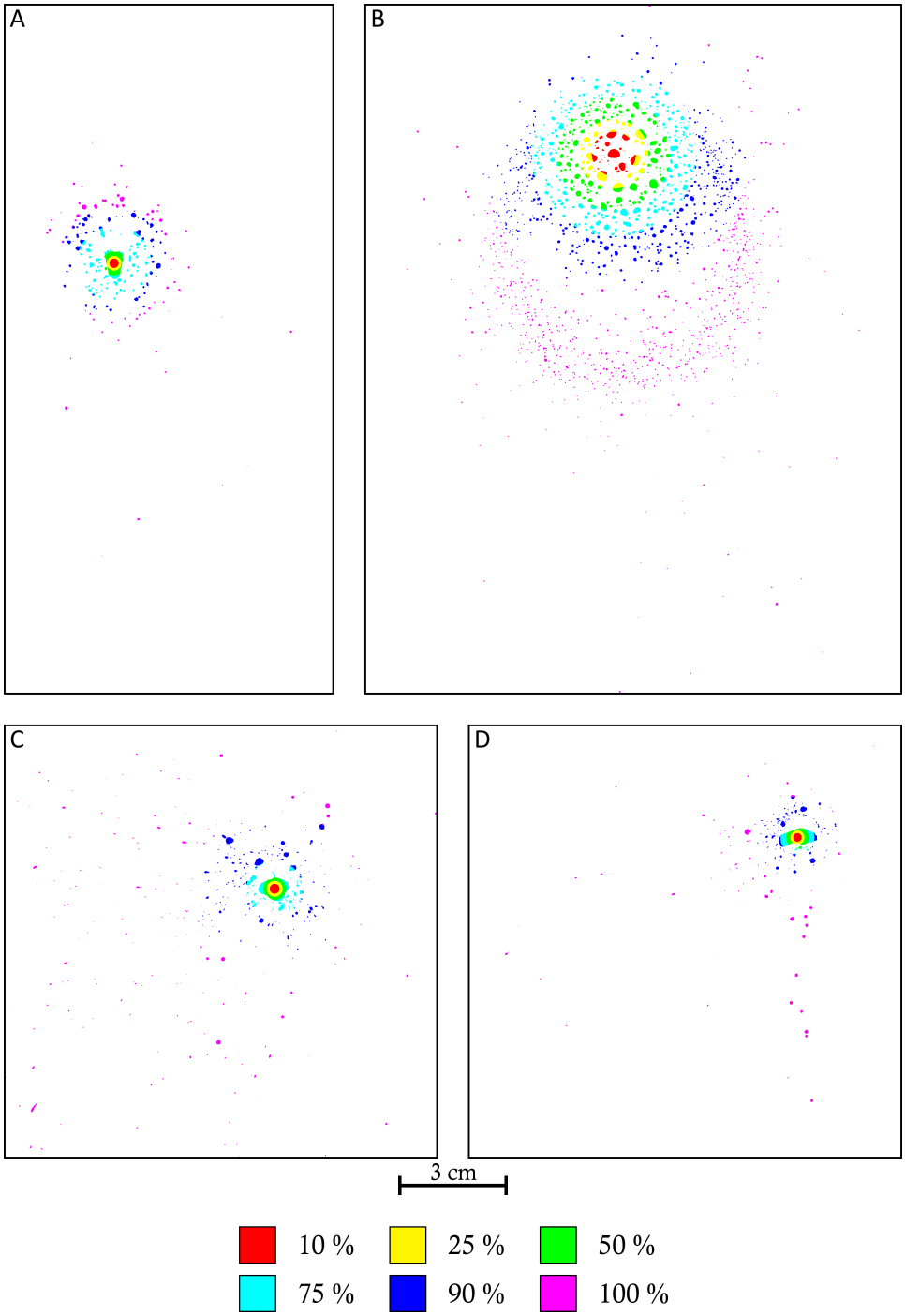


**Figure 51:** Distribution of the venom. A 75 % aqueous solution of glycerol was used. The outermost droplets in C and D are not shown. The boxes separate the patterns. They do not describe the expanse of the patterns.

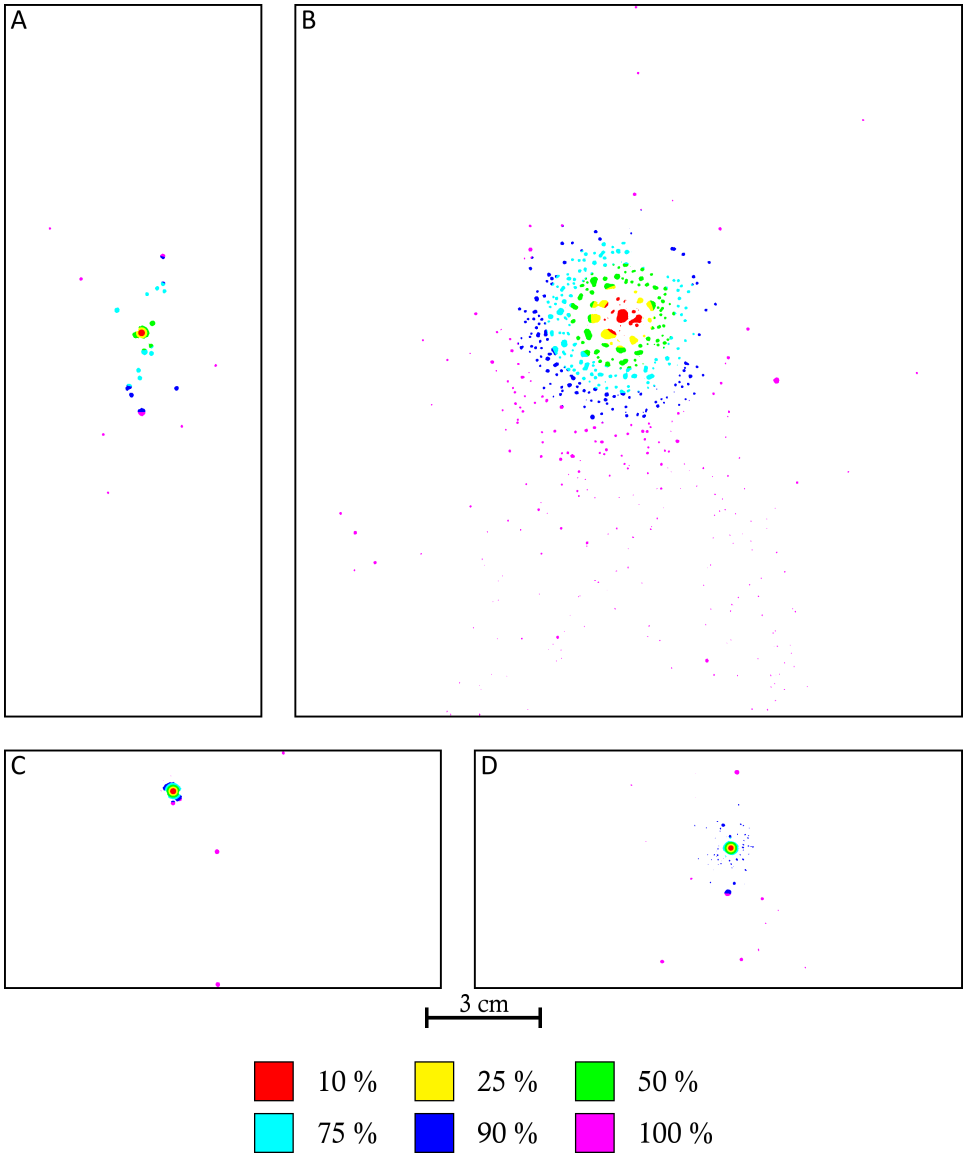


**Figure 52:** Distribution of the venom. A 85 % aqueous solution of glycerol was used. The outermost droplets in D are not shown. The boxes separate the patterns. They do not describe the expanse of the patterns.

# 4 Discussion

## 4.1 Behaviour

The experiments show that the eyes of a possible threat do not play a prominent role in the performance of the spitting behaviour. Species adapting their behaviour to the presence of mock eyes are known in crabs (O'Brien and Dunlap, 1975), chicken (Gallup et al., 1971), iguanas (Burger et al., 1991), and snakes (Herzog and Bern, 1992). Since different animals and even other snakes react to mock eyes, it can be assumed that spitting cobras also can recognize eyes. The experiments described in section 2.2.1 demonstrate two things: First, eyes are not a necessary stimulus for a spitting cobra to eject its venom towards an aggressor; second, eyes do not alter a cobra's defensive behaviour. Both, reaction times and spitting frequencies did not differ whether the target was equipped with glass eyes or not. The cobras' responses did not differ between different characters of glass eyes. The "unimportance" of an aggressors eyes were further illustrated by the results of the experiments shown in section 3.1.3 in which the distribution of the venom on the target was examined. The distribution was unaffected by the position of the eyes as the venom was generally directed at the centre of the larger of the two disks even if the eyes were placed on the smaller disk.

At first glance the results of these experiments are surprising since cobra venom is ineffective on unharmed skin. But when given more thought the need to aim at the eyes diminishes. Spitting cobras distribute their venom over a certain area by performing rapid head movements (Westhoff et al., 2005; Young et al., 2009a, and see section 3.1.4). This heightens the probability that at least one eye is hit. Furthermore an animal attacking a cobra most likely will look at the cobra to see what the snake is doing. During one spitting act a cobra uses only a small amount of its venom and could spit at least forty times consecutively (Cascardi et al., 1999; Greene, 1999, and personal observation). If the first spit has no effect, the cobra can spit again. Some cobras tend to directly spit two times (Rasmussen et al., 1995, and personal observation) which increases the area struck. Therefore the evolution might have driven the behaviour of the spitting cobras in aiming at eyes indirectly through aiming at the head or face of an aggressor or just at its centre assuming that an attacker is directly facing the cobra. Recognition of eyes as a requirement for spitting could even be disadvantageous in a situation when the cobra would not be able to discriminate the eyes. This could especially be the case when the cobra is going to shed its skin and the eyes become opaque.

When a cobras spits at a mammal but misses the eyes, the mammal is likely to clean its fur after the encounter and thereby perceive the bitter taste of the venom (Warrell and David Omerod, 1976, and personal opinion). The venom could also enter the mouth of the attacker, having a repellent aspect, even if the eyes are not reached. It would be interesting to know if spitting cobra venom is more unpleasant in taste than other elapid venom.

The results described in section 3.1.2 affirm the first result of the aforementioned experiments since all the targets presented were not equipped with glass eyes. The experiments showed that the shape of the target but not the eyes elicits spitting behaviour. A target with straight edges and sharp corners resulted in a lower spitting frequency (23.9 % for triangles) than a target that had a rounded shape (80.0 %).

The shape of a target (threat) could be the clue for a spitting cobra to discern dangerous and harmless things. While being able to spit numerous times it would be a drawback for a snake to become aroused by every movement in its surrounding. A leaf moved by the wind, for example, should not elicit venom spitting under normal circumstances. Needless performance of defensive behaviour would be a waste of energy and could interfere with other activities or draw unwanted attention. The triangles and rectangles used in the experiments obviously do not fit in the snakes' category for danger.

To our surprise, the cobras did not spit at mounted birds. The explanation that the cobras had never experienced a bird and thus would not spit can be dismissed as the cobras did not know black plates of different shapes before the experiments, either. Because the small face-shaped plate was smaller than the birds presented, size should not be the reason, either. Maybe one explanation is the unnatural "behaviour" of the mounted birds. They did not flap their wings and because of their low stability they were not moved as fast as the other targets presented.

The design of the experiments involves few sources of mistakes. First of all the experimenter knew which target he was presenting to the snakes. That way the experimenter's expectation of the outcome of the experiments could have unconsciously influenced the movements performed with the target. It was not possible to remove this source as the target had to be moved according to the position and reaction of the tested snake. While observing the behaviour of the snake the experimenter was generally able to see the target as well. In fact although the movement of the target was not traced one can sure that the movement of the target differed between presentation which resulted in a spitting event and those

which did not. If the snake did not react to the target within the first seconds of presentation (more than 50 % of the spitting events occurred in the first ten seconds; compare figure 15) different kind of movement were performed to find the movement that would eventually provoke the snake to spit. Nonetheless, it is unlikely that the results were influenced by the fact that the experimenter knew the target because the results of the comparison of different eyes did not match the expectations. It was expected that a target without eyes would lead to a much lower spitting frequency and that there would be differences between different kinds of glass eyes, especially regarding their shininess as this would be a strong hint for an eye.

Secondly, the behaviour of the snakes could have been influenced by the fact that most of the experimental animals were raised in captivity or had been in captivity for a long time. The cobras raised in captivity do not know their possible natural enemies, as they know only humans and other snakes. Furthermore their spitting never had a natural consequence as either the annoyance (in the form of a human wearing a visor or being behind the closed terrarium door) was unaffected by the venom or, during the experiments, was removed after spitting independent of the aim of the spitting. If spitting cobras need to learn parts of the spitting behaviour to effectively repel an attacker, these snake were never able to do so due to the possibly improper responses of the targets. This issue is not easily dismissed but there are two considerations which controvert the demand of learning on the side of the cobra. First, the cobras are able to hit a moving target without training. Second, in their early days of life spitting cobras are in immediate danger of being preyed upon by many other animals. Hatchling spitting cobras rear up and spit at annoyances even before completely leaving their eggs (Young et al., 2009b). If they had to learn the appropriate way of spitting during that time they would probably being eaten before they could achieve that.

The results of the experiments in section 3.1.4 (spitting distance) shows another aspect of the spitting behaviour. Assuming that spitting cobras aim at the face the spitting pattern should cover the face of the antagonist but not exceed its width and height. Thus the dimensions of the spitting patterns should only depend on the size of the antagonist's face but not on target distance. To achieve this the spitting angles of spitting cobras should not be constant but follow the function $2 \cdot \arctan\left(\frac{a}{2 \cdot d}\right)$ ($d$ = distance between the snake and the aggressor's face, $a$ = height or width of the face). The regression analysis using this function (adjusted by the distance between face and visor, table 8) shows that the snakes adjusted their spitting angles to match a face of 24 cm height and 16 cm width.

These values correspond well with the height (21.5 cm) and width (13.5 cm) of the experimenters face.

Two cobras showed a positive correlation between target distance and spitting pattern dimension. This is not what would be expected if the cobra compensates for target distance. Despite the positive correlation, the slopes of the regression lines were considerably less than 1. Thus these two snakes did compensate for target distance, but not perfectly.

This shows that the spitting behaviour is not stereotyped and could be adjusted to the situation.

The video experiments were not successful. The frame rate of the videos fails to be the reason because the flicker fusion frequency of cobras is below 50 Hz (47 Hz for *N. nigricollis*; Kohl and Young, 2011). Other reasons could be the presented stimulus, colour, and depth perception. It is unlikely that the stimulus was the reason because cobras spat at the experimenter but failed to spit at equally sized videos of the experimenter. Cobras are likely to have two to three classes of cones and one class of rods; a condition found in many snake genera including two elapids (*Acanthophis* and *Micrurus*; Underwood, 1970). However, it is not known if they have colour vision. The retina of garter snakes (*Thamnophis sirtalis* and *T. marcianus*) contains three morphologically distinct classes of cone photoreceptors but only a single class of cone pigment. Therefore it is unlikely that these species have color vision (Jacobs et al., 1992). The same could be the case in cobras. If the cobras have colour vision, the colours depicted on the screen would be unnatural for them because the base colours are adjusted to human vision (Fleishman and Endler, 2000). Videos miss depth cues like motion parallax and accommodation which are used by a variety of animals (Walls, 1942; D'Eath, 1998; Zeil, 2000). Because the distance between a threat and the cobra is an important factor for the spitting behaviour (cf. section 3.1.4) these missing depth clues could be the main reason for the failure.

## 4.2 Fang morphology

The results of the measuring of cobra fangs (section 3.2) show differences between the fangs of spitting cobras and non-spitting cobras. The shorter discharge orifices and the presence of ridges inside the venom canal were known earlier (Bogert, 1943; Wüster and Thorpe, 1992; Young et al., 2009a). The shorter discharge orifices are necessary for the spitting behaviour because they cause the venom to be ejected forward instead of downward. By influencing the internal flow of a

liquid, structures upstream of a nozzle's orifice can influence the disintegration of liquid jets (Begenir, 2002), as used in swirl nozzles (Nonnenmacher and Piesche, 2000). This and the observation that the ridges were found only in spitting species to date, encourage the assumption that these structures have a favourable influence on the ejected venom jets. The finding that the species *N. nigricollis* and *N. pallida* which show a variable spitting behaviour and have a greater disposition to spit than the species *N. siamensis* and *H. haemachatus* have more pronounced ridges within the venom canals further supports this assumption. Changes in the cross section of a tube or deflections of a tube reduce the pressure of the liquid that runs through the tube (Kamaier and Paschereit, 2007). Therefore the pressure at the discharge orifice of spitting cobras would be reduced compared to non-spitting species. The wider venom canal of spitting cobras could be an adaption to counteract the pressure reduction. While there are some pressure measurements and theoretical estimations to be found in literature for spitting cobras, vipers and a colubrid (Horton, 1948; Kardong and Lavin-Murcio, 1993; Young et al., 2000, 2001, 2004) no values are given for non-spitting cobras or closely related elapids.

The experiments of section 3.2.2 (artificial spraying) give a hint about the effect the ridges inside the venom canal have on the ejected liquid stream. The discharge orifices of the fangs used in the experiments have similar dimensions. The most noticeable difference between fang nig5 which produced the patterns with the highest number of droplets and the other fangs is the disparity of the ridges within this fang (compare figures 34, 35, 38, and 39). The ridge on the right side of fang nig5 was very small compared to the one on the left side. This leads to possible effects of the ridges. Ridges on both sides of the venom canal could cause vortices that stabilize the ejected liquid stream. If there is a big difference in the extent of the two ridges this stabilization might be hindered. As no fangs with similar discharge orifices as the ones tested but without ridges were available this hypothesis could not be tested further.

An additional structure within the venom canal is the suture. In the African cobras the suture was more pronounced than in the Asian species but there was no difference between spitting and non-spitting cobras. This implies that the influence of the suture on the liquid stream is much lower than the influence of the ridges.

This is the first work to measure snake fangs in such detail. Measurement of more species including species from other families may give more insight in the adaptions of venom fangs.

# Summary

Spitting cobras are unique within the snakes for their ability to "spit" venom at an aggressor. This is used solely as a defensive behaviour. When the venom enters the eyes it causes intense pain and may lead to permanent blindness when not washed out. On the skin the venom has no effect. Because of this mode of action it could be assumed that eyes are an important stimulus for the control of the spitting behaviour. Earlier experiments showed that cobras do not spit at moving objects indiscriminately. While presentation of faces generally elicited spitting behaviour, presentation of hands did not.

To investigate the influence of different parameters on the venom spitting, behavioural experiments were conducted. In these experiments different targets were presented to the cobras. The results showed that contrary to the assumption, eyes had no effect on the spitting behaviour. Neither was their presence required to elicit spitting, nor did the cobras aim their spitting at them. The shape of the target was of importance. While targets with a rounded shape generally elicited spitting, simple targets with straight edges did not. This was not affected by the size of the target. An explanation may be that the shape of an object could be a good indicator of the danger it poses to the snake. The eyes instead my not always be discernable and therefore it could make more sense to distribute the venom over the target. Spitting cobras could spit repeatedly in a short time which should lead to a high probability to hit an eye.

In additional experiments it was tested whether the spitting behaviour is stereotyped or could be adjusted. The experiments demonstrated that spitting cobras are able to adjust the venom distribution to target distance. The further away the target was, the smaller was the angle over which the venom was sprayed.

Besides the behavioural experiments, the morphology of the venom fangs was investigated. It was known that the fangs of spitting cobras differ from fangs of non-spitting cobras in having smaller discharge orifices. Additionally it was known that they possess ridges inside the venom canal. By the use of microtomography the fangs of spitting cobras and non-spitting species were measured and compared in detail, for the first time. The data could be used to investigate the hydrodynamic aspects of the venom spitting. Artificial spitting experiments provide grounds for the assumption that uniform pairs of ridges may reduce the breakup of the venom streams.

# References

Alexander, RMcN (1963) The evolution of the basilisk. *Greece & Rome* 10:170–191

Barbour, T (1922) Rattlesnakes and spitting snakes. *Copeia* 1922:36–38

Begenir, A (2002) *The role of orifice design in hydroentanglement.* Master's thesis, North Carolina State University

Bogert, CM (1943) Dentitional phenomena in cobras and other elapids with notes on adaptive modifications of fangs. *Bulletin American Museum of Natural History* 81:285–363

Boie, F (1827) Bemerkungen über Merrem's Versuch eines Systems der Amphibien. *Isis von Oken* 20:508–566

Broadley, DG (1959) The herpetology of Southern Rhodesia. part 1. snakes. *Buletin of the Museum of Comparative Zoology* 102:3–100

Broadley, DG (1968) A review of the African cobras of the genus *Naja* (*Serpentes: Elapinae*). *Arnoldia* 29:1–13

Broadley, DG, Wüster, W (2004) A review of the southern African 'non-spitting' cobras (Serpentes: Elapidae: *Naja*). *African Journal of Herpetology* 53:101–122

Burger, J, Gochfeld, M, Murray, BG, Jr. (1991) Role of a predoator's eye size in risk perception by basking black iguana, *Ctenosaura similis*. *Animal Behaviour* 42:471–476

Cascardi, J, Young, BA, Husic, HD, Sherma, J (1999) Protein variation in the venom spat by the red spitting cobra, *Naja pallida* (Reptilia: Serpentes). *Toxicon* 37:1271–1279

Castoe, TA, Smith, EN, Brown, RM, Parkinson, CL (2007) Highter-level phylogeny of Asian and American coralsnakes, their placement within the Elapidae (Squamata), and the systematic affinities of the enigmatic Asian coralsnake *Hemibungarus calligaster* (Wiegmann, 1834). *Zoological Journal of the Linnean Society* 151:809–831

CHAN-ARD, T, STUART, BL, WÜSTER, W (2000) First record of Indochinese spitting cobra *Naja siamensis* Laurenti (Serpentes: Elapidaae) from Laos, with comments on the genus in the country. *The Natural History Bulletin of the Siam Society* 48:149–152

CHEN, YH (1997) A new species of snake in China – *Trimeresurus mangshanensis*. In DAVID, P, TONG, H (eds.), *Translations of recent descriptions of Chinese pitvipers of the Trimeresurus-complex (Serpentes, Viperidae), with a key to the complex in China and adjacent areas*, volume 112 of *Smithsonian Herpetological Information Service*. Smithsonian Institution

D'EATH, RB (1998) Can video images imitate real stimuli in animal behaviour experiments? *Biological Reviews* 73:267–292

FITZSIMONS, FW (1912) *The snakes of South Africa*. T. Maskew Miller, Cape Town & Pretoria

FLEISHMAN, LJ, ENDLER, JA (2000) Some comments on visual perception and the use of video playback in animal studies. *Acta Ethologica* 3:15–27

FREYVOGEL, TA, HONEGGER, CG (1965) Der «Speiakt» von *Naja nigricollis*. *Acta Tropica* 22:289–302

GALLUP, GG, JR., NASH, RF, ELLISON, AL, JR. (1971) Tonic immobility as a reaction to predation: Artificial eyes as a fear stimulus for chickens. *Psychonomic Science* 23:79–80

GORING JONES, MD (1900) Can a cobra eject its poison. *Journal of the Bombay Natural History Society* 13:376

GREENE, HW (1999) *Schlangen: Faszination einer unbekannten Welt*. Birkhäuser

GRÜNTZIG, J, LENZ, W, BERKEMEIER, B, MEBS, D (1985) Experimental studies on the spitting cobra ophthalmia (*Naja nigricollis*). *Graefes Archive for Clinical and Experimantal Ophthalmology* 223:196–201

HERZOG, HA, JR., BERN, C (1992) Do garter snakes strike at the eyes of predators? *Animal Behaviour* 44:771–773

HOBLEY, CW (1911) Spitting cobra. *The Journal of the East Africa and Uganda Natural History Society* 1:98–101

Horton, CW (1948) On the mechanics of spitting in the African spitting cobra. *Copeia* 1948:23–25

Ismail, M, Al-Bekairi, AM, El-Bedaiwy, AM, Abd-El Salam, MA (1993a) The occular effects of spitting cobras: I. The ringhals cobra (*Hemachatus haemachatus*) venom incduced corneal opacification syndrome. *Clinical Toxicology* 31:31–41

Ismail, M, Al-Bekairi, AM, El-Bedaiwy, AM, Abd-El Salam, MA (1993b) The occular effects of spitting cobras: II. Evidence that coardiotoxins are responcible for the corneal opacification syndrome. *Clinical Toxicology* 31:45–62

Jacobs, GH, Fenwick, JA, Crognale, MA, Deegan II, JF (1992) The all-cone retina of the garter snake: spectral mechanisms and photopigment. *Journal of Comparative Physiology A* 170:701–707

Kamaier, F, Paschereit, CO (2007) *Strömungslehre*. Walter de Gruyter, Berlin

Kardong, KV, Lavin-Murcio, PA (1993) Venom delivery of snakes as high-pressure and low-pressure systems. *Copeia* 1993:644–650

Koch, M, Sachs, WB (1927) Über zwei giftspeiende Schlangen, *Sepedon haemachates* und *Naia nigricollis*. *Zoologischer Anzeiger* 70:155–159

Kohl, T, Young, BA (2011) Electrophysiology of the snake retina. poster presented at the 9$^{th}$ Göttingen Meeting of the German Neuroscience Society

Kopstein, F (1930) Die Giftschlangen Javas und ihre Bedeutung für den Menschen. *Zoomorphologie* 19:339–353

Laurenti, JN (1768) Specimen Medicum, Exhibens Synopsin Reptilium Emendatam cum Experimentis circa Venena. Vienna

Lawson, R, Slowinski, JB, Crother, BI, Burbrink, FT (2005) Phylogeny of the Colubroidea (Serpentes): New evidence from mitochondrial and nuclear genes. *Molecular Phylogenetics and Evolution* 37:581–601

Lee, MSY, Hugall, AF, Lawson, R, Scanlon, JD (2007) Phylogeny of snakes (Serpentes): Combining morphological and molecular data in likelyhood, Bayesian and parsimony analyses. *Systematics and Biodiversity* 5:371–389

LUISELLI, L, ANGELICI, FM (2000) Ecological relationships in two Afrotropical cobra species (*Naja melanoleuca* and *Naja nigricollis*). *Canadien Journal of Zoology* 78:191–198

MATTISON, C. (1995) *The encyclopedia of snakes.* Blandford, London

MURRAY, MA (1948) The serpent hieroglyph. *The Journal of Egyptian Archaeology* 34:117–118

NONNENMACHER, S, PIESCHE, M (2000) Design of hollow cone pressure swirl nozzles to atomize Newtonian fluids. *Chemincal Engineering Science* 55:4339–4348

O'BRIEN, TJ, DUNLAP, WP (1975) Tonic immobility in the blue crab (*Callinectes sapidus*, Rathbun): Its relation to threat of predation. *Journal of Comparative and Physiological Psychology* 89:86–94

DE PURY, S (2006) *Spuckverhalten und Spuckmuster von Speikobras (Naja pallida und Naja nigricollis).* diploma thesis, Westfälische Wilhelms-Universität Münster

PYRON, RA, BURBRINK, FT, COLLI, GR, NIETO MONTES DE OCA, A, VITT, LJ, KUCZYNSKI, CA, WIENS, JJ (2011) The phylogeny of advanced snakes (Colubroidea), with discovery of a new subfamily and comparison of support methods for likelihood tree. *Molecular Biology and Evolution* 58:329–342

RASMUSSEN, S, YOUNG, B, KRIMM, H (1995) On the 'spitting' behavior in cobras (Serpentes: Elapidae). *Journal of Zoology* 237:27–35

SEGUR, JB, OBERSTAR, HE (1951) Viscosity of glycerol and its aqueous solutions. *Industrial and Engineering Chemistry* 43:2117–2120

SLOWINSKI, JB, KNIGHT, A, ROONEY, AP (1997) Inferring species trees from gene trees: A phylogenetic analysis of the Elapidae (Serpentes) based on the amino acid sequences of venom proteins. *Molecular Phylogenetics and Evolution* 8:349–362

SLOWINSKI, JB, WÜSTER, W (2000) A new cobra (Elapidae: *Naja*) from Myanmar (Burma). *Herpetologica* 56:257–270

SPAWLS, S, HOWELL, K, DREWES, R, ASHE, J (2002) *A field guide to the reptiles of East Africa.* Academic Press, London

TRUTNAU, L (1998) *Giftschlangen.* Ulmer, Stuttgart

UNDERWOOD, G (1970) The eye. In GANS, C, PARSONS, TS (eds.), *Biology of the Reptilia*, volume 2, pp. 1–97. Academic Press, London

WALLACH, V, WÜSTER, W, BROADLEY, DG (2009) In praise of subgenera: taxonomic status of cobras of the genus *Naja* Laurenti (Serpentes: Elapidae). *Zootaxa* 2236:26–36

WALLS, GL (1942) *The vertebrate eye and its adaptive radiation.* Cranbrook Institute of Science, Bloomfield Hills, Michigan

WARRELL, DA, DAVID OMEROD, L (1976) Snake venom ophthalmia and blindness caused by the spitting cobra *Naja nigricollis* in Nigeria. *The American Journal of Hygiene* 25:525–529

WESTHOFF, G, TZSCHÄTZSCH, K, BLECKMANN, H (2005) The spitting behavior of two species of spitting cobras. *Journal of Comparative Physiology A* 191:873–881

WÜSTER, W (1996) Taxonomic changes and toxicology: systematic revision of the asiatic cobras (*Naja naja* species complex). *Toxicon* 34:399–406

WÜSTER, W, CROOKES, S, INEICH, I, MANÉ, Y, POOK, CE, TRAPE, J-F, BROADLEY, DG (2007) The phylogeny of cobras inferred from mitochondrial DNA sequences: Evolution of venom spitting and the phylogeography of the African spitting cobras (Serpentes: Elapidae: *Naja nigricollis* complex). *Molecular Phylogenetics and Evolution* 45:437–453

WÜSTER, W, THORPE, RS (1992) Dentitional phenomena in cobras revisited: Spitting and fang structure in the asiatic species of *Naja* (Serpentes: Elapidae). *Herpetologica* 48:424–434

WÜSTER, W, THORPE, RS (1994) *Naja siamensis*, a cryptic species of venomous snake revealed by mtDNA sequencing. *Experimentia* 50:75–79

WÜSTER, W, WARRELL, DA, COX, MJ, JINTAKUNE, P, NABHITABHATA, J (1997) Redescription of *Naja siamensis* (Serpentes: Elapidae), a widely overlooked spitting cobra from S.E. Asia: geographic variation, medical importance and designation of a neotype. *Journal of Zoology* 243:771–788

YOUNG, BA, BLAIR, M, ZAHN, K, MARVIN, J (2001) Mechanics of venom expulsion in *Crotalus*, with special reference to the role of the fang sheath. *The Anatomical Record* 264:415–426

YOUNG, BA, BOETIG, M, WESTHOFF, G (2009a) Functional bases of the spatial dispersal of venom during cobra "spitting". *Physiological and Biochemical Zoology* 82:80–89

YOUNG, BA, BOETIG, M, WESTHOFF, G (2009b) Spitting behaviour of hatchling red spitting cobras *Naja pallida*. *The Herpetological Journal* 19:185–191

YOUNG, BA, DUNLAP, K, KOENIG, K, SINGER, M (2004) The buccal buckle: the functional morphology of venom spitting in cobras. *Journal of Experimental Biology* 207:3483–3494

YOUNG, BA, ZAHN, K, BLAIR, M, LALOR, J (2000) Functional subdivision of the venom gland musculature and the regulation of venom expulsion in rattlesnakes. *Journal of Morphology* 246:249–259

ZEIL, J (2000) Depth cues, behavioural context, and natural illumination: some potential limitations of video playback techniques. *Acta Ethologica* 3:39–48

# A Test results

**Table 14:** Spitting at eyes. The characteristics of the glass eyes are abbreviated as: c = colourless iris, b = black, m = matt, s = shiny, h = horizontal, and v = vertical. The tests performed were Fisher exact tests for the spitting frequency and Kolmogorov-Smirnov tests for time and adjusted time. $Z$ is the test statistic of the Kolmogorov-Smirnov test.

| Eyes 1 | Eyes 2 | Parameter | $N_1$ | $N_2$ | $Z$ | $p$ |
|---|---|---|---|---|---|---|
| cmh | cmv | spitting frequency | 36 | 36 | | 0.357 |
| cmh | cmv | time | 32 | 35 | 0.767 | 0.599 |
| cmh | cmv | adjusted time | 32 | 35 | 0.759 | 0.612 |
| cmh | csh | spitting frequency | 36 | 36 | | > 0.999 |
| cmh | csh | time | 32 | 31 | 0.744 | 0.637 |
| cmh | csh | adjusted time | 32 | 31 | 0.596 | 0.875 |
| cmh | bmh | spitting frequency | 36 | 36 | | > 0.999 |
| cmh | bmh | time | 32 | 33 | 0.886 | 0.413 |
| cmh | bmh | adjusted time | 32 | 33 | 1.160 | 0.135 |
| cmv | csv | spitting frequency | 36 | 36 | | 0.357 |
| cmv | csv | time | 35 | 32 | 0.653 | 0.787 |
| cmv | csv | adjusted time | 35 | 32 | 0.559 | 0.914 |
| cmv | bmv | spitting frequency | 36 | 36 | | 0.107 |
| cmv | bmv | time | 35 | 30 | 0.995 | 0.275 |
| cmv | bmv | adjusted time | 35 | 30 | 0.785 | 0.569 |
| csh | csv | spitting frequency | 36 | 36 | | > 0.999 |
| csh | csv | time | 31 | 32 | 0.652 | 0.789 |
| csh | csv | adjusted time | 31 | 32 | 0.820 | 0.512 |
| csh | bsh | spitting frequency | 36 | 36 | | 0.710 |
| csh | bsh | time | 31 | 33 | 0.836 | 0.486 |
| csh | bsh | adjusted time | 31 | 33 | 1.071 | 0.202 |
| csv | bsv | spitting frequency | 36 | 36 | | 0.514 |
| csv | bsv | time | 32 | 29 | 0.693 | 0.722 |
| csv | bsv | adjusted time | 32 | 29 | 1.093 | 0.183 |
| bmh | bmv | spitting frequency | 36 | 36 | | 0.478 |
| bmh | bmv | time | 33 | 30 | 0.673 | 0.756 |
| bmh | bmv | adjusted time | 33 | 30 | 0.637 | 0.812 |

*continuation on the next page*

*continuation of table 14*

| Eyes 1 | Eyes 2 | Parameter | $N_1$ | $N_2$ | $Z$ | $p$ |
|---|---|---|---|---|---|---|
| bmh | bsh | spitting frequency | 36 | 36 | | > 0.999 |
| bmh | bsh | time | 33 | 33 | 0.615 | 0.843 |
| bmh | bsh | adjusted time | 33 | 33 | 0.615 | 0.843 |
| bmv | bsv | spitting frequency | 36 | 36 | | > 0.999 |
| bmv | bsv | time | 30 | 29 | 0.680 | 0.745 |
| bmv | bsv | adjusted time | 30 | 29 | 1.223 | 0.101 |
| bsh | bsv | spitting frequency | 36 | 36 | | 0.307 |
| bsh | bsv | time | 33 | 29 | 0.690 | 0.728 |
| bsh | bsv | adjusted time | 33 | 29 | 1.334 | 0.057 |
| absent | cmh | spitting frequency | 36 | 36 | | 0.514 |
| absent | cmh | time | 29 | 32 | 0.731 | 0.659 |
| absent | cmh | adjusted time | 29 | 32 | 1.143 | 0.146 |
| absent | cmv | spitting frequency | 36 | 36 | | 0.055 |
| absent | cmv | time | 29 | 35 | 0.812 | 0.524 |
| absent | cmv | adjusted time | 29 | 35 | 0.745 | 0.635 |
| absent | csh | spitting frequency | 36 | 36 | | 0.753 |
| absent | csh | time | 29 | 31 | 0.672 | 0.758 |
| absent | csh | adjusted time | 29 | 31 | 0.947 | 0.331 |
| absent | csv | spitting frequency | 36 | 36 | | 0.514 |
| absent | csv | time | 29 | 32 | 0.862 | 0.448 |
| absent | csv | adjusted time | 29 | 32 | 0.899 | 0.393 |
| absent | bmh | spitting frequency | 36 | 36 | | 0.307 |
| absent | bmh | time | 29 | 33 | 0.710 | 0.694 |
| absent | bmh | adjusted time | 29 | 33 | 0.801 | 0.543 |
| absent | bmv | spitting frequency | 36 | 36 | | > 0.999 |
| absent | bmv | time | 29 | 30 | 0.856 | 0.456 |
| absent | bmv | adjusted time | 29 | 30 | 0.887 | 0.411 |
| absent | bsh | spitting frequency | 36 | 36 | | 0.307 |
| absent | bsh | time | 29 | 33 | 0.993 | 0.277 |
| absent | bsh | adjusted time | 29 | 33 | 1.039 | 0.231 |
| absent | bsv | spitting frequency | 36 | 36 | | > 0.999 |
| absent | bsv | time | 29 | 29 | 0.788 | 0.564 |
| absent | bsv | adjusted time | 29 | 29 | 1.182 | 0.122 |

# B Source code

The following source code is written in C++. The program was used to process the pictures gained from the CT-scans of the fangs. It served as a tool to identify the fang itself and the venom canal within the fang and to determine their edges in an objektive manner that stayed the same for every fang. This was used for two tasks. The first was to generate pictures of the fang that could be used for the 3D reconstruction. The second was to measure the venom canal.

```
1  #include <iostream>
2  #include <vector>
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <math.h>
6  #include <time.h>
7  #include <fstream>
8  #include <string>
9
10 #define PI 3.14159265358979323846264338327 95
11
12 using namespace std;
13
14 struct listHead;
15
16 struct listElement
17 {
18     int number;
19     int emark;
20     int emark2;
21     int x;
22     int y;
23     listElement* next;
24     listElement* prev;
25     listHead* head;
26 };
27
28 struct listHead
29 {
30     int count;
31     int number;
32     int hmark;
33     int hmark2;
34     listHead* prev;
35     listHead* next;
36     listElement* first;
37     listElement* last;
38     listElement* p1;
39     listElement* p2;
40     listElement* p3;
41     listElement* internp1;
42     listElement* internp2;
43 };
44
```

```
45 struct fuenfFloat{
46     float el;
47     float al;
48     float bl;
49     float w;
50     float d;
51 };
52 
53 class Bild {
54     public:
55         Bild ( int width, int height, int layers );
56         ~Bild ();
57         Bild ( const Bild &bp );
58         void operator = ( const Bild &bp );
59         int GetPixelValue ( int xpos, int ypos, int layer );
60         void SetPixelValue ( int xpos, int ypos, int layer, int value );
61         int GetWidth ( );
62         int GetHeight ( );
63         int GetLayers ( );
64     private:
65         int m_width;
66         int m_height;
67         int m_layers;
68         int*** m_pixel;
69 };
70 
71 class ListList {
72     public:
73         ListList ();
74         ~ListList ();
75         ListList ( const ListList &ll );
76         listHead* operator[] (int index);
77         int GetCount ();
78         void IncCount ();
79         void DecCount ();
80         void SetCount ( int value );
81         listHead* GetList ( int index );
82         int InsertList ( listHead* lh, int pos );
83         int DeleteList ( int position );
84         int DeleteList ( listHead* lh );
85     private:
86         int m_count;
87         listHead* m_first;
88         listHead* m_last;
89         listHead* m_internp1;
90         listHead* m_internp2;
91 };
92 
93 class Vec2D {
94     private:
95         float m_x;
96         float m_y;
97         int m_ix;
98         int m_iy;
99         float m_l;
```

```
100         float m_w;
101     public:
102         Vec2D ( int ix, int iy );
103         Vec2D ( float xl, float yw, char art );
104         Vec2D ( double xl, double yw, char art );
105         Vec2D ( );
106         Vec2D operator+ ( Vec2D v1 );
107         Vec2D operator- ( Vec2D v1 );
108         Vec2D operator/ ( float sk );
109         void operator= ( Vec2D v1 );
110         friend Vec2D operator* ( float sk, Vec2D v1 );
111         friend Vec2D operator* ( Vec2D v1, float sk );
112         float dotp ( Vec2D v1 );
113         int GetIX ();
114         int GetIY ();
115         float GetX ();
116         float GetY ();
117         float GetL ();
118         float GetW ();
119         float CompL ();
120         float CompW ();
121         void SetIX ( int ix );
122         void SetIY ( int iY );
123         void SetIXIY ( int ix, int iy );
124         void SetX ( float x );
125         void SetY ( float y );
126         void SetXY ( float x, float y );
127         void SetL ( float l );
128         void SetW ( float w );
129         void SetLW ( float l, float w );
130 };
131
132 class Vec3D {
133     private:
134         float m_x;
135         float m_y;
136         float m_z;
137         int m_ix;
138         int m_iy;
139         int m_iz;
140         float m_l;
141     public:
142         Vec3D ( int ix, int iy, int iz );
143         Vec3D ( float x, float y, float z );
144         Vec3D ( double x, double y, double z );
145         Vec3D ( );
146         Vec3D operator+ ( Vec3D v1 );
147         Vec3D operator- ( Vec3D v1 );
148         Vec3D operator/ ( float sk );
149         void operator= ( Vec3D v1 );
150         friend Vec3D operator* ( float sk, Vec3D v1 );
151         friend Vec3D operator* ( Vec3D v1, float sk );
152         float dotp ( Vec3D v1 );
153         Vec3D crossp ( Vec3D v1 );
154         int GetIX ();
```

```
155         int GetIY ();
156         int GetIZ ();
157         float GetX ();
158         float GetY ();
159         float GetZ ();
160         float GetL ();
161         float CompL ();
162         void SetIX ( int ix );
163         void SetIY ( int iY );
164         void SetIZ ( int iZ );
165         void SetIXIYIZ ( int ix, int iy, int iz );
166         void SetX ( float x );
167         void SetY ( float y );
168         void SetZ ( float z );
169         void SetXYZ ( float x, float y, float z );
170         void SetL ( float l );
171 };
172
173 listHead* appendLists ( listHead* listHead1, listHead* listHead2, int pointer1,
        int pointer2, int pointer3 );
174 int insertElement ( listHead* listHead1, int position, int xpos, int ypos );
175 int deleteElement ( listHead* listHead1, int position );
176 listHead* createList ();
177 void deleteList ( listHead* &listHead1 );
178 void emptyList ( listHead* &listHead1 );
179 void setListPointer ( listElement* &pointer, listHead* &lhead, int num );
180 void reverseList ( listHead* &listHead1 );
181 listHead* copyList ( listHead* oldList );
182 void copyList ( listHead* sourceList, listHead* &targetList );
183 bool compareLists ( listHead* listHead1, listHead* listHead2 );
184 listElement* teileSortList ( listHead* listHead1, listElement* links,
        listElement* rechts, int was );
185 void quickSortList ( listHead* listHead1, listElement* links, listElement*
        rechts, int was );
186 int writeList( listHead* listHead1, FILE* datei );
187 int readfour ( ifstream& datei, int anfang );
188 int readnumber ( ifstream& datei, int anfang, int anzahl );
189 int teile38 ( int links, int rechts, int liste[3][8], int pos );
190 void quicksort38 ( int links, int rechts, int liste[3][8], int pos );
191 int teile68 ( int links, int rechts, int liste[5][8], int pos );
192 void quicksort68 ( int links, int rechts, int liste[5][8], int pos );
193 int berechnungS ( listHead* liPos, Bild &bp, int l1, int l2, int l3, int l4, int
         l5, FILE* datei1, FILE* datei2, int einrueck, int number );
194 void printFloat ( float zahl, FILE* datei, float nachkomma );
195 int flaechen ( Bild &bp, int xpos, int ypos, int mark, int ersteEbene, int
        ersterWert, int ersterBereich, int zweiteEbene, int zweiterWert, int
        zweiterBereich, int dritteEbene, int dritterWert, int dritterBereich );
196 void putNumberInFile ( int number, FILE* file );
197 int getNumberFromFile ( FILE* file);
198 float ellDiff ( Bild &bp, listHead* liPos, float para[5], int parameter, float
        startWerte[2], int bpRi, bool einzel );
199 int maxKantensuche ( Bild &bp2, listHead* liRand[2], bool firstNumber, int hist
        [256], int grauGrenzeAlt, float oeffnungAlt, float ruecktemp[3], bool
        keinGiftkanal, bool keinMarkkanal, bool mark[1], FILE* mess );
200 int sortListByCoordinates ( listHead* listHead1, listHead* liAus );
```

```
201 int readfour ( ifstream& datei, int anfang );
202
203 int main ()
204 {
205     FILE* diff;
206     FILE* mess;
207     FILE* dif2;
208     FILE* mes2;
209
210     bool gefunden[4] = {false, false, false, false};
211     bool gleicheRichtung = false;
212     bool keinGiftkanal = false;
213     bool keinMarkkanal = false;
214     bool keinMarkkanalStart = false;
215     bool verbGrau = false;
216
217     char zahl = 0;
218     char c = 0;
219     char pause = 'x';
220
221     int einr = 1;
222     int height = 0;
223     int i = 0;
224     int ia = 0;
225     int ib = 0;
226     int ic = 0;
227     int id = 0;
228     int ig = 0;
229     int ih = 0;
230     int ii = 0;
231     int ij = 0;
232     int ik = 0;
233     int iarray[9] = {0};
234     int iRand = 0;
235     int j = 0;
236     int kante = -1;
237     int changeNumber = 0;
238     int maxnumber = 0;
239     int minnumber = 0;
240     int number = 0;
241     int numziff = 0;
242     int posx = 0;
243     int posy = 0;
244     int snumber = 0;
245     int startnumber = 0;
246     int startnumber2 = 0;
247     int stepsize = 1;
248     int verbLinie[4][2] = {0};
249     int width = 0;
250
251     int maxkante = 0;
252
253     long start = 1262;
254
255     float blackwhite = 0.0;
```

```
256     float ftemp = 0.0;
257     float ftemp2 = 0.0;
258     float ftemp3 = 0.0;
259     float ftemp4 = 0.0;
260     float ftemp5 = 0.0;
261     float ftemp6 = 0.0;
262     float ruecktemp[3] = {0, 0, 0};
263
264     time_t startzeit;
265     char zeit [29];
266     char zeit2 [30];
267     int jahr = 2007;
268     int monat = 0;
269     int tag = 0;
270     int rechentag = 0;
271     int stunde = 0;
272     int minute = 0;
273     int sekunde = 0;
274     int rest = 0;
275     int sowi = 0;
276     int monate[12];
277     monate[0] = 31;
278     monate[1] = 28;
279     monate[2] = 31;
280     monate[3] = 30;
281     monate[4] = 31;
282     monate[5] = 30;
283     monate[6] = 31;
284     monate[7] = 31;
285     monate[8] = 30;
286     monate[9] = 31;
287     monate[10] = 30;
288     monate[11] = 31;
289
290     startzeit = time(NULL);
291     rechentag = (startzeit-1199145600)/86400;
292     tag = rechentag + 1;
293     while (tag > 0) {
294         if ( monat%12 == 0 ) {
295             jahr++;
296             monat = 0;
297             monate[1] = ( ( jahr%4 == 0 && ( jahr%100 != 0 || jahr%400 == 0 ) )
                    ? 29 : 28 );
298         };
299         if ( tag > monate[monat] ) {
300             tag -= monate[monat];
301             monat++;
302         }
303         else {
304             monat++;
305             break;
306         };
307     };
308     i = jahr%100;
309     j = (i/12+i%12+(i%12)/4+2)%7;
```

```
310     int wotali[2];
311     wotali[0] = j;
312     wotali[1] = (j+35-10)%7;
313
314     if ( monat < 3 || monat > 10 ) {
315         sowi = 1;
316     }
317     else {
318         if ( monat > 3 && monat < 10 ) {
319             sowi = 2;
320         }
321         else {
322             if ( monat == 3 ) {
323                 sowi = (tag < 31-(j+3)%7 ? 1 : 2);
324             }
325             else {
326                 sowi = (tag < 31-j ? 2 : 1);
327             };
328         };
329     };
330
331     rest = (startzeit-1199145600-(rechentag*86400))%86400;
332     stunde = rest/3600 +sowi;
333     rest = rest%3600;
334     minute = rest/60;
335     sekunde = rest%60;
336
337     zeit[0] = 'd';
338     zeit[1] = 'i';
339     zeit[2] = 'f';
340     zeit[3] = 'f';
341     zeit[4] = '_';
342     zeit[5] = (jahr/1000)+48;
343     zeit[6] = ((jahr/100)%10)+48;
344     zeit[7] = ((jahr/10)%10)+48;
345     zeit[8] = (jahr%10)+48;
346     zeit[9] = '-';
347     zeit[10] = (monat/10)+48;
348     zeit[11] = (monat%10)+48;
349     zeit[12] = '-';
350     zeit[13] = (tag/10)+48;
351     zeit[14] = (tag%10)+48;
352     zeit[15] = '_';
353     zeit[16] = (stunde/10)+48;
354     zeit[17] = (stunde%10)+48;
355     zeit[18] = '-';
356     zeit[19] = (minute/10)+48;
357     zeit[20] = (minute%10)+48;
358     zeit[21] = '-';
359     zeit[22] = (sekunde/10)+48;
360     zeit[23] = (sekunde%10)+48;
361     zeit[24] = '.';
362     zeit[25] = 't';
363     zeit[26] = 'x';
364     zeit[27] = 't';
```

```
365     zeit[28] = '\0';
366
367     diff = fopen(zeit, "w");
368
369     zeit[3] = '2';
370
371     dif2 = fopen(zeit, "w");
372
373     zeit[0] = 'm';
374     zeit[1] = 'e';
375     zeit[2] = 's';
376     zeit[3] = 's';
377
378     mess = fopen(zeit, "w");
379
380     zeit[3] = '2';
381
382     mes2 = fopen(zeit, "w");
383
384     cout << "Bitte Dateinamen (der *.bmp) eingeben: ";
385     string sname;
386     cin >> sname;
387     int strlen = sname.length();
388     char name[strlen+5];
389     sname.copy(name,strlen);
390
391     numziff = 0;
392     i = strlen - 1;
393     while ( name[i] > 47 && name[i] < 58 && i >= 0 ) {
394         numziff++;
395         i--;
396     };
397     cout << "Ermittelte Anzahl der Ziffern: " << numziff << endl;
398
399     char nameelli[strlen+13];
400     char nameell2[strlen+14];
401     char namerand[strlen+10];
402     char nameran2[strlen+11];
403     char nameamir[strlen+11];
404     char nameami2[strlen+12];
405     char namekasu[strlen+10];
406     char namelog[strlen+9];
407     char namesuch[strlen+9];
408     for ( i = 0; i < strlen; i++ ) {
409         nameamir[i+6] = name[i];
410         nameami2[i+7] = name[i];
411         namerand[i+5] = name[i];
412         nameran2[i+6] = name[i];
413         nameelli[i+8] = name[i];
414         nameell2[i+9] = name[i];
415         namekasu[i+5] = name[i];
416         namelog[i+4]  = name[i];
417         namesuch[i+4] = name[i];
418     };
419     nameamir[0] = 'a';
```

```
420       nameamir[1] = 'm';
421       nameamir[2] = 'i';
422       nameamir[3] = 'r';
423       nameamir[4] = 'a';
424       nameami2[0] = 'a';
425       nameami2[1] = 'm';
426       nameami2[2] = 'i';
427       nameami2[3] = 'r';
428       nameami2[4] = 'a';
429       nameami2[5] = '2';
430       nameelli[0] = 'e';
431       nameelli[1] = 'l';
432       nameelli[2] = 'l';
433       nameelli[3] = 'i';
434       nameelli[4] = 'p';
435       nameelli[5] = 's';
436       nameelli[6] = 'e';
437       nameell2[0] = 'e';
438       nameell2[1] = 'l';
439       nameell2[2] = 'l';
440       nameell2[3] = 'i';
441       nameell2[4] = 'p';
442       nameell2[5] = 's';
443       nameell2[6] = 'e';
444       nameell2[7] = '2';
445       namekasu[0] = 'k';
446       namekasu[1] = 'a';
447       namekasu[2] = 's';
448       namekasu[3] = 'u';
449       namelog[0]  = 'l';
450       namelog[1]  = 'o';
451       namelog[2]  = 'g';
452       namerand[0] = 'r';
453       namerand[1] = 'a';
454       namerand[2] = 'n';
455       namerand[3] = 'd';
456       nameran2[0] = 'r';
457       nameran2[1] = 'a';
458       nameran2[2] = 'n';
459       nameran2[3] = 'd';
460       nameran2[4] = '2';
461       namesuch[0] = 'r';
462       namesuch[1] = 'z';
463       namesuch[2] = 'w';
464       namerand[4] = nameran2[5] = namekasu[4] = nameamir[5] = nameami2[6] =
              nameelli[7] = nameell2[8] = namesuch[3] = namelog[3] = '-';
465       name[strlen]   = namesuch[strlen+4] = namerand[strlen+5] = nameran2[strlen
              +6]  = namekasu[strlen+5] = namelog[strlen+4] = nameelli[strlen+8]  =
              nameell2[strlen+9]  = nameamir[strlen+6]  = nameami2[strlen+7]  = '.';
466       name[strlen+1] = namesuch[strlen+5] = namerand[strlen+6] = nameran2[strlen
              +7]                                          = nameelli[strlen+9]  =
              nameell2[strlen+10] = nameamir[strlen+7]  = nameami2[strlen+8]  = 'b';
467       name[strlen+2] = namesuch[strlen+6] = namerand[strlen+7] = nameran2[strlen
              +8]                                          = nameelli[strlen+10] =
              nameell2[strlen+11] = nameamir[strlen+8]  = nameami2[strlen+9]  = 'm';
```

```
468     name[strlen+3] = namesuch[strlen+7] = namerand[strlen+8] = nameran2[strlen
            +9]                                          = nameelli[strlen+11] =
            nameell2[strlen+12] = nameamir[strlen+9]  = nameami2[strlen+10] = 'p';
469     name[strlen+4] = namesuch[strlen+8] = namerand[strlen+9] = nameran2[strlen
            +10] = namekasu[strlen+9] = namelog[strlen+8] = nameelli[strlen+12] =
            nameell2[strlen+13] = nameamir[strlen+10] = nameami2[strlen+11] = '\0';
470     namelog[strlen+5] = namekasu[strlen+6] = namelog[strlen+7] = namekasu[strlen
            +8] = 't';
471     namelog[strlen+6] = namekasu[strlen+7] = 'x';
472
473     for ( j = 0 ; j < numziff ; j++ ) {
474         number += ((static_cast<int>(name[strlen-1-j])-48)*static_cast<int>(pow
                (10,j)+.5));
475     };
476     startnumber = number;
477     startnumber2 = startnumber;
478
479     i = static_cast<int>(pow(10.0,numziff)+0.5) - 1;
480     do {
481         cout << "Bitte minimale Dateinummer eingeben: ";
482         cin >> minnumber;
483     }
484     while ( minnumber > number || minnumber < 0 || minnumber > i );
485     do {
486         cout << "Bitte maximale Dateinummer eingeben: ";
487         cin >> maxnumber;
488     }
489     while ( maxnumber < number || maxnumber > i );
490     int scheibenAnzahl = maxnumber - minnumber + 1;
491
492     do {
493         cout << "Bitte angeben ob der Markkanal gesucht werden soll (0=nein;1=ja
                ): ";
494         cin >> i;
495     }
496     while ( i < 0 || i > 1 );
497     if ( i == 0 ) {
498         keinMarkkanal = true;
499         keinMarkkanalStart = true;
500     };
501
502     bool nurBilder = false;
503     do {
504         cout << "Bitte angeben ob nur Bilder ausgegeben werden soll (0=nein;1=ja
                ): ";
505         cin >> i;
506     }
507     while ( i < 0 || i > 1 );
508     if ( i == 1 ) {
509         nurBilder = true;
510     };
511
512     ifstream fbild( name, ios::binary );
513     if ( fbild.is_open() != true ) {
514         printf("  Datei %s konnte nicht geoeffnet werden!", name);
```

```
515         return -1;
516     };
517
518     int rUmfeld[2][9];
519     rUmfeld[0][1] = rUmfeld[0][2] = rUmfeld[0][3] = rUmfeld[1][1] = rUmfeld
            [1][7] = rUmfeld[1][8] = -1;
520     rUmfeld[0][0] = rUmfeld[0][4] = rUmfeld[0][8] = rUmfeld[1][0] = rUmfeld
            [1][2] = rUmfeld[1][6] = 0;
521     rUmfeld[0][5] = rUmfeld[0][6] = rUmfeld[0][7] = rUmfeld[1][3] = rUmfeld
            [1][4] = rUmfeld[1][5] = 1;
522
523     int umwandel[8];
524     umwandel[0] = 1; umwandel[1] = 2; umwandel[2] = 3; umwandel[3] = 5; umwandel
            [4] = 8; umwandel[5] = 7; umwandel[6] = 6; umwandel[7] = 4;
525
526     int zw[3][4];
527     zw[1][3] = -1;
528     zw[0][2] = zw[1][0] = zw[2][0] = zw[2][3] = 0;
529     zw[0][0] = zw[0][1] = zw[0][3] = zw[1][1] = zw[1][2] = zw[2][1] = zw[2][2] =
             1;
530
531     start = readfour(fbild,10);
532     width = readfour(fbild,18);
533     height = readfour(fbild,22);
534     fbild.close();
535
536     int nullen = ( ( 4 - ( width % 4 ) ) % 4 );
537
538     listHead* randList[2];
539     randList[0] = createList();
540     randList[1] = createList();
541
542     float giftMitte[scheibenAnzahl][2];
543     float giftMitteM[scheibenAnzahl][2];
544     Bild* bp = new Bild(width, height, scheibenAnzahl);
545     Bild* bpe = new Bild(width, height, scheibenAnzahl);
546     Bild* bpe2 = new Bild(width, height, 6);
547     Bild* bp2 = new Bild(width, height, 9);
548     fprintf(diff, "Nummer%cZaehler%cWinkel%cDifferenz", 9, 9, 9);
549     for ( i = startnumber; i > minnumber - 1; i-- ) {
550         fprintf(diff, "%cN-%i", 9, i);
551     };
552     for ( i = startnumber + 1; i < maxnumber + 1; i++ ) {
553         fprintf(diff, "%cN-%i", 9, i);
554     };
555     for ( i = minnumber; i < maxnumber + 1; i++ ) {
556         fprintf(dif2, "%cN-%i", 9, i);
557     };
558     fprintf(diff, "\n");
559     fprintf(dif2, "\n");
560     fprintf(mess, "Nummer%cBreite der Giftkanaloeffnung%cmaximaler Durchmesser%
            cminimaler Durchmesser%cUmfang%cFlaeche%cEllipsenwinkel%cEllipsenflaeche
            %cnum. Exzentrizitaet%cDiff%cWinkel 1%cWinkel 2%cWinkel 3%cDifferenz 1%
            cDifferenz 2%cDifferenz 3%cWinkel min zwischen%cDifferenz min zwischen%
            cWinkel min absolut%cDifferenz min absolut%cmittlere Differenz (Betrag)%
```

```
            cStandardabweichung (Betrag)%cmittlere Differenz%cStandardabweichung\n",
             9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 );
561         fprintf(mes2, "Nummer%cmaximaler Durchmesser%cminimaler Durchmesser%cUmfang%
            cFlaeche%cEllipsenwinkel%cEllipsenflaeche%cnum. Exzentrizitaet%cDiff%
            cWinkel 1%cWinkel 2%cWinkel 3%cDifferenz 1%cDifferenz 2%cDifferenz 3%
            cWinkel min zwischen%cDifferenz min zwischen%cWinkel min absolut%
            cDifferenz min absolut%cmittlere Differenz (Betrag)%cStandardabweichung
            (Betrag)%cmittlere Differenz%cStandardabweichung%cNormalenvektor x%
            cNormalenvektor y%cNormalenvektor z\n", 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
             9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 );
562
563         for ( number = minnumber; number <= maxnumber; number++ ) {
564             cout << "number: " << number << endl;
565             for ( j = 0; j < numziff; j++ ) {
566                 zahl = ( number - number / static_cast<int>(pow(10,numziff-j)+.5) *
                        static_cast<int>(pow(10,numziff-j)+.5) ) / static_cast<int>(pow
                        (10,numziff-1-j)+.5) +48;
567                 name[strlen-numziff+j] = zahl;
568                 nameamir[strlen+6-numziff+j] = zahl;
569                 nameelli[strlen+8-numziff+j] = zahl;
570                 namerand[strlen+5-numziff+j] = zahl;
571                 namesuch[strlen+4-numziff+j] = zahl;
572                 namelog[strlen+4-numziff+j]  = zahl;
573                 namekasu[strlen+5-numziff+j] = zahl;
574             };
575             ifstream fbild( name, ios::binary );
576             if ( fbild.is_open() != true ) {
577                 printf("  Datei %s konnte nicht geoeffnet werden!", name);
578                 return -1;
579             }
580
581             fbild.seekg(start);
582
583             for ( i = height - 1; i >= 0; i-- ) {
584                 for ( j = 0; j < width; j++ ) {
585                     fbild.get(c);
586                     if ( fbild.fail() ) {
587                         cout << "  Lesefehler aus Bilddatei." << endl;
588                         break;
589                     };
590                     ia = static_cast<int>(c);
591                     ia = ( ia < 0 ? 256 + ia : ia );
592                     bp->SetPixelValue(j, i, number - minnumber, ia);
593                 };
594                 for ( j = 0; j < nullen; j++ ) {
595                     fbild.get(c);
596                 };
597             };
598             fbild.close();
599         };
600
601         cout << "  Jetzt sind alle Bildpunkte eingetragen." << endl;
602
603         int hist[256] = {0};
604         int grauGrenze = 0;
```

```
605
606     number = startnumber;
607     snumber = number - minnumber;
608
609     Bild* bpStart = new Bild(width, height, 1);
610
611     listHead* listStart[2];
612     listStart[0] = createList();
613     listStart[1] = createList();
614
615     listHead* listTest;
616     listTest = createList();
617
618     ifstream fbild2 (name, ios::binary);
619     if ( fbild2.is_open() != true ) {
620         cout << "  Bilddatei konnte nicht geoeffnet werden." << endl;
621         return -1;
622     };
623
624     int grauGrenzenListe[scheibenAnzahl];
625     float kanalOeffnung = 0.0;
626     float kanalMittelXStart = 0;
627     float kanalMittelYStart = 0;
628     bool mark[1];
629     mark[0] = keinMarkkanal;
630
631     while ( number <= maxnumber ) {
632         cout << "number = " << number << endl;
633         fprintf(mess, "%i", number);
634         snumber = number - minnumber;
635         for ( i = 0; i < height; i++ ) {
636             for ( j = 0; j < width; j++ ) {
637                 bp2->SetPixelValue(j, i, 0, -1);
638                 bp2->SetPixelValue(j, i, 1, bp->GetPixelValue(j, i, snumber));
639             };
640         };
641
642         emptyList(randList[0]);
643         emptyList(randList[1]);
644
645         grauGrenze = maxKantensuche(*bp2, randList, (number == startnumber ?
                true : false), hist, grauGrenze, kanalOeffnung, ruecktemp,
                keinGiftkanal, keinMarkkanal, mark, mess);
646         grauGrenzenListe[snumber] = grauGrenze;
647         kanalOeffnung = ruecktemp[0];
648
649         id = 0;
650         ftemp = 0.0;
651         ftemp2 = 0.0;
652         ftemp3 = 0.0;
653         for ( i = 1; i < height - 1; i++ ) {
654             for ( j = 1; j < width - 1; j++ ) {
655                 id = bp2->GetPixelValue(j, i, 4);
656                 bpe->SetPixelValue(j, i, snumber, id);
657                 if ( id == 3 ) {
```

```
658                     ftemp2 += static_cast<float>(j);
659                     ftemp3 += static_cast<float>(i);
660                     ftemp += 1.0;
661                 };
662             };
663         };
664         if ( ftemp > 0.5 ) {
665             ftemp2 /= ftemp;
666             ftemp3 /= ftemp;
667             giftMitte[snumber][0] = ftemp2;
668             giftMitte[snumber][1] = ftemp3;
669         }
670         else {
671             giftMitte[snumber][0] = 0.0;
672             giftMitte[snumber][1] = 0.0;
673         };
674 
675         id = 0;
676         for ( i = 1; i < height - 1; i++ ) {
677             for ( j = 1; j < width - 1; j++ ) {
678                 ia = ( bp2->GetPixelValue(j-1, i-1, 1) + bp2->GetPixelValue(j-1,
                         i+1, 1) - bp2->GetPixelValue(j+1, i-1, 1) - bp2->
                     GetPixelValue(j+1, i+1, 1) + 2 * bp2->GetPixelValue(j-1, i,
                     1) - 2 * bp2->GetPixelValue(j+1, i, 1) );
679                 ib = ( -1 * bp2->GetPixelValue(j-1, i-1, 1) + bp2->GetPixelValue
                     (j-1, i+1, 1) - bp2->GetPixelValue(j+1, i-1, 1) + bp2->
                     GetPixelValue(j+1, i+1, 1) - 2 * bp2->GetPixelValue(j, i-1,
                     1) + 2 * bp2->GetPixelValue(j, i+1, 1) );
680                 ftemp = sqrt(static_cast<float>(ia * ia + ib * ib));
681                 ic = static_cast<int>(ftemp+0.5);
682                 bp2->SetPixelValue(j, i, 2, ic);
683                 id = ( ic > id ? ic : id );
684                 if ( kante < ic ) {
685                     kante = ic;
686                 };
687                 if ( ia != 0 ) {
688                     ftemp = atan( static_cast<float>(ib) / static_cast<float>(ia
                         ) );
689                     if ( ftemp <= -1.178 || ftemp >= 1.178 ) {
690                         bp2->SetPixelValue(j, i, 5, 1);
691                     }
692                     else if ( ftemp > -0.393 && ftemp < 0.393 ) {
693                         bp2->SetPixelValue(j, i, 5, 3);
694                     }
695                     else if ( ftemp >= 0.393 && ftemp < 1.178 ) {
696                         bp2->SetPixelValue(j, i, 5, 2);
697                     }
698                     else if ( ftemp <= -0.393 && ftemp > -1.178 ) {
699                         bp2->SetPixelValue(j, i, 5, 4);
700                     }
701                     else {
702                         cout << "  FEHLER!!" << endl;
703                     };
704                 }
705                 else {
```

```
706                     if ( ib != 0 ) {
707                         bp2->SetPixelValue(j, i, 5, 1);
708                     }
709                     else {
710                         bp2->SetPixelValue(j, i, 5, 3);
711                     };
712                 };
713             };
714         };
715 
716         if ( !keinGiftkanal && !nurBilder ) {
717             cout << "  Berechnung des Giftkanals." << endl;
718             emptyList(listTest);
719             if ( randList[1]->count > 0 ) {
720                 sortListByCoordinates(randList[1], listTest);
721                 einr = berechnungS(randList[1], *bp2, 2, 5, 0, 3, 4, diff, mess,
                         einr, number);
722                 fprintf(mess, "\n");
723                 if ( randList[1]->count < 9 ) {
724                     keinGiftkanal = true;
725                 };
726                 for ( i = 0; i < height; i++ ) {
727                     for ( j = 0; j < width; j++ ) {
728                         bp2->SetPixelValue(j, i, 8, bp2->GetPixelValue(j, i, 1))
                             ;
729                     };
730                 };
731                 randList[1]->p1 = randList[1]->first;
732                 while ( randList[1]->p1 != 0 ) {
733                     bp2->SetPixelValue(randList[1]->p1->x, randList[1]->p1->y,
                         8, 255 );
734                     randList[1]->p1 = randList[1]->p1->next;
735                 };
736             }
737             else {
738                 cout << "  Kein Giftkanal" << endl;
739                 keinGiftkanal = true;
740                 einr++;
741             };
742         }
743         else {
744             for ( i = 0; i < height; i++ ) {
745                 for ( j = 0; j < width; j++ ) {
746                     bp2->SetPixelValue(j, i, 3, bp2->GetPixelValue(j, i, 1));
747                     bp2->SetPixelValue(j, i, 8, bp2->GetPixelValue(j, i, 1));
748                 };
749             };
750             einr++;
751         };
752 
753         for ( j = 0; j < numziff; j++ ) {
754             zahl = ( number - number / static_cast<int>(pow(10,numziff-j)+.5) *
                 static_cast<int>(pow(10,numziff-j)+.5) ) / static_cast<int>(pow
                 (10,numziff-1-j)+.5) +48;
755             nameamir[strlen+6-numziff+j] = zahl;
```

```
756             nameelli[strlen+8-numziff+j] = zahl;
757             namerand[strlen+5-numziff+j] = zahl;
758         };
759         ofstream ambild (nameamir, ios::binary);
760         if ( ambild.is_open() != true ) {
761             cout << "  Neue Bilddatei konnte nicht erstellt werden." << endl;
762             fbild2.close();
763             return -1;
764         }
765         ofstream elbild (nameelli, ios::binary);
766         if ( elbild.is_open() != true ) {
767             cout << "  Neue Bilddatei konnte nicht erstellt werden." << endl;
768             fbild2.close();
769             ambild.close();
770             return -1;
771         }
772         ofstream rabild (namerand, ios::binary);
773         if ( rabild.is_open() != true ) {
774             cout << "  Neue Bilddatei konnte nicht erstellt werden." << endl;
775             fbild2.close();
776             ambild.close();
777             elbild.close();
778             return -1;
779         }
780         fbild2.seekg(0);
781 
782         for ( i = 0; i < start; i++ ) {
783             fbild2.get(c);
784             if ( fbild2.fail() ) {
785                 cout << "  Lesefehler aus Bilddatei." << endl;
786                 break;
787             };
788             ambild.put(c);
789             if ( ambild.fail() ) {
790                 cout << "  Schreibfehler in neuer Bilddatei. (Amira)" << endl;
791                 break;
792             };
793             elbild.put(c);
794             if ( elbild.fail() ) {
795                 cout << "  Schreibfehler in neuer Bilddatei. (Ellipse)" << endl;
796                 break;
797             };
798             rabild.put(c);
799             if ( rabild.fail() ) {
800                 cout << "  Schreibfehler in neuer Bilddatei. (Rand)" << endl;
801                 break;
802             };
803         };
804         for ( i = height - 1; i >= 0; i-- ) {
805             for ( j = 0; j < width; j++ ) {
806                 ambild.put(255 - (bp2->GetPixelValue(j, i, 4)*50));
807                 elbild.put(bp2->GetPixelValue(j, i, 3));
808                 rabild.put(bp2->GetPixelValue(j, i, 8));
809             };
810             for ( j = 0; j < nullen; j++ ) {
```

```
811                 ambild.put(0);
812                 elbild.put(0);
813                 rabild.put(0);
814             };
815         };
816         ambild.close();
817         elbild.close();
818         rabild.close();
819 
820         if ( number == startnumber ) {
821             for ( i = 0; i < height; i++ ) {
822                 for ( j = 0; j < width; j++ ) {
823                     ia = bp2->GetPixelValue(j, i, 4);
824                     bpStart->SetPixelValue(j, i, 0, ia);
825                     bp2->SetPixelValue(j, i, 7, ia);
826                 };
827             };
828             copyList(randList[0], listStart[0]);
829             copyList(randList[1], listStart[1]);
830             kanalMittelXStart = ruecktemp[1];
831             kanalMittelYStart = ruecktemp[2];
832         }
833         else {
834             for ( i = 0; i < height; i++ ) {
835                 for ( j = 0; j < width; j++ ) {
836                     ia = bp2->GetPixelValue(j, i, 4);
837                     bp2->SetPixelValue(j, i, 7, ia);
838                 };
839             };
840         };
841 
842         if ( number <= minnumber ) {
843             number = startnumber + 1;
844             stepsize *= -1;
845             keinGiftkanal = false;
846             keinMarkkanal = keinMarkkanalStart;
847             mark[0] = keinMarkkanalStart;
848             for ( iRand = 0; iRand < 2; iRand++ ) {
849                 emptyList(randList[iRand]);
850                 copyList(listStart[iRand], randList[iRand]);
851             };
852             for ( i = 0; i < height; i++ ) {
853                 for ( j = 0; j < width; j++ ) {
854                     ia = bpStart->GetPixelValue(j, i, 0);
855                     bp2->SetPixelValue(j, i, 7, ia);
856                 };
857             };
858             ruecktemp[1] = kanalMittelXStart;
859             ruecktemp[2] = kanalMittelYStart;
860         }
861         else {
862             number -= stepsize;
863             keinMarkkanal = mark[0];
864         };
865     };
```

```
866
867     if ( !nurBilder ) {
868         float giAround[9][2] = {{0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0},
                {0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0}};
869         bool bAround[10] = {false, false, false, false, false, false, false,
                false, false, false};
870         for ( i = 0; i < scheibenAnzahl; i++ ) {
871             giftMitteM[i][0] = 0.0;
872             giftMitteM[i][1] = 0.0;
873             if ( giftMitte[i][0] > 0.5 ) {
874                 giAround[i%9][0] = giftMitte[i][0];
875                 giAround[i%9][1] = giftMitte[i][1];
876                 bAround[i%9] = true;
877                 bAround[9] = ( bAround[0] && bAround[1] && bAround[2] && bAround
                        [3] && bAround[4] && bAround[5] && bAround[6] && bAround[7]
                        && bAround[8] );
878             }
879             else {
880                 bAround[i%9] = false;
881                 bAround[9] = false;
882             };
883             if ( bAround[9] ) {
884                 giftMitteM[i-4][0] = ( giAround[0][0] + giAround[1][0] +
                        giAround[2][0] + giAround[3][0] + giAround[4][0] + giAround
                        [5][0] + giAround[6][0] + giAround[7][0] + giAround[8][0] )
                        / 9.0;
885                 giftMitteM[i-4][1] = ( giAround[0][1] + giAround[1][1] +
                        giAround[2][1] + giAround[3][1] + giAround[4][1] + giAround
                        [5][1] + giAround[6][1] + giAround[7][1] + giAround[8][1] )
                        / 9.0;
886             };
887         };
888
889         Vec3D vm1(0, 0, 0);
890         Vec3D vm2(0, 0, 0);
891         Vec3D vn(0, 0, 0);
892         Vec3D va(0, 0, 0);
893         Vec3D vb(0, 0, 0);
894         Vec3D vp(0, 0, 0);
895
896         einr = 1;
897
898         for ( number = minnumber + 1; number < maxnumber; number++ ) {
899             snumber = number - minnumber;
900             cout << "Normalenvektor snumber = " << snumber << endl;
901             if ( giftMitteM[snumber][0] < 0.5 ) {
902                 einr++;
903             }
904             else {
905                 gefunden[1] = false;
906                 vm1.SetXYZ(giftMitteM[snumber][0], giftMitteM[snumber][1],
                        static_cast<float>(snumber));
907                 if ( giftMitteM[snumber-1][0] > 0.5 ) {
908                     vm2.SetXYZ(giftMitteM[snumber-1][0], giftMitteM[snumber
                            -1][1], static_cast<float>(snumber-1));
```

```
909                     vn = vm2 - vm1;
910                 }
911                 else if ( number < maxnumber ) {
912                     vm2.SetXYZ(giftMitteM[snumber+1][0], giftMitteM[snumber
                        +1][1], static_cast<float>(snumber+1));
913                     vn = vm1 - vm2;
914                 }
915                 else {
916                     cout << "  nicht auswertbar" << endl;
917                     break;
918                 };
919                 va.SetX(vn.GetZ() / sqrt( vn.GetX() * vn.GetX() + vn.GetZ() * vn
                    .GetZ()));
920                 va.SetY(0.0);
921                 va.SetZ(vn.GetX() / sqrt( vn.GetX() * vn.GetX() + vn.GetZ() * vn
                    .GetZ()));
922                 vb.SetX(0.0);
923                 vb.SetY(vn.GetZ() / sqrt( vn.GetY() * vn.GetY() + vn.GetZ() * vn
                    .GetZ()));
924                 vb.SetZ(vn.GetY() / sqrt( vn.GetY() * vn.GetY() + vn.GetZ() * vn
                    .GetZ()));
925                 for ( j = 0; j < static_cast<int>(giftMitteM[snumber][0] + 0.5);
                     j++ ) {
926                     ia = static_cast<int>(giftMitteM[snumber][0] + 0.5) - j;
927                     for ( i = 0; i < static_cast<int>(giftMitteM[snumber][1] +
                        0.5); i++ ) {
928                         ib = static_cast<int>(giftMitteM[snumber][1] + 0.5) - i;
929                         if ( va.GetX() < 0 ) {
930                             if ( vb.GetY() < 0 ) {
931                                 vp = vm1 + ia * va + ib * vb;
932                             }
933                             else {
934                                 vp = vm1 + ia * va - ib * vb;
935                             };
936                         }
937                         else {
938                             if ( vb.GetY() < 0 ) {
939                                 vp = vm1 - ia * va + ib * vb;
940                             }
941                             else {
942                                 vp = vm1 - ia * va - ib * vb;
943                             };
944                         };
945                         if ( vp.GetIZ() >= 0 && vp.GetIZ() <= maxnumber -
                            minnumber && vp.GetIY() >= 0 && vp.GetIY() < height
                            && vp.GetIX() >= 0 && vp.GetIX() < width ) {
946                             ij = bp->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                                GetIZ());
947                             bpe2->SetPixelValue(j, i, 1, ij);
948                             ij = bpe->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                                GetIZ());
949                             bpe2->SetPixelValue(j, i, 0, ij);
950                             if ( ij == 3 ) {
951                                 gefunden[1] = true;
952                             };
```

```
953                         }
954                         else {
955                             bpe2->SetPixelValue(j, i, 1, 0 );
956                             bpe2->SetPixelValue(j, i, 0, 5 );
957                         };
958                     };
959                     for ( i = static_cast<int>(giftMitteM[snumber][1] + 0.5); i
                        < height; i++ ) {
960                     ib = i - static_cast<int>(giftMitteM[snumber][1] + 0.5);
961                     if ( va.GetX() < 0 ) {
962                         if ( vb.GetY() < 0 ) {
963                             vp = vm1 + ia * va - ib * vb;
964                         }
965                         else {
966                             vp = vm1 + ia * va + ib * vb;
967                         };
968                     }
969                     else {
970                         if ( vb.GetY() < 0 ) {
971                             vp = vm1 - ia * va - ib * vb;
972                         }
973                         else {
974                             vp = vm1 - ia * va + ib * vb;
975                         };
976                     };
977                     if ( vp.GetIZ() >= 0 && vp.GetIZ() <= maxnumber -
                        minnumber && vp.GetIY() >= 0 && vp.GetIY() < height
                        && vp.GetIX() >= 0 && vp.GetIX() < width ) {
978                         ij = bp->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                            GetIZ());
979                         bpe2->SetPixelValue(j, i, 1, ij);
980                         ij = bpe->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                            GetIZ());
981                         bpe2->SetPixelValue(j, i, 0, ij);
982                         if ( ij == 3 ) {
983                             gefunden[1] = true;
984                         };
985                     }
986                     else {
987                         bpe2->SetPixelValue(j, i, 1, 0 );
988                         bpe2->SetPixelValue(j, i, 0, 5 );
989                     };
990                 };
991             };
992             for ( j = static_cast<int>(giftMitteM[snumber][0] + 0.5); j <
                width; j++ ) {
993                 ia = j - static_cast<int>(giftMitteM[snumber][0] + 0.5);
994                 for ( i = 0; i < static_cast<int>(giftMitteM[snumber][1] +
                    0.5); i++ ) {
995                 ib = static_cast<int>(giftMitteM[snumber][1] + 0.5) - i;
996                 if ( va.GetX() < 0 ) {
997                     if ( vb.GetY() < 0 ) {
998                         vp = vm1 - ia * va + ib * vb;
999                     }
1000                    else {
```

```
1001                                vp = vm1 - ia * va - ib * vb;
1002                            };
1003                        }
1004                        else {
1005                            if ( vb.GetY() < 0 ) {
1006                                vp = vm1 + ia * va + ib * vb;
1007                            }
1008                            else {
1009                                vp = vm1 + ia * va - ib * vb;
1010                            };
1011                        };
1012                        if ( vp.GetIZ() >= 0 && vp.GetIZ() <= maxnumber -
                            minnumber && vp.GetIY() >= 0 && vp.GetIY() < height
                            && vp.GetIX() >= 0 && vp.GetIX() < width ) {
1013                            ij = bp->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                                GetIZ());
1014                            bpe2->SetPixelValue(j, i, 1, ij);
1015                            ij = bpe->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                                GetIZ());
1016                            bpe2->SetPixelValue(j, i, 0, ij);
1017                            if ( ij == 3 ) {
1018                                gefunden[1] = true;
1019                            };
1020                        }
1021                        else {
1022                            bpe2->SetPixelValue(j, i, 0, 5 );
1023                            bpe2->SetPixelValue(j, i, 1, 0 );
1024                        };
1025                    };
1026                    for ( i = static_cast<int>(giftMitteM[snumber][1] + 0.5); i
                        < height; i++ ) {
1027                        ib = i - static_cast<int>(giftMitteM[snumber][1] + 0.5);
1028                        if ( va.GetX() < 0 ) {
1029                            if ( vb.GetY() < 0 ) {
1030                                vp = vm1 - ia * va - ib * vb;
1031                            }
1032                            else {
1033                                vp = vm1 - ia * va + ib * vb;
1034                            };
1035                        }
1036                        else {
1037                            if ( vb.GetY() < 0 ) {
1038                                vp = vm1 + ia * va - ib * vb;
1039                            }
1040                            else {
1041                                vp = vm1 + ia * va + ib * vb;
1042                            };
1043                        };
1044                        if ( vp.GetIZ() >= 0 && vp.GetIZ() <= maxnumber -
                            minnumber && vp.GetIY() >= 0 && vp.GetIY() < height
                            && vp.GetIX() >= 0 && vp.GetIX() < width ) {
1045                            ij = bp->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                                GetIZ());
1046                            bpe2->SetPixelValue(j, i, 1, ij);
```

```
1047                            ij = bpe->GetPixelValue(vp.GetIX(), vp.GetIY() , vp.
                                    GetIZ());
1048                            bpe2->SetPixelValue(j, i, 0, ij);
1049                            if ( ij == 3 ) {
1050                                gefunden[1] = true;
1051                            };
1052                        }
1053                        else {
1054                            bpe2->SetPixelValue(j, i, 0, 5 );
1055                            bpe2->SetPixelValue(j, i, 1, 0 );
1056                        };
1057                    };
1058                };
1059                if ( gefunden[1] ) {
1060                    for ( i = 1; i < height - 1; i++ ) {
1061                        for ( j = 1; j < width - 1; j++ ) {
1062                            ia = bpe2->GetPixelValue(j, i, 0);
1063                            if ( ia == 2 || ia == 3 ) {
1064                                ig = 0;
1065                                ih = 0;
1066                                ii = 0;
1067                                ij = 0;
1068                                for ( ic = -1; ic < 2; ic++  ) {
1069                                    for ( id = -1; id < 2; id++ ) {
1070                                        ib = bpe2->GetPixelValue(j + id, i + ic,
                                                0);
1071                                        switch ( ib ) {
1072                                            case ( 1 ) : ig++; break;
1073                                            case ( 2 ) : ih++; break;
1074                                            case ( 3 ) : ii++; break;
1075                                            default : ij++; break;
1076                                        };
1077                                    };
1078                                };
1079                                if ( ia == 2 && ih == 1 || ia == 3 && ii == 1 )
                                        {
1080                                    if ( ij > ig ) {
1081                                        bpe2->SetPixelValue(j, i, 0, 0);
1082                                    }
1083                                    else {
1084                                        bpe2->SetPixelValue(j, i, 0, 1);
1085                                    };
1086                                };
1087                            };
1088                        };
1089                    };
1090
1091                    fprintf(mes2, "%i", number);
1092                    emptyList(randList[1]);
1093                    gefunden[0] = false;
1094
1095                    for ( i = 0; i < height; i++ ) {
1096                        for ( j = 0; j < width; j++ ) {
1097                            gefunden[0] = false;
1098                            ia = bpe2->GetPixelValue(j, i, 0);
```

```
1099                        if ( j > 0 ) {
1100                            ib = bpe2->GetPixelValue(j-1, i, 0);
1101                            if ( ia != ib ) {
1102                                if ( ia == 1 && ib == 3 ) {
1103                                    insertElement(randList[1], -1, j, i);
1104                                    gefunden[0] = true;
1105                                }
1106                                else if ( ia == 5 && ib == 3 ) {
1107                                    insertElement(randList[1], -1, j, i);
1108                                    randList[1]->last->emark = -1;
1109                                    gefunden[0] = true;
1110                                };
1111                            };
1112                        };
1113                        if ( j < width - 1 && !gefunden[0] ) {
1114                            ib = bpe2->GetPixelValue(j+1, i, 0);
1115                            if ( ia != ib ) {
1116                                if ( ia == 1 && ib == 3 ) {
1117                                    insertElement(randList[1], -1, j, i);
1118                                    gefunden[0] = true;
1119                                }
1120                                else if ( ia == 5 && ib == 3 ) {
1121                                    insertElement(randList[1], -1, j, i);
1122                                    randList[1]->last->emark = -1;
1123                                    gefunden[0] = true;
1124                                };
1125                            };
1126                        };
1127                        if ( i > 0 && !gefunden[0] ) {
1128                            ib = bpe2->GetPixelValue(j, i-1, 0);
1129                            if ( ia != ib ) {
1130                                if ( ia == 1 && ib == 3 ) {
1131                                    insertElement(randList[1], -1, j, i);
1132                                    gefunden[0] = true;
1133                                }
1134                                else if ( ia == 5 && ib == 3 ) {
1135                                    insertElement(randList[1], -1, j, i);
1136                                    randList[1]->last->emark = -1;
1137                                    gefunden[0] = true;
1138                                };
1139                            };
1140                        };
1141                        if ( i < height - 1 && !gefunden[0] ) {
1142                            ib = bpe2->GetPixelValue(j, i+1, 0);
1143                            if ( ia != ib ) {
1144                                if ( ia == 1 && ib == 3 ) {
1145                                    insertElement(randList[1], -1, j, i);
1146                                }
1147                                else if ( ia == 5 && ib == 3 ) {
1148                                    insertElement(randList[1], -1, j, i);
1149                                    randList[1]->last->emark = -1;
1150                                    gefunden[0] = true;
1151                                };
1152                            };
1153                        };
```

```
1154                        };
1155                    };
1156
1157                    id = 0;
1158                    for ( i = 1; i < height - 1; i++ ) {
1159                        for ( j = 1; j < width - 1; j++ ) {
1160                            ia = ( bpe2->GetPixelValue(j-1, i-1, 1) + bpe2->
                                    GetPixelValue(j-1, i+1, 1) - bpe2->GetPixelValue
                                    (j+1, i-1, 1) - bpe2->GetPixelValue(j+1, i+1, 1)
                                     + 2 * bpe2->GetPixelValue(j-1, i, 1) - 2 * bpe2
                                    ->GetPixelValue(j+1, i, 1) );
1161                            ib = ( -1 * bpe2->GetPixelValue(j-1, i-1, 1) + bpe2
                                    ->GetPixelValue(j-1, i+1, 1) - bpe2->
                                    GetPixelValue(j+1, i-1, 1) + bpe2->GetPixelValue
                                    (j+1, i+1, 1) - 2 * bpe2->GetPixelValue(j, i-1,
                                    1) + 2 * bpe2->GetPixelValue(j, i+1, 1) );
1162                            ftemp = sqrt(static_cast<float>(ia * ia + ib * ib));
1163                            ic = static_cast<int>(ftemp+0.5);
1164                            id = ( ic > id ? ic : id );
1165                            if ( ia != 0 ) {
1166                                ftemp = atan( static_cast<float>(ib) /
                                        static_cast<float>(ia) );
1167                                if ( ftemp <= -1.178 || ftemp >= 1.178 ) {
1168                                    bpe2->SetPixelValue(j, i, 2, 1);
1169                                }
1170                                else if ( ftemp > -0.393 && ftemp < 0.393 ) {
1171                                    bpe2->SetPixelValue(j, i, 2, 3);
1172                                }
1173                                else if ( ftemp >= 0.393 && ftemp < 1.178 ) {
1174                                    bpe2->SetPixelValue(j, i, 2, 2);
1175                                }
1176                                else if ( ftemp <= -0.393 && ftemp > -1.178 ) {
1177                                    bpe2->SetPixelValue(j, i, 2, 4);
1178                                }
1179                                else {
1180                                    cout << "  FEHLER!!" << endl;
1181                                };
1182                            }
1183                            else {
1184                                if ( ib != 0 ) {
1185                                    bpe2->SetPixelValue(j, i, 2, 1);
1186                                }
1187                                else {
1188                                    bpe2->SetPixelValue(j, i, 2, 3);
1189                                };
1190                            };
1191                        };
1192                    };
1193
1194                    sortListByCoordinates(randList[1], listTest);
1195                    randList[1]->p1 = randList[1]->first;
1196                    while ( randList[1]->p1 != 0 ) {
1197                        if ( randList[1]->p1->emark == -1 ) {
1198                            i = randList[1]->p1->number;
1199                            randList[1]->p2 = randList[1]->p1->next;
```

```
1200                            deleteElement(randList[1], i);
1201                            randList[1]->p1 = randList[1]->p2;
1202                        }
1203                        else {
1204                            randList[1]->p1 = randList[1]->p1->next;
1205                        };
1206                    };
1207                    einr = berechnungS(randList[1], *bpe2, 4, 2, 5, 3, 0, dif2,
                        mes2, einr, number);
1208                    fprintf(mes2, "%c", 9);
1209                    printFloat(vn.GetX(), mes2, 5);
1210                    fprintf(mes2, "%c", 9);
1211                    printFloat(vn.GetY(), mes2, 5);
1212                    fprintf(mes2, "%c", 9);
1213                    printFloat(vn.GetZ(), mes2, 5);
1214                    fprintf(mes2, "\n");
1215
1216                    for ( j = 0; j < width; j++ ) {
1217                        for ( i = 0; i < height; i++ ) {
1218                            bpe2->SetPixelValue(j, i, 5, bpe2->GetPixelValue(j,
                                i, 1));
1219                        };
1220                    };
1221                    randList[1]->p1 = randList[1]->first;
1222                    while ( randList[1]->p1 != 0 ) {
1223                        j = randList[1]->p1->x;
1224                        i = randList[1]->p1->y;
1225                        bpe2->SetPixelValue(j, i, 5, 255);
1226                        randList[1]->p1 = randList[1]->p1->next;
1227                    };
1228
1229                    for ( j = 0; j < numziff; j++ ) {
1230                        zahl = ( number - number / static_cast<int>(pow(10,
                            numziff-j)+.5) * static_cast<int>(pow(10,numziff-j)
                            +.5) ) / static_cast<int>(pow(10,numziff-1-j)+.5)
                            +48;
1231                        nameami2[strlen+7-numziff+j] = zahl;
1232                        nameell2[strlen+9-numziff+j] = zahl;
1233                        nameran2[strlen+6-numziff+j] = zahl;
1234                    };
1235                    ofstream a2bild (nameami2, ios::binary);
1236                    if ( a2bild.is_open() != true ) {
1237                        cout << "  Neue Bilddatei konnte nicht erstellt werden.
                            (Amira)" << endl;
1238                        fbild2.close();
1239                        return -1;
1240                    }
1241                    ofstream e2bild (nameell2, ios::binary);
1242                    if ( e2bild.is_open() != true ) {
1243                        cout << "  Neue Bilddatei konnte nicht erstellt werden.
                            (Ellipse)" << endl;
1244                        fbild2.close();
1245                        a2bild.close();
1246                        return -1;
1247                    }
```

```
1248                    ofstream r2bild (nameran2, ios::binary);
1249                    if ( r2bild.is_open() != true ) {
1250                        cout << "  Neue Bilddatei konnte nicht erstellt werden.
                                (Rand)" << endl;
1251                        fbild2.close();
1252                        a2bild.close();
1253                        e2bild.close();
1254                        return -1;
1255                    }
1256                    fbild2.seekg(0);
1257
1258                    for ( i = 0; i < start; i++ ) {
1259                        fbild2.get(c);
1260                        if ( fbild2.fail() ) {
1261                            cout << "  Lesefehler aus Bilddatei." << endl;
1262                            break;
1263                        };
1264                        a2bild.put(c);
1265                        if ( a2bild.fail() ) {
1266                            cout << "  Schreibfehler in neuer Bilddatei. (Amira)
                                    " << endl;
1267                            break;
1268                        };
1269                        e2bild.put(c);
1270                        if ( e2bild.fail() ) {
1271                            cout << "  Schreibfehler in neuer Bilddatei. (
                                    Ellipse)" << endl;
1272                            break;
1273                        };
1274                        r2bild.put(c);
1275                        if ( r2bild.fail() ) {
1276                            cout << "  Schreibfehler in neuer Bilddatei. (Rand)"
                                     << endl;
1277                            break;
1278                        };
1279                    };
1280                    for ( i = height - 1; i >= 0; i-- ) {
1281                        for ( j = 0; j < width; j++ ) {
1282                            a2bild.put(255 - (bpe2->GetPixelValue(j, i, 0)*50));
1283                            e2bild.put(bpe2->GetPixelValue(j, i, 3));
1284                            r2bild.put(bpe2->GetPixelValue(j, i, 5));
1285                        };
1286                        for ( j = 0; j < nullen; j++ ) {
1287                            a2bild.put(0);
1288                            e2bild.put(0);
1289                            r2bild.put(0);
1290                        };
1291                    };
1292                    a2bild.close();
1293                    e2bild.close();
1294                    r2bild.close();
1295                }
1296                else {
1297                    einr++;
1298                };
```

```
1299                };
1300            };
1301        };
1302
1303        fbild2.close();
1304
1305        delete(bp);
1306        delete(bp2);
1307
1308        fbild.close();
1309        fclose(diff);
1310        fclose(mess);
1311
1312        printf("%c", 7);
1313
1314        return 0;
1315    }
1316
1317    listHead* appendLists ( listHead* listHead1, listHead* listHead2, int pointer1,
        int pointer2, int pointer3 )
1318    {
1319        if ( listHead1->count == 0 ) {
1320            if ( listHead2->count == 0 ) {
1321                return 0;
1322            }
1323            else {
1324                return listHead2;
1325            };
1326        }
1327        else if ( listHead2->count == 0 ) {
1328            return listHead1;
1329        }
1330        else {
1331            listHead1->count += listHead2->count;
1332            listHead1->internp1 = listHead1->last;
1333            listHead1->last->next = listHead2->first;
1334            listHead1->last->next->prev = listHead1->last;
1335            listHead1->last = listHead2->last;
1336            switch ( pointer1 ) {
1337                case 11 : break;
1338                case 12 : listHead1->p1 = listHead1->p2; break;
1339                case 13 : listHead1->p1 = listHead1->p3; break;
1340                case 21 : listHead1->p1 = listHead2->p1; break;
1341                case 22 : listHead1->p1 = listHead2->p2; break;
1342                case 23 : listHead1->p1 = listHead2->p3; break;
1343                default : printf("  Fehler bei der Zuweisung des variablen
                        listElement-Pointers p1.\n  Es wird p1 von der ersten Liste
                        uebernommen.\n"); break;
1344            };
1345            switch ( pointer2 ) {
1346                case 11 : listHead1->p2 = listHead1->p1; break;
1347                case 12 : break;
1348                case 13 : listHead1->p2 = listHead1->p3; break;
1349                case 21 : listHead1->p2 = listHead2->p1; break;
1350                case 22 : listHead1->p2 = listHead2->p2; break;
```

```
1351                case 23 : listHead1->p2 = listHead2->p3; break;
1352                default : printf("  Fehler bei der Zuweisung des variablen
                        listElement-Pointers p2.\n  Es wird p2 von der ersten Liste
                        uebernommen.\n"); break;
1353            };
1354            switch ( pointer3 ) {
1355                case 11 : listHead1->p3 = listHead1->p1; break;
1356                case 12 : listHead1->p3 = listHead1->p2; break;
1357                case 13 : break;
1358                case 21 : listHead1->p3 = listHead2->p1; break;
1359                case 22 : listHead1->p3 = listHead2->p2; break;
1360                case 23 : listHead1->p3 = listHead2->p3; break;
1361                default : printf("  Fehler bei der Zuweisung des variablen
                        listElement-Pointers p2.\n  Es wird p2 von der ersten Liste
                        uebernommen.\n"); break;
1362            };
1363            listHead1->internp1 = listHead1->internp1->next;
1364            while ( listHead1->internp1 != 0 ) {
1365                listHead1->internp1->number = listHead1->internp1->prev->number + 1;
1366                listHead1->internp1->head = listHead1;
1367                listHead1->internp1 = listHead1->internp1->next;
1368            };
1369            delete listHead2;
1370        };
1371        return listHead1;
1372    };
1373
1374    int insertElement ( listHead* listHead1, int position, int xpos, int ypos )
1375    {
1376        if ( position > listHead1->count || listHead1 == 0 ) {
1377            printf("  insertElement: Beim Einfuegen des Elements ist ein Fehler
                    aufgetreten: ");
1378            if ( listHead1 == 0 ) {
1379                printf("  Die Liste existiert nicht.\n");
1380            }
1381            else {
1382                printf("  Die vorgesehene Position (%i) ist zu gross. Maximal
                        moeglich ist %i.\n", position, listHead1->count - 1);
1383            };
1384            return -1;
1385        }
1386        else {
1387            listElement* listElement1 = new listElement;
1388            if ( listHead1->count == 0 ) {
1389                listHead1->first = listElement1;
1390                listHead1->last = listElement1;
1391                listElement1->prev = 0;
1392                listElement1->next = 0;
1393            }
1394            else if ( position == listHead1->count || position < 0 ) {
1395                listHead1->last->next = listElement1;
1396                listElement1->prev = listHead1->last;
1397                listHead1->last = listElement1;
1398                listElement1->next = 0;
1399            }
```

```
1400          else {
1401              listHead1->internp1 = listHead1->first;
1402              if ( position == 0 ) {
1403                  listHead1->first->prev = listElement1;
1404                  listElement1->next = listHead1->first;
1405                  listHead1->first = listElement1;
1406                  listElement1->prev = 0;
1407              }
1408              else {
1409                  while ( listHead1->internp1->number != position ) {
1410                      listHead1->internp1 = listHead1->internp1->next;
1411                  };
1412                  listElement1->next = listHead1->internp1;
1413                  listElement1->prev = listHead1->internp1->prev;
1414                  listHead1->internp1->prev->next = listElement1;
1415                  listHead1->internp1->prev = listElement1;
1416              };
1417              while ( listHead1->internp1 != 0 ) {
1418                  listHead1->internp1->number += 1;
1419                  listHead1->internp1 = listHead1->internp1->next;
1420              };
1421          };
1422          listElement1->head = listHead1;
1423          listElement1->x = xpos;
1424          listElement1->y = ypos;
1425          listElement1->number = ( position < 0 ? listHead1->count : position );
1426          listElement1->emark = 0;
1427          listElement1->emark2 = 0;
1428          listHead1->count += 1;
1429          return 0;
1430      };
1431  };
1432
1433  int deleteElement ( listHead* listHead1, int position )
1434  {
1435      if ( listHead1 == 0 ) {
1436          printf("  deleteList: Ein Fehler ist beim Loeschen eines Elementes
                  aufgetreten:\n  Die Liste existiert nicht.\n");
1437          return -1;
1438      }
1439      else if ( listHead1->count == 0 ) {
1440          printf("  deleteList: Ein Fehler ist beim Loeschen eines Elementes
                  aufgetreten:\n  Die Liste enthaelt keine Elemente.\n");
1441          return -1;
1442      }
1443      else if ( listHead1->count <= position ) {
1444          printf("  deleteList: Ein Fehler ist beim Loeschen eines Elementes
                  aufgetreten:\n  Die angegebene Position (%i) ist zu gross. maximal
                  moeglich ist %i.\n", position, listHead1->count - 1);
1445          return -1;
1446      }
1447      else {
1448          if ( listHead1->p1 != 0 ) {
1449              if ( position == -1 ) {
1450                  if ( listHead1->p1 == listHead1->last ) {
```

```
1451                    listHead1->p1 = 0;
1452                };
1453            }
1454            else if ( listHead1->p1->number == position ) {
1455                if ( listHead1->p1 == listHead1->last || listHead1->count == 1 )
                         {
1456                    listHead1->p1 = 0;
1457                }
1458                else {
1459                    listHead1->p1 = listHead1->p1->next;
1460                };
1461            };
1462        };
1463        if ( listHead1->p2 != 0 ) {
1464            if ( position == -1 ) {
1465                if ( listHead1->p2 == listHead1->last ) {
1466                    listHead1->p2 = 0;
1467                };
1468            }
1469            else if ( listHead1->p2->number == position ) {
1470                if ( listHead1->p2 == listHead1->last || listHead1->count == 1 )
                         {
1471                    listHead1->p2 = 0;
1472                }
1473                else {
1474                    listHead1->p2 = listHead1->p2->next;
1475                };
1476            };
1477        };
1478        if ( listHead1->p3 != 0 ) {
1479            if ( position == -1 ) {
1480                if ( listHead1->p3 == listHead1->last ) {
1481                    listHead1->p3 = 0;
1482                };
1483            }
1484            else if ( listHead1->p3->number == position ) {
1485                if ( listHead1->p3 == listHead1->last || listHead1->count == 1 )
                         {
1486                    listHead1->p3 = 0;
1487                }
1488                else {
1489                    listHead1->p3 = listHead1->p3->next;
1490                };
1491            };
1492        };
1493        if ( listHead1->count == 1 ) {
1494            delete listHead1->first;
1495            listHead1->first = 0;
1496            listHead1->last = 0;
1497            listHead1->internp1 = 0;
1498            listHead1->internp2 = 0;
1499            listHead1->p1 = 0;
1500            listHead1->p2 = 0;
1501            listHead1->p3 = 0;
1502        }
```

```
1503          else if ( listHead1->count == position + 1 || position < 0 ) {
1504              listHead1->internp1 = listHead1->last;
1505              listHead1->last->prev->next = 0;
1506              listHead1->last = listHead1->last->prev;
1507              delete listHead1->internp1;
1508              listHead1->internp1 = 0;
1509              listHead1->internp2 = 0;
1510          }
1511          else {
1512              listHead1->internp1 = listHead1->first;
1513              if ( position == 0 ) {
1514                  listHead1->first = listHead1->first->next;
1515                  listHead1->first->prev = 0;
1516                  delete listHead1->internp1;
1517                  listHead1->internp1 = listHead1->first;
1518              }
1519              else {
1520                  while ( listHead1->internp1->number != position ) {
1521                      listHead1->internp1 = listHead1->internp1->next;
1522                  };
1523                  listHead1->internp1->prev->next = listHead1->internp1->next;
1524                  listHead1->internp1->next->prev = listHead1->internp1->prev;
1525                  listHead1->internp2 = listHead1->internp1;
1526                  listHead1->internp1 = listHead1->internp2->next;
1527                  delete listHead1->internp2;
1528                  listHead1->internp2 = 0;
1529              };
1530              while ( listHead1->internp1 != 0 ) {
1531                  listHead1->internp1->number -= 1;
1532                  listHead1->internp1 = listHead1->internp1->next;
1533              };
1534          };
1535          listHead1->count -= 1;
1536          return 0;
1537      };
1538  };
1539
1540  listHead* createList ()
1541  {
1542      listHead* listHead1 = new listHead;
1543      listHead1->count = 0;
1544      listHead1->number = 0;
1545      listHead1->hmark = 0;
1546      listHead1->hmark2 = 0;
1547      listHead1->first = 0;
1548      listHead1->last = 0;
1549      listHead1->prev = 0;
1550      listHead1->next = 0;
1551      listHead1->internp1 = 0;
1552      listHead1->internp2 = 0;
1553      listHead1->p1 = 0;
1554      listHead1->p2 = 0;
1555      listHead1->p3 = 0;
1556      return listHead1;
1557  };
```

```
1558
1559 void deleteList ( listHead* &listHead1 )
1560 {
1561     if ( listHead1 != 0 ) {
1562         if ( listHead1->count != 0 ) {
1563             listHead1->internp1 = listHead1->last->prev;
1564             while ( listHead1->internp1 != 0 ) {
1565                 delete listHead1->internp1->next;
1566                 listHead1->internp1->next = 0;
1567                 listHead1->internp1 = listHead1->internp1->prev;
1568             };
1569             delete listHead1->first;
1570             listHead1->first = 0;
1571         };
1572         listHead1->last = 0;
1573         listHead1->prev = 0;
1574         listHead1->next = 0;
1575         delete listHead1;
1576         listHead1 = 0;
1577     };
1578 };
1579
1580 void emptyList ( listHead* &listHead1 )
1581 {
1582     if ( listHead1 != 0 ) {
1583         while ( listHead1->count > 0 ) {
1584             deleteElement(listHead1, -1);
1585         };
1586     };
1587     return;
1588 };
1589
1590 void setListPointer ( listElement* &pointer, listHead* &lhead, int num )
1591 {
1592     if ( pointer == 0 ) {
1593         pointer = lhead->first;
1594     };
1595     if ( num >= lhead->count || lhead->count == 0 ) {
1596         return;
1597     }
1598     else {
1599         if ( pointer->number == num ) {
1600             return;
1601         }
1602         else if ( num < 0 ) {
1603             pointer = lhead->last;
1604         }
1605         else {
1606             while ( pointer->number < num ) {
1607                 pointer = pointer->next;
1608             };
1609             while ( pointer->number > num ) {
1610                 pointer = pointer->prev;
1611             };
1612         };
```

```
1613      };
1614  };
1615
1616  void reverseList ( listHead* &listHead1 )
1617  {
1618      if ( listHead1 == 0 ) {
1619          cout << "  Die Liste konnte nicht umgedreht werden, da sie nicht
                  existiert." << endl;
1620          return;
1621      };
1622      if ( listHead1->count < 2 ) {
1623          if ( listHead1->count == 0 ) {
1624              cout << "  Die Liste konnte nicht umgedreht werden, da sie leer ist.
                      " << endl;
1625          };
1626          return;
1627      }
1628      else {
1629          listHead1->internp1 = listHead1->last;
1630          listHead1->last = listHead1->first;
1631          listHead1->first = listHead1->internp1;
1632          if ( listHead1->count == 2 ) {
1633              listHead1->first->number = 0;
1634              listHead1->last->number = 1;
1635              listHead1->first->next = listHead1->last;
1636              listHead1->first->prev = 0;
1637              listHead1->last->next = 0;
1638              listHead1->last->prev = listHead1->first;
1639          }
1640          else {
1641              listHead1->internp1 = listHead1->first->prev;
1642              listHead1->internp2 = listHead1->internp1;
1643              listHead1->first->next = listHead1->first->prev;
1644              listHead1->first->prev = 0;
1645              listHead1->first->number = 0;
1646              while ( listHead1->internp2 != listHead1->last ) {
1647                  listHead1->internp2 = listHead1->internp2->prev;
1648                  listHead1->internp1->prev = listHead1->internp1->next;
1649                  listHead1->internp1->next = listHead1->internp2;
1650                  listHead1->internp1->number = listHead1->count - 1 - listHead1->
                          internp1->number;
1651                  listHead1->internp1 = listHead1->internp2;
1652              };
1653              listHead1->last->prev = listHead1->last->next;
1654              listHead1->last->next = 0;
1655              listHead1->last->number = listHead1->count -1;
1656          };
1657      };
1658      return;
1659  };
1660
1661  listHead* copyList ( listHead* oldList )
1662  {
1663      listHead* newList = createList();
1664      if ( oldList != 0 ) {
```

```
1665          oldList->internp1 = oldList->first;
1666          while ( oldList->internp1 != 0 ) {
1667              insertElement(newList, -1, oldList->internp1->x, oldList->internp1->
                      y);
1668              newList->last->emark = oldList->internp1->emark;
1669              newList->last->emark2 = oldList->internp1->emark2;
1670              oldList->internp1 = oldList->internp1->next;
1671          };
1672      };
1673      if ( oldList->p1 != 0 ) {
1674          newList->p1 = newList->first;
1675          while ( newList->p1->number != oldList->p1->number ) {
1676              newList->p1 = newList->p1->next;
1677          };
1678      };
1679      if ( oldList->p2 != 0 ) {
1680          newList->p2 = newList->first;
1681          while ( newList->p2->number != oldList->p2->number ) {
1682              newList->p2 = newList->p2->next;
1683          };
1684      };
1685      if ( oldList->p3 != 0 ) {
1686          newList->p3 = newList->first;
1687          while ( newList->p3->number != oldList->p3->number ) {
1688              newList->p3 = newList->p3->next;
1689          };
1690      };
1691      newList->hmark = oldList->hmark;
1692      newList->hmark2 = oldList->hmark2;
1693      return newList;
1694  };
1695
1696  void copyList ( listHead* sourceList, listHead* &targetList )
1697  {
1698      emptyList(targetList);
1699      if ( sourceList != 0 ) {
1700          sourceList->internp1 = sourceList->first;
1701          while ( sourceList->internp1 != 0 ) {
1702              insertElement(targetList, -1, sourceList->internp1->x, sourceList->
                      internp1->y);
1703              targetList->last->emark = sourceList->internp1->emark;
1704              targetList->last->emark2 = sourceList->internp1->emark2;
1705              sourceList->internp1 = sourceList->internp1->next;
1706          };
1707      };
1708      if ( sourceList->p1 != 0 ) {
1709          targetList->p1 = targetList->first;
1710          while ( targetList->p1->number != sourceList->p1->number ) {
1711              targetList->p1 = targetList->p1->next;
1712          };
1713      };
1714      if ( sourceList->p2 != 0 ) {
1715          targetList->p2 = targetList->first;
1716          while ( targetList->p2->number != sourceList->p2->number ) {
1717              targetList->p2 = targetList->p2->next;
```

```
1718            };
1719        };
1720        if ( sourceList->p3 != 0 ) {
1721            targetList->p3 = targetList->first;
1722            while ( targetList->p3->number != sourceList->p3->number ) {
1723                targetList->p3 = targetList->p3->next;
1724            };
1725        };
1726        targetList->hmark = sourceList->hmark;
1727        targetList->hmark2 = sourceList->hmark2;
1728    };
1729
1730    bool compareLists ( listHead* listHead1, listHead* listHead2 )
1731    {
1732        if ( listHead1 == 0 || listHead2 == 0 ) {
1733            cout << "  Mindestens eine der beiden Listen existiert nicht." << endl;
1734            return false;
1735        }
1736        else {
1737            if ( listHead1 == listHead2 ) {
1738                return true;
1739            }
1740            else {
1741                return false;
1742            };
1743        };
1744    };
1745
1746    listElement* teileSortList ( listHead* listHead1, listElement* links,
        listElement* rechts, int was )
1747    {
1748        listElement* elinks = links;
1749        listElement* erechts = rechts->prev;
1750        int pivot = 0;
1751        int temp = 0;
1752        int i = 0;
1753        int j = 0;
1754        switch ( was ) {
1755            case 1 : pivot = rechts->emark; break;
1756            case 2 : pivot = rechts->x; break;
1757            case 3 : pivot = rechts->y; break;
1758            default : cout << "  Fehler beim Teilen der Liste fuer Quicksort.
                Fehlerhafte Angabe von 'was'." << endl; break;
1759        };
1760        do {
1761            switch ( was ) {
1762                case 1 :
1763                    while ( elinks->emark >= pivot && elinks->number < rechts->
                        number ) {
1764                        elinks = elinks->next;
1765                    };
1766                    while ( erechts->emark <= pivot && erechts->number > links->
                        number ) {
1767                        erechts = erechts->prev;
1768                    };
```

```
1769                break;
1770            case 2 :
1771                while ( elinks->x >= pivot && elinks->number < rechts->number )
                        {
1772                    elinks = elinks->next;
1773                };
1774                while ( erechts->x <= pivot && erechts->number > links->number )
                         {
1775                    erechts = erechts->prev;
1776                };
1777                break;
1778            case 3 :
1779                while ( elinks->y >= pivot && elinks->number < rechts->number )
                        {
1780                    elinks = elinks->next;
1781                };
1782                while ( erechts->y <= pivot && erechts->number > links->number )
                         {
1783                    erechts = erechts->prev;
1784                };
1785                break;
1786            default : cout << "  Fehler beim Teilen der Liste fuer Quicksort.
                    Fehlerhafte Angabe von 'was'." << endl; break;
1787        };
1788        if ( elinks->number < erechts->number ) {
1789            temp = elinks->emark;
1790            elinks->emark = erechts->emark;
1791            erechts->emark = temp;
1792            temp = elinks->emark2;
1793            elinks->emark2 = erechts->emark2;
1794            erechts->emark2 = temp;
1795            temp = elinks->x;
1796            elinks->x = erechts->x;
1797            erechts->x = temp;
1798            temp = elinks->y;
1799            elinks->y = erechts->y;
1800            erechts->y = temp;
1801
1802            elinks = elinks->next;
1803            erechts = erechts->prev;
1804        };
1805    }
1806    while ( elinks->number < erechts->number );
1807    switch ( was ) {
1808        case 1 : temp = elinks->emark; break;
1809        case 2 : temp = elinks->x; break;
1810        case 3 : temp = elinks->y; break;
1811        default : cout << "  Fehler beim Teilen der Liste fuer Quicksort.
                Fehlerhafte Angabe von 'was'." << endl; break;
1812    };
1813    if ( temp < pivot ) {
1814        temp = elinks->emark;
1815        elinks->emark = rechts->emark;
1816        rechts->emark = temp;
1817        temp = elinks->emark2;
```

```
1818          elinks->emark2 = rechts->emark2;
1819          rechts->emark2 = temp;
1820          temp = elinks->x;
1821          elinks->x = rechts->x;
1822          rechts->x = temp;
1823          temp = elinks->y;
1824          elinks->y = rechts->y;
1825          rechts->y = temp;
1826      };
1827      return elinks;
1828  };
1829
1830  void quickSortList ( listHead* listHead1, listElement* links, listElement*
          rechts, int was )
1831  {
1832      listElement* teiler = 0;
1833      if ( links != 0 && rechts != 0 ) {
1834          if ( links->number < rechts->number ) {
1835              teiler = teileSortList(listHead1,links, rechts, was);
1836              quickSortList(listHead1, links, teiler->prev, was);
1837              quickSortList(listHead1, teiler->next, rechts, was);
1838          };
1839      };
1840  };
1841
1842  int writeList( listHead* listHead1, FILE* datei )
1843  {
1844      if ( listHead1 == 0 ) {
1845          fprintf(datei, "Die Liste konnte nicht geschrieben werden, da sie nicht
                  existiert.\n\n");
1846          return -1;
1847      }
1848      else {
1849          fprintf(datei, "Listenkopf:\n  Anzahl Elemente: %i\n  Listenmarkierung:
                  %i\n  Listenmarkierung 2: %i\n\n", listHead1->count, listHead1->
                  hmark, listHead1->hmark2 );
1850          if ( listHead1->count != 0 ) {
1851              listHead1->internp1 = listHead1->first;
1852              while ( listHead1->internp1 != 0 ) {
1853                  fprintf(datei, "  Elementnummer: %i\n  x-Position: %i\n  y-
                          Position: %i\n  Elementmarkierung %i:\n\n", listHead1->
                          internp1->number, listHead1->internp1->x, listHead1->
                          internp1->y, listHead1->internp1->emark );
1854                  listHead1->internp1 = listHead1->internp1->next;
1855              };
1856          };
1857          return 0;
1858      };
1859  };
1860
1861  Bild::Bild ( int width, int height, int layers )
1862  :   m_width(width), m_height(height), m_layers(layers)
1863  {
1864      m_pixel = new int**[width];
1865      for ( int i = 0; i < width; i++ ) {
```

```
1866          m_pixel[i] = new int*[height];
1867          for ( int j = 0; j < height; j++ ) {
1868              m_pixel[i][j] = new int[layers];
1869              for ( int k = 0; k < layers; k++ ) {
1870                  m_pixel[i][j][k] = 0;
1871              };
1872          };
1873      };
1874  };
1875
1876  Bild::~Bild ()
1877  {
1878      if ( m_pixel != 0 ) {
1879          for ( int i = 0; i < m_width; i++ ) {
1880              for ( int j = 0; j < m_height; j++ ) {
1881                  delete[] m_pixel[i][j];
1882              };
1883          };
1884          for ( int i = 0; i < m_width; i++ ) {
1885              delete[] m_pixel[i];
1886          };
1887          delete[] m_pixel;
1888          m_pixel = 0;
1889      };
1890  };
1891
1892  Bild::Bild ( const Bild &bp )
1893  {
1894      printf("  Copy-Ctor\n");
1895      m_width = bp.m_width;
1896      m_height = bp.m_height;
1897      m_layers = bp.m_layers;
1898      m_pixel = new int**[m_width];
1899      for ( int i = 0; i < m_width; i++ ) {
1900          m_pixel[i] = new int*[m_height];
1901          for ( int j = 0; j < m_height; j++ ) {
1902              m_pixel[i][j] = new int[m_layers];
1903              for ( int k = 0; k < m_layers; k++ ) {
1904                  m_pixel[i][j][k] = bp.m_pixel[i][j][k];
1905              };
1906          };
1907      };
1908  };
1909
1910  void Bild::operator = ( const Bild &bp )
1911  {
1912      if ( &bp == this ) {
1913          printf("  this\n");
1914          return;
1915      };
1916      if ( m_width == bp.m_width && m_height == bp.m_height && m_layers == bp.
          m_layers ) {
1917          for ( int i = 0; i < m_width; i++ ) {
1918              for ( int j = 0; j < m_height; j++ ) {
1919                  for ( int k = 0; k < m_layers; k++ ) {
```

```
1920                        m_pixel[i][j][k] = bp.m_pixel[i][j][k];
1921                    };
1922                };
1923            };
1924        }
1925        else {
1926            printf("  Fehler beim Anwenden des operators =, da die Arrays nicht
                gleichgross sind.\n");
1927        };
1928    };
1929
1930    int Bild::GetPixelValue ( int xpos, int ypos, int layer )
1931    {
1932        if ( xpos < 0 || ypos < 0 || layer < 0 || xpos > m_width - 1 || ypos >
                m_height - 1 || layer > m_layers - 1 ) {
1933            throw 1;
1934        }
1935        else {
1936            return m_pixel[xpos][ypos][layer];
1937        };
1938    };
1939
1940    void Bild::SetPixelValue ( int xpos, int ypos, int layer, int value )
1941    {
1942        m_pixel[xpos][ypos][layer] = value;
1943    };
1944
1945    int Bild::GetWidth ( )
1946    {
1947        return m_width;
1948    };
1949
1950    int Bild::GetHeight ( )
1951    {
1952        return m_height;
1953    };
1954
1955    int Bild::GetLayers ( )
1956    {
1957        return m_layers;
1958    };
1959
1960    ListList::ListList ()
1961    {
1962        m_count = 0;
1963        m_first = 0;
1964        m_last = 0;
1965        m_internp1 = 0;
1966        m_internp2 = 0;
1967    };
1968
1969    ListList::~ListList ()
1970    {
1971        while ( m_count > 0 ) {
1972            m_internp1 = m_first->next;
```

```
1973          deleteList(m_first);
1974          m_first = m_internp1;
1975          m_count--;
1976      };
1977  };
1978
1979  ListList::ListList (const ListList &ll )
1980  {
1981      listHead* lh = ll.m_first;
1982      while ( lh != 0 ) {
1983          InsertList(copyList(lh), -1);
1984          lh = lh->next;
1985      };
1986  };
1987
1988  listHead* ListList::operator[] ( int index )
1989  {
1990      if ( index >= m_count ) {
1991          cout << "  ListList::[]: Der vorgegebene Index (" << index << ") ist zu
                gross. Maximal moeglich ist " << (m_count - 1) << "." << endl;
1992          return 0;
1993      }
1994      else if ( index < 0 ) {
1995          return m_last;
1996      }
1997      else {
1998          m_internp1 = m_first;
1999          while ( m_internp1->number < index ) {
2000              if ( m_internp1->next != 0 ) {
2001                  m_internp1 = m_internp1->next;
2002              }
2003              else {
2004                  cout << "  ListList::[]: Die Listenliste hat einen Fehler." << "
                            Obwohl die Anzahl der Listen gross genug sein sollte
                        wurde keine Liste mit passender Nummer gefunden." << endl;
2005              };
2006          };
2007          if ( m_internp1->number > index ) {
2008              cout << "  ListList::[]: Die Listenliste hat einen Fehler. Es kommt
                    keine Liste mit der vorgegebenen Nummer vor." << endl;
2009              return 0;
2010          }
2011          else {
2012              return m_internp1;
2013          };
2014      };
2015  };
2016
2017  listHead* ListList::GetList ( int index )
2018  {
2019      if ( index >= m_count ) {
2020          cout << "  ListList::GetList: Der vorgegebene Index (" << index << ")
                ist zu gross. Maximal moeglich ist " << (m_count - 1) << "." << endl
                ;
2021          return 0;
```

```
2022      }
2023      else if ( index < 0 ) {
2024          return m_last;
2025      }
2026      else {
2027          m_internp1 = m_first;
2028          while ( m_internp1->number < index ) {
2029              if ( m_internp1->next != 0 ) {
2030                  m_internp1 = m_internp1->next;
2031              }
2032              else {
2033                  cout << "  ListList::GetList: Die Listenliste hat einen Fehler."
                             << "   Obwohl die Anzahl der Listen gross genug sein sollte
                             wurde keine Liste mit passender Nummer gefunden." << endl;
2034              };
2035          };
2036          if ( m_internp1->number > index ) {
2037              cout << "  ListList::GetList: Die Listenliste hat einen Fehler. Es
                         kommt keine Liste mit der vorgegebenen Nummer vor." << endl;
2038              return 0;
2039          }
2040          else {
2041              return m_internp1;
2042          };
2043      };
2044  };
2045
2046  int ListList::DeleteList ( int position )
2047  {
2048      if ( m_count == 0 ) {
2049          printf("  ListList::DeleteList: Ein Fehler ist beim Loeschen einer Liste
                       aufgetreten:\n  Die Listenliste enthaelt keine Listen.\n");
2050          return -1;
2051      }
2052      else if ( m_count <= position ) {
2053          printf("  ListList::DeleteList: Ein Fehler ist beim Loeschen einer Liste
                       aufgetreten:\n  Die angegebene Position (%i) ist zu gross. Maximal
                      moeglich ist %i.\n", position, m_count - 1);
2054          return -1;
2055      }
2056      else {
2057          if ( m_count == 1 ) {
2058              deleteList(m_first);
2059              m_first = 0;
2060              m_last = 0;
2061              m_internp1 = 0;
2062          }
2063          else if ( m_count == position + 1 || position < 0 ) {
2064              m_internp1 = m_last;
2065              m_last->prev->next = 0;
2066              m_last = m_last->prev;
2067              deleteList(m_internp1);
2068              m_internp1 = 0;
2069          }
2070          else {
```

```
2071                m_internp1 = m_first;
2072                if ( position == 0 ) {
2073                    m_first = m_first->next;
2074                    m_first->prev = 0;
2075                    deleteList(m_internp1);
2076                    m_internp1 = m_first;
2077                }
2078                else {
2079                    while ( m_internp1->number != position ) {
2080                        m_internp1 = m_internp1->next;
2081                    };
2082                    m_internp1->prev->next = m_internp1->next;
2083                    m_internp1->next->prev = m_internp1->prev;
2084                    m_internp2 = m_internp1;
2085                    m_internp1 = m_internp2->next;
2086                    deleteList(m_internp2);
2087                    m_internp2 = 0;
2088                };
2089                while ( m_internp1 != 0 ) {
2090                    m_internp1->number -= 1;
2091                    m_internp1 = m_internp1->next;
2092                };
2093            };
2094            m_count -= 1;
2095            return 0;
2096        };
2097    };
2098
2099    int ListList::DeleteList ( listHead* lh )
2100    {
2101        if ( m_count == 1 ) {
2102            deleteList(lh);
2103            m_first = 0;
2104            m_last = 0;
2105            m_internp1 = 0;
2106        }
2107        else if ( compareLists(lh, m_last) ) {
2108            m_internp1 = m_last;
2109            m_last->prev->next = 0;
2110            m_last = m_last->prev;
2111            deleteList(m_internp1);
2112            m_internp1 = 0;
2113        }
2114        else {
2115            m_internp1 = m_first;
2116            if ( compareLists(lh, m_first) ) {
2117                m_first = m_first->next;
2118                m_first->prev = 0;
2119                deleteList(m_internp1);
2120                m_internp1 = m_first;
2121            }
2122            else {
2123                lh->prev->next = lh->next;
2124                lh->next->prev = lh->prev;
2125                m_internp1 = lh->next;
```

```
2126                deleteList(lh);
2127            };
2128            while ( m_internp1 != 0 ) {
2129                m_internp1->number -= 1;
2130                m_internp1 = m_internp1->next;
2131            };
2132        };
2133        m_count -= 1;
2134        return 0;
2135    };
2136
2137    int ListList::InsertList ( listHead* lh, int position )
2138    {
2139        if ( position > m_count ) {
2140            cout << "  ListList::InsertList: Beim Einfuegen des Elements ist ein
                        Fehler aufgetreten: Die vorgesehene Position (" << position << ")
                        ist zu gross. Maximal moeglich ist " << m_count << "." << endl;
2141            return -1;
2142        }
2143        else {
2144            if ( m_count == 0 ) {
2145                m_first = lh;
2146                m_last = lh;
2147                lh->prev = 0;
2148                lh->next = 0;
2149            }
2150            else if ( position == m_count || position < 0 ) {
2151                m_last->next = lh;
2152                lh->prev = m_last;
2153                m_last = lh;
2154                lh->next = 0;
2155            }
2156            else {
2157                m_internp1 = m_first;
2158                if ( position == 0 ) {
2159                    m_first->prev = lh;
2160                    lh->next = m_first;
2161                    m_first = lh;
2162                    lh->prev = 0;
2163                }
2164                else {
2165                    while ( m_internp1->number != position ) {
2166                        m_internp1 = m_internp1->next;
2167                    };
2168                    lh->next = m_internp1;
2169                    lh->prev = m_internp1->prev;
2170                    m_internp1->prev->next = lh;
2171                    m_internp1->prev = lh;
2172                };
2173                while ( m_internp1 != 0 ) {
2174                    m_internp1->number += 1;
2175                    m_internp1 = m_internp1->next;
2176                };
2177            };
2178            lh->number = ( position < 0 ? m_count : position );
```

```
2179            m_count += 1;
2180            return 0;
2181        };
2182    };
2183
2184    int ListList::GetCount ()
2185    {
2186        return m_count;
2187    };
2188
2189    void ListList::SetCount ( int value )
2190    {
2191        m_count = value;
2192        return;
2193    };
2194
2195    void ListList::IncCount ()
2196    {
2197        m_count++;
2198        return;
2199    };
2200
2201    void ListList::DecCount ()
2202    {
2203        m_count--;
2204        return;
2205    };
2206
2207    Vec2D::Vec2D ( )
2208    {
2209        m_x = 0.0;
2210        m_y = 0.0;
2211        m_ix = 0;
2212        m_iy = 0;
2213        m_l = 0.0;
2214        m_w = 0.0;
2215    };
2216
2217    Vec3D::Vec3D ( )
2218    {
2219        m_x = 0.0;
2220        m_y = 0.0;
2221        m_z = 0.0;
2222        m_ix = 0;
2223        m_iy = 0;
2224        m_iz = 0;
2225        m_l = 0.0;
2226    };
2227
2228    Vec2D::Vec2D ( int ix, int iy )
2229    {
2230        m_ix = ix;
2231        m_iy = iy;
2232        m_x = static_cast<float>( ix );
2233        m_y = static_cast<float>( iy );
```

```
2234      m_l = CompL();
2235      m_w = CompW();
2236  };
2237
2238  Vec3D::Vec3D ( int ix, int iy, int iz )
2239  {
2240      m_ix = ix;
2241      m_iy = iy;
2242      m_iz = iz;
2243      m_x = static_cast<float>( ix );
2244      m_y = static_cast<float>( iy );
2245      m_z = static_cast<float>( iz );
2246      m_l = CompL();
2247  };
2248
2249  Vec2D::Vec2D ( float xl, float yw, char art )
2250  {
2251      if ( art == 'l' || art == 'w' ) {
2252          m_l = xl;
2253          m_w = yw;
2254          float fx = cos(yw) * xl;
2255          m_x = fx;
2256          m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2257          fx = sin(yw) * xl;
2258          m_y = fx;
2259          m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2260      }
2261      else if ( art == 'x' || art == 'y' ) {
2262          m_x = xl;
2263          m_y = yw;
2264          m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2265          m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2266          m_l = CompL();
2267          m_w = CompW();
2268      }
2269      else {
2270          cout << "  FEHLER BEIM ERSTELLEN DES VEKTORS!!!" << endl;
2271      };
2272  };
2273
2274  Vec3D::Vec3D ( float x, float y, float z )
2275  {
2276      m_x = x;
2277      m_y = y;
2278      m_z = z;
2279      m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2280      m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2281      m_iz = static_cast<int>( m_z < 0 ? ( m_z - 0.5 ) : ( m_z + 0.5 ) );
2282      m_l = CompL();
2283  };
2284
2285  Vec2D::Vec2D ( double xl, double yw, char art )
2286  {
2287      if ( art == 'l' || art == 'w' ) {
2288          m_l = xl;
```

```
2289            m_w = yw;
2290            float fx = cos(yw) * xl;
2291            m_x = fx;
2292            m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2293            fx = sin(yw) * xl;
2294            m_y = fx;
2295            m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2296        }
2297        else if ( art == 'x' || art == 'y' ) {
2298            m_x = xl;
2299            m_y = yw;
2300            m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2301            m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2302            m_l = CompL();
2303            m_w = CompW();
2304        }
2305        else {
2306            cout << "  FEHLER BEIM ERSTELLEN DES VEKTORS!!!" << endl;
2307        };
2308    };
2309
2310    Vec3D::Vec3D ( double x, double y, double z )
2311    {
2312        m_x = x;
2313        m_y = y;
2314        m_z = z;
2315        m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2316        m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2317        m_iz = static_cast<int>( m_z < 0 ? ( m_z - 0.5 ) : ( m_z + 0.5 ) );
2318        m_l = CompL();
2319    };
2320
2321    Vec2D Vec2D::operator+ ( Vec2D v1 )
2322    {
2323        Vec2D tv;
2324        tv.m_x = m_x + v1.m_x;
2325        tv.m_y = m_y + v1.m_y;
2326        tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
                );
2327        tv.m_iy = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
                );
2328        tv.m_l = tv.CompL();
2329        tv.m_w = tv.CompW();
2330        return tv;
2331    };
2332
2333    Vec3D Vec3D::operator+ ( Vec3D v1 )
2334    {
2335        Vec3D tv;
2336        tv.m_x = m_x + v1.m_x;
2337        tv.m_y = m_y + v1.m_y;
2338        tv.m_z = m_z + v1.m_z;
2339        tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
                );
```

```
2340      tv.m_iy = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2341      tv.m_iz = static_cast<int>( tv.m_z < 0 ? ( tv.m_z - 0.5 ) : ( tv.m_z + 0.5 )
              );
2342      tv.m_l = tv.CompL();
2343      return tv;
2344  };
2345
2346  Vec2D Vec2D::operator- ( Vec2D v1 )
2347  {
2348      Vec2D tv;
2349      tv.m_x = m_x - v1.m_x;
2350      tv.m_y = m_y - v1.m_y;
2351      tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
              );
2352      tv.m_iy = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2353      tv.m_l = tv.CompL();
2354      tv.m_w = tv.CompW();
2355      return tv;
2356  };
2357
2358  Vec3D Vec3D::operator- ( Vec3D v1 )
2359  {
2360      Vec3D tv;
2361      tv.m_x = m_x - v1.m_x;
2362      tv.m_y = m_y - v1.m_y;
2363      tv.m_z = m_z - v1.m_z;
2364      tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
              );
2365      tv.m_iy = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2366      tv.m_iz = static_cast<int>( tv.m_z < 0 ? ( tv.m_z - 0.5 ) : ( tv.m_z + 0.5 )
              );
2367      tv.m_l = tv.CompL();
2368      return tv;
2369  };
2370
2371  Vec2D Vec2D::operator/ ( float sk )
2372  {
2373      Vec2D tv;
2374      tv.m_l = m_l / sk;
2375      float fx = cos(m_w) * tv.m_l;
2376      tv.m_x = fx;
2377      tv.m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2378      fx = sin(m_w) * tv.m_l;
2379      tv.m_y = fx;
2380      tv.m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2381      tv.m_l = tv.CompL();
2382      tv.m_w = tv.CompW();
2383      return tv;
2384  };
2385
2386  Vec3D Vec3D::operator/ ( float sk )
2387  {
```

```
2388      Vec3D tv;
2389      tv.m_x = m_x / sk;
2390      tv.m_y = m_y / sk;
2391      tv.m_z = m_z / sk;
2392      tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
              );
2393      tv.m_iy = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2394      tv.m_iz = static_cast<int>( tv.m_z < 0 ? ( tv.m_z - 0.5 ) : ( tv.m_z + 0.5 )
              );
2395      tv.m_l = tv.CompL();
2396      return tv;
2397  };
2398
2399  void Vec2D::operator= ( Vec2D v1 )
2400  {
2401      m_x = v1.m_x;
2402      m_y = v1.m_y;
2403      m_ix = v1.m_ix;
2404      m_iy = v1.m_iy;
2405      m_l = v1.m_l;
2406      m_w = v1.m_w;
2407  };
2408
2409  void Vec3D::operator= ( Vec3D v1 )
2410  {
2411      m_x = v1.m_x;
2412      m_y = v1.m_y;
2413      m_z = v1.m_z;
2414      m_ix = v1.m_ix;
2415      m_iy = v1.m_iy;
2416      m_iz = v1.m_iz;
2417      m_l = v1.m_l;
2418  };
2419
2420  Vec2D operator* ( float sk, Vec2D v1 )
2421  {
2422      Vec2D tv;
2423      float fx = sk * v1.m_x;
2424      tv.m_x = fx;
2425      tv.m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2426      fx = sk * v1.m_y;
2427      tv.m_y = fx;
2428      tv.m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2429      tv.m_l = tv.CompL();
2430      tv.m_w = tv.CompW();
2431      return tv;
2432  };
2433
2434  Vec3D operator* ( float sk, Vec3D v1 )
2435  {
2436      Vec3D tv;
2437      float fx = sk * v1.m_x;
2438      tv.m_x = fx;
2439      fx = sk * v1.m_y;
```

```
2440      tv.m_y = fx;
2441      fx = sk * v1.m_z;
2442      tv.m_z = fx;
2443      tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
              );
2444      tv.m_iy = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2445      tv.m_iz = static_cast<int>( tv.m_z < 0 ? ( tv.m_z - 0.5 ) : ( tv.m_z + 0.5 )
              );
2446      tv.m_l = tv.CompL();
2447      return tv;
2448  };
2449
2450  Vec2D operator* ( Vec2D v1, float sk )
2451  {
2452      Vec2D tv;
2453      float fx = sk * v1.m_x;
2454      tv.m_x = fx;
2455      tv.m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2456      fx = sk * v1.m_y;
2457      tv.m_y = fx;
2458      tv.m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2459      tv.m_l = tv.CompL();
2460      tv.m_w = tv.CompW();
2461      return tv;
2462  };
2463
2464  Vec3D operator* ( Vec3D v1, float sk )
2465  {
2466      Vec3D tv;
2467      float fx = sk * v1.m_x;
2468      tv.m_x = fx;
2469      fx = sk * v1.m_y;
2470      tv.m_y = fx;
2471      fx = sk * v1.m_z;
2472      tv.m_z = fx;
2473      tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
              );
2474      tv.m_ix = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2475      tv.m_ix = static_cast<int>( tv.m_z < 0 ? ( tv.m_z - 0.5 ) : ( tv.m_z + 0.5 )
              );
2476      tv.m_l = tv.CompL();
2477      return tv;
2478  };
2479
2480  float Vec2D::dotp ( Vec2D v1 )
2481  {
2482      float fx = 0.0;
2483      fx = m_x * v1.m_x + m_y * v1.m_y;
2484      return fx;
2485  };
2486
2487  float Vec3D::dotp ( Vec3D v1 )
2488  {
```

```
2489      float fx = 0.0;
2490      fx = m_x * v1.m_x + m_y * v1.m_y + m_z * v1.m_z;
2491      return fx;
2492  };
2493
2494  Vec3D Vec3D::crossp ( Vec3D v1 )
2495  {
2496      Vec3D tv;
2497      tv.m_x = m_y * v1.m_z - m_z * v1.m_y;
2498      tv.m_y = m_z * v1.m_x - m_x * v1.m_z;
2499      tv.m_z = m_x * v1.m_y - m_y * v1.m_x;
2500      tv.m_ix = static_cast<int>( tv.m_x < 0 ? ( tv.m_x - 0.5 ) : ( tv.m_x + 0.5 )
              );
2501      tv.m_ix = static_cast<int>( tv.m_y < 0 ? ( tv.m_y - 0.5 ) : ( tv.m_y + 0.5 )
              );
2502      tv.m_ix = static_cast<int>( tv.m_z < 0 ? ( tv.m_z - 0.5 ) : ( tv.m_z + 0.5 )
              );
2503      return tv;
2504  };
2505
2506  int Vec2D::GetIX ()
2507  {
2508      return m_ix;
2509  };
2510
2511  int Vec3D::GetIX ()
2512  {
2513      return m_ix;
2514  };
2515
2516  int Vec2D::GetIY ()
2517  {
2518      return m_iy;
2519  };
2520
2521  int Vec3D::GetIY ()
2522  {
2523      return m_iy;
2524  };
2525
2526  int Vec3D::GetIZ ()
2527  {
2528      return m_iz;
2529  };
2530
2531  float Vec2D::GetX ()
2532  {
2533      return m_x;
2534  };
2535
2536  float Vec2D::GetY ()
2537  {
2538      return m_y;
2539  };
2540
```

```
2541  float Vec3D::GetX ()
2542  {
2543      return m_x;
2544  };
2545
2546  float Vec3D::GetY ()
2547  {
2548      return m_y;
2549  };
2550
2551  float Vec3D::GetZ ()
2552  {
2553      return m_z;
2554  };
2555
2556  float Vec2D::GetL ()
2557  {
2558      return m_l;
2559  };
2560
2561  float Vec3D::GetL ()
2562  {
2563      return m_l;
2564  };
2565
2566  float Vec2D::GetW ()
2567  {
2568      return m_w;
2569  };
2570
2571  float Vec2D::CompL ()
2572  {
2573      float l = 0.0;
2574      l = sqrt ( m_x * m_x + m_y * m_y );
2575      return l;
2576  };
2577
2578  float Vec3D::CompL ()
2579  {
2580      float l = 0.0;
2581      l = sqrt ( m_x * m_x + m_y * m_y + m_z * m_z );
2582      return l;
2583  };
2584
2585  float Vec2D::CompW ()
2586  {
2587      float a = 0.0;
2588      if ( m_x == 0 ) {
2589          if ( m_y < 0 ) {
2590              a = -PI / 2.0;
2591          }
2592          else {
2593              a = PI / 2.0;
2594          };
2595      }
```

```
2596        else {
2597            a = atan( m_y / m_x );
2598            if ( m_x < 0 ) {
2599                if ( m_y < 0 ) {
2600                    a -= PI;
2601                }
2602                else {
2603                    a += PI;
2604                };
2605            };
2606        };
2607        return a;
2608    };
2609
2610    void Vec2D::SetIX ( int ix )
2611    {
2612        m_ix = ix;
2613        m_x = static_cast<float>(m_ix);
2614        m_l = CompL();
2615        m_w = CompW();
2616    };
2617
2618    void Vec3D::SetIX ( int ix )
2619    {
2620        m_ix = ix;
2621        m_x = static_cast<float>(m_ix);
2622        m_l = CompL();
2623    };
2624
2625    void Vec2D::SetIY ( int iy )
2626    {
2627        m_iy = iy;
2628        m_y = static_cast<float>(m_iy);
2629        m_l = CompL();
2630        m_w = CompW();
2631    };
2632
2633    void Vec3D::SetIY ( int iy )
2634    {
2635        m_iy = iy;
2636        m_y = static_cast<float>(m_iy);
2637        m_l = CompL();
2638    };
2639
2640    void Vec3D::SetIZ ( int iz )
2641    {
2642        m_iz = iz;
2643        m_z = static_cast<float>(m_iz);
2644        m_l = CompL();
2645    };
2646
2647    void Vec2D::SetIXIY ( int ix, int iy )
2648    {
2649        m_ix = ix;
2650        m_iy = iy;
```

```
2651      m_x = static_cast<float>(m_ix);
2652      m_y = static_cast<float>(m_iy);
2653      m_l = CompL();
2654      m_w = CompW();
2655  };
2656
2657  void Vec3D::SetIXIYIZ ( int ix, int iy, int iz )
2658  {
2659      m_ix = ix;
2660      m_iy = iy;
2661      m_iz = iz;
2662      m_x = static_cast<float>(m_ix);
2663      m_y = static_cast<float>(m_iy);
2664      m_z = static_cast<float>(m_iz);
2665      m_l = CompL();
2666  };
2667
2668  void Vec2D::SetX ( float x )
2669  {
2670      m_x = x;
2671      m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2672      m_l = CompL();
2673      m_w = CompW();
2674  };
2675
2676  void Vec3D::SetX ( float x )
2677  {
2678      m_x = x;
2679      m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2680      m_l = CompL();
2681  };
2682
2683  void Vec2D::SetY ( float y )
2684  {
2685      m_y = y;
2686      m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2687      m_l = CompL();
2688      m_w = CompW();
2689  };
2690
2691  void Vec3D::SetY ( float y )
2692  {
2693      m_y = y;
2694      m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2695      m_l = CompL();
2696  };
2697
2698  void Vec3D::SetZ ( float z )
2699  {
2700      m_z = z;
2701      m_iz = static_cast<int>( m_z < 0 ? ( m_z - 0.5 ) : ( m_z + 0.5 ) );
2702      m_l = CompL();
2703  };
2704
2705  void Vec2D::SetXY ( float x, float y )
```

```
2706 {
2707     m_x = x;
2708     m_y = y;
2709     m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2710     m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2711     m_l = CompL();
2712     m_w = CompW();
2713 };
2714
2715 void Vec3D::SetXYZ ( float x, float y, float z )
2716 {
2717     m_x = x;
2718     m_y = y;
2719     m_z = z;
2720     m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2721     m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2722     m_iz = static_cast<int>( m_z < 0 ? ( m_z - 0.5 ) : ( m_z + 0.5 ) );
2723     m_l = CompL();
2724 };
2725
2726 void Vec2D::SetL ( float l )
2727 {
2728     m_l = l;
2729     float fx = cos(m_w) * l;
2730     m_x = fx;
2731     m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2732     fx = sin(m_w) * l;
2733     m_y = fx;
2734     m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2735 };
2736
2737 void Vec3D::SetL ( float l )
2738 {
2739     float fx = l / m_l;
2740     m_l = l;
2741     m_x = fx * m_x;
2742     m_ix = static_cast<int>( m_x < 0 ? ( m_x - 0.5 ) : ( m_x + 0.5 ) );
2743     m_y = fx * m_y;
2744     m_iy = static_cast<int>( m_y < 0 ? ( m_y - 0.5 ) : ( m_y + 0.5 ) );
2745     m_z = fx * m_z;
2746     m_iz = static_cast<int>( m_z < 0 ? ( m_z - 0.5 ) : ( m_z + 0.5 ) );
2747     l = CompL();
2748 };
2749
2750 void Vec2D::SetW ( float w )
2751 {
2752     m_w = w;
2753     float fx = cos(w) * m_l;
2754     m_x = fx;
2755     m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2756     fx = sin(w) * m_l;
2757     m_y = fx;
2758     m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2759 };
2760
```

```
2761 void Vec2D::SetLW ( float l, float w )
2762 {
2763     m_l = l;
2764     m_w = w;
2765     float fx = cos(w) * l;
2766     m_x = fx;
2767     m_ix = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2768     fx = sin(w) * l;
2769     m_y = fx;
2770     m_iy = static_cast<int>( fx < 0 ? ( fx - 0.5 ) : ( fx + 0.5 ) );
2771 };
2772
2773 int readfour ( ifstream& datei, int anfang )
2774 {
2775     char swidth [5];
2776     int ia = 0;
2777     int ib = 0;
2778     int ic = 0;
2779     int id = 0;
2780
2781     datei.seekg(anfang);
2782     for ( int i = 0; i < 4; i++ ) {
2783         datei.get(swidth[i]);
2784     };
2785     ia = static_cast<int>(swidth[0]);
2786     ia = ( ia < 0 ? ia+256 : ia );
2787     ib = static_cast<int>(swidth[1]);
2788     ib = ( ib < 0 ? ib+256 : ib );
2789     ic = static_cast<int>(swidth[2]);
2790     ic = ( ic < 0 ? ic+256 : ic );
2791     id = static_cast<int>(swidth[3]);
2792     id = ( id < 0 ? id+256 : id );
2793     return (ia + ib * 256 + ic * 65536 + id * 16777216);
2794 };
2795
2796 int readnumber ( ifstream& datei, int anfang, int anzahl )
2797 {
2798     char swidth [anzahl + 1];
2799     int ia = 0;
2800     int ib = 0;
2801     int ic = 0;
2802     int id = 0;
2803
2804     datei.seekg(anfang);
2805     for ( int i = 0; i < anzahl; i++ ) {
2806         datei.get(swidth[i]);
2807         ia = static_cast<int>(swidth[i]);
2808         ia = ( ia < 0 ? ia+256 : ia );
2809         ib += ia * ( 1 << (i*8) );
2810     };
2811     return ib;
2812 };
2813
2814 int teile38 ( int links, int rechts, int liste[3][8], int pos )
2815 {
```

```
2816      int i = links;
2817      int j = rechts - 1;
2818      int pivot = liste[pos][rechts];
2819      int temp = 0;
2820      int x = 0;
2821      do {
2822          while ( liste[pos][i] >= pivot && i < rechts ) {
2823              i++;
2824          };
2825          while ( liste[pos][j] <= pivot && j > links ) {
2826              j--;
2827          };
2828          if ( i < j ) {
2829              for ( x = 0; x < 3; x++ ) {
2830                  temp = liste[x][i];
2831                  liste[x][i] = liste[x][j];
2832                  liste[x][j] = temp;
2833              };
2834          };
2835      }
2836      while ( i < j );
2837      if ( liste [pos][i] < pivot ) {
2838          for ( x = 0; x < 3; x++ ) {
2839              temp = liste[x][i];
2840              liste[x][i] = liste[x][rechts];
2841              liste[x][rechts] = temp;
2842          };
2843      };
2844      return i;
2845  };
2846
2847  void quicksort38 ( int links, int rechts, int liste[3][8], int pos )
2848  {
2849      int teiler = 0;
2850      if ( links < rechts ) {
2851          teiler = teile38 ( links, rechts, liste, pos );
2852          quicksort38 ( links, teiler-1, liste, pos );
2853          quicksort38 ( teiler+1, rechts, liste, pos );
2854      };
2855  };
2856
2857  int teile68 ( int links, int rechts, int liste[6][8], int pos )
2858  {
2859      int i = links;
2860      int j = rechts - 1;
2861      int pivot = liste[pos][rechts];
2862      int temp = 0;
2863      int x = 0;
2864      do {
2865          while ( liste[pos][i] >= pivot && i < rechts ) {
2866              i++;
2867          };
2868          while ( liste[pos][j] <= pivot && j > links ) {
2869              j--;
2870          };
```

```
2871          if ( i < j ) {
2872              for ( x = 0; x < 6; x++ ) {
2873                  temp = liste[x][i];
2874                  liste[x][i] = liste[x][j];
2875                  liste[x][j] = temp;
2876              };
2877          };
2878      }
2879      while ( i < j );
2880      if ( liste [pos][i] < pivot ) {
2881          for ( x = 0; x < 6; x++ ) {
2882              temp = liste[x][i];
2883              liste[x][i] = liste[x][rechts];
2884              liste[x][rechts] = temp;
2885          };
2886      };
2887      return i;
2888  };
2889
2890  void quicksort68 ( int links, int rechts, int liste[6][8], int pos )
2891  {
2892      int teiler = 0;
2893      if ( links < rechts ) {
2894          teiler = teile68 ( links, rechts, liste, pos );
2895          quicksort68 ( links, teiler-1, liste, pos );
2896          quicksort68 ( teiler+1, rechts, liste, pos );
2897      };
2898  };
2899
2900  int berechnungS ( listHead* liPos, Bild &bp, int l1, int l2, int l3, int l4, int
          l5, FILE* datei1, FILE* datei2, int einrueck, int number )
2901  {
2902      cout << "  Start berechnungS" << endl;
2903      float a = 0.0;
2904      float cxy = 0;
2905      float ftemp = 0.0;
2906      float ftemp2 = 0.0;
2907      float ftemp3 = 0.0;
2908      float laengeA1 = 0.0;
2909      float laengeA2 = 0.0;
2910      float laengeA3 = 0.0;
2911      float laengeA4 = 0.0;
2912      float maxDurchmesser = 0.0;
2913      float minDurchmesser = sqrt(bp.GetHeight() * bp.GetHeight() + bp.GetWidth()
             * bp.GetWidth());
2914      float sx = 0;
2915      float sy = 0;
2916      float winkelL = 0.0;
2917      float winkelB = 0.0;
2918      float winkelT1 = 0.0;
2919      float winkelT2 = 0.0;
2920      float winkelT3 = 0.0;
2921      float winkelT4 = 0.0;
2922
2923      int dx, dy = 0;
```

```
2924        int e1x, e1y, e2x, e2y, mx, my = 0;
2925        int itemp = 0;
2926        int i = 0;
2927        int j = 0;
2928        int x1, x2, y1, y2 = 0;
2929        int zaehler = 0;
2930        Vec2D vp (0, 0);
2931        Vec2D v1 (0, 0);
2932        Vec2D v2 (0, 0);
2933
2934        liPos->p1 = liPos->first;
2935        while ( liPos->p1 != 0 ) {
2936            liPos->p2 = liPos->first;
2937            ftemp2 = 0;
2938            while ( liPos->p2 != 0 ) {
2939                ftemp = sqrt( (liPos->p1->x - liPos->p2->x) * (liPos->p1->x - liPos
                        ->p2->x) + (liPos->p1->y - liPos->p2->y) * (liPos->p1->y - liPos
                        ->p2->y) );
2940                if ( ftemp > maxDurchmesser ) {
2941                    maxDurchmesser = ftemp;
2942                    v1.SetIXIY(liPos->p1->x - liPos->p2->x, liPos->p1->y - liPos->p2
                            ->y);
2943                };
2944                if ( ftemp > ftemp2 ) {
2945                    ftemp2 = ftemp;
2946                };
2947                liPos->p2 = liPos->p2->next;
2948            };
2949            if ( ftemp2 < minDurchmesser ) {
2950                minDurchmesser = ftemp2;
2951            };
2952            if ( liPos->p1->emark != -1 ) {
2953                sx += liPos->p1->x;
2954                sy += liPos->p1->y;
2955                cxy++;
2956            };
2957            liPos->p1 = liPos->p1->next;
2958        };
2959        fprintf(datei2, "%c", 9);
2960        printFloat(maxDurchmesser, datei2, 5.0);
2961        fprintf(datei2, "%c", 9);
2962        printFloat(minDurchmesser, datei2, 5.0);
2963        fprintf(datei2, "%c%i", 9, liPos->count);
2964
2965        if ( cxy > 1.0 ) {
2966            sx = sx / cxy;
2967            sy = sy / cxy;
2968
2969            Vec2D vm (sx, sy, 'x');
2970
2971            bool gefunden = false;
2972            bool gefunden2 = false;
2973            bool gefunden3 = false;
2974
2975            int u = 0;
```

```
2976            int v = 0;
2977            int w = 0;
2978
2979            for ( i = 0; i < bp.GetHeight(); i++ ) {
2980                for ( j = 0; j < bp.GetWidth(); j++ ) {
2981                    bp.SetPixelValue(j, i, l3, 0);
2982                };
2983            };
2984            liPos->p1 = liPos->first;
2985            while ( liPos->p1 != 0 ) {
2986                bp.SetPixelValue(liPos->p1->x, liPos->p1->y, l3, 1);
2987                liPos->p1 = liPos->p1->next;
2988            };
2989
2990            if ( bp.GetPixelValue(vm.GetIX(), vm.GetIY(), l3) != 1 ) {
2991                x2 = 11;
2992                y2 = 11;
2993                itemp = 0;
2994                for ( i = 0; i < bp.GetHeight(); i++ ) {
2995                    for ( j = 0; j < bp.GetWidth(); j++ ) {
2996                        w = flaechen(bp, j, i, x2, l3, 0, 0, l5, 3, 0, 0, 0, 2);
2997                        if ( w > 0 ) {
2998                            if ( w > itemp ) {
2999                                itemp = w;
3000                                y2 = x2;
3001                            };
3002                            x2++;
3003                        };
3004                    };
3005                };
3006                sx = 0.0;
3007                sy = 0.0;
3008                cxy = 0.0;
3009                for ( i = 0; i < bp.GetHeight(); i++ ) {
3010                    for ( j = 0; j < bp.GetWidth(); j++ ) {
3011                        x2 = bp.GetPixelValue(j, i, l3);
3012                        if ( x2 > 10 ) {
3013                            if ( x2 == y2 ) {
3014                                bp.SetPixelValue(j, i, l3, 3);
3015                                sx += static_cast<float>(j);
3016                                sy += static_cast<float>(i);
3017                                cxy += 1.0;
3018                            }
3019                            else {
3020                                bp.SetPixelValue(j, i, l3, 0);
3021                            };
3022                        };
3023                    };
3024                };
3025                fprintf(datei2, "%c%i", 9, itemp);
3026                vm.SetXY(sx / cxy, sy / cxy);
3027            }
3028            else {
3029                fprintf(datei2, "%c%i", 9, static_cast<int>(cxy));
3030            };
```

```
3031
3032        float winkelT = 0.0;
3033
3034        float fu = 0.0;
3035        float ftemp4 = (1<<30);
3036        float ftemp5 = 0;
3037        float ftemp6 = 0;
3038        float ftemp7 = 0;
3039
3040        Vec2D vt (0, 0);
3041        Vec2D vf1 (0, 0);
3042        Vec2D vf2 (0, 0);
3043        Vec2D va (0, 0);
3044        Vec2D ves (0, 0);
3045        Vec2D vas (0, 0);
3046        Vec2D vw (0, 0);
3047        Vec2D ve (0, 0);
3048
3049        ftemp2 = (1<<30);
3050
3051        va.SetL(maxDurchmesser/2.0);
3052        va.SetW(v1.GetW());
3053        float lb = minDurchmesser / 2.0;
3054
3055        float pcount[2][5] = {{0.0, 0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0,
                0.0}};
3056
3057        float para[5] = {vm.GetX(), vm.GetY(), va.GetL(), lb, va.GetW()};
3058
3059        float startWerte[2] = { 0.0, 0.0};
3060
3061        ftemp2 = ellDiff(bp, liPos, para, 2, startWerte, l2, true);
3062        pcount[0][0] = startWerte[1];
3063
3064        float lri[5] = {1.0, 1.0, 1.0, 1.0, 1.0};
3065        bool better[5] = {true, true, true, true, true};
3066        bool changeri[5] = {true, true, true, true, true};
3067        float step[5][5] = {{1.0, 0.0, 0.0, 0.0, 0.0}, {0.0, 1.0, 0.0, 0.0,
                0.0}, {0.0, 0.0, 1.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 1.0, 0.0}, {0.0,
                0.0, 0.0, 0.0, PI / 72.0}};
3068        float diff[2][5] = {{99999999.0, 99999999.0, 99999999.0, 99999999.0,
                99999999.0}, {0.0, 0.0, 0.0, 0.0, 0.0}};
3069
3070        ftemp = -1.0;
3071
3072        startWerte[0] = ftemp2;
3073        startWerte[1] = pcount[0][0];
3074
3075        for ( i = 2; i < 25; i++ ) {
3076            ftemp = ellDiff(bp, liPos, para, (i%5), startWerte, l2, false);
3077            startWerte[0] = ftemp;
3078        };
3079
3080        ftemp2 = ellDiff(bp, liPos, para, 2, startWerte, l2, true);
3081
```

```
3082            vm.SetXY(para[0], para[1]);
3083            if ( para[2] > para[3] ) {
3084                va.SetL(para[2]);
3085                lb = para[3];
3086                va.SetW(para[4]);
3087            }
3088            else {
3089                va.SetL(para[3]);
3090                lb = para[2];
3091                va.SetW(para[4]+PI/2.0);
3092            };
3093
3094            fprintf(datei2, "%c", 9);
3095            printFloat(va.GetW(), datei2, 5.0);
3096
3097            fprintf(datei2, "%c", 9);
3098            ftemp = PI * va.GetL() * lb;
3099            printFloat(ftemp, datei2, 5.0);
3100
3101            fprintf(datei2, "%c", 9);
3102            ftemp = sqrt( va.GetL() * va.GetL() - lb * lb ) / va.GetL();
3103            printFloat(ftemp, datei2, 5.0);
3104
3105            fprintf(datei2, "%c", 9);
3106            printFloat(ftemp2, datei2, 5.0);
3107
3108            for ( i = 0; i < bp.GetHeight(); i++ ) {
3109                for ( j = 0; j < bp.GetWidth(); j++ ) {
3110                    bp.SetPixelValue(j, i, l4, bp.GetPixelValue(j, i, 1));
3111                };
3112            };
3113            for ( i = 0; i < 720; i++ ) {
3114                ftemp2 = static_cast<float>(i) * PI / 360.0;
3115                ftemp3 = para[0] + para[2] * cos(i) * cos(para[4]) - para[3] * sin(i
                        ) * sin(para[4]);
3116                ftemp4 = para[1] + para[2] * cos(i) * sin(para[4]) + para[3] * sin(i
                        ) * cos(para[4]);
3117                u = static_cast<int>(ftemp3 + 0.5);
3118                v = static_cast<int>(ftemp4 + 0.5);
3119                if ( u >= 0 && u < bp.GetWidth() && v >= 0 && v < bp.GetHeight() ) {
3120                    bp.SetPixelValue(u, v, l4, 255);
3121                };
3122            };
3123
3124            vas.SetLW(va.GetL(), va.GetW());
3125            ves.SetLW(sqrt(va.GetL() * va.GetL() - lb * lb), va.GetW());
3126            vf1 = vm + ves;
3127            vf2 = vm - ves;
3128
3129            i = 0;
3130            gefunden = false;
3131            gefunden2 = false;
3132            ftemp = 0.0;
3133
3134            j = liPos->count;
```

```
3135
3136            float diffsA[j][3];
3137            float wiffsA[j];
3138            j = 0;
3139            ftemp3 = 0.0;
3140            ftemp5 = 0.0;
3141
3142            liPos->p1 = liPos->first;
3143            while ( liPos->p1 != 0 ) {
3144                vt.SetIXIY(liPos->p1->x, liPos->p1->y);
3145                v1 = vt - vm;
3146                ftemp  = vm.GetX() + vas.GetL() * cos(v1.GetW() - vas.GetW()) * cos(
                        vas.GetW()) - lb * sin(v1.GetW() - vas.GetW()) * sin(vas.GetW())
                        ;
3147                ftemp2 = vm.GetY() + vas.GetL() * cos(v1.GetW() - vas.GetW()) * sin(
                        vas.GetW()) + lb * sin(v1.GetW() - vas.GetW()) * cos(vas.GetW())
                        ;
3148                vw.SetXY(ftemp, ftemp2);
3149                v2 = vw - vm;
3150                v2.SetW(v1.GetW());
3151                vp = v2 - v1;
3152                fprintf(datei1, "%i%c%i%c", number, 9, zaehler, 9);
3153                zaehler++;
3154                if ( v1.GetL() > v2.GetL() ) {
3155                    if ( vp.GetW() < 0 ) {
3156                        ftemp = vp.GetW() + PI;
3157                    }
3158                    else {
3159                        ftemp = vp.GetW() - PI;
3160                    };
3161                }
3162                else {
3163                    ftemp = vp.GetW();
3164                };
3165                wiffsA[j] = ftemp;
3166                printFloat(ftemp, datei1, 5.0);
3167                fprintf(datei1, "%c", 9);
3168                if ( v1.GetL() > v2.GetL() ) {
3169                    ftemp = -1.0 * vp.GetL();
3170                }
3171                else {
3172                    ftemp = vp.GetL();
3173                };
3174                ftemp3 += fabs(ftemp);
3175                ftemp5 += ftemp;
3176                printFloat(ftemp, datei1, 5.0);
3177                for ( i = 0; i < einrueck; i++ ) {
3178                    fprintf(datei1, "%c", 9);
3179                };
3180                printFloat(ftemp, datei1, 5.0);
3181                fprintf(datei1, "\n");
3182                diffsA[j][0] = ftemp;
3183                j++;
3184                liPos->p1 = liPos->p1->next;
3185            };
```

```
3186
3187            float diff11[11];
3188            itemp = liPos->count;
3189            for ( i = 0; i < 5; i++ ) {
3190                diff11[4-i] = diffsA[itemp-i][0];
3191            };
3192            for ( i = 0; i < 5; i++ ) {
3193                diff11[i+5] = diffsA[i][0];
3194            };
3195            j = 10;
3196            for ( i = 0; i < itemp - 5; i++ ) {
3197                diff11[j%11] = diffsA[i+5][0];
3198                diffsA[i][1] = ( diff11[0] + diff11[1] + diff11[2] + diff11[3] +
                        diff11[4] + diff11[5] + diff11[6] + diff11[7] + diff11[8] +
                        diff11[9] + diff11[10] ) / 11.0;
3199                diffsA[i][2] = 0.0;
3200                j++;
3201            };
3202            while ( i < itemp ) {
3203                diff11[j%11] = diffsA[i + 5 - itemp][0];
3204                diffsA[i][1] = ( diff11[0] + diff11[1] + diff11[2] + diff11[3] +
                        diff11[4] + diff11[5] + diff11[6] + diff11[7] + diff11[8] +
                        diff11[9] + diff11[10] ) / 11.0;
3205                diffsA[i][2] = 0.0;
3206                i++;
3207                j++;
3208            };
3209
3210            j = 0;
3211            ftemp = diffsA[0][1];
3212            for ( i = 1; i < itemp; i++ ) {
3213                if ( diffsA[i][1] > ftemp ) {
3214                    ftemp = diffsA[i][1];
3215                    j = i;
3216                };
3217            };
3218            diffsA[j][2] = 1.0;
3219            i = j + 1;
3220            while ( diffsA[i%itemp][1] < diffsA[(i-1)%itemp][1] && (i%itemp) != j )
                    {
3221                diffsA[i%itemp][2] = 1.0;
3222                i++;
3223            };
3224            i = j - 1;
3225            while ( diffsA[i%itemp][1] < diffsA[(i+1)%itemp][1] && (i%itemp) != j )
                    {
3226                diffsA[i%itemp][2] = 1.0;
3227                i--;
3228            };
3229
3230            i = 0;
3231            while ( diffsA[i][2] > 0.5 && i < itemp ) {
3232                i++;
3233            };
3234            if ( i < itemp ) {
```

```
3235            j = i;
3236            ftemp = diffsA[i][1];
3237        };
3238        for ( i = j; i < itemp; i++ ) {
3239            if ( diffsA[i][1] > ftemp && diffsA[i][2] < 0.5 ) {
3240                ftemp = diffsA[i][1];
3241                j = i;
3242            };
3243        };
3244        diffsA[j][2] = 2.0;
3245        i = itemp + j + 1;
3246        while ( diffsA[i%itemp][1] < diffsA[(i-1)%itemp][1] && (i%itemp) != j )
                {
3247            diffsA[i%itemp][2] = 2.0;
3248            i++;
3249        };
3250        i = itemp + j - 1;
3251        while ( diffsA[i%itemp][1] < diffsA[(i+1)%itemp][1] && (i%itemp) != j )
                {
3252            diffsA[i%itemp][2] = 2.0;
3253            i--;
3254        };
3255        i = 0;
3256        while ( diffsA[i][2] > 0.5 && i < itemp ) {
3257            i++;
3258        };
3259        if ( i < itemp ) {
3260            j = i;
3261            ftemp = diffsA[i][1];
3262        };
3263        for ( i = j; i < itemp; i++ ) {
3264            if ( diffsA[i][1] > ftemp && diffsA[i][2] < 0.5 ) {
3265                ftemp = diffsA[i][1];
3266                j = i;
3267            };
3268        };
3269        diffsA[j][2] = 3.0;
3270        i =  itemp + j + 1;
3271        while ( diffsA[i%itemp][1] < diffsA[(i-1)%itemp][1] && (i%itemp) != j )
                {
3272            diffsA[i%itemp][2] = 3.0;
3273            i++;
3274        };
3275        i = itemp + j - 1;
3276        while ( diffsA[i%itemp][1] < diffsA[(i+1)%itemp][1] && (i%itemp) != j )
                {
3277            diffsA[i%itemp][2] = 3.0;
3278            i--;
3279        };
3280
3281        u = 0;
3282        ftemp = -100.0;
3283        for ( i = 0; i < itemp; i++ ) {
3284            if ( diffsA[i][0] > ftemp && fabs( diffsA[i][2] - 1.0 ) < 0.1 ) {
3285                ftemp = diffsA[i][0];
```

```
3286                    u = i;
3287                };
3288            };
3289
3290            v = 0;
3291            ftemp = -100.0;
3292            for ( i = 0; i < itemp; i++ ) {
3293                if ( diffsA[i][0] > ftemp && fabs( diffsA[i][2] - 2.0 ) < 0.1 ) {
3294                    ftemp = diffsA[i][0];
3295                    v = i;
3296                };
3297            };
3298
3299            int u3 = 0;
3300            ftemp = -100.0;
3301            for ( i = 0; i < itemp; i++ ) {
3302                if ( diffsA[i][0] > ftemp && fabs( diffsA[i][2] - 3.0 ) < 0.1 ) {
3303                    ftemp = diffsA[i][0];
3304                    u3 = i;
3305                };
3306            };
3307
3308            j = abs( u - v );
3309            int i2 = 0;
3310            if ( j < itemp / 2 ) {
3311                i2 = ( u < v ? u : v );
3312            }
3313            else {
3314                i2 = ( u < v ? v : u );
3315            };
3316            w = i2;
3317            ftemp = diffsA[u][0];
3318            for ( i = itemp; i < j + itemp + 1; i++ ) {
3319                if ( diffsA[(i2+i)%itemp][0] < ftemp ) {
3320                    ftemp = diffsA[(i2+i)%itemp][0];
3321                    w = ( ( i2 + i ) % itemp );
3322                };
3323            };
3324
3325            i2 = 0;
3326            ftemp = diffsA[0][0];
3327            for ( i = 1; i < itemp; i++ ) {
3328                if ( diffsA[i][0] < ftemp ) {
3329                    ftemp = diffsA[i][0];
3330                    i2 = i;
3331                };
3332            };
3333
3334            fprintf(datei2, "%c", 9);
3335            printFloat(wiffsA[u], datei2, 5.0);
3336            fprintf(datei2, "%c", 9);
3337            printFloat(wiffsA[v], datei2, 5.0);
3338            fprintf(datei2, "%c", 9);
3339            printFloat(wiffsA[u3], datei2, 5.0);
3340            fprintf(datei2, "%c", 9);
```

```
3341            printFloat(diffsA[u][0], datei2, 5.0);
3342            fprintf(datei2, "%c", 9);
3343            printFloat(diffsA[v][0], datei2, 5.0);
3344            fprintf(datei2, "%c", 9);
3345            printFloat(diffsA[u3][0], datei2, 5.0);
3346            fprintf(datei2, "%c", 9);
3347            printFloat(wiffsA[w], datei2, 5.0);
3348            fprintf(datei2, "%c", 9);
3349            printFloat(diffsA[w][0], datei2, 5.0);
3350            fprintf(datei2, "%c", 9);
3351            printFloat(wiffsA[i2], datei2, 5.0);
3352            fprintf(datei2, "%c", 9);
3353            printFloat(diffsA[i2][0], datei2, 5.0);
3354            fprintf(datei2, "%c", 9);
3355            ftemp4 = ftemp3 / static_cast<float>(itemp);
3356            printFloat(ftemp4, datei2, 5.0);
3357            fprintf(datei2, "%c", 9);
3358            ftemp3 = 0.0;
3359            for ( i = 0; i < itemp; i++ ) {
3360                ftemp3 += ( ( fabs(diffsA[i][0]) - ftemp4 ) * ( fabs(diffsA[i][0]) -
                        ftemp4 ) );
3361            };
3362            ftemp4 = ftemp3 / ( static_cast<float>(itemp) - 1);
3363            ftemp3 = sqrt(ftemp4);
3364            printFloat(ftemp3, datei2, 5.0);
3365            fprintf(datei2, "%c", 9);
3366            ftemp4 = ftemp5 / static_cast<float>(itemp);
3367            printFloat(ftemp4, datei2, 5.0);
3368            fprintf(datei2, "%c", 9);
3369            ftemp3 = 0.0;
3370            for ( i = 0; i < itemp; i++ ) {
3371                ftemp3 += ( ( diffsA[i][0] - ftemp4 ) * ( diffsA[i][0] - ftemp4 ) );
3372            };
3373            ftemp4 = ftemp3 / ( static_cast<float>(itemp) - 1);
3374            ftemp3 = sqrt(ftemp4);
3375            printFloat(ftemp3, datei2, 5.0);
3376        }
3377        else {
3378            fprintf(datei1, "%i", 0);
3379            for ( i = 0; i < einrueck; i++ ) {
3380                fprintf(datei1, "%c", 9);
3381            };
3382            printFloat(0.0, datei1, 5.0);
3383
3384            fprintf(datei2, "%c%i", 9, 1);
3385            fprintf(datei2, "%c", 9);
3386            printFloat(0.0, datei2, 5.0);
3387            fprintf(datei2, "%c", 9);
3388            ftemp = PI * 0.25;
3389            printFloat(ftemp, datei2, 5.0);
3390            fprintf(datei2, "%c", 9);
3391            printFloat(0.0, datei2, 5.0);
3392            fprintf(datei2, "%c", 9);
3393            printFloat(0.0, datei2, 5.0);
3394            fprintf(datei2, "%c", 9);
```

```
3395            printFloat(3.2, datei2, 5.0);
3396            fprintf(datei2, "%c", 9);
3397            printFloat(3.3, datei2, 5.0);
3398            fprintf(datei2, "%c", 9);
3399            printFloat(3.4, datei2, 5.0);
3400            fprintf(datei2, "%c", 9);
3401            printFloat(0.0, datei2, 5.0);
3402            fprintf(datei2, "%c", 9);
3403            printFloat(0.0, datei2, 5.0);
3404            fprintf(datei2, "%c", 9);
3405            printFloat(0.0, datei2, 5.0);
3406            fprintf(datei2, "%c", 9);
3407            printFloat(3.3, datei2, 5.0);
3408            fprintf(datei2, "%c", 9);
3409            printFloat(0.0, datei2, 5.0);
3410            fprintf(datei2, "%c", 9);
3411            printFloat(3.3, datei2, 5.0);
3412            fprintf(datei2, "%c", 9);
3413            printFloat(0.0, datei2, 5.0);
3414            fprintf(datei2, "%c", 9);
3415            printFloat(0.0, datei2, 5.0);
3416            fprintf(datei2, "%c", 9);
3417            printFloat(0.0, datei2, 5.0);
3418            fprintf(datei2, "%c", 9);
3419            printFloat(0.0, datei2, 5.0);
3420            fprintf(datei2, "%c", 9);
3421            printFloat(0.0, datei2, 5.0);
3422        };
3423
3424        cout << "  ende berechnungS" << endl;
3425
3426        return (einrueck + 1);
3427    };
3428
3429    void printFloat ( float zahl, FILE* datei, float nachkomma )
3430    {
3431        float ftemp = zahl;
3432        float ftemp2 = pow(10.0, nachkomma);
3433        int itemp = 0;
3434        if ( ftemp < 0 ) {
3435            fprintf(datei, "-");
3436        };
3437        ftemp = fabs(ftemp);
3438        itemp = static_cast<int>(ftemp);
3439        ftemp -= static_cast<float>(itemp);
3440        fprintf(datei, "%i,", itemp );
3441        ftemp *= ftemp2;
3442        ftemp += 0.5;
3443        itemp = static_cast<int>(ftemp);
3444        fprintf(datei, "%0*i", static_cast<int>(nachkomma+.5), itemp);
3445        return;
3446    };
3447
3448    int flaechen ( Bild &bp, int xpos, int ypos, int mark, int ersteEbene, int
        ersterWert, int ersterBereich, int zweiteEbene, int zweiterWert, int
```

```
        zweiterBereich, int dritteEbene, int dritterWert, int dritterBereich )
3449 {
3450     int bpe = 0;
3451     int bpz = 0;
3452     int bpd = 0;
3453     bpe = bp.GetLayers();
3454     if ( xpos < 0 || ypos < 0 || xpos >= bp.GetWidth() || ypos >= bp.GetHeight()
              || zweiteEbene >= bpe && zweiterBereich < 2 || dritteEbene >= bpe &&
             dritterBereich < 2 ) {
3455         return 0;
3456     }
3457     else {
3458         bpe = bp.GetPixelValue(xpos, ypos, ersteEbene);
3459         if ( zweiterBereich < 2 ) {
3460             bpz = bp.GetPixelValue(xpos, ypos, zweiteEbene);
3461         };
3462         if ( dritterBereich < 2 ) {
3463             bpd = bp.GetPixelValue(xpos, ypos, dritteEbene);
3464         };
3465         if ( ersterBereich  == 0 && bpe != ersterWert  || ersterBereich  == -1
                 && bpe > ersterWert  || ersterBereich  == 1 && bpe < ersterWert
3466           || zweiterBereich == 0 && bpz != zweiterWert || zweiterBereich == -1
                   && bpz > zweiterWert || zweiterBereich == 1 && bpz < zweiterWert
3467           || dritterBereich == 0 && bpd != dritterWert || dritterBereich == -1
                   && bpd > dritterWert || dritterBereich == 1 && bpd < dritterWert )
                    {
3468             return 0;
3469         }
3470         else {
3471             int counter = 1;
3472             int dx = 0;
3473             int dy = 0;
3474             bool weiter = false;
3475             bp.SetPixelValue(xpos, ypos, ersteEbene, mark);
3476             listHead* liste = createList();
3477             insertElement(liste, -1, xpos, ypos);
3478             liste->first->emark = 4;
3479             while ( liste->count > 0 ) {
3480                 switch ( liste->last->emark ) {
3481                     case 0 : deleteElement(liste, -1);
3482                         weiter = true;
3483                         break;
3484                     case 1 : dx = 0;
3485                         dy = 1;
3486                         break;
3487                     case 2 : dx = 1;
3488                         dy = 0;
3489                         break;
3490                     case 3 : dx = 0;
3491                         dy = -1;
3492                         break;
3493                     case 4 : dx = -1;
3494                         dy = 0;
3495                         break;
3496                     default:
```

```
3497                            break;
3498                    };
3499                    if ( weiter ) {
3500                        weiter = false;
3501                    }
3502                    else {
3503                        liste->last->emark -= 1;
3504                        if ( liste->last->x + dx < 0 || liste->last->y + dy < 0 ||
                                liste->last->x + dx >= bp.GetWidth() || liste->last->y +
                                 dy >= bp.GetHeight() ) {
3505                        }
3506                        else {
3507                            bpe = bp.GetPixelValue(liste->last->x + dx, liste->last
                                    ->y + dy, ersteEbene);
3508                            if ( zweiterBereich < 2 ) {
3509                                bpz = bp.GetPixelValue(liste->last->x + dx, liste->
                                        last->y + dy, zweiteEbene);
3510                            };
3511                            if ( dritterBereich < 2 ) {
3512                                bpd = bp.GetPixelValue(liste->last->x + dx, liste->
                                        last->y + dy, dritteEbene);
3513                            };
3514                            if ( bpe != mark && ( bpe == ersterWert  || bpe <
                                    ersterWert  && ersterBereich  == -1 || bpe >
                                    ersterWert  && ersterBereich  == 1 )
3515                                                 && ( bpz == zweiterWert || bpz <
                                                         zweiterWert && zweiterBereich == -1
                                                          || bpz > zweiterWert &&
                                                         zweiterBereich == 1 ||
                                                         zweiterBereich > 1 )
3516                                                 && ( bpd == dritterWert || bpd <
                                                         dritterWert && dritterBereich == -1
                                                          || bpd > dritterWert &&
                                                         dritterBereich == 1 ||
                                                         dritterBereich > 1 ) ) {
3517                                insertElement(liste, -1, liste->last->x + dx, liste
                                        ->last->y + dy);
3518                                liste->last->emark = 4;
3519                                bp.SetPixelValue(liste->last->x, liste->last->y,
                                        ersteEbene, mark);
3520                                counter++;
3521                            };
3522                        };
3523                    };
3524                };
3525                deleteList(liste);
3526                return counter;
3527            };
3528        };
3529    };
3530
3531    void putNumberInFile ( int number, FILE* file )
3532    {
3533        char big = 0;
3534        char little = 0;
```

```
3535      if ( number < 0 ) {
3536          big = 128;
3537          number *= -1;
3538      };
3539      little = ( number & 255 );
3540      number = number / 256;
3541      big = ( big | ( number & 127 ) );
3542      fprintf(file, "%c%c", big, little);
3543      return;
3544  };
3545
3546  int getNumberFromFile ( FILE* file )
3547  {
3548      char big = fgetc(file);
3549      char little = fgetc(file);
3550      int nbl = static_cast<int>(big);
3551      int number = 0;
3552      if ( nbl < 0 ) {
3553          number = ( nbl + 128 ) * 256;
3554      }
3555      else {
3556          number = nbl * 256;
3557      };
3558      if ( nbl < 0 ) {
3559          number *= -1;
3560          nbl =  static_cast<int>(little);
3561          if ( nbl < 0 ) {
3562              nbl += 256;
3563          };
3564          number -= nbl;
3565      }
3566      else {
3567          nbl =  static_cast<int>(little);
3568          if ( nbl < 0 ) {
3569              nbl += 256;
3570          };
3571          number += nbl;
3572      };
3573      return number;
3574  };
3575
3576  float ellDiff ( Bild &bp, listHead* liPos, float basPara[5], int parameter,
          float startWerte[2], int bpRi, bool einzel )
3577  {
3578      bool besser = true;
3579      bool kleiner = false;
3580      bool turn = false;
3581      int i = 0;
3582      float diffDiff = 0.0;
3583      float diffDiffAlt = 0.0;
3584      float minDiff = startWerte[0];
3585      float aktDiff = 0.0;
3586      float winkel = 0.0;
3587      Vec2D vaend (0, 0);
3588      if ( parameter == 0 ) {
```

```
3589            vaend.SetL(1.0);
3590            vaend.SetW(basPara[4]);
3591        }
3592        else if ( parameter == 1 ) {
3593            vaend.SetL(1.0);
3594            vaend.SetW(basPara[4] + PI / 2.0);
3595        };
3596
3597        float aenderung = 0.0;
3598        if ( !einzel ) {
3599            if ( parameter < 2 ) {
3600                aenderung = vaend.GetX();
3601            }
3602            else if ( parameter < 4 ) {
3603                aenderung = 1.0;
3604            }
3605            else {
3606                aenderung = PI / 90.0;
3607            };
3608        };
3609        float aktPara[5];
3610        float ftemp1 = 0.0;
3611        float ftemp2 = 0.0;
3612        float pcount = 0.0;
3613        float pcountMin = startWerte[1];
3614        float paraRi = 0.5;
3615        for ( i = 0; i < 5; i++ ) {
3616            aktPara[i] = basPara[i];
3617        };
3618        Vec2D vp (0, 0);
3619        Vec2D vt (0, 0);
3620        Vec2D vw (0, 0);
3621        Vec2D v2 (0, 0);
3622        do {
3623            if ( diffDiff > 0.0 ) {
3624                minDiff = aktDiff;
3625            };
3626            aktDiff = 0.0;
3627            pcount = 0.0;
3628            if ( parameter < 2 ) {
3629                aktPara[parameter] += vaend.GetX();
3630                aenderung = vaend.GetX();
3631                aktPara[1-parameter] += vaend.GetY();
3632            }
3633            else {
3634                aktPara[parameter] += aenderung;
3635            };
3636            liPos->p1 = liPos->first;
3637            while ( liPos->p1 != 0 ) {
3638                vt.SetIXIY(liPos->p1->x, liPos->p1->y);
3639                vp.SetXY(vt.GetX() - aktPara[0], vt.GetY() - aktPara[1]);
3640                ftemp1 = aktPara[0] + aktPara[2] * cos(vp.GetW() - aktPara[4]) * cos
                        (aktPara[4]) - aktPara[3] * sin(vp.GetW() - aktPara[4]) * sin(
                        aktPara[4]);
```

```
3641            ftemp2 = aktPara[1] + aktPara[2] * cos(vp.GetW() - aktPara[4]) * sin
                    (aktPara[4]) + aktPara[3] * sin(vp.GetW() - aktPara[4]) * cos(
                    aktPara[4]);
3642            vw.SetXY(ftemp1, ftemp2);
3643            winkel = atan ( aktPara[2] * aktPara[2] / ( aktPara[3] * aktPara[3]
                    ) * tan( vp.GetW() + aktPara[4] ) ) - aktPara[4];
3644            while ( winkel < -( PI / 2.0 ) ) {
3645                winkel += PI;
3646            };
3647            while ( winkel > ( PI / 2.0 ) ) {
3648                winkel -= PI;
3649            };
3650            if ( winkel <= -1.178 || winkel >= 1.178 ) {
3651                i = 1;
3652            }
3653            else if ( winkel > -0.393 && winkel < 0.393 ) {
3654                i = 3;
3655            }
3656            else if ( winkel >= 0.393 && winkel < 1.178 ) {
3657                i = 4;
3658            }
3659            else if ( winkel <= -0.393 && winkel > -1.178 ) {
3660                i = 2;
3661            }
3662            else {
3663                cout << "  Fehler bei Kantenrichtungsbestimmung eines Punktes."
                        << endl << "winkel   va.GetL   lb   vp.GetW   va.GetW
                        liPos->p1->x   vm.GetX   liPos->p1->y   vm.GetY" << endl <<
                        winkel << "  " << ( aktPara[2] ) << "  " << ( aktPara[3] )
                        << "  " << ( vp.GetW() ) << "  " << ( aktPara[4] ) << "  "
                        << ( liPos->p1->x ) << "  " << ( aktPara[0] ) << "  " << (
                        liPos->p1->y ) << "  " << ( aktPara[1] ) << endl;
3664                printf("%c%c%c%c", 7, 7, 7, 7);
3665                cout << i << endl;
3666                cin >> i;
3667            };
3668            v2 = vw - vt;
3669            if ( i == bp.GetPixelValue(liPos->p1->x, liPos->p1->y, bpRi) ) {
3670                aktDiff += v2.GetL();
3671                pcount += 1.0;
3672            };
3673            liPos->p1 = liPos->p1->next;
3674        };
3675        aktDiff /= pcount;
3676        diffDiff = minDiff - aktDiff;
3677        if ( einzel ) {
3678            pcountMin = pcount;
3679        }
3680        else {
3681            if ( pcount > pcountMin * 0.95 ) {
3682                if ( diffDiff < 0.0 && fabs( diffDiff - diffDiffAlt ) < 0.00001
                        ) {
3683                    diffDiff *= 0.7;
3684                };
3685                if ( diffDiff > 0.0 ) {
```

```
3686                        besser = true;
3687                        kleiner = true;
3688                        paraRi = ( aenderung < 0.0 ? -1.0 : 1.0 );
3689                        if ( pcount > pcountMin ) {
3690                            pcountMin = pcount;
3691                        };
3692                    }
3693                    else {
3694                        besser = false;
3695                    };
3696                }
3697                else {
3698                    besser = false;
3699                    if ( fabs( diffDiff - fabs(diffDiffAlt) ) < 0.00001 ) {
3700                        diffDiff *= 0.7;
3701                    };
3702                };
3703
3704                if ( !besser ) {
3705                    aktPara[parameter] -= aenderung;
3706                    if ( parameter < 2 ) {
3707                        aktPara[1-parameter] -= vaend.GetY();
3708                    };
3709                    if ( kleiner ) {
3710                        if ( parameter < 2 ) {
3711                            vaend.SetL(vaend.GetL() / 2.0);
3712                            aenderung = vaend.GetX();
3713                        }
3714                        else {
3715                            aenderung /= 2.0;
3716                        };
3717                        kleiner = false;
3718                    }
3719                    else {
3720                        if ( parameter < 2 ) {
3721                            vaend.SetL(vaend.GetL() * -1.0);
3722                            aenderung = vaend.GetX();
3723                        }
3724                        else {
3725                            aenderung *= -1.0;
3726                        };
3727                        kleiner = true;
3728                    };
3729                };
3730            };
3731        }
3732        while ( ( diffDiff > 0.001 || diffDiff < 0.0 ) && !einzel && ( fabs(
                aenderung) > 0.00001 && parameter < 3.5 || fabs(aenderung) > 0.0000001
                && parameter > 3.5) );
3733        if ( !einzel ) {
3734            basPara[parameter] = aktPara[parameter];
3735            if ( parameter < 2 ) {
3736                basPara[1-parameter] = aktPara[1-parameter];
3737            }
```

```
3738          else if ( ( parameter == 2 || parameter == 3 ) && fabs(basPara[parameter
                  ]) < 0.00001 ) {
3739              basPara[parameter] = 0.00001;
3740          };
3741      };
3742      startWerte[1] = pcountMin;
3743      return aktDiff;
3744  };
3745
3746  int maxKantensuche ( Bild &bp2, listHead* liRand[2], bool firstNumber, int hist
          [256], int grauGrenzeAlt, float oeffnungAlt, float ruecktemp[3], bool
          keinGiftkanal, bool keinMarkkanal, bool mark[1], FILE* mess )
3747  {
3748      int grauGrenze = 0;
3749      vector<int> vecFl(0);
3750      vector<int> vecX1(0);
3751      vector<int> vecX2(0);
3752      vector<int> vecY1(0);
3753      vector<int> vecY2(0);
3754      int i = 0;
3755      int j = 0;
3756
3757      FILE* ausg;
3758
3759      float ftemp1 = 0.0;
3760      float ftemp2 = 0.0;
3761      float ftemp3 = 0.0;
3762      float ftemp4 = 0.0;
3763      float ftemp5 = 0.0;
3764      float ftemp6 = 0.0;
3765
3766      bool gefunden = false;
3767      int ia = 0;
3768      int ib = 0;
3769      int ic = 0;
3770      int id = 0;
3771      int ig = 0;
3772      int ih = 0;
3773      int ii = 0;
3774      int ij = 0;
3775
3776      for ( i = 1; i < bp2.GetHeight() - 1; i++ ) {
3777          for ( j = 1; j < bp2.GetWidth() - 1; j++ ) {
3778              ia = ( bp2.GetPixelValue(j-1, i-1, 1) + bp2.GetPixelValue(j-1, i+1,
                      1) - bp2.GetPixelValue(j+1, i-1, 1) - bp2.GetPixelValue(j+1, i
                      +1, 1) + 2 * bp2.GetPixelValue(j-1, i, 1) - 2 * bp2.
                      GetPixelValue(j+1, i, 1) );
3779              ib = ( -1 * bp2.GetPixelValue(j-1, i-1, 1) + bp2.GetPixelValue(j-1,
                      i+1, 1) - bp2.GetPixelValue(j+1, i-1, 1) + bp2.GetPixelValue(j
                      +1, i+1, 1) - 2 * bp2.GetPixelValue(j, i-1, 1) + 2 * bp2.
                      GetPixelValue(j, i+1, 1) );
3780              ftemp1 = sqrt(static_cast<float>(ia * ia + ib * ib));
3781              ic = static_cast<int>(ftemp1+0.5);
3782              bp2.SetPixelValue(j, i, 2, ic);
3783              bp2.SetPixelValue(j, i, 4, -1);
```

```
3784            };
3785        };
3786
3787        ii = 0;
3788
3789        for ( ih = 0; ih < 256; ih++ ) {
3790            ij = 0;
3791            ig = 0;
3792            for ( i = 0; i < bp2.GetHeight(); i++ ) {
3793                for ( j = 0; j < bp2.GetWidth(); j++ ) {
3794                    gefunden = false;
3795                    ia = bp2.GetPixelValue(j, i, 1);
3796                    if ( j > 0 ) {
3797                        ib = bp2.GetPixelValue(j-1, i, 1);
3798                        if ( ia >= ih && ib < ih ) {
3799                            gefunden = true;
3800                            ij += bp2.GetPixelValue(j, i, 2);
3801                            ig++;
3802                        };
3803                    };
3804                    if ( j < bp2.GetWidth() - 1 && !gefunden ) {
3805                        ib = bp2.GetPixelValue(j+1, i, 1);
3806                        if ( ia >= ih && ib < ih ) {
3807                            gefunden = true;
3808                            ij += bp2.GetPixelValue(j, i, 2);
3809                            ig++;
3810                        };
3811                    };
3812                    if ( i > 0 && !gefunden ) {
3813                        ib = bp2.GetPixelValue(j, i-1, 1);
3814                        if ( ia >= ih && ib < ih ) {
3815                            gefunden = true;
3816                            ij += bp2.GetPixelValue(j, i, 2);
3817                            ig++;
3818                        };
3819                    };
3820                    if ( i < bp2.GetHeight() - 1 && !gefunden ) {
3821                        ib = bp2.GetPixelValue(j, i+1, 1);
3822                        if ( ia >= ih && ib < ih ) {
3823                            ij += bp2.GetPixelValue(j, i, 2);
3824                            ig++;
3825                        };
3826                    };
3827                };
3828            };
3829            if ( ig > 0 ) {
3830                ij = ij / ig;
3831            };
3832            if ( ij >= ii ) {
3833                ii = ij;
3834                grauGrenze = ih;
3835            }
3836            else if ( ij < ii / 2 ) {
3837                break;
3838            };
```

```
3839        };
3840
3841        cout << "  grauGrenze: " << grauGrenze << endl;
3842
3843        for ( i = 0; i < bp2.GetHeight(); i++ ) {
3844            for ( j = 0; j < bp2.GetWidth(); j++ ) {
3845                bp2.SetPixelValue(j, i, 0, -1);
3846                bp2.SetPixelValue(j, i, 8, (firstNumber ? 0 : 1000));
3847            };
3848        };
3849
3850        ia = 0;
3851        ib = 0;
3852
3853        if ( !firstNumber ) {
3854            for ( i = 0; i < bp2.GetHeight(); i++ ) {
3855                for ( j = 0; j < bp2.GetWidth(); j++ ) {
3856                    for ( ia = (i < 5 ? -1 * i : -5); ia < (i > bp2.GetHeight() - 6
                            ? bp2.GetHeight() - i : 6); ia++ ) {
3857                        for ( ib = (j < 5 ? -1 * j : -5); ib < (j > bp2.GetWidth() -
                                 6 ? bp2.GetWidth() - j : 6); ib++ ) {
3858                            if ( bp2.GetPixelValue(j+ib, i+ia, 7) == 1 ) {
3859                                if ( bp2.GetPixelValue(j, i, 8) > ( abs(ia) > abs(ib
                                    ) ? abs(ia) : abs(ib) ) ) {
3860                                    bp2.SetPixelValue(j, i, 8, ( abs(ia) > abs(ib) ?
                                         abs(ia) : abs(ib) ));
3861                                };
3862                            };
3863                        };
3864                    };
3865                };
3866            };
3867        };
3868
3869        ia = 0;
3870        ib = 0;
3871        id = 0;
3872        ig = 1;
3873        ii = 0;
3874        vecFl.clear();
3875        vecX1.clear();
3876        vecX2.clear();
3877        vecY1.clear();
3878        vecY2.clear();
3879
3880        for ( i = 0; i < bp2.GetHeight(); i++ ) {
3881            for ( j = 0; j < bp2.GetWidth(); j++ ) {
3882                id = flaechen(bp2, j, i, ia, 0, -1, 0, 1, grauGrenze, 1, 8, 5, -1);
3883                if ( id > 0 ) {
3884                    vecFl.push_back(id);
3885                    vecX1.push_back(j);
3886                    vecX2.push_back(j);
3887                    vecY1.push_back(i);
3888                    vecY2.push_back(i);
3889                    if ( id > ib ) {
```

```
3890                    ib = id;
3891                    ig = ia;
3892                };
3893                ia++;
3894            }
3895            else {
3896                ii = bp2.GetPixelValue(j, i, 0);
3897                if ( ii > -1 ) {
3898                    if ( j > vecX2[ii] ) {
3899                        vecX2[ii] = j;
3900                    };
3901                    if ( i > vecY2[ii] ) {
3902                        vecY2[ii] = i;
3903                    };
3904                    if ( j < vecX1[ii] ) {
3905                        vecX1[ii] = j;
3906                    };
3907                    if ( i < vecY1[ii] ) {
3908                        vecY1[ii] = i;
3909                    };
3910                };
3911            };
3912        };
3913    };
3914
3915    int minX = vecX1[ig];
3916    int maxX = vecX2[ig];
3917    int minY = vecY1[ig];
3918    int maxY = vecY2[ig];
3919
3920    float zahnMittX = 0.0;
3921    float zahnMittY = 0.0;
3922    ftemp1 = 0.0;
3923
3924    for ( i = 0; i < bp2.GetHeight(); i++ ) {
3925        for ( j = 0; j < bp2.GetWidth(); j++ ) {
3926            if ( bp2.GetPixelValue(j, i, 0) == ig ) {
3927                bp2.SetPixelValue(j, i, 4, 1);
3928                zahnMittX += static_cast<float>(j);
3929                zahnMittY += static_cast<float>(i);
3930                ftemp1 += 1.0;
3931            }
3932            else {
3933                bp2.SetPixelValue(j, i, 4, 0);
3934            };
3935            bp2.SetPixelValue(j, i, 0, -1);
3936            bp2.SetPixelValue(j, i, 8, (firstNumber ? 0 : 1000));
3937        };
3938    };
3939
3940    zahnMittX /= ftemp1;
3941    zahnMittY /= ftemp1;
3942
3943
3944    listHead* liAus = createList();
```

```
3945
3946      if ( !keinGiftkanal ) {
3947
3948          vecFl.clear();
3949          vecX1.clear();
3950          vecX2.clear();
3951          vecY1.clear();
3952          vecY2.clear();
3953          ia = 0;
3954          ib = 0;
3955
3956          if ( !firstNumber ) {
3957              for ( i = 0; i < bp2.GetHeight(); i++ ) {
3958                  for ( j = 0; j < bp2.GetWidth(); j++ ) {
3959                      for ( ia = (i < 2 ? -1 * i : -2); ia < (i > bp2.GetHeight()
                              - 3 ? bp2.GetHeight() - i : 3); ia++ ) {
3960                          for ( ib = (j < 2 ? -1 * j : -2); ib < (j > bp2.GetWidth
                                  () - 3 ? bp2.GetWidth() - j : 3); ib++ ) {
3961                              if ( bp2.GetPixelValue(j+ib, i+ia, 7) == 3 ) {
3962                                  if ( bp2.GetPixelValue(j, i, 8) > ( abs(ia) >
                                          abs(ib) ? abs(ia) : abs(ib) ) ) {
3963                                      bp2.SetPixelValue(j, i, 8, ( abs(ia) > abs(
                                              ib) ? abs(ia) : abs(ib) ));
3964                                  };
3965                              };
3966                          };
3967                      };
3968                  };
3969              };
3970              id = flaechen(bp2, 0, 0, -2, 0, -1, 0, 1, (grauGrenze - 1) , -1, 8,
                      3, 1);
3971          }
3972          else {
3973              id = flaechen(bp2, 0, 0, -2, 0, -1, 0, 1, (grauGrenze - 1) , -1, 8,
                      1, 2);
3974          };
3975
3976          ia = 0;
3977          for ( i = minY; i < maxY + 1; i++ ) {
3978              for ( j = minX; j < maxX + 1; j++ ) {
3979                  id = flaechen(bp2, j, i, ia, 0, -1, 0, 1, (grauGrenze - 1) , -1,
                           8, 2, -1);
3980                  if ( id > 0 ) {
3981                      vecFl.push_back(id);
3982                      vecX1.push_back(j);
3983                      vecX2.push_back(j);
3984                      vecY1.push_back(i);
3985                      vecY2.push_back(i);
3986                      ia++;
3987                  }
3988                  else {
3989                      ib = bp2.GetPixelValue(j, i, 0);
3990                      if ( ib > -1 ) {
3991                          if ( j > vecX2[ib] ) {
3992                              vecX2[ib] = j;
```

```
3993                        };
3994                        if ( i > vecY2[ib] ) {
3995                            vecY2[ib] = i;
3996                        };
3997                        if ( j < vecX1[ib] ) {
3998                            vecX1[ib] = j;
3999                        };
4000                        if ( i < vecY1[ib] ) {
4001                            vecY1[ib] = i;
4002                        };
4003                    };
4004                };
4005            };
4006        };
4007
4008        ib = 0;
4009        ic = 0;
4010        ftemp2 = static_cast<float>(bp2.GetWidth() * bp2.GetHeight());
4011        for ( i = 0; i < ia; i++ ) {
4012            ftemp1 = PI * static_cast<float>(vecX2[i] - vecX1[i]) * static_cast<
                float>(vecY2[i] - vecY1[i]) / 4.0;
4013            if ( fabs( ftemp1 / static_cast<float>(vecFl[i]) - 1.0 ) < ftemp2 &&
                 vecFl[i] > 50 ) {
4014                ftemp2 = fabs( ftemp1 / static_cast<float>(vecFl[i]) - 1.0 );
4015                ib = i;
4016            };
4017        };
4018        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4019            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4020                if ( bp2.GetPixelValue(j, i, 0) == ib ) {
4021                    bp2.SetPixelValue(j, i, 4, 3);
4022                }
4023                else if ( bp2.GetPixelValue(j, i, 4) != 1 ) {
4024                    bp2.SetPixelValue(j, i, 4, 0);
4025                };
4026                bp2.SetPixelValue(j, i, 0, -1);
4027            };
4028        };
4029
4030        vecX1.clear();
4031        vecX2.clear();
4032        vecY1.clear();
4033        vecY2.clear();
4034
4035        ia = 0;
4036        ib = 0;
4037        ii = 0;
4038        ij = 0;
4039        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4040            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4041                id = flaechen(bp2, j, i, ia, 0, -1, 0, 4, 0, 0, 2, 0, 2);
4042                if ( id > 0 ) {
4043                    if ( id > ij ) {
4044                        ij = id;
4045                        ii = ia;
```

```
4046                    };
4047                    vecX1.push_back(id);
4048                    vecY1.push_back(0);
4049                    vecY2.push_back(0);
4050                    ia++;
4051                };
4052            };
4053        };
4054        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4055            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4056                ib = bp2.GetPixelValue(j, i, 0);
4057                if ( ib > -1 && ib != ii ) {
4058                    for ( ic = (i > 0 ? -1 : 0); ic < (i < (bp2.GetHeight() - 1)
                             ? 2 : 1); ic++ ) {
4059                        for ( id = (j > 0 ? -1 : 0); id < (j < (bp2.GetWidth() -
                                 1) ? 2 : 1); id++ ) {
4060                            ig = bp2.GetPixelValue(j+id, i+ic, 4);
4061                            if ( ig == 1 ) {
4062                                vecY1[ib] += 1;
4063                            }
4064                            else if ( ig == 3 ) {
4065                                vecY2[ib] += 1;
4066                            };
4067                        };
4068                    };
4069                };
4070            };
4071        };
4072        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4073            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4074                ib = bp2.GetPixelValue(j, i, 0);
4075                if ( ib > -1 && ib != ii ) {
4076                    if ( vecY2[ib] > vecY1[ib] ) {
4077                        bp2.SetPixelValue(j, i, 4, 3);
4078                    }
4079                    else if ( keinMarkkanal ) {
4080                        bp2.SetPixelValue(j, i, 4, 1);
4081                    };
4082                };
4083            };
4084        };
4085
4086        gefunden = false;
4087
4088        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4089            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4090                gefunden = false;
4091                ia = bp2.GetPixelValue(j, i, 4);
4092                if ( j > 0 ) {
4093                    ib = bp2.GetPixelValue(j-1, i, 4);
4094                    if ( ia != ib ) {
4095                        if ( ia == 1 && ib == 0 ) {
4096                            insertElement(liRand[0], -1, j, i);
4097                            gefunden = true;
4098                        }
```

```
4099                        else if ( ia == 1 && ib == 3 ) {
4100                            insertElement(liRand[1], -1, j, i);
4101                            gefunden = true;
4102                        };
4103                    };
4104                };
4105                if ( j < bp2.GetWidth() - 1 && !gefunden ) {
4106                    ib = bp2.GetPixelValue(j+1, i, 4);
4107                    if ( ia != ib ) {
4108                        if ( ia == 1 && ib == 0 ) {
4109                            insertElement(liRand[0], -1, j, i);
4110                            gefunden = true;
4111                        }
4112                        else if ( ia == 1 && ib == 3 ) {
4113                            insertElement(liRand[1], -1, j, i);
4114                            gefunden = true;
4115                        };
4116                    };
4117                };
4118                if ( i > 0 && !gefunden ) {
4119                    ib = bp2.GetPixelValue(j, i-1, 4);
4120                    if ( ia != ib ) {
4121                        if ( ia == 1 && ib == 0 ) {
4122                            insertElement(liRand[0], -1, j, i);
4123                            gefunden = true;
4124                        }
4125                        else if ( ia == 1 && ib == 3 ) {
4126                            insertElement(liRand[1], -1, j, i);
4127                            gefunden = true;
4128                        };
4129                    };
4130                };
4131                if ( i < bp2.GetHeight() - 1 && !gefunden ) {
4132                    ib = bp2.GetPixelValue(j, i+1, 4);
4133                    if ( ia != ib ) {
4134                        if ( ia == 1 && ib == 0 ) {
4135                            insertElement(liRand[0], -1, j, i);
4136                        }
4137                        else if ( ia == 1 && ib == 3 ) {
4138                            insertElement(liRand[1], -1, j, i);
4139                        };
4140                    };
4141                };
4142            };
4143        };
4144
4145        if ( liRand[1]->count > 4 ) {
4146            sortListByCoordinates(liRand[1], liAus);
4147
4148            liAus->p1 = liAus->first;
4149            while ( liAus->p1 != 0 ) {
4150                bp2.SetPixelValue(liAus->p1->x, liAus->p1->y, 4, 1);
4151                liAus->p1 = liAus->p1->next;
4152            };
4153
```

```
4154            emptyList(liAus);
4155
4156            Vec2D vm( ruecktemp[1], ruecktemp[2], 'x' );
4157            Vec2D v1(0,0);
4158            Vec2D v2(0,0);
4159            Vec2D vp(0,0);
4160            Vec2D vt(0,0);
4161            bool bUnd = true;
4162            bool firstIsOne = true;
4163            ia = liRand[1]->count / 3 + ( liRand[1]->count % 3 < 2 ? 0 : 1 );
4164            if ( abs ( liRand[1]->first->x - liRand[1]->last->x ) > 1 || abs (
                    liRand[1]->first->y - liRand[1]->last->y ) > 1 ) {
4165                if ( firstNumber ) {
4166                    cout << "  offener Giftkanal bei der ersten Scheibe. Das
                            darf nicht vorkommen." << endl;
4167                }
4168                else {
4169                    cout << "  offener Giftkanal" << endl;
4170                    i = liRand[1]->count / 2;
4171                    liRand[1]->p1 = liRand[1]->first;
4172                    while ( liRand[1]->p1->number < i ) {
4173                        liRand[1]->p1 = liRand[1]->p1->next;
4174                    };
4175                    vt.SetIXIY(liRand[1]->p1->x, liRand[1]->p1->y);
4176                    vp.SetIXIY(liRand[1]->first->x, liRand[1]->first->y);
4177                    v1 = vp - vt;
4178                    vp.SetIXIY(liRand[1]->last->x, liRand[1]->last->y);
4179                    v2 = vp - vt;
4180                    ftemp1 = v1.GetW() + PI;
4181                    ftemp2 = v2.GetW() + PI;
4182                    if ( ftemp1 < ftemp2 ) {
4183                        ftemp3 = ftemp1;
4184                        ftemp4 = ftemp2;
4185                        firstIsOne = true;
4186                    }
4187                    else {
4188                        ftemp3 = ftemp2;
4189                        ftemp4 = ftemp1;
4190                        firstIsOne = false;
4191                    };
4192                    if ( ftemp4 - ftemp3 < PI ) {
4193                        ftemp1 = ftemp3;
4194                        ftemp2 = ftemp4;
4195                    }
4196                    else {
4197                        ftemp1 = ftemp4;
4198                        ftemp2 = ftemp3;
4199                        bUnd = false;
4200                        firstIsOne = !firstIsOne;
4201                    };
4202                    liRand[1]->p1 = liRand[1]->first;
4203                    liRand[1]->p3 = liRand[1]->first;
4204                    for ( i = 0; i < ia; i++ ) {
4205                        vp.SetIXIY(liRand[1]->p3->x, liRand[1]->p3->y);
4206                        v1 = vp - vt;
```

```
4207                        ftemp3 = v1.GetW() + PI;
4208                        if ( ( ftemp3 > ftemp1 && ftemp3 < ftemp2 && bUnd ) || (
                                ( ftemp3 > ftemp1 || ftemp3 < ftemp2 ) && !bUnd ) )
                                {
4209                            if ( firstIsOne ) {
4210                                ftemp1 = ftemp3;
4211                            }
4212                            else {
4213                                ftemp2 = ftemp3;
4214                            };
4215                            liRand[1]->p1 = liRand[1]->p3;
4216                        };
4217                        liRand[1]->p3 = liRand[1]->p3->next;
4218                        if ( liRand[1]->p3 == 0 ) {
4219                            i = 1000;
4220                        };
4221                    };
4222                    while ( liRand[1]->p1 != liRand[1]->first ) {
4223                        insertElement(liRand[0], -1, liRand[1]->first->x, liRand
                            [1]->first->y);
4224                        liRand[0]->last->emark = liRand[1]->first->emark;
4225                        liRand[0]->last->emark2 = liRand[1]->first->emark2;
4226                        deleteElement(liRand[1], 0);
4227                        ib--;
4228                    };
4229                    liRand[1]->p2 = liRand[1]->last;
4230                    liRand[1]->p3 = liRand[1]->last;
4231                    for ( i = 0; i < ia; i++ ) {
4232                        vp.SetIXIY(liRand[1]->p3->x, liRand[1]->p3->y);
4233                        v2 = vp - vt;
4234                        ftemp3 = v2.GetW() + PI;
4235                        if ( ( ftemp3 > ftemp1 && ftemp3 < ftemp2 && bUnd ) || (
                                ( ftemp3 > ftemp1 || ftemp3 < ftemp2 ) && !bUnd ) )
                                {
4236                            if ( firstIsOne ) {
4237                                ftemp2 = ftemp3;
4238                            }
4239                            else {
4240                                ftemp1 = ftemp3;
4241                            };
4242                            liRand[1]->p2 = liRand[1]->p3;
4243                        };
4244                        liRand[1]->p3 = liRand[1]->p3->prev;
4245                        if ( liRand[1]->p3 == 0 ) {
4246                            i = 1000;
4247                        };
4248                    };
4249                    while ( liRand[1]->p2 != liRand[1]->last ) {
4250                        insertElement(liRand[0], -1, liRand[1]->last->x, liRand
                            [1]->last->y);
4251                        liRand[0]->last->emark = liRand[1]->last->emark;
4252                        liRand[0]->last->emark2 = liRand[1]->last->emark2;
4253                        deleteElement(liRand[1], -1);
4254                    };
4255                    cout << "  Verbindungslinie erstellen" << endl;
```

```
4256                    int dx = liRand[1]->first->x - liRand[1]->last->x;
4257                    int dy = liRand[1]->first->y - liRand[1]->last->y;
4258                    fprintf(mess, "%c", 9);
4259                    printFloat(sqrt(static_cast<float>( dx * dx + dy * dy )),
                            mess, 5.0);
4260                    float dXdY = 0.0;
4261                    if ( abs(dy) > abs(dx) ) {
4262                        dXdY = fabs( static_cast<float>(dx) / static_cast<float
                                >(dy) );
4263                    }
4264                    else {
4265                        dXdY = fabs( static_cast<float>(dy) / static_cast<float
                                >(dx) );
4266                    };
4267                    float dXdYS = 0.5;
4268                    ia = liRand[1]->first->x;
4269                    ib = liRand[1]->first->y;
4270                    int addx = (dx > 0 ? -1 : 1);
4271                    int addy = (dy > 0 ? -1 : 1);
4272                    for ( int verb = 1; verb < ( abs(dy) > abs(dx) ? abs(dy) :
                            abs(dx) ); verb++ ) {
4273                        dXdYS += dXdY;
4274                        if ( abs(dy) > abs(dx) ) {
4275                            ib += addy;
4276                            if ( dXdYS >= 1 ) {
4277                                ia += addx;
4278                                dXdYS -= 1;
4279                            };
4280                        }
4281                        else {
4282                            ia += addx;
4283                            if ( dXdYS >= 1 ) {
4284                                ib += addy;
4285                                dXdYS -= 1;
4286                            };
4287                        };
4288                        bp2.SetPixelValue(ia, ib, 4, 4);
4289                    };
4290
4291                    for ( i = 0; i < bp2.GetHeight(); i++ ) {
4292                        for ( j = 0; j < bp2.GetWidth(); j++ ) {
4293                            ia = bp2.GetPixelValue(j, i, 4);
4294                            if ( ia == 3 ) {
4295                                bp2.SetPixelValue(j, i, 4, 0);
4296                            }
4297                            else if ( ia == 4 ) {
4298                                bp2.SetPixelValue(j, i, 4, 3);
4299                            };
4300                        };
4301                    };
4302
4303                    v2.SetXY( static_cast<float>(liRand[1]->first->x + liRand
                            [1]->last->x) / 2.0, static_cast<float>(liRand[1]->first
                            ->y + liRand[1]->last->y) / 2.0 );
4304                    vp = v2 - vt;
```

```
4305                    vp.SetL(1.0);
4306                    Vec2D v3(0,0);
4307                    v3.SetIXIY(liRand[1]->last->y - liRand[1]->first->y, liRand
                            [1]->first->x - liRand[1]->last->x);
4308                    v1.SetIXIY(liRand[1]->first->x, liRand[1]->first->y);
4309                    ftemp1 = v1.dotp(v3);
4310                    ftemp2 = ( ftemp1 - vt.dotp(v3) ) / v3.dotp(v3);
4311                    v1 = vt;
4312                    do {
4313                        v1 = v1 + vp;
4314                        j = v1.GetIX();
4315                        i = v1.GetIY();
4316                        ftemp3 = ( ftemp1 - v1.dotp(v3) ) / v3.dotp(v3);
4317                        cout << "j / i = " << j << " / " << i << endl;
4318                        cout << "E4: " << ( bp2.GetPixelValue(j, i, 4) ) << endl
                                ;
4319                        cout << "ftemp2 / ftemp3 = " << ftemp2 << " / " <<
                                ftemp3 << endl;
4320                    }
4321                    while ( bp2.GetPixelValue(j, i, 4) == 1 && ftemp2 * ftemp3
                            >= 0.0 );
4322
4323                    if ( ftemp2 * ftemp3 >= 0.0 ) {
4324                        id = flaechen(bp2, j, i, 3, 4, 0, 0, 0, 4, 2, 8, 1, 2);
4325                    }
4326                    else {
4327                        cout << "Kein flaechen, da auf der falschen Seite." <<
                                endl;
4328                    };
4329
4330                    cout << "fl-test: " << id << endl;
4331                    cout << "ra-test: " << ( liRand[1]->count ) << endl;
4332                };
4333            }
4334            else {
4335                ruecktemp[0] = 0.0;
4336                fprintf(mess, "%c0", 9);
4337            };
4338            sortListByCoordinates(liRand[0], liAus);
4339            liRand[0]->p1 = liRand[0]->first;
4340            while ( liRand[0]->p1 != 0 ) {
4341                liRand[0]->p1 = liRand[0]->p1->next;
4342            };
4343
4344            ftemp1 = 0.0;
4345            ftemp2 = 0.0;
4346            ftemp3 = 0.0;
4347            for ( i = 0; i < bp2.GetHeight(); i++ ) {
4348                for ( j = 0; j < bp2.GetWidth(); j++ ) {
4349                    if ( bp2.GetPixelValue(j, i, 4) == 3 ) {
4350                        ftemp2 += static_cast<float>(j);
4351                        ftemp3 += static_cast<float>(i);
4352                        ftemp1 += 1.0;
4353                    };
4354                    bp2.SetPixelValue(j, i, 0, -1);
```

```
4355                 };
4356             };
4357             ruecktemp[1] = ftemp2 / ftemp1;
4358             ruecktemp[2] = ftemp3 / ftemp1;
4359         }
4360         else {
4361             for ( i = 0; i < bp2.GetHeight(); i++ ) {
4362                 for ( j = 0; j < bp2.GetWidth(); j++ ) {
4363                     bp2.SetPixelValue(j, i, 0, -1);
4364                 };
4365             };
4366         };
4367     };
4368
4369     if ( !keinMarkkanal ) {
4370         ia = 0;
4371         ib = 0;
4372
4373         if ( !firstNumber ) {
4374             for ( i = 0; i < bp2.GetHeight(); i++ ) {
4375                 for ( j = 0; j < bp2.GetWidth(); j++ ) {
4376                     bp2.SetPixelValue(j, i, 8, 1000);
4377                     for ( ia = (i < 5 ? -1 * i : -5); ia < (i > bp2.GetHeight()
                            - 6 ? bp2.GetHeight() - i : 6); ia++ ) {
4378                     for ( ib = (j < 5 ? -1 * j : -5); ib < (j > bp2.GetWidth
                            () - 6 ? bp2.GetWidth() - j : 6); ib++ ) {
4379                         if ( bp2.GetPixelValue(j+ib, i+ia, 7) == 2 ) {
4380                             if ( bp2.GetPixelValue(j, i, 8) > ( abs(ia) >
                                    abs(ib) ? abs(ia) : abs(ib) ) ) {
4381                                 bp2.SetPixelValue(j, i, 8, ( abs(ia) > abs(
                                        ib) ? abs(ia) : abs(ib) ));
4382                             };
4383                         };
4384                     };
4385                     };
4386                 };
4387             };
4388             id = flaechen(bp2, 0, 0, -2, 0, 0, -1, 1, (grauGrenze - 1) , -1, 8,
                    6, 1);
4389         }
4390         else {
4391             id = flaechen(bp2, 0, 0, -2, 0, 0, -1, 1, (grauGrenze - 1) , -1, 4,
                    0, 0);
4392         };
4393
4394         ia = 0;
4395         ib = 0;
4396         ig = 0;
4397
4398         for ( i = minY; i < maxY + 1; i++ ) {
4399             for ( j = minX; j < maxX + 1; j++ ) {
4400                 if ( firstNumber ) {
4401                     id = flaechen(bp2, j, i, ia, 0, -1, 0, 1, (grauGrenze - 1) ,
                            -1, 4, 2, -1);
4402                 }
```

```
4403                    else {
4404                        id = flaechen(bp2, j, i, ia, 0, -1, 0, 1, (grauGrenze - 1) ,
                                -1, 8, 5, -1);
4405                    };
4406                    if ( id > 0 ) {
4407                        if ( id > ib ) {
4408                            ib = id;
4409                            ig = ia;
4410                        };
4411                        ia++;
4412                    };
4413                };
4414            };
4415
4416            for ( i = minY; i < maxY + 1; i++ ) {
4417                for ( j = minX; j < maxX + 1; j++ ) {
4418                    if ( bp2.GetPixelValue(j, i, 0) == ig ) {
4419                        bp2.SetPixelValue(j, i, 4, 2);
4420                    };
4421                    ia = bp2.GetPixelValue(j, i, 0);
4422                    if ( ia != -2 ) {
4423                        bp2.SetPixelValue(j, i, 0, -1);
4424                    };
4425                    bp2.SetPixelValue(j, i, 8, (firstNumber ? 0 : 1000));
4426                };
4427            };
4428
4429            vecX1.clear();
4430            vecX2.clear();
4431            vecY1.clear();
4432            vecY2.clear();
4433
4434            ia = 0;
4435            ib = 0;
4436            ii = 0;
4437            ij = 0;
4438            for ( i = 0; i < bp2.GetHeight(); i++ ) {
4439                for ( j = 0; j < bp2.GetWidth(); j++ ) {
4440                    id = flaechen(bp2, j, i, ia, 0, -1, 0, 4, 0, 0, 2, 0, 2);
4441                    if ( id > 0 ) {
4442                        vecY1.push_back(0);
4443                        vecY2.push_back(0);
4444                        ia++;
4445                    };
4446                };
4447            };
4448            for ( i = 0; i < bp2.GetHeight(); i++ ) {
4449                for ( j = 0; j < bp2.GetWidth(); j++ ) {
4450                    ib = bp2.GetPixelValue(j, i, 0);
4451                    if ( ib > -1 ) {
4452                        for ( ic = (i > 0 ? -1 : 0); ic < (i < (bp2.GetHeight() - 1)
                                ? 2 : 1); ic++ ) {
4453                            for ( id = (j > 0 ? -1 : 0); id < (j < (bp2.GetWidth() -
                                    1) ? 2 : 1); id++ ) {
4454                                ig = bp2.GetPixelValue(j+id, i+ic, 4);
```

```
4455                            if ( ig == 1 ) {
4456                                vecY1[ib] += 1;
4457                            }
4458                            else if ( ig == 2 ) {
4459                                vecY2[ib] += 1;
4460                            };
4461                        };
4462                    };
4463                };
4464            };
4465        };
4466        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4467            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4468                ib = bp2.GetPixelValue(j, i, 0);
4469                if ( ib > -1 ) {
4470                    if ( vecY2[ib] > vecY1[ib] ) {
4471                        bp2.SetPixelValue(j, i, 4, 2);
4472                    }
4473                    else if ( vecY1[ib] > 0 ) {
4474                        bp2.SetPixelValue(j, i, 4, 1);
4475                    };
4476                };
4477            };
4478        };
4479
4480        listHead* markLi = createList();
4481
4482        gefunden = false;
4483
4484        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4485            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4486                gefunden = false;
4487                ia = bp2.GetPixelValue(j, i, 4);
4488                if ( j > 0 ) {
4489                    ib = bp2.GetPixelValue(j-1, i, 4);
4490                    if ( ia != ib ) {
4491                        if ( ia == 1 && ib == 2 ) {
4492                            insertElement(markLi, -1, j, i);
4493                            gefunden = true;
4494                        };
4495                    };
4496                };
4497                if ( j < bp2.GetWidth() - 1 && !gefunden ) {
4498                    ib = bp2.GetPixelValue(j+1, i, 4);
4499                    if ( ia != ib ) {
4500                        if ( ia == 1 && ib == 2 ) {
4501                            insertElement(markLi, -1, j, i);
4502                            gefunden = true;
4503                        };
4504                    };
4505                };
4506                if ( i > 0 && !gefunden ) {
4507                    ib = bp2.GetPixelValue(j, i-1, 4);
4508                    if ( ia != ib ) {
4509                        if ( ia == 1 && ib == 2 ) {
```

```
4510                            insertElement(markLi, -1, j, i);
4511                            gefunden = true;
4512                        };
4513                    };
4514                };
4515                if ( i < bp2.GetHeight() - 1 && !gefunden ) {
4516                    ib = bp2.GetPixelValue(j, i+1, 4);
4517                    if ( ia != ib ) {
4518                        if ( ia == 1 && ib == 2 ) {
4519                            insertElement(markLi, -1, j, i);
4520                        };
4521                    };
4522                };
4523            };
4524        };
4525
4526        if ( markLi > 0 ) {
4527
4528            gefunden = false;
4529            ig = 1000000;
4530            ih = 1000000;
4531            ii = 0;
4532            ij = 0;
4533            ftemp3 = 0.0;
4534            ftemp4 = 0.0;
4535            ftemp5 = 0.0;
4536
4537            ib = 0;
4538            for ( i = 0; i < bp2.GetHeight(); i++ ) {
4539                for ( j = 0; j < bp2.GetWidth(); j++ ) {
4540                    ia = bp2.GetPixelValue(j, i, 4);
4541                    if ( ia == 2 ) {
4542                        if ( j < ih ) {
4543                            ih = j;
4544                        };
4545                        if ( j > ij ) {
4546                            ij = j;
4547                        };
4548                        if ( i < ig ) {
4549                            ig = i;
4550                        };
4551                        if ( i > ii ) {
4552                            ii = i;
4553                        };
4554                        ftemp3 += static_cast<float>(j);
4555                        ftemp4 += static_cast<float>(i);
4556                        ftemp5 += 1.0;
4557                        ib++;
4558                    };
4559                };
4560            };
4561            if ( ib < 5 ) {
4562                mark[0] = true;
4563            };
4564
```

```
4565            emptyList(liAus);
4566
4567            sortListByCoordinates(markLi, liAus);
4568
4569            markLi->p1 = markLi->first;
4570            while ( markLi->p1 != 0 ) {
4571                markLi->p1 = markLi->p1->next;
4572            };
4573
4574            liAus->p1 = liAus->first;
4575            while ( liAus->p1 != 0 ) {
4576                bp2.SetPixelValue(liAus->p1->x, liAus->p1->y, 4, 1);
4577                liAus->p1 = liAus->p1->next;
4578            };
4579            emptyList(liAus);
4580
4581            if ( abs ( markLi->first->x - markLi->last->x ) > 1 || abs ( markLi
                    ->first->y - markLi->last->y ) > 1 ) {
4582                if ( firstNumber ) {
4583                    cout << "  offener Markkanal bei der ersten Scheibe. Das
                            darf nicht vorkommen." << endl;
4584                }
4585                else {
4586                    cout << "  offener Markkanal" << endl;
4587
4588                    ftemp2 = static_cast<float>( ig + ii ) / 2.0;
4589                    ftemp1 = static_cast<float>( ih + ij ) / 2.0;
4590                    ftemp3 /= ftemp5;
4591                    ftemp4 /= ftemp5;
4592
4593                    j = static_cast<int>(ftemp1 + 0.5);
4594                    i = static_cast<int>(ftemp2 + 0.5);
4595                    ftemp5 = ftemp3 - ftemp1;
4596                    ftemp6 = ftemp4 - ftemp2;
4597
4598                    if ( fabs(ftemp5) > fabs(ftemp6) ) {
4599                        ftemp6 /= fabs(ftemp5);
4600                        ftemp5 /= fabs(ftemp5);
4601                    }
4602                    else {
4603                        ftemp5 /= fabs(ftemp6);
4604                        ftemp6 /= fabs(ftemp6);
4605                    };
4606
4607                    float count = 0.0;
4608                    ftemp3 = 0.0;
4609                    ftemp4 = 0.0;
4610
4611                    if ( bp2.GetPixelValue(j, i, 4) == 2 ) {
4612                        ftemp1 += 0.5;
4613                        ftemp2 += 0.5;
4614                        while ( bp2.GetPixelValue(static_cast<int>(ftemp1),
                                static_cast<int>(ftemp2), 4) == 2 ) {
4615                            ftemp3 += ftemp1;
4616                            ftemp4 += ftemp2;
```

```
4617                            count += 1.0;
4618                            ftemp1 += ftemp5;
4619                            ftemp2 += ftemp6;
4620                        };
4621                        ftemp2 = static_cast<float>( ig + ii ) / 2.0 + 0.5;
4622                        ftemp1 = static_cast<float>( ih + ij ) / 2.0 + 0.5;
4623                        ftemp1 -= ftemp5;
4624                        ftemp2 -= ftemp6;
4625                        while ( bp2.GetPixelValue(static_cast<int>(ftemp1),
                                static_cast<int>(ftemp2), 4) == 2 ) {
4626                            ftemp3 += ftemp1;
4627                            ftemp4 += ftemp2;
4628                            count += 1.0;
4629                            ftemp1 -= ftemp5;
4630                            ftemp2 -= ftemp6;
4631                        };
4632                        ftemp1 = ftemp3 / count;
4633                        ftemp2 = ftemp4 / count;
4634                    }
4635                    else {
4636                        ftemp1 += 0.5;
4637                        ftemp2 += 0.5;
4638                        while ( bp2.GetPixelValue(static_cast<int>(ftemp1),
                                static_cast<int>(ftemp2), 4) != 2 ) {
4639                            ftemp1 += ftemp5;
4640                            ftemp2 += ftemp6;
4641                            if ( ftemp1 < 0 || ftemp2 < 0 || static_cast<int>(
                                    ftemp1) > bp2.GetWidth() || static_cast<int>(
                                    ftemp2) > bp2.GetHeight() ) {
4642                                cout << "  FEHLER!" << endl;
4643                                break;
4644                            };
4645                        };
4646                        while ( bp2.GetPixelValue(static_cast<int>(ftemp1),
                                static_cast<int>(ftemp2), 4) == 2 ) {
4647                            ftemp3 += ftemp1;
4648                            ftemp4 += ftemp2;
4649                            count += 1.0;
4650                            ftemp1 += ftemp5;
4651                            ftemp2 += ftemp6;
4652                            if ( ftemp1 < 0 || ftemp2 < 0 || static_cast<int>(
                                    ftemp1) > bp2.GetWidth() || static_cast<int>(
                                    ftemp2) > bp2.GetHeight() ) {
4653                                cout << "  FEHLER!" << endl;
4654                                break;
4655                            };
4656                        };
4657                        ftemp1 = ftemp3 / count;
4658                        ftemp2 = ftemp4 / count;
4659                    };
4660
4661                    Vec2D vm( ftemp1, ftemp2, 'x' );
4662                    Vec2D v1(0,0);
4663                    Vec2D v2(0,0);
4664                    Vec2D vp(0,0);
```

```
4665                    bool bUnd = true;
4666                    bool firstIsOne = true;
4667                    ia = markLi->count / 2;
4668                    ib = ia + ( markLi->count % 2 );
4669
4670                    vp.SetIXIY(markLi->first->x, markLi->first->y);
4671                    v1 = vp - vm;
4672                    vp.SetIXIY(markLi->last->x, markLi->last->y);
4673                    v2 = vp - vm;
4674                    ftemp1 = v1.GetW() + PI;
4675                    ftemp2 = v2.GetW() + PI;
4676                    if ( ftemp1 < ftemp2 ) {
4677                        ftemp3 = ftemp1;
4678                        ftemp4 = ftemp2;
4679                        firstIsOne = true;
4680                    }
4681                    else {
4682                        ftemp3 = ftemp2;
4683                        ftemp4 = ftemp1;
4684                        firstIsOne = false;
4685                    };
4686                    if ( ftemp4 - ftemp3 < PI ) {
4687                        ftemp1 = ftemp3;
4688                        ftemp2 = ftemp4;
4689                    }
4690                    else {
4691                        ftemp1 = ftemp4;
4692                        ftemp2 = ftemp3;
4693                        bUnd = false;
4694                        firstIsOne = !firstIsOne;
4695                    };
4696                    markLi->p1 = markLi->first;
4697                    markLi->p3 = markLi->first;
4698                    for ( i = 0; i < 10; i++ ) {
4699                        vp.SetIXIY(markLi->p3->x, markLi->p3->y);
4700                        v1 = vp - vm;
4701                        ftemp3 = v1.GetW() + PI;
4702                        if ( ( ftemp3 > ftemp1 && ftemp3 < ftemp2 && bUnd ) || (
                                 ( ftemp3 > ftemp1 || ftemp3 < ftemp2 ) && !bUnd ) )
                                 {
4703                            if ( firstIsOne ) {
4704                                ftemp1 = ftemp3;
4705                            }
4706                            else {
4707                                ftemp2 = ftemp3;
4708                            };
4709                            markLi->p1 = markLi->p3;
4710                        };
4711                        markLi->p3 = markLi->p3->next;
4712                        if ( markLi->p3 == 0 || markLi->p3->number > ia ) {
4713                            i = 100;
4714                        };
4715                    };
4716                    while ( markLi->p1 != markLi->first ) {
4717                        deleteElement(markLi, 0);
```

```
4718                        ib--;
4719                    };
4720                    markLi->p2 = markLi->last;
4721                    markLi->p3 = markLi->last;
4722                    for ( i = 0; i < 10; i++ ) {
4723                        vp.SetIXIY(markLi->p3->x, markLi->p3->y);
4724                        v2 = vp - vm;
4725                        ftemp3 = v2.GetW() + PI;
4726                        if ( ( ftemp3 > ftemp1 && ftemp3 < ftemp2 && bUnd ) || (
                                ( ftemp3 > ftemp1 || ftemp3 < ftemp2 ) && !bUnd ) )
                                {
4727                            if ( firstIsOne ) {
4728                                ftemp2 = ftemp3;
4729                            }
4730                            else {
4731                                ftemp1 = ftemp3;
4732                            };
4733                            markLi->p2 = markLi->p3;
4734                        };
4735                        markLi->p3 = markLi->p3->prev;
4736                        if ( markLi->p3 == 0 || markLi->p3->number < ib ) {
4737                            i = 100;
4738                        };
4739                    };
4740                    while ( markLi->p2 != markLi->last ) {
4741                        deleteElement(markLi, -1);
4742                    };
4743                    cout << "  Verbindungslinie erstellen" << endl;
4744                    int dx = markLi->first->x - markLi->last->x;
4745                    int dy = markLi->first->y - markLi->last->y;
4746                    float dXdY = 0.0;
4747                    if ( abs(dy) > abs(dx) ) {
4748                        dXdY = fabs( static_cast<float>(dx) / static_cast<float
                                >(dy) );
4749                    }
4750                    else {
4751                        dXdY = fabs( static_cast<float>(dy) / static_cast<float
                                >(dx) );
4752                    };
4753                    float dXdYS = 0.5;
4754                    ia = markLi->first->x;
4755                    ib = markLi->first->y;
4756                    int addx = (dx > 0 ? -1 : 1);
4757                    int addy = (dy > 0 ? -1 : 1);
4758                    for ( int verb = 1; verb < ( abs(dy) > abs(dx) ? abs(dy) :
                        abs(dx) ); verb++ ) {
4759                        dXdYS += dXdY;
4760                        if ( abs(dy) > abs(dx) ) {
4761                            ib += addy;
4762                            if ( dXdYS >= 1 ) {
4763                                ia += addx;
4764                                dXdYS -= 1;
4765                            };
4766                        }
4767                        else {
```

```
4768                                ia += addx;
4769                                if ( dXdYS >= 1 ) {
4770                                    ib += addy;
4771                                    dXdYS -= 1;
4772                                };
4773                            };
4774                            bp2.SetPixelValue(ia, ib, 4, 4);
4775                        };
4776
4777                        ib = 0;
4778                        for ( i = 0; i < bp2.GetHeight(); i++ ) {
4779                            for ( j = 0; j < bp2.GetWidth(); j++ ) {
4780                                ia = bp2.GetPixelValue(j, i, 4);
4781                                if ( ia == 2 ) {
4782                                    bp2.SetPixelValue(j, i, 4, 0);
4783                                }
4784                                else if ( ia == 4 ) {
4785                                    bp2.SetPixelValue(j, i, 4, 2);
4786                                    ib++;
4787                                };
4788                            };
4789                        };
4790                        if ( ib < 5 ) {
4791                            mark[0] = true;
4792                        };
4793                        j = static_cast<int>(vm.GetIX());
4794                        i = static_cast<int>(vm.GetIY());
4795                        id = flaechen(bp2, j, i, 2, 4, 0, 0, 0, 4, 2, 8, 1, 2);
4796                    };
4797                };
4798            }
4799            else {
4800                mark[0] = true;
4801            };
4802        }
4803        else {
4804            id = flaechen(bp2, 0, 0, -2, 0, -1, 0, 1, grauGrenze - 1, -1, 8, 5, 2);
4805            for ( i = 0; i < bp2.GetHeight(); i++ ) {
4806                for ( j = 0; j < bp2.GetWidth(); j++ ) {
4807                    if ( bp2.GetPixelValue(j, i, 0) != -2 && bp2.GetPixelValue(j, i,
                             4) != 3 && bp2.GetPixelValue(j, i, 1) < grauGrenze ) {
4808                        bp2.SetPixelValue(j, i, 4, 1);
4809                    };
4810                };
4811            };
4812        };
4813
4814        return grauGrenze;
4815    };
4816
4817    int sortListByCoordinates ( listHead* listHead1, listHead* liAus )
4818    {
4819        cout << "  Begin Sortierung" << endl;
4820        if ( listHead1 == 0 ) {
4821            cout << "  Fehler bei Sortierung. Liste existiert nicht." << endl;
```

```
4822            return -1;
4823        }
4824        else if ( listHead1->count == 0 ) {
4825            cout << "  Fehler bei Sortierung. Liste ist leer." << endl;
4826            return -1;
4827        }
4828        else if ( listHead1->count < 3 ) {
4829            cout << "  Liste zu kurz fuer eine Sortierung." << endl;
4830            return 1;
4831        }
4832        else {
4833            ListList* zwLists = new ListList;
4834            zwLists->InsertList(copyList(listHead1), 0);
4835            listHead* liPos = zwLists->GetList(0);
4836            liPos->hmark = 0;
4837            int xmin = numeric_limits< int >::max();
4838            int ymin = numeric_limits< int >::max();
4839            int xmax = 0;
4840            int ymax = 0;
4841            int x1 = 0;
4842            int y1 = 0;
4843            int i = 0;
4844            int j = 0;
4845            listHead1->p1 = listHead1->first;
4846            while ( listHead1->p1 != 0 ) {
4847                i = listHead1->p1->x;
4848                if ( i < xmin ) {
4849                    xmin = i;
4850                };
4851                if ( i > xmax ) {
4852                    xmax = i;
4853                };
4854                i = listHead1->p1->y;
4855                if ( i < ymin ) {
4856                    ymin = i;
4857                };
4858                if ( i > ymax ) {
4859                    ymax = i;
4860                };
4861                listHead1->p1 = listHead1->p1->next;
4862            };
4863            int liPix[xmax-xmin+1][ymax-ymin+1][2];
4864            vector<listHead*> hePix[xmax-xmin+1][ymax-ymin+1];
4865            vector<listHead*> zwVec(0);
4866            listHead* zwHead = 0;
4867            for ( i = 0; i < ymax-ymin+1; i++ ) {
4868                for ( j = 0; j < xmax-xmin+1; j++ ) {
4869                    liPix[j][i][0] = 0;
4870                    liPix[j][i][1] = 0;
4871                };
4872            };
4873            listHead1->p1 = listHead1->first;
4874            while ( listHead1->p1 != 0 ) {
4875                liPix[listHead1->p1->x - xmin][listHead1->p1->y - ymin][0] = 1;
4876                listHead1->p1 = listHead1->p1->next;
```

```
4877        };
4878        int ic = 0;
4879        listHead1->p1 = listHead1->first;
4880        while ( listHead1->p1 != 0 ) {
4881            j = listHead1->p1->x - xmin;
4882            i = listHead1->p1->y - ymin;
4883            ic = -1;
4884            for ( y1 = ( i > 0 ? -1 : 0 ); y1 < ( i < ( ymax - ymin ) ? 2 : 1 );
                     y1++ ) {
4885                for ( x1 = ( j > 0 ? -1 : 0 ); x1 < ( j < ( xmax - xmin ) ? 2 :
                        1 ); x1++ ) {
4886                    if ( liPix[j+x1][i+y1][0] > 0 ) {
4887                        ic++;
4888                    };
4889                };
4890            };
4891            liPix[j][i][0] = ic;
4892            listHead1->p1 = listHead1->p1->next;
4893        };
4894
4895        int em = 0;
4896        int em2 = 0;
4897        int number = 0;
4898        listHead* liPos2;
4899
4900        liPos->p3 = 0;
4901        while ( liPos->p3 == 0 && liPos->count > 2 ) {
4902            liPos->p3 = liPos->first->next;
4903            while ( abs( liPos->p3->x - liPos->first->x ) > 1 || abs( liPos->p3
                    ->y - liPos->first->y ) > 1 ) {
4904                liPos->p3 = liPos->p3->next;
4905                if ( liPos->p3 == 0 ) {
4906                    cout << "Hier wird geloescht.   " << ( liPos->count ) << " (
                             " << ( liPos->first->x ) << " / " << ( liPos->first->y
                            ) << " )" << endl;
4907                    deleteElement(liPos, 0);
4908                    break;
4909                };
4910            };
4911        };
4912        if ( liPos->p3 != 0 ) {
4913            for ( i = 1; i < liPix[liPos->first->x - xmin][liPos->first->y -
                    ymin][0]; i++ ) {
4914                zwLists->InsertList(copyList(liPos), -1);
4915                liPos2 = zwLists->GetList(-1);
4916                liPos2->p3 = liPos2->p3->next;
4917                if ( liPos2->p3 != 0 ) {
4918                    while ( abs( liPos2->p3->x - liPos2->first->x ) > 1 || abs(
                            liPos2->p3->y - liPos2->first->y ) > 1 ) {
4919                        liPos2->p3 = liPos2->p3->next;
4920                        if ( liPos2->p3 == 0 ) {
4921                            break;
4922                        };
4923                    };
4924                    if ( liPos2->p3 != 0 ) {
```

```
4925                        number = liPos2->p3->number;
4926                        em = liPos2->p3->emark;
4927                        em2 = liPos2->p3->emark2;
4928                        x1 = liPos2->p3->x;
4929                        y1 = liPos2->p3->y;
4930                        deleteElement(liPos2, number);
4931                        insertElement(liPos2, 1, x1, y1);
4932                        liPos2->first->next->emark = em;
4933                        liPos2->first->next->emark2 = em2;
4934                    };
4935                }
4936                else {
4937                    cout << "  Das duerfte am Anfang nicht sein. (a)" << endl;
4938                };
4939            };
4940        }
4941        else {
4942            cout << "  Das duerfte am Anfang nicht sein. (b)   " << (liPos->
                count) << endl;
4943        };
4944        if ( liPos->p3 != 0 ) {
4945            number = liPos->p3->number;
4946            em = liPos->p3->emark;
4947            em2 = liPos->p3->emark2;
4948            x1 = liPos->p3->x;
4949            y1 = liPos->p3->y;
4950            deleteElement(liPos, number);
4951            insertElement(liPos, 1, x1, y1);
4952            liPos->first->next->emark = em;
4953            liPos->first->next->emark2 = em2;
4954        };
4955
4956        for ( i = 0; i < zwLists->GetCount(); i++ ) {
4957            (zwLists->GetList(i))->p1 = (zwLists->GetList(i))->first;
4958        };
4959
4960        bool gefunden = false;
4961        bool bL = true;
4962        int lNum = 0;
4963        int iL = 0;
4964        int x2 = 0;
4965        int y2 = 0;
4966        int vpos = 0;
4967        while ( bL ) {
4968            bL = false;
4969            iL = zwLists->GetCount();
4970            for ( lNum = 0; lNum < iL; lNum++ ) {
4971                liPos = zwLists->GetList(lNum);
4972                liPos->p3 = liPos->p1->next;
4973                x2 = liPos->p1->x;
4974                y2 = liPos->p1->y;
4975                while ( abs( liPos->p3->x - x2 ) > 1 || abs( liPos->p3->y - y2 )
                     > 1 ) {
4976                    liPos->p3 = liPos->p3->next;
4977                    if ( liPos->p3 == 0 ) {
```

```
4978                        if  ( abs( liPos->first->x - x2 ) > 1 || abs( liPos->
                                first->y - y2 ) > 1 || liPos->hmark == 2 ) {
4979                            liPos->hmark = 1;
4980                        }
4981                        else {
4982                            liPos->hmark = 2;
4983                        };
4984                        break;
4985                    };
4986                };
4987                if ( liPos->p3 != 0 ) {
4988                    number = liPos->p3->number;
4989                    bL = true;
4990                    for ( i = 2; i < liPix[x2 - xmin][y2 - ymin][0]; i++ ) {
4991                        zwLists->InsertList(copyList(liPos), -1);
4992                        liPos2 = zwLists->GetList(-1);
4993                        while ( liPos2->p3->number != number + 1 ) {
4994                            liPos2->p3 = liPos2->p3->next;
4995                            if ( liPos2->p3 == 0 ) {
4996                                if  ( abs( liPos2->first->x - x2 ) > 1 || abs(
                                        liPos2->first->y - y2 ) > 1 || liPos2->hmark
                                         == 2 ) {
4997                                    liPos2->hmark = 1;
4998                                }
4999                                else {
5000                                    liPos2->hmark = 2;
5001                                };
5002                                break;
5003                            };
5004                        };
5005                        if ( liPos2->p3 != 0 ) {
5006                            while ( abs( liPos2->p3->x - x2 ) > 1 || abs( liPos2
                                    ->p3->y - y2 ) > 1 ) {
5007                                liPos2->p3 = liPos2->p3->next;
5008                                if ( liPos2->p3 == 0 ) {
5009                                    liPos2->hmark = 1;
5010                                    break;
5011                                };
5012                            };
5013                        };
5014                        if ( liPos2->p3 != 0 ) {
5015                            x1 = liPos2->p3->x;
5016                            y1 = liPos2->p3->y;
5017                            number = liPos2->p3->number;
5018                            em = liPos2->p3->emark;
5019                            em2 = liPos2->p3->emark2;
5020                            deleteElement(liPos2, number);
5021                            insertElement(liPos2, liPos2->p1->number + 1, x1, y1
                                    );
5022                            liPos2->p1->next->emark = em;
5023                            liPos2->p1->next->emark2 = em2;
5024                            for ( j = static_cast<int>((hePix[x1 - xmin][y1 -
                                    ymin]).size()) - 1; j >= 0 ; j-- ) {
5025                                zwHead = (hePix[x1 - xmin][y1 - ymin])[j];
5026                                for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
```

```
5027                                    for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5028                                        if ( (hePix[ix][iy]).size() > 0 ) {
5029                                            vpos = 0;
5030                                            while ( vpos < (hePix[ix][iy]).size
                                                    () && (hePix[ix][iy])[vpos] !=
                                                    zwHead ) {
5031                                                vpos++;
5032                                            };
5033                                            if ( vpos != (hePix[ix][iy]).size()
                                                    ) {
5034                                                (hePix[ix][iy]).erase((hePix[ix
                                                        ][iy]).begin()+vpos);
5035                                            };
5036                                        };
5037                                    };
5038                                };
5039                                if ( zwHead->number < iL ) {
5040                                    iL--;
5041                                };
5042                                if ( zwHead->number < lNum ) {
5043                                    lNum--;
5044                                };
5045                                zwLists->DeleteList(zwHead);
5046                                zwHead = 0;
5047                            };
5048                            (hePix[x1 - xmin][y1 - ymin]).push_back(liPos2);
5049                            liPos2->p1 = liPos2->p1->next;
5050                        }
5051                        else {
5052                            zwLists->DeleteList(liPos2);
5053                        };
5054                    };
5055                    x1 = liPos->p3->x;
5056                    y1 = liPos->p3->y;
5057                    number = liPos->p3->number;
5058                    em = liPos->p3->emark;
5059                    em2 = liPos->p3->emark2;
5060                    deleteElement(liPos, number);
5061                    insertElement(liPos, liPos->p1->number + 1, x1, y1);
5062                    liPos->p1->next->emark = em;
5063                    liPos->p1->next->emark2 = em2;
5064                    if ( liPos->hmark != 2 ) {
5065                        for ( j = static_cast<int>((hePix[x1 - xmin][y1 - ymin])
                                .size()) - 1; j >= 0 ; j-- ) {
5066                            zwHead = (hePix[x1 - xmin][y1 - ymin])[j];
5067                            for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
5068                                for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5069                                    if ( (hePix[ix][iy]).size() > 0 ) {
5070                                        vpos = 0;
5071                                        while ( vpos < (hePix[ix][iy]).size() &&
                                                 (hePix[ix][iy])[vpos] != zwHead ) {
5072                                            vpos++;
5073                                        };
5074                                        if ( vpos != (hePix[ix][iy]).size() ) {
```

```
5075                                        (hePix[ix][iy]).erase((hePix[ix][iy
                                                ]).begin()+vpos);
5076                                    };
5077                                };
5078                            };
5079                        };
5080                        if ( zwHead->number < iL ) {
5081                            iL--;
5082                        };
5083                        if ( zwHead->number < lNum ) {
5084                            lNum--;
5085                        };
5086                        zwLists->DeleteList(zwHead->number);
5087                        zwHead = 0;
5088                    };
5089                    (hePix[x1 - xmin][y1 - ymin]).push_back(liPos);
5090                };
5091                liPos->p1 = liPos->p1->next;
5092                if ( liPos->p1 == liPos->last ) {
5093                    bL = false;
5094                };
5095            };
5096        };
5097        if ( bL || liPos->p1 == liPos->last ) {
5098            i = 0;
5099            while ( i < zwLists->GetCount() ) {
5100                if ( (zwLists->GetList(i))->hmark == 1 ) {
5101                    zwHead = (zwLists->GetList(i));
5102                    for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
5103                        for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5104                            if ( (hePix[ix][iy]).size() > 0 ) {
5105                                vpos = 0;
5106                                while ( vpos < (hePix[ix][iy]).size() && (
                                        hePix[ix][iy])[vpos] != zwHead ) {
5107                                    vpos++;
5108                                };
5109                                if ( vpos != (hePix[ix][iy]).size() ) {
5110                                    (hePix[ix][iy]).erase((hePix[ix][iy]).
                                            begin()+vpos);
5111                                };
5112                            };
5113                        };
5114                    };
5115                    zwLists->DeleteList(i);
5116                }
5117                else {
5118                    i++;
5119                };
5120            };
5121        };
5122    };
5123    gefunden = false;
5124    for ( i = 0; i < zwLists->GetCount(); i++ ) {
5125        liPos = zwLists->GetList(i);
5126        if ( liPos->hmark == 2) {
```

```
5127                    gefunden = true;
5128                };
5129            };
5130            for ( i = 0; i < zwLists->GetCount(); i++ ) {
5131                liPos = zwLists->GetList(i);
5132                j = liPos->hmark;
5133                if ( gefunden ) {
5134                    if ( j != 2 ) {
5135                        zwHead = (zwLists->GetList(i));
5136                        for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
5137                            for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5138                                if ( (hePix[ix][iy]).size() > 0 ) {
5139                                    vpos = 0;
5140                                    while ( vpos < (hePix[ix][iy]).size() && (hePix[
                                        ix][iy])[vpos] != zwHead ) {
5141                                        vpos++;
5142                                    };
5143                                    if ( vpos != (hePix[ix][iy]).size() ) {
5144                                        (hePix[ix][iy]).erase((hePix[ix][iy]).begin
                                            ()+vpos);
5145                                    };
5146                                };
5147                            };
5148                        };
5149                        zwLists->DeleteList(i);
5150                        i--;
5151                    };
5152                }
5153                else {
5154                    liPos->hmark = 0;
5155                };
5156            };
5157            if ( liPos->p1 != liPos->last && liPos->p1 != 0 && !gefunden ) {
5158                bL = true;
5159            }
5160            else {
5161                bL = false;
5162            };
5163            while ( bL ) {
5164                bL = false;
5165                iL = zwLists->GetCount();
5166                for ( lNum = 0; lNum < iL; lNum++ ) {
5167                    liPos = zwLists->GetList(lNum);
5168                    liPos->p3 = liPos->p1->next;
5169                    x2 = liPos->first->x;
5170                    y2 = liPos->first->y;
5171                    while ( abs( liPos->p3->x - x2 ) > 1 || abs( liPos->p3->y - y2 )
                         > 1 ) {
5172                        liPos->p3 = liPos->p3->next;
5173                        if ( liPos->p3 == 0 ) {
5174                            liPos->hmark = 1;
5175                            break;
5176                        };
5177                    };
5178                    if ( liPos->p3 != 0 ) {
```

```
5179                    number = liPos->p3->number;
5180                    bL = true;
5181                    for ( i = 2; i < liPix[x2 - xmin][y2 - ymin][0]; i++ ) {
5182                        zwLists->InsertList(copyList(liPos), -1);
5183                        liPos2 = zwLists->GetList(-1);
5184                        while ( liPos2->p3->number != number + 1 ) {
5185                            liPos2->p3 = liPos2->p3->next;
5186                            if ( liPos2->p3 == 0 ) {
5187                                liPos2->hmark = 1;
5188                                break;
5189                            };
5190                        };
5191                        if ( liPos2->p3 != 0 ) {
5192                            while ( abs( liPos2->p3->x - liPos2->first->x ) > 1
                                    || abs( liPos2->p3->y - liPos2->first->y ) > 1 )
                                     {
5193                                liPos2->p3 = liPos2->p3->next;
5194                                if ( liPos2->p3 == 0 ) {
5195                                    liPos2->hmark = 1;
5196                                    break;
5197                                };
5198                            };
5199                        };
5200                        if ( liPos2->p3 != 0 ) {
5201                            x1 = liPos2->p3->x;
5202                            y1 = liPos2->p3->y;
5203                            number = liPos2->p3->number;
5204                            em = liPos2->p3->emark;
5205                            em2 = liPos2->p3->emark2;
5206                            deleteElement(liPos2, number);
5207                            insertElement(liPos2, 0, x1, y1);
5208                            liPos2->first->emark = em;
5209                            liPos2->first->emark2 = em2;
5210                            for ( j = static_cast<int>((hePix[x1 - xmin][y1 -
                                    ymin]).size()) - 1; j >= 0 ; j-- ) {
5211                                zwHead = (hePix[x1 - xmin][y1 - ymin])[j];
5212                                for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
5213                                    for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5214                                        if ( (hePix[ix][iy]).size() > 0 ) {
5215                                            vpos = 0;
5216                                            while ( vpos < (hePix[ix][iy]).size
                                                () && (hePix[ix][iy])[vpos] !=
                                                zwHead ) {
5217                                                vpos++;
5218                                            };
5219                                            if ( vpos != (hePix[ix][iy]).size()
                                                ) {
5220                                                (hePix[ix][iy]).erase((hePix[ix
                                                    ][iy]).begin()+vpos);
5221                                            };
5222                                        };
5223                                    };
5224                                };
5225                                if ( zwHead->number < iL ) {
5226                                    iL--;
```

```
5227                                    };
5228                                    if ( zwHead->number < lNum ) {
5229                                        lNum--;
5230                                    };
5231                                    zwLists->DeleteList(zwHead->number);
5232                                    zwHead = 0;
5233                                };
5234                                (hePix[x1 - xmin][y1 - ymin]).push_back(liPos2);
5235                            }
5236                            else {
5237                                zwLists->DeleteList(liPos2);
5238                            };
5239                        };
5240                        x1 = liPos->p3->x;
5241                        y1 = liPos->p3->y;
5242                        number = liPos->p3->number;
5243                        em = liPos->p3->emark;
5244                        em2 = liPos->p3->emark2;
5245                        deleteElement(liPos, number);
5246                        insertElement(liPos, 0, x1, y1);
5247                        liPos->first->emark = em;
5248                        liPos->first->emark2 = em2;
5249                        for ( j = static_cast<int>((hePix[x1 - xmin][y1 - ymin]).
                                size()) - 1; j >= 0 ; j-- ) {
5250                            zwHead = (hePix[x1 - xmin][y1 - ymin])[j];
5251                            for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
5252                                for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5253                                    if ( (hePix[ix][iy]).size() > 0 ) {
5254                                        vpos = 0;
5255                                        while ( vpos < (hePix[ix][iy]).size() && (
                                                hePix[ix][iy])[vpos] != zwHead ) {
5256                                            vpos++;
5257                                        };
5258                                        if ( vpos != (hePix[ix][iy]).size() ) {
5259                                            (hePix[ix][iy]).erase((hePix[ix][iy]).
                                                    begin()+vpos);
5260                                        };
5261                                    };
5262                                };
5263                            };
5264                            if ( zwHead->number < iL ) {
5265                                iL--;
5266                            };
5267                            if ( zwHead->number < lNum ) {
5268                                lNum--;
5269                            };
5270                            zwLists->DeleteList(zwHead);
5271                            zwHead = 0;
5272                        };
5273                        (hePix[x1 - xmin][y1 - ymin]).push_back(liPos);
5274                        if ( liPos->p1 == liPos->last ) {
5275                            bL = false;
5276                        };
5277                    };
5278                };
```

```
5279            if ( bL || liPos->p1 == liPos->last ) {
5280                i = 0;
5281                while ( i < zwLists->GetCount() ) {
5282                    if ( (zwLists->GetList(i))->hmark == 1 ) {
5283                        zwHead = (zwLists->GetList(i));
5284                        for ( int ix = 0; ix < xmax-xmin+1; ix++ ) {
5285                            for ( int iy = 0; iy < ymax-ymin+1; iy++ ) {
5286                                if ( (hePix[ix][iy]).size() > 0 ) {
5287                                    vpos = 0;
5288                                    while ( vpos < (hePix[ix][iy]).size() && (
                                            hePix[ix][iy])[vpos] != zwHead ) {
5289                                        vpos++;
5290                                    };
5291                                    if ( vpos != (hePix[ix][iy]).size() ) {
5292                                        (hePix[ix][iy]).erase((hePix[ix][iy]).
                                                begin()+vpos);
5293                                    };
5294                                };
5295                            };
5296                        };
5297                        zwLists->DeleteList(i);
5298                    }
5299                    else {
5300                        i++;
5301                    };
5302                };
5303            };
5304        };
5305
5306        while ( zwLists->GetCount() > 1 ) {
5307            zwLists->DeleteList(1);
5308        };
5309
5310        liPos = zwLists->GetList(0);
5311        while ( liPos->p1 != liPos->last ) {
5312            gefunden = false;
5313            x1 = liPos->last->x;
5314            y1 = liPos->last->y;
5315            liPos->p3 = liPos->first;
5316            while ( ( abs ( liPos->p3->x - x1 ) > 1 || abs ( liPos->p3->y - y1 )
                     > 1 ) && !gefunden ) {
5317                if ( abs ( liPos->p3->next->x - x1 ) > 1 || abs ( liPos->p3->
                        next->y - y1 ) > 1 ) {
5318                    liPos->p3 = liPos->p3->next;
5319                    if ( liPos->p3->next == 0 || liPos->p3 == liPos->p1 ) {
5320                        break;
5321                    };
5322                }
5323                else {
5324                    gefunden = true;
5325                    number = liPos->p3->next->number;
5326                    em = liPos->last->emark;
5327                    em2 = liPos->last->emark2;
5328                    insertElement(liPos, number, x1, y1);
5329                    liPos->p3->next->emark = em;
```

```
5330                        liPos->p3->next->emark2 = em2;
5331                    };
5332                };
5333                deleteElement(liPos, -1);
5334            };
5335
5336            if ( abs ( liPos->first->x - liPos->last->x ) > 1 || abs ( liPos->first
                    ->y - liPos->last->y ) > 1 ) {
5337                zwLists->InsertList(copyList(liPos), 1);
5338                liPos = zwLists->GetList(1);
5339                liPos->p1 = liPos->last;
5340                while ( abs ( liPos->first->x - liPos->p1->x ) > 1 || abs ( liPos->
                        first->y - liPos->p1->y ) > 1 ) {
5341                    liPos->p1 = liPos->p1->prev;
5342                };
5343                if ( liPos->p1->number > 8 ) {
5344                    while ( liPos->p1 != 0 ) {
5345                        liPos->p2 = liPos->p1->next;
5346                        x1 = liPos->p1->x;
5347                        y1 = liPos->p1->y;
5348                        em = liPos->p1->emark;
5349                        em2 = liPos->p1->emark2;
5350                        number = liPos->p1->number;
5351                        deleteElement(liPos, number);
5352                        insertElement(liPos, 0, x1, y1);
5353                        liPos->first->emark = em;
5354                        liPos->first->emark2 = em2;
5355                        liPos->p1 = liPos->p2;
5356                    };
5357                    if ( abs ( liPos->first->x - liPos->last->x ) > 1 || abs ( liPos
                            ->first->y - liPos->last->y ) > 1 ) {
5358                        printf("%c%c%c", 7, 7, 7);
5359                        copyList(zwLists->GetList(0), listHead1);
5360                    }
5361                    else {
5362                        copyList(zwLists->GetList(1), listHead1);
5363                    };
5364                }
5365                else {
5366                    copyList(zwLists->GetList(1), listHead1);
5367                };
5368            }
5369            else {
5370                copyList(zwLists->GetList(0), listHead1);
5371            };
5372            delete zwLists;
5373        };
5374        cout << "  Ende Sortierung" << endl;
5375        return 0;
5376    };
```

# C Computers

**Generally used computer**
Dell Optiplex 740
AMD Athlon 64 X2 Dual Core Processor 3800+ 2.00 GHz
3.43 GB ($2 \times 2$ GB, DDR2) RAM physical address extension
Microsoft Windows XP Professional SP 3 (32 Bit)

**Computer used for Avizo and the fang measuring program**
Dell Precision T3400
Intel Core 2 Quad Q9550 2.83 GHz
16 GB (DDR2) RAM
Microsoft Windows Vista Business SP 2 (64 Bit)

# D 3D reconstruction of fangs

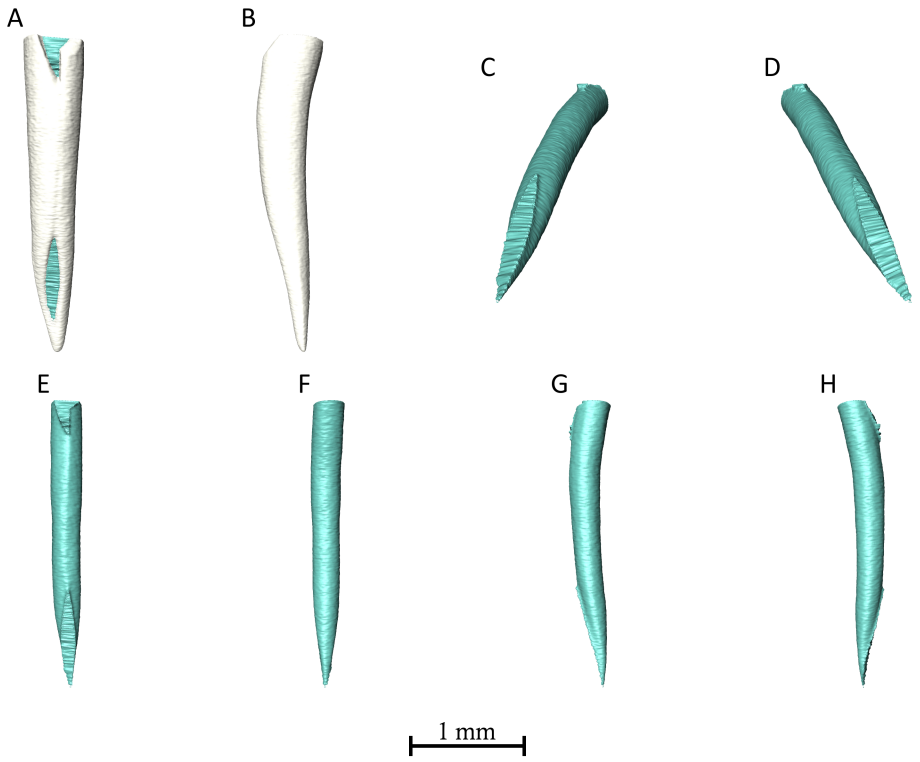


**Figure 53:** (*Dendroaspis angusticeps*) 3D reconstruction of fang ang1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1. The fang was broken and the basal part of the fang was not scanned.

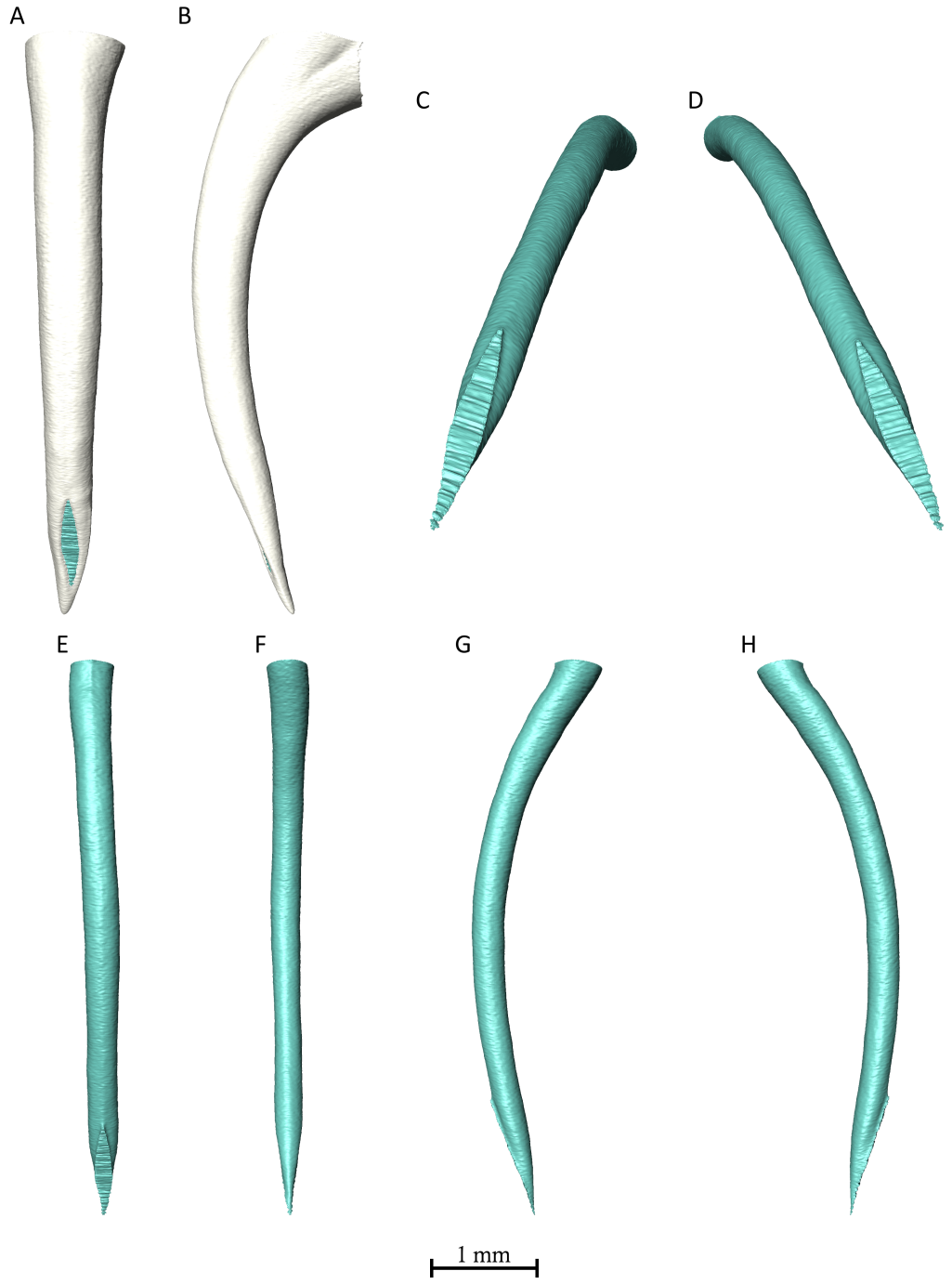


**Figure 54:** (*Dendroaspis angusticeps*) 3D reconstruction of fang ang2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

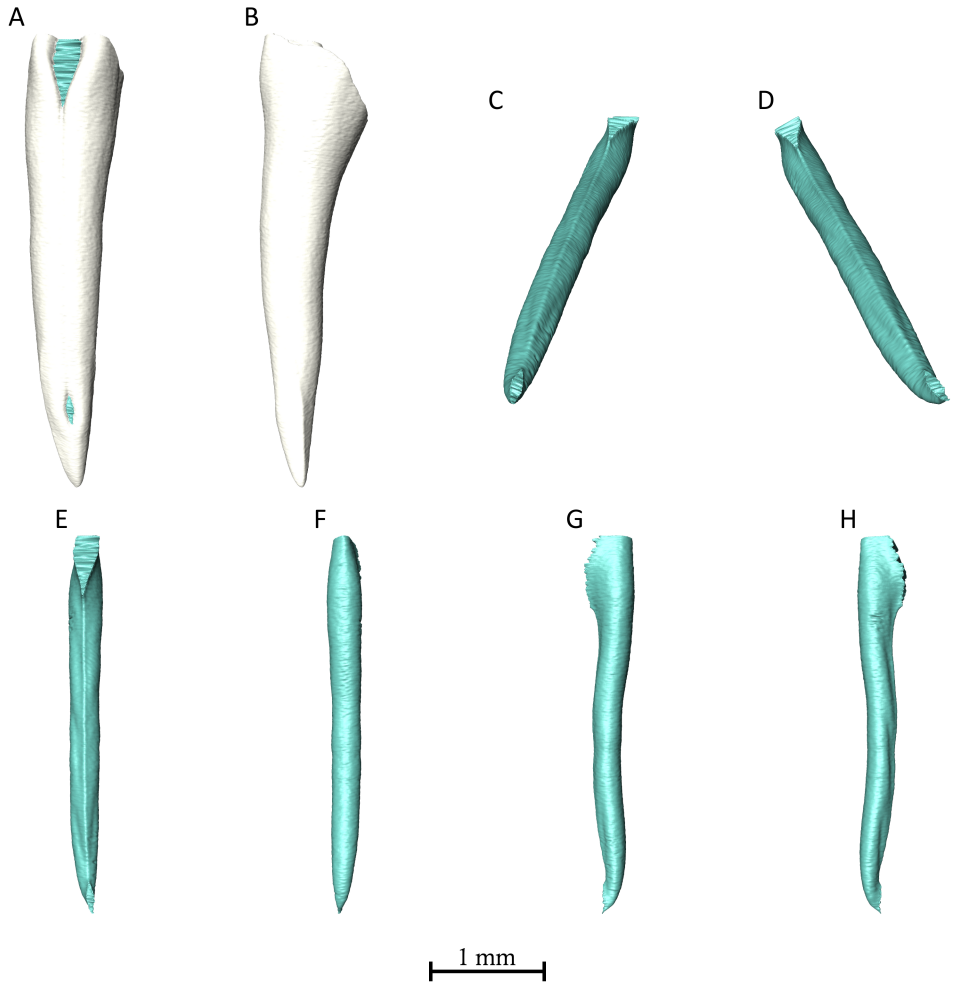


**Figure 55:** (*Hemachatus haemachatus*) 3D reconstruction of fang hae1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

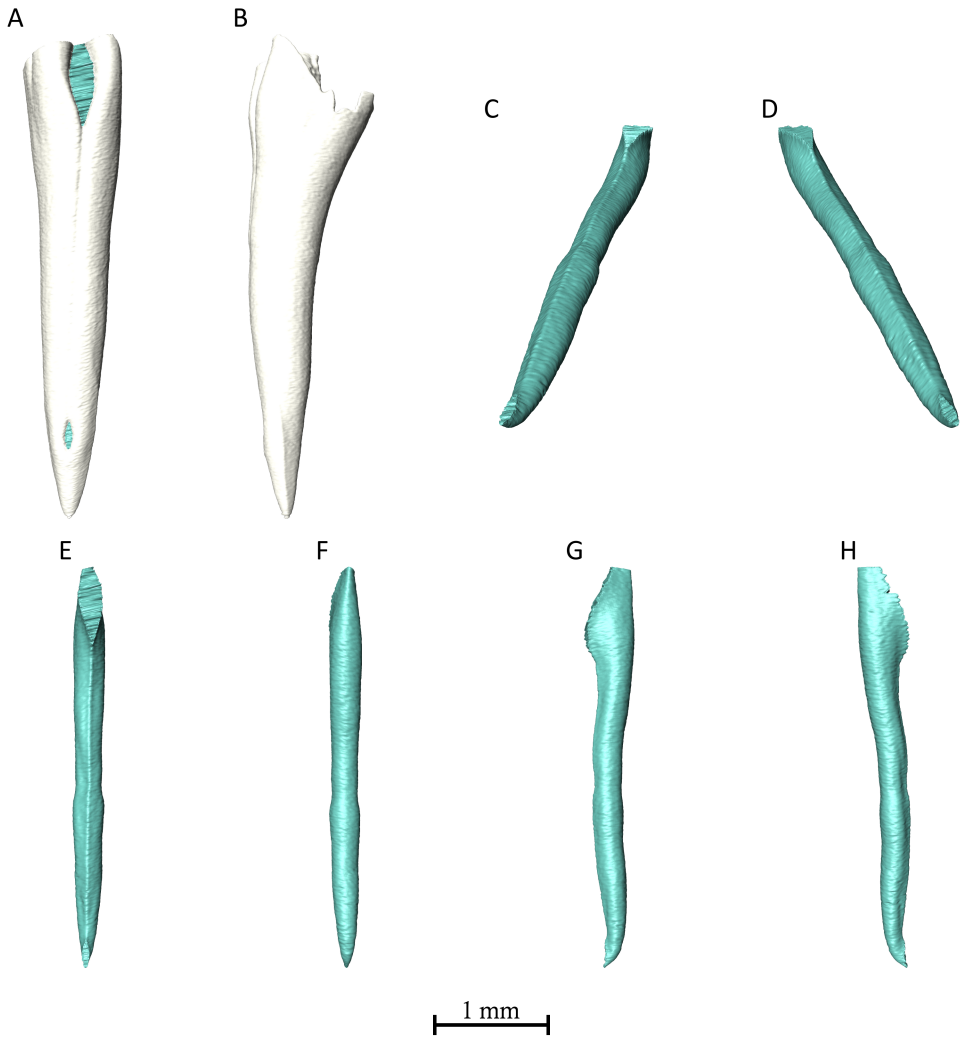


**Figure 56:** (*Hemachatus haemachatus*) 3D reconstruction of fang hae2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

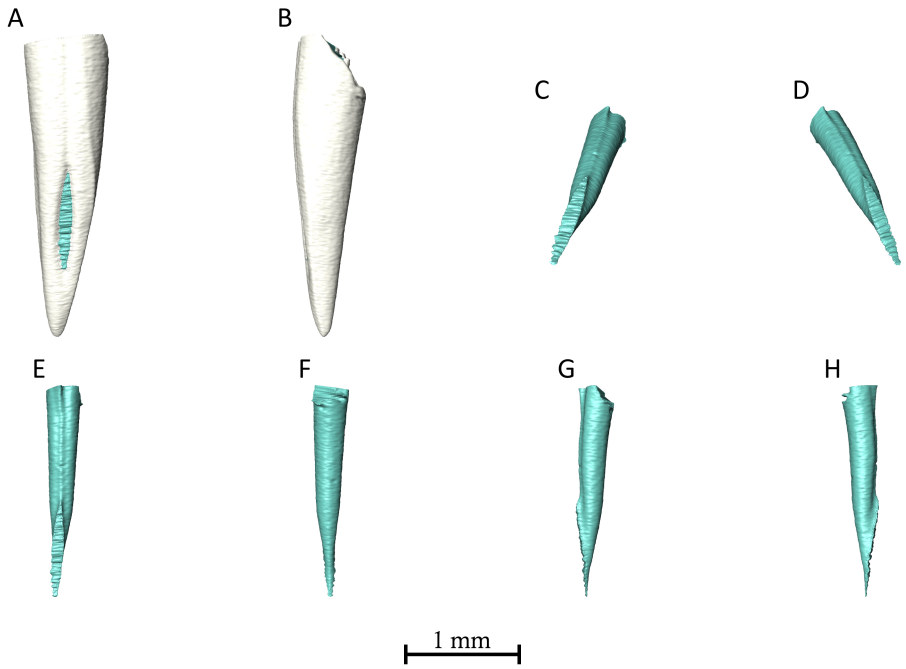


**Figure 57:** (*Naja haje*) 3D reconstruction of fang haj1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1. The fang was broken and the basal part of the fang was not scanned.

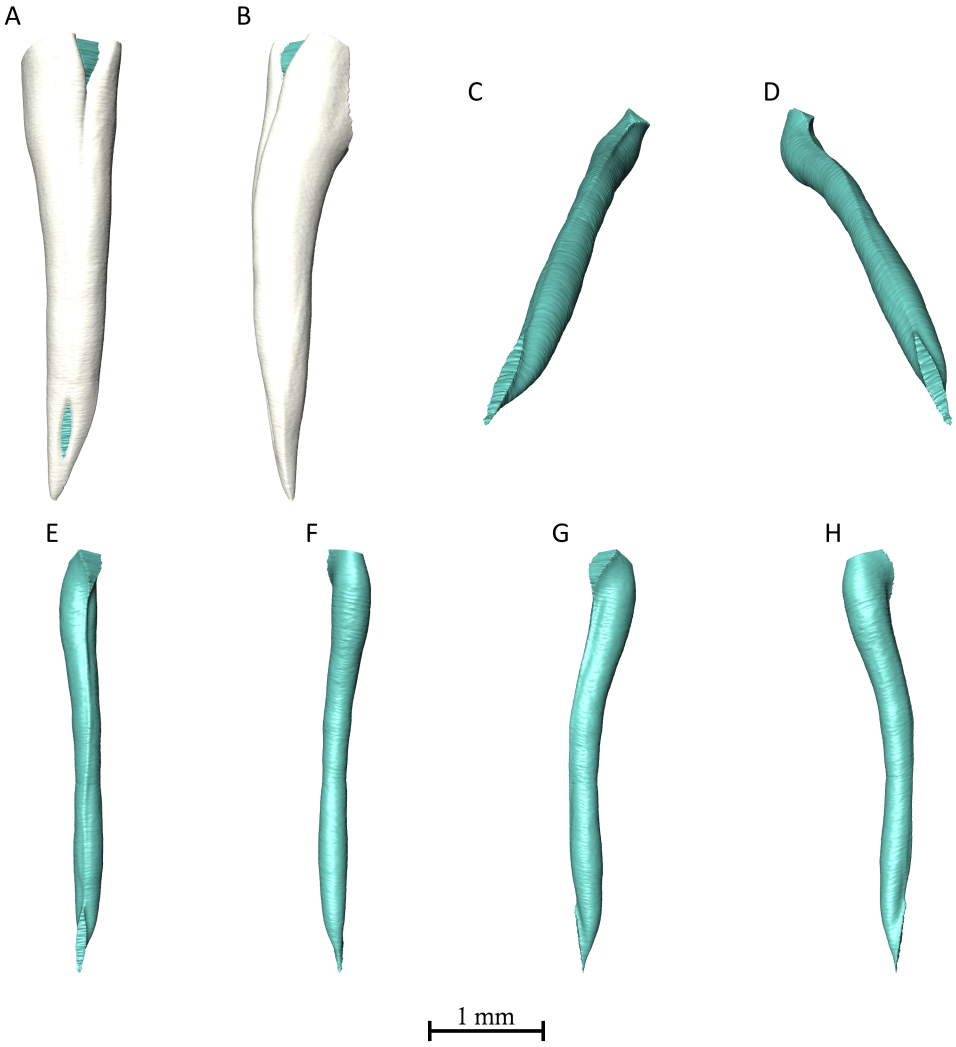


**Figure 58:** (*Naja kaouthia*) 3D reconstruction of fang kao1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast in this figure and figures 22 to 25. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

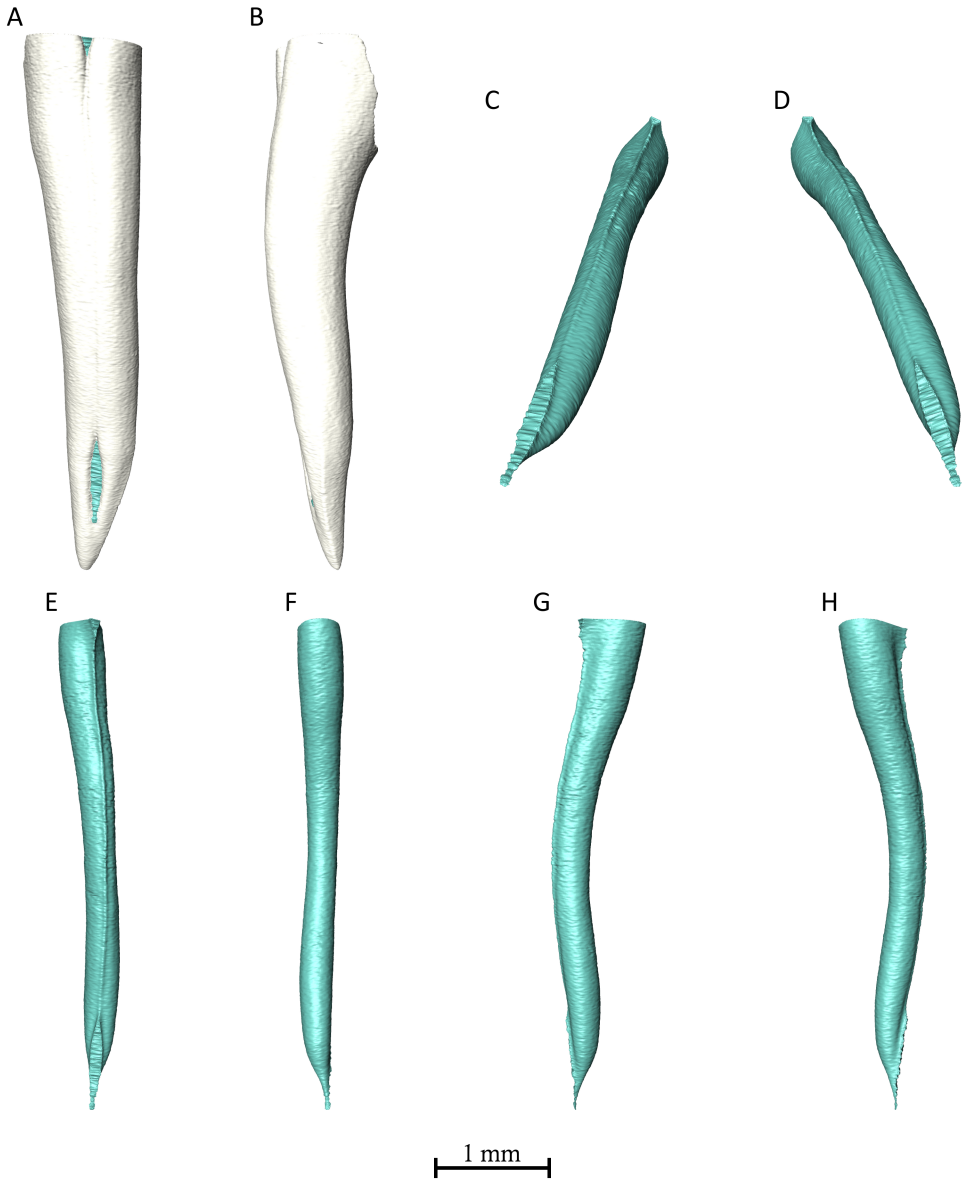


**Figure 59:** (*Naja kaouthia*) 3D reconstruction of fang kao2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

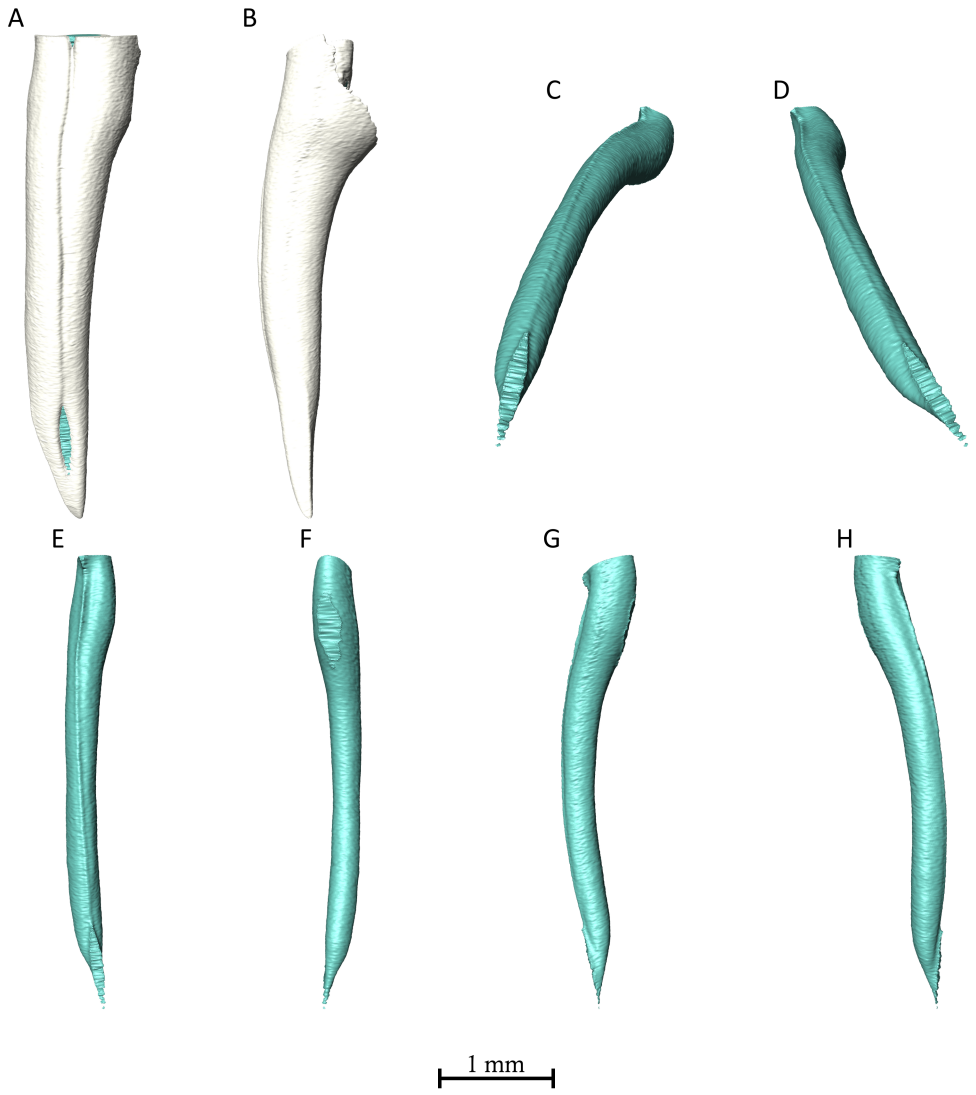


**Figure 60:** (*Naja kaouthia*) 3D reconstruction of fang kao3. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

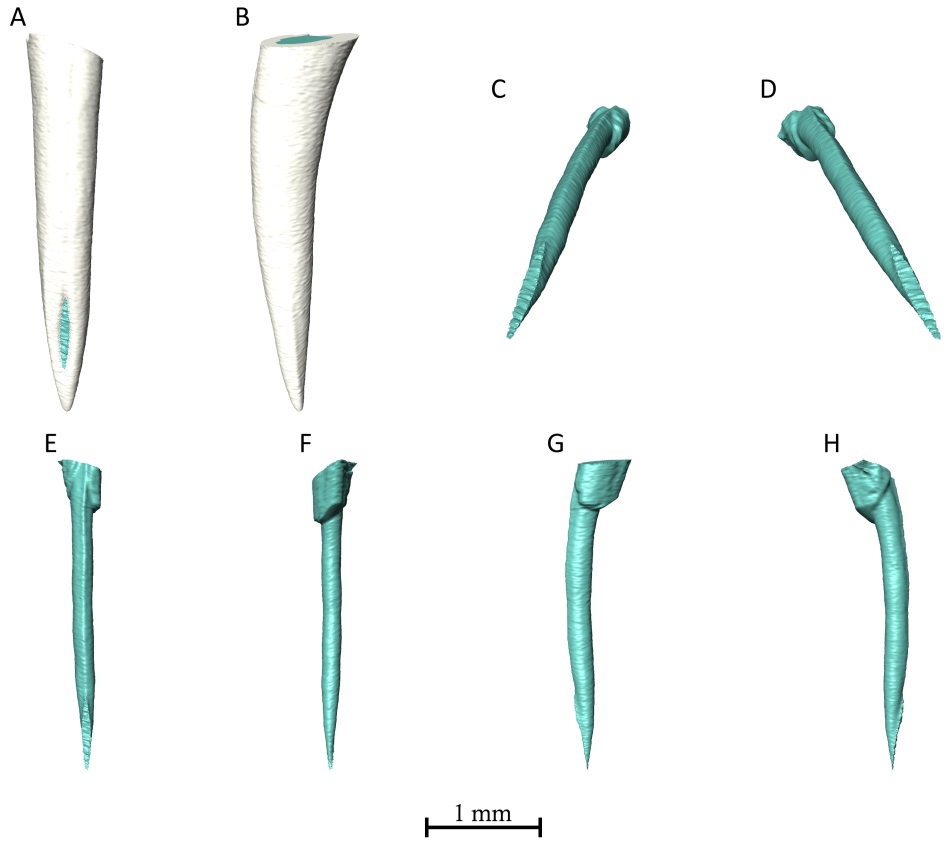


**Figure 61:** (*Naja melanoleuca*) 3D reconstruction of fang mel1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1. The apparent widening of the venom canal at its basal end is a result of the fang's basal part being broken.

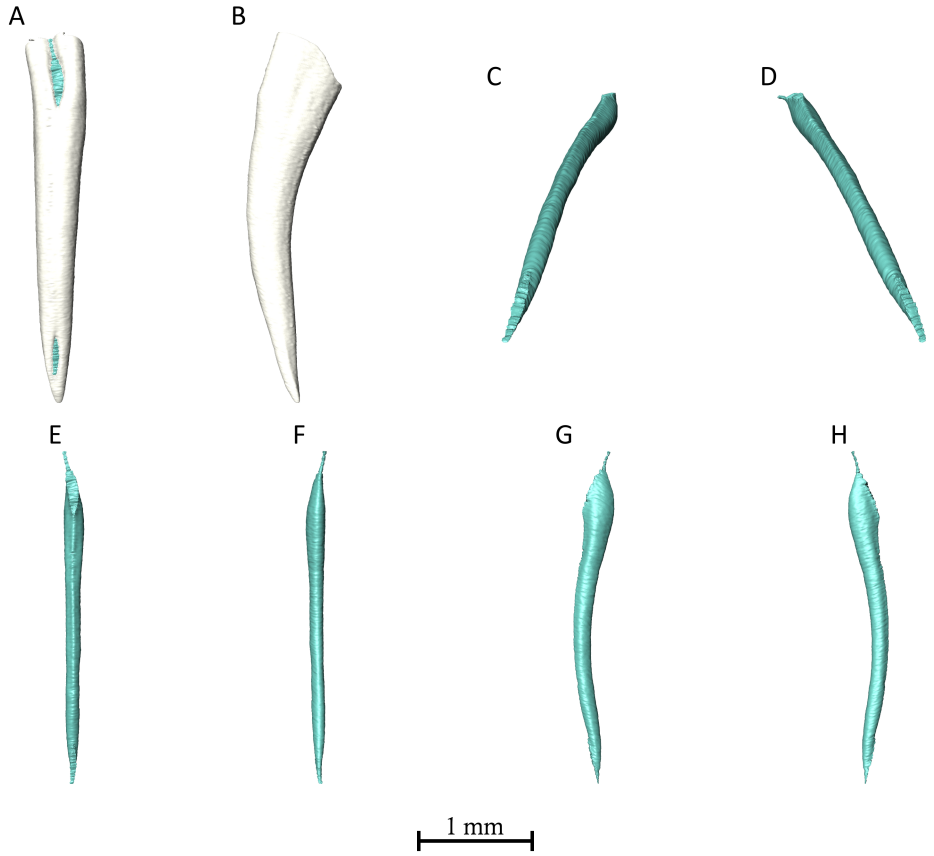


**Figure 62:** (*Naja melanoleuca*) 3D reconstruction of fang mel2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

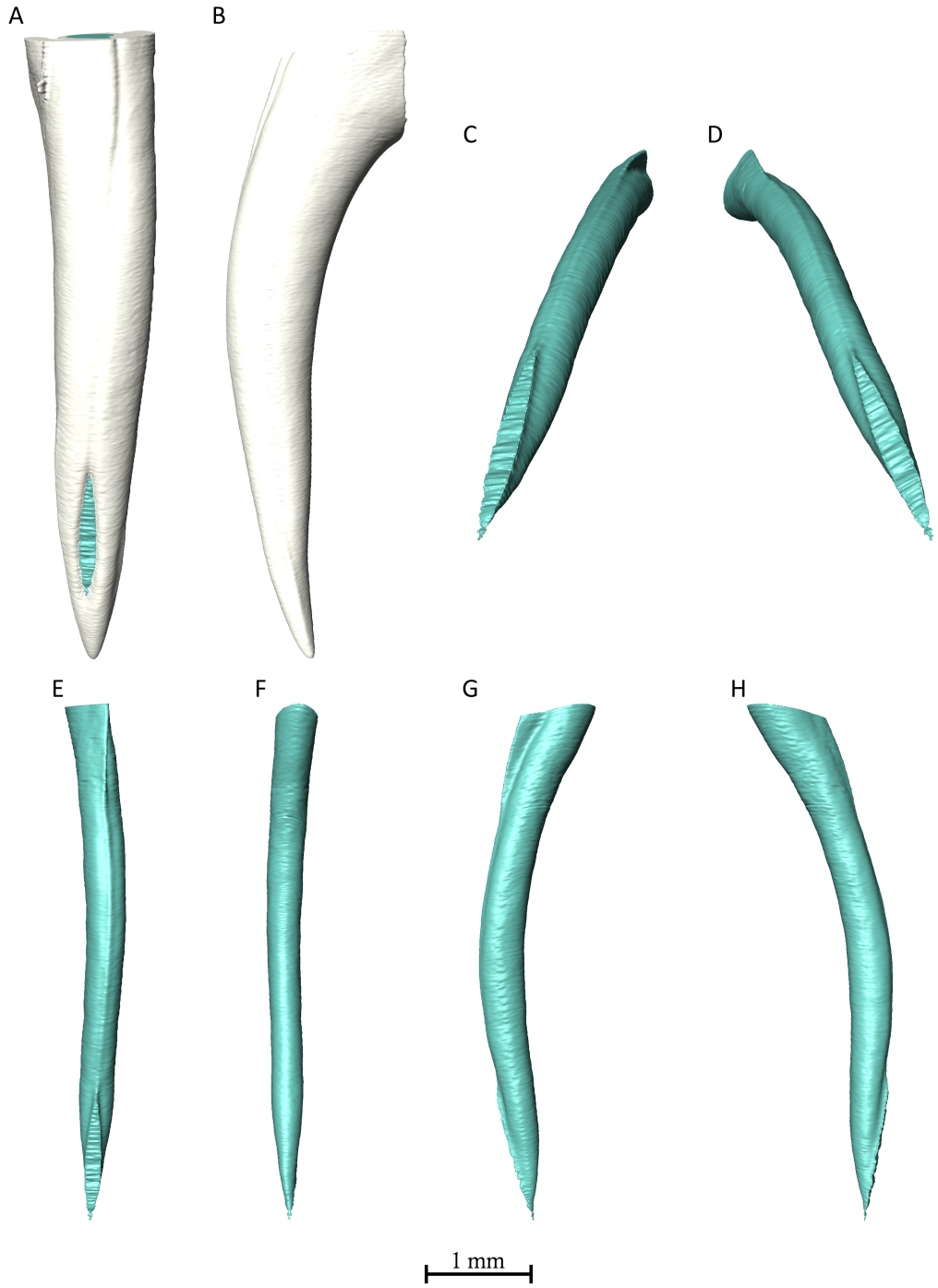


**Figure 63:** (*Naja naja*) 3D reconstruction of fang naj1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

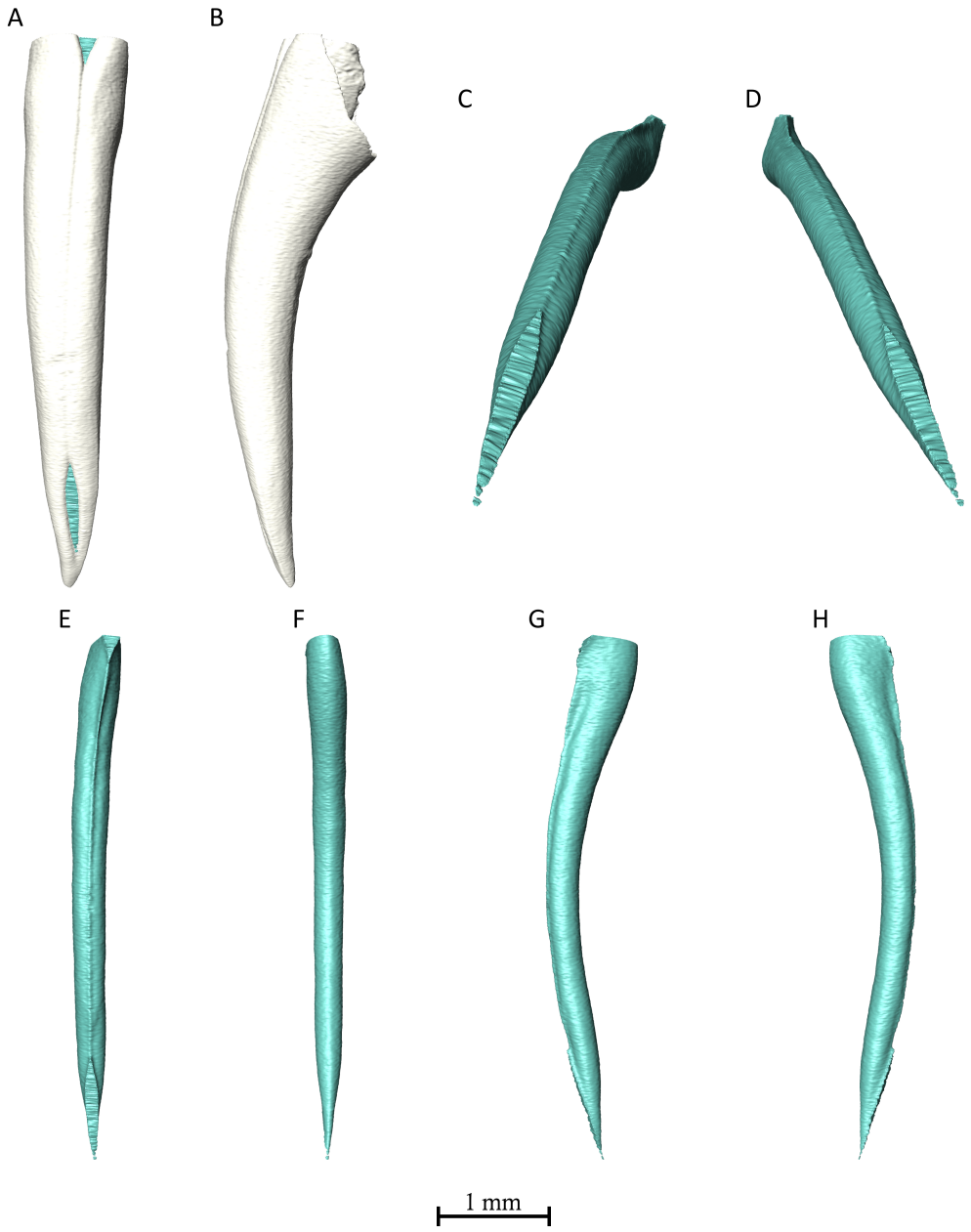


**Figure 64:** (*Naja naja*) 3D reconstruction of fang naj2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

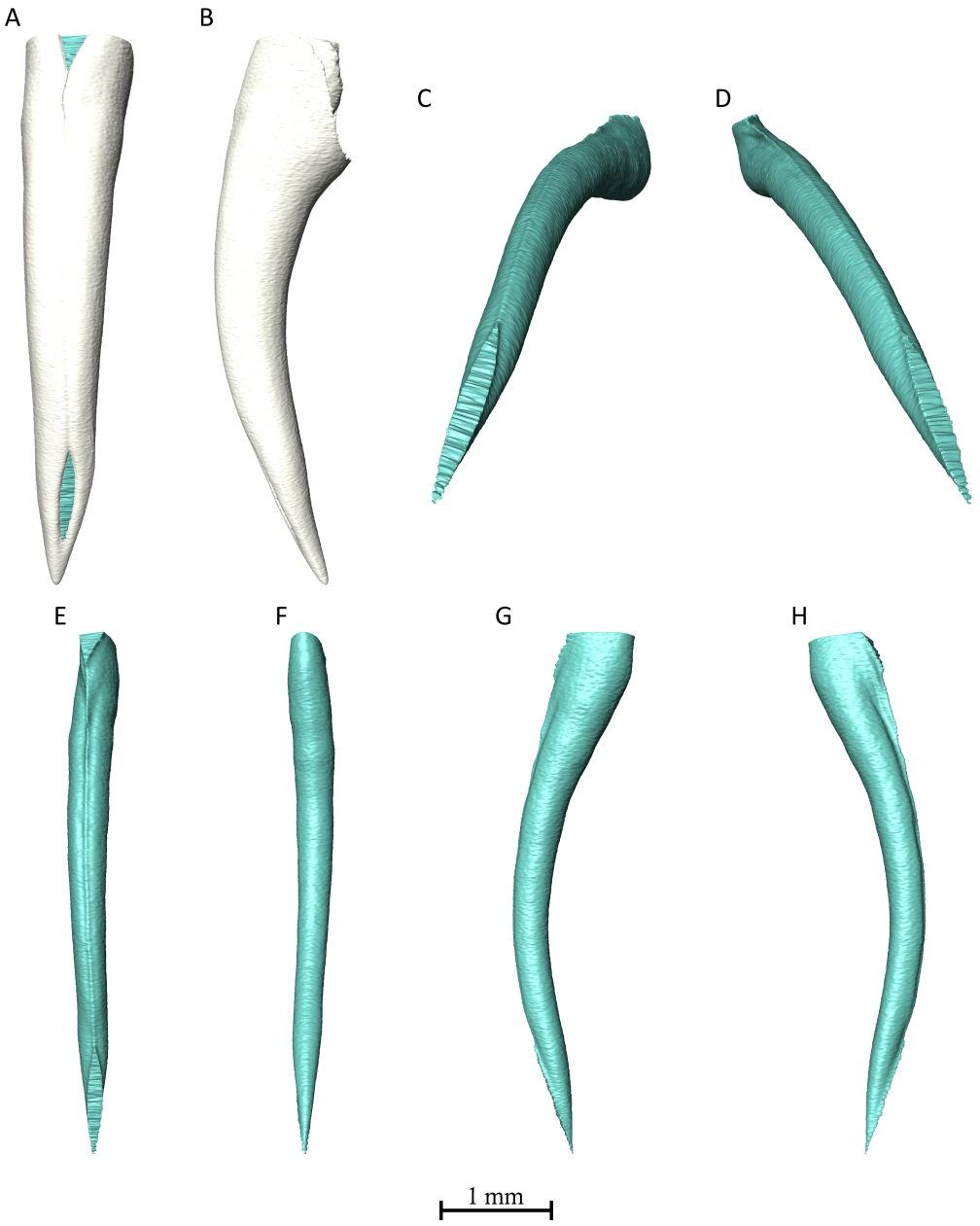


**Figure 65:** (*Naja naja*) 3D reconstruction of fang naj3. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

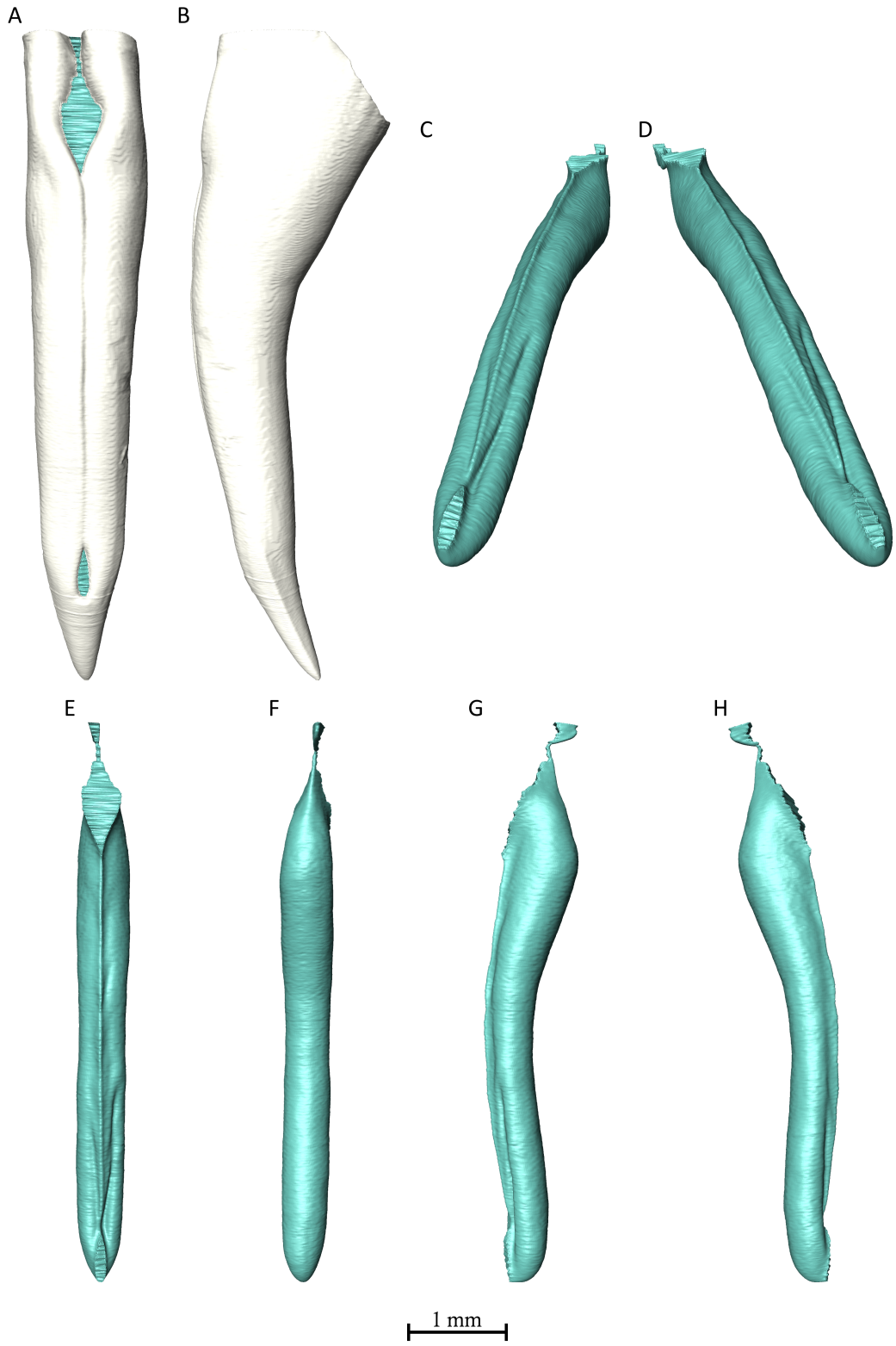


**Figure 66:** (*Naja nigricollis*) 3D reconstruction of fang nig1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

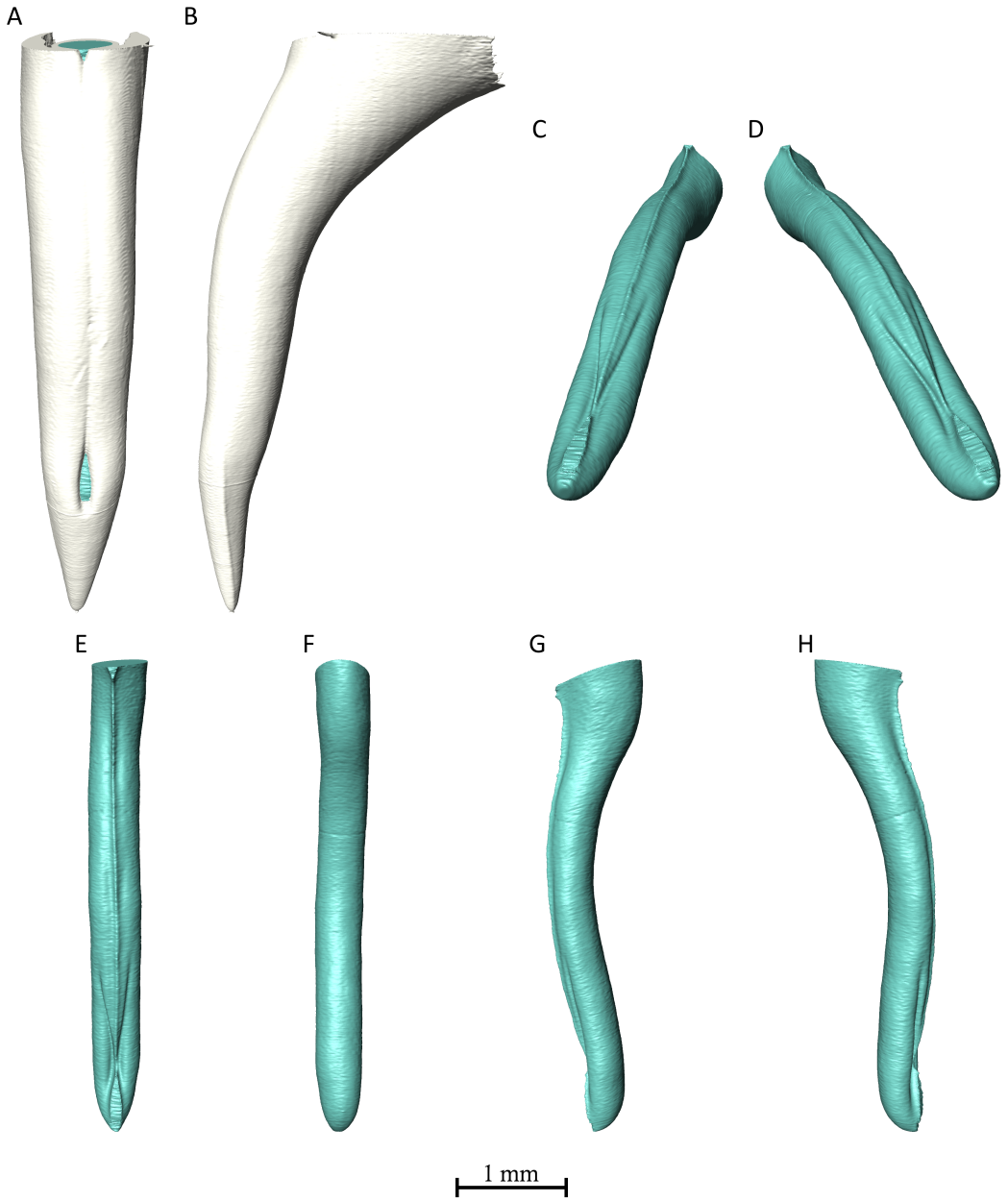


**Figure 67:** (*Naja nigricollis*) 3D reconstruction of fang nig2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

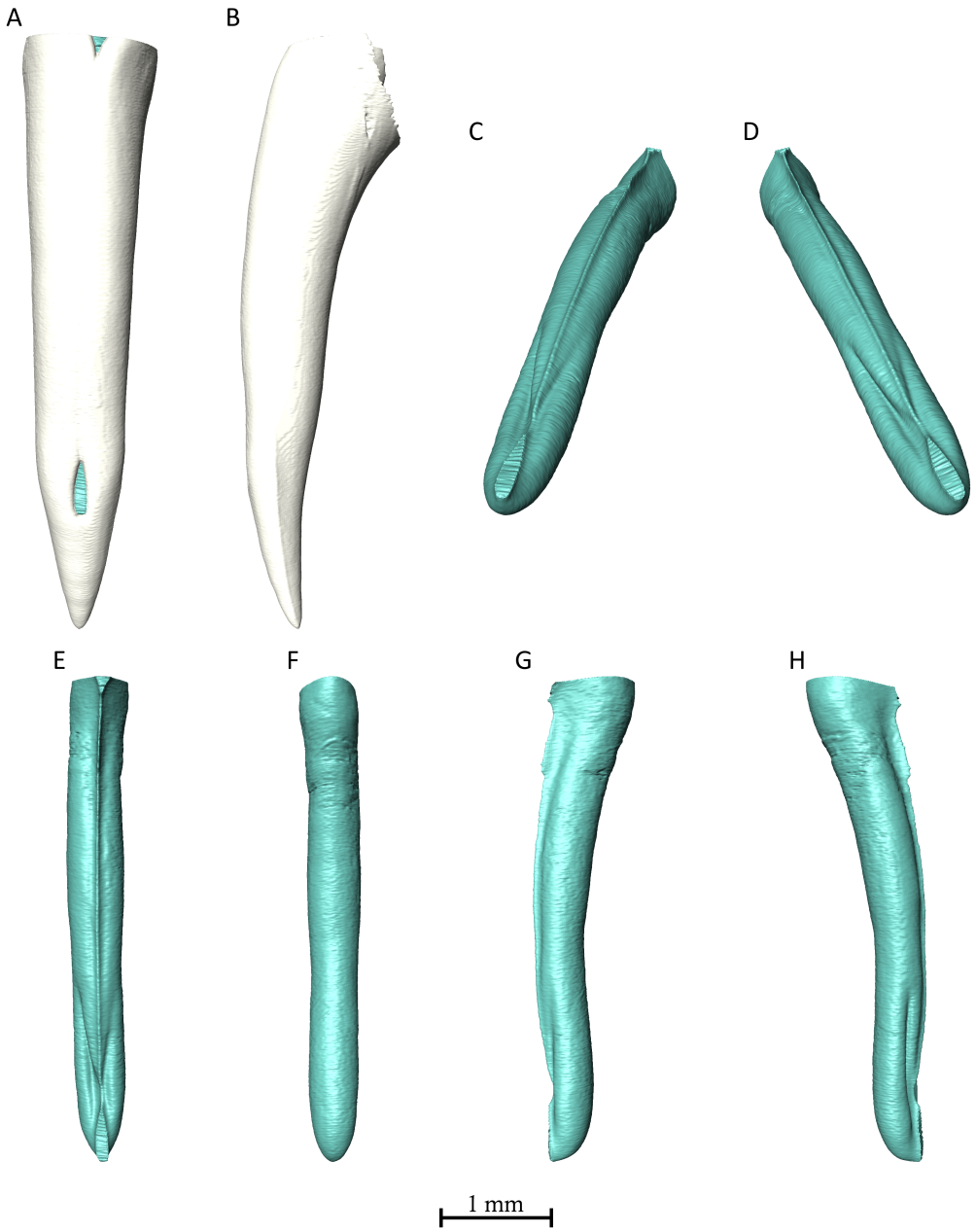


**Figure 68:** (*Naja nigricollis*) 3D reconstruction of fang nig3. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

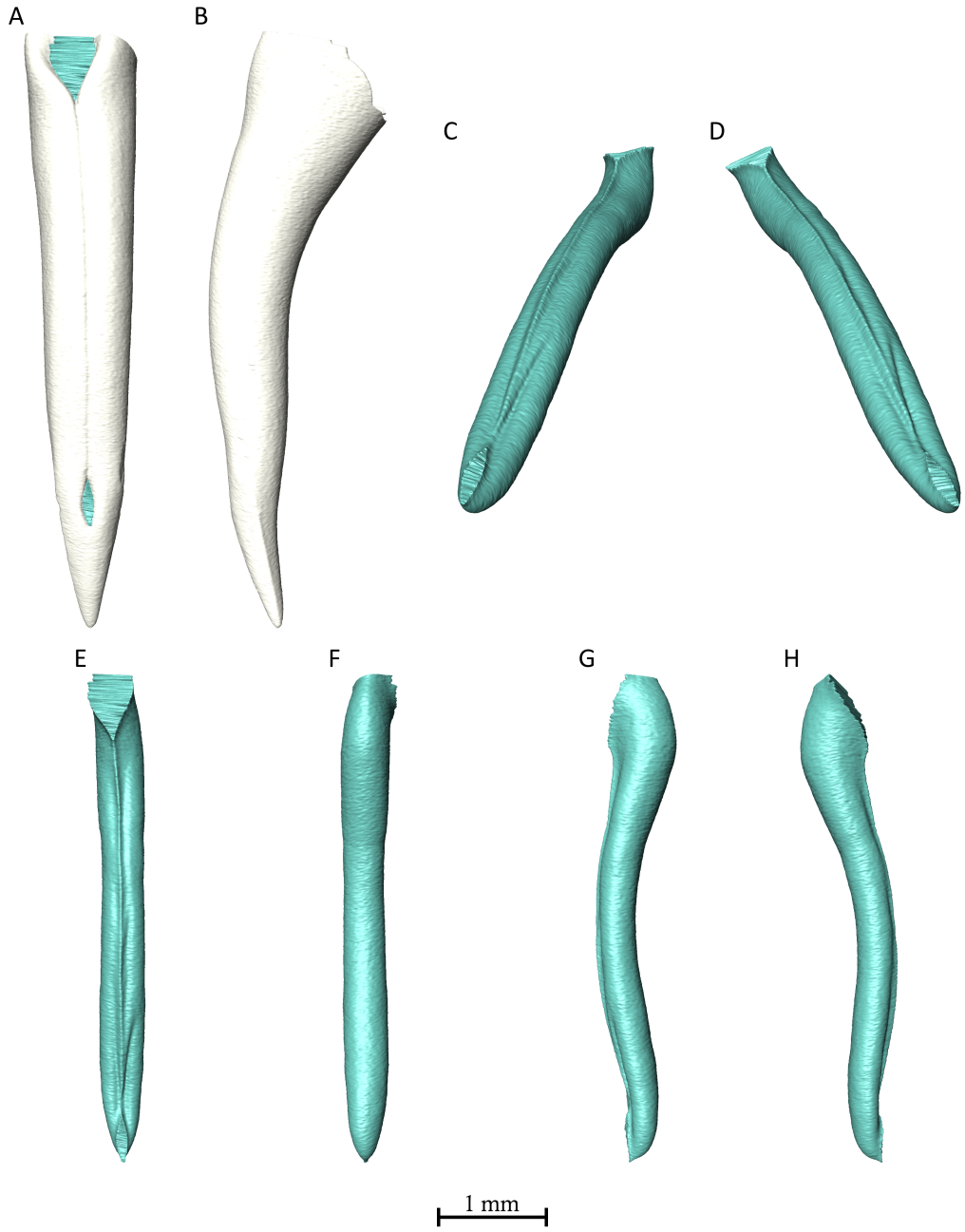


**Figure 69:** (*Naja nigricollis*) 3D reconstruction of fang nig4. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

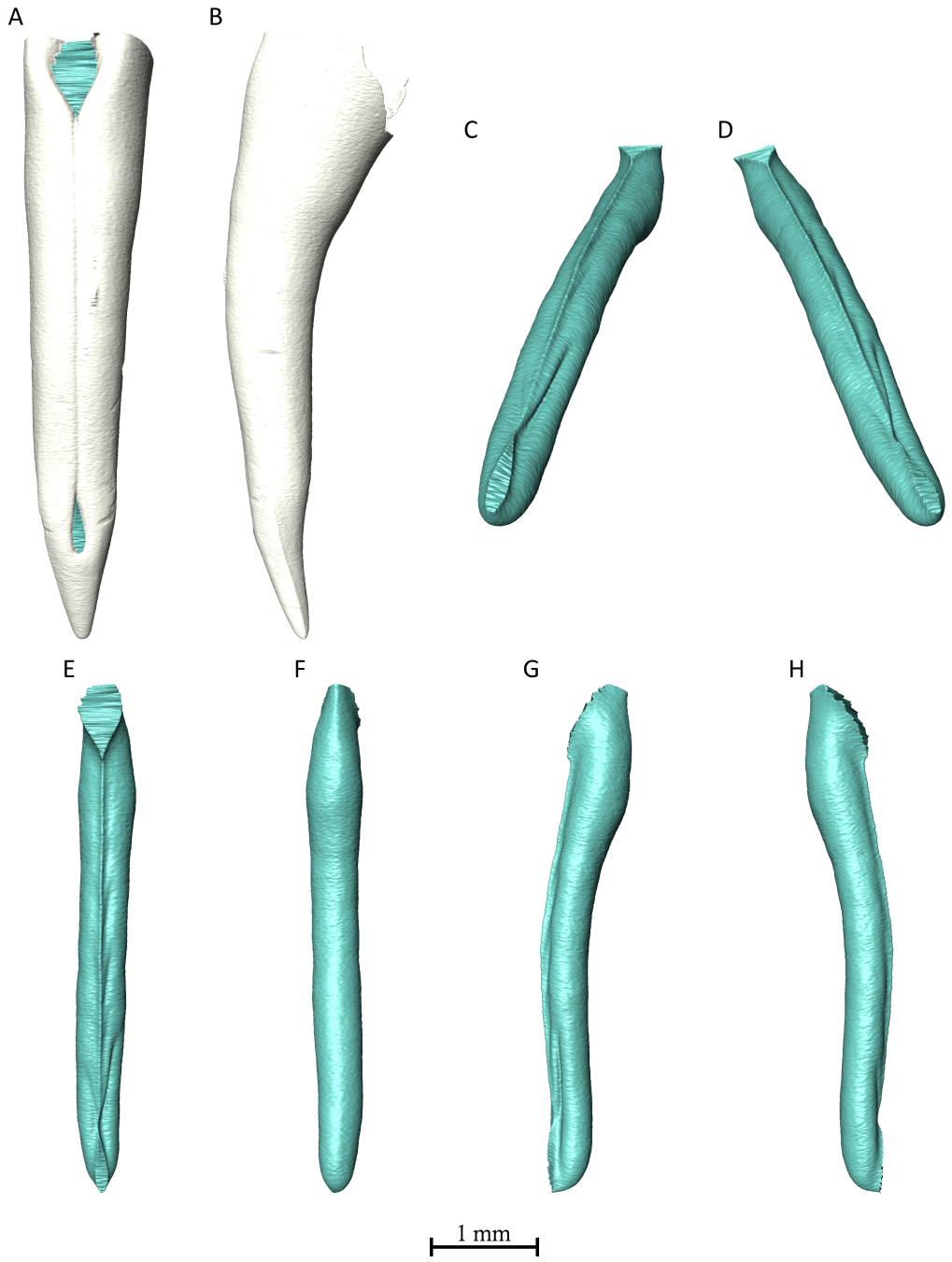


**Figure 70:** (*Naja nigricollis*) 3D reconstruction of fang nig5. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

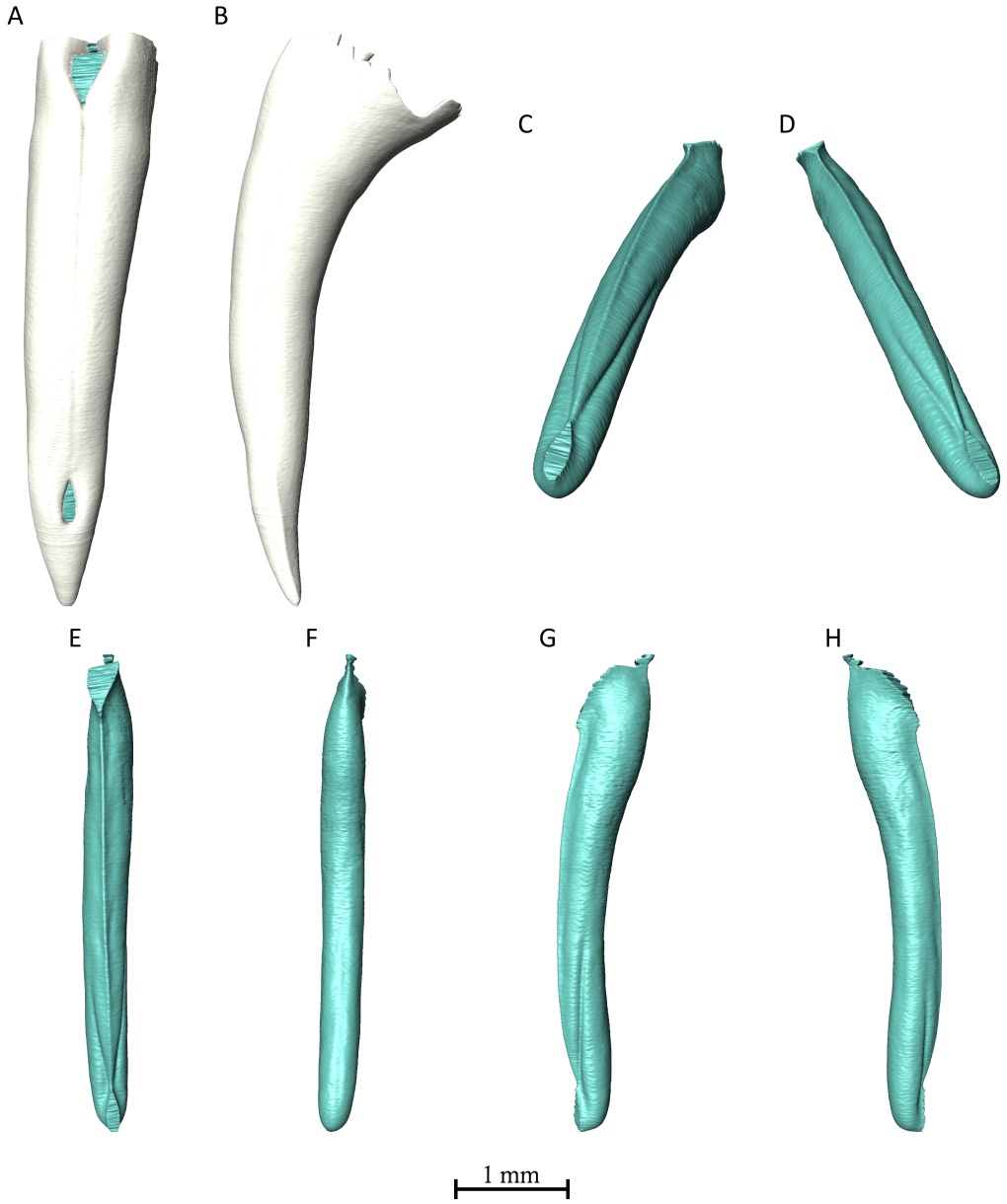


**Figure 71:** (*Naja pallida*) 3D reconstruction of fang pal1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

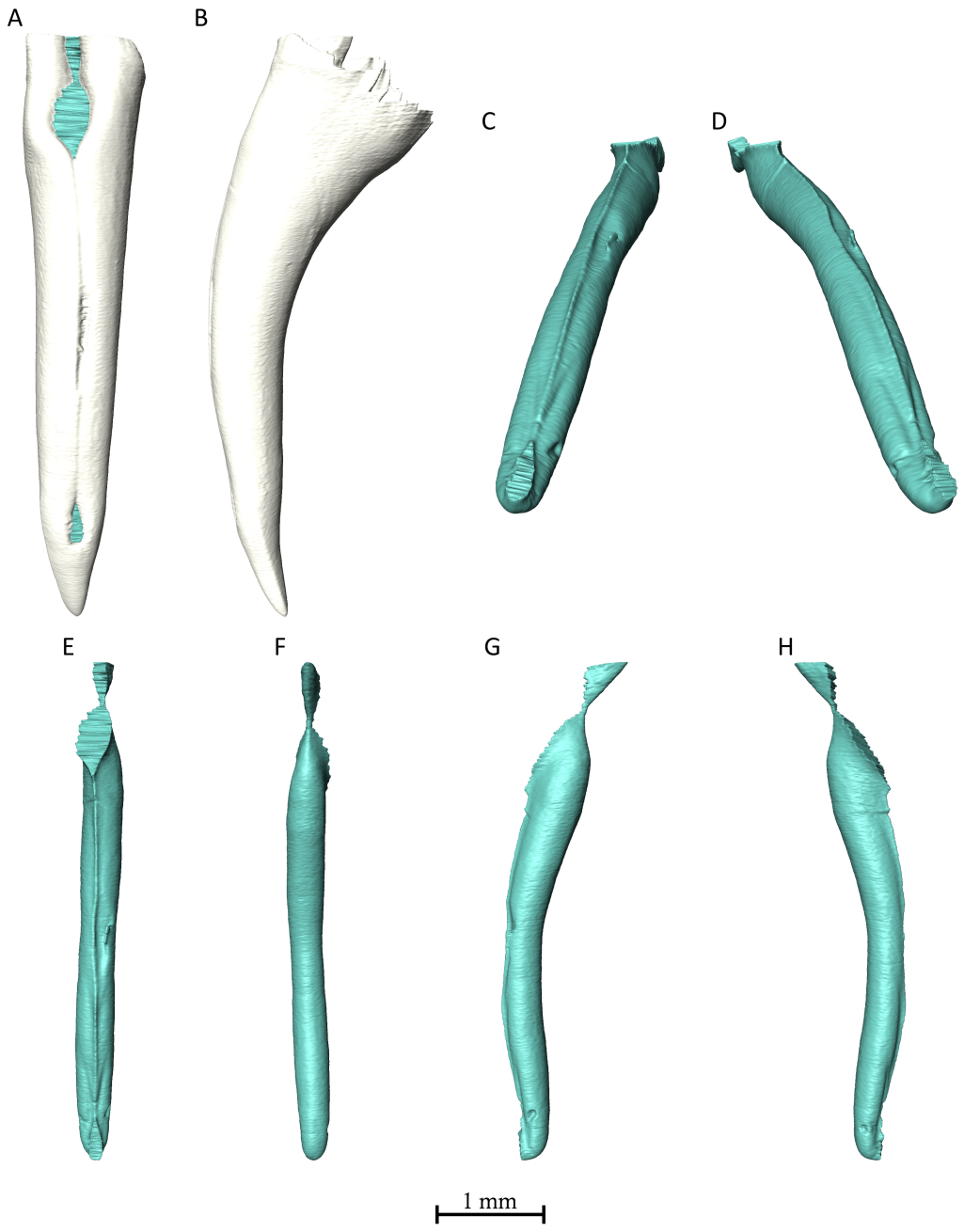


**Figure 72:** (*Naja pallida*) 3D reconstruction of fang pal2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

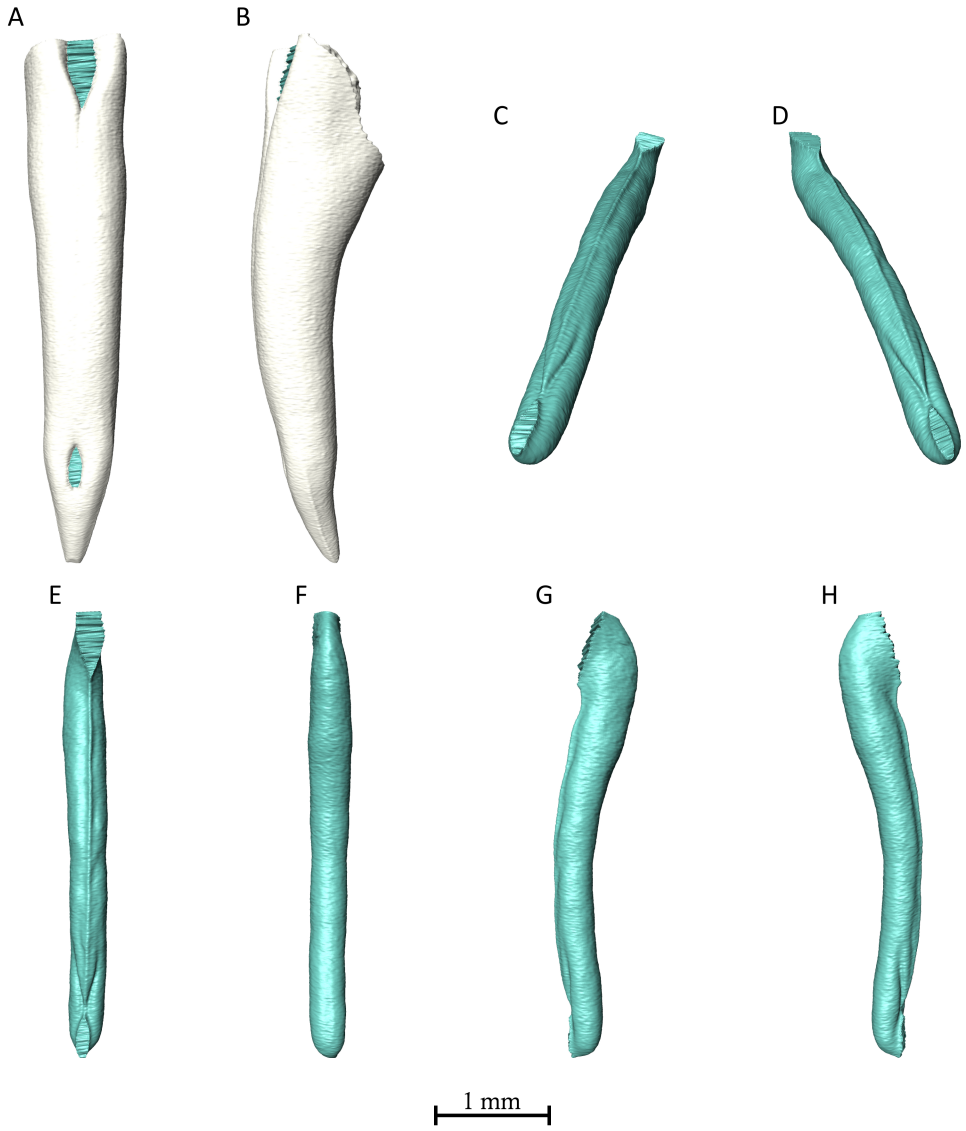


**Figure 73:** (*Naja pallida*) 3D reconstruction of fang pal3. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

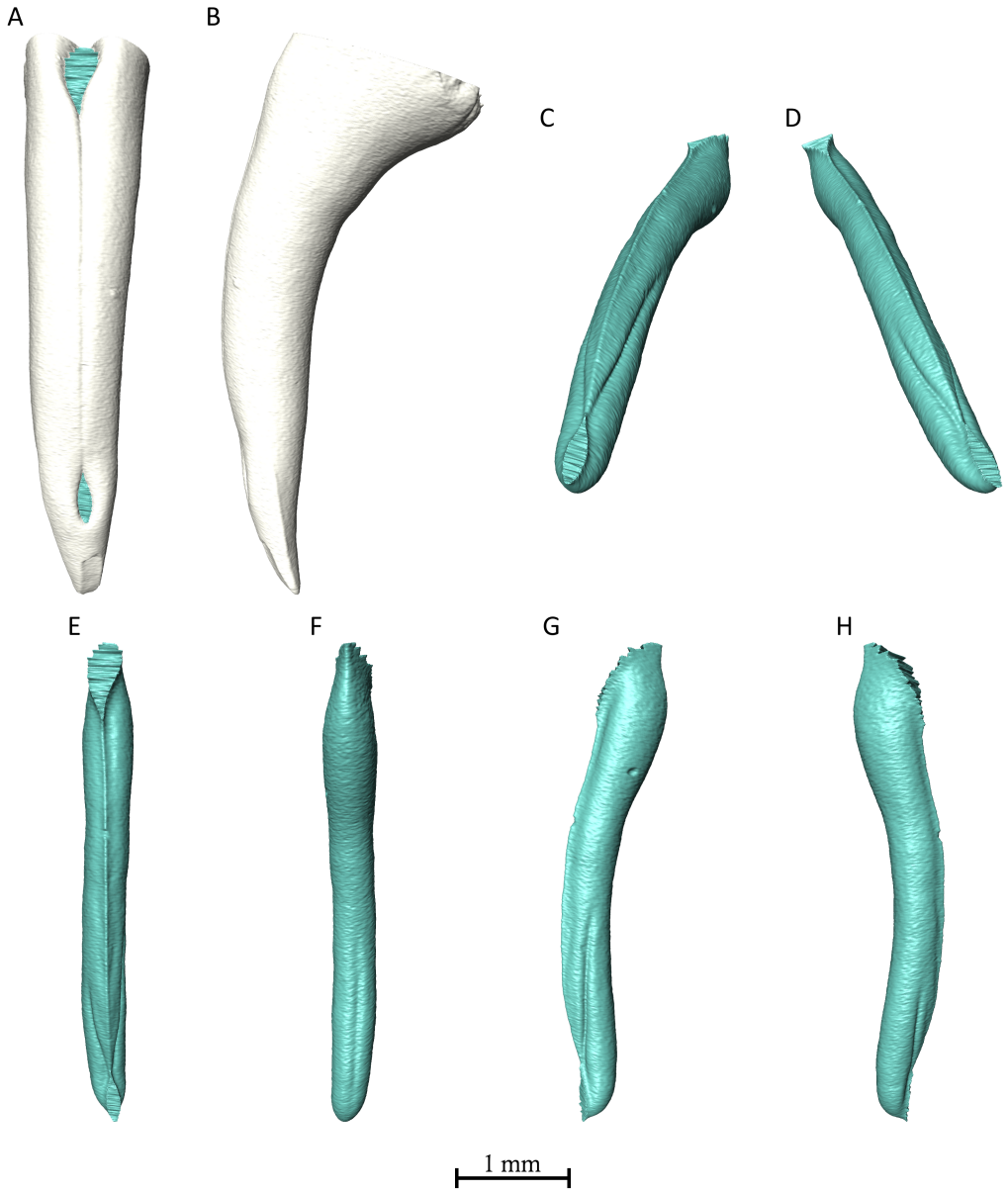


**Figure 74:** (*Naja pallida*) 3D reconstruction of fang pal4. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1. The fang tip is broken.

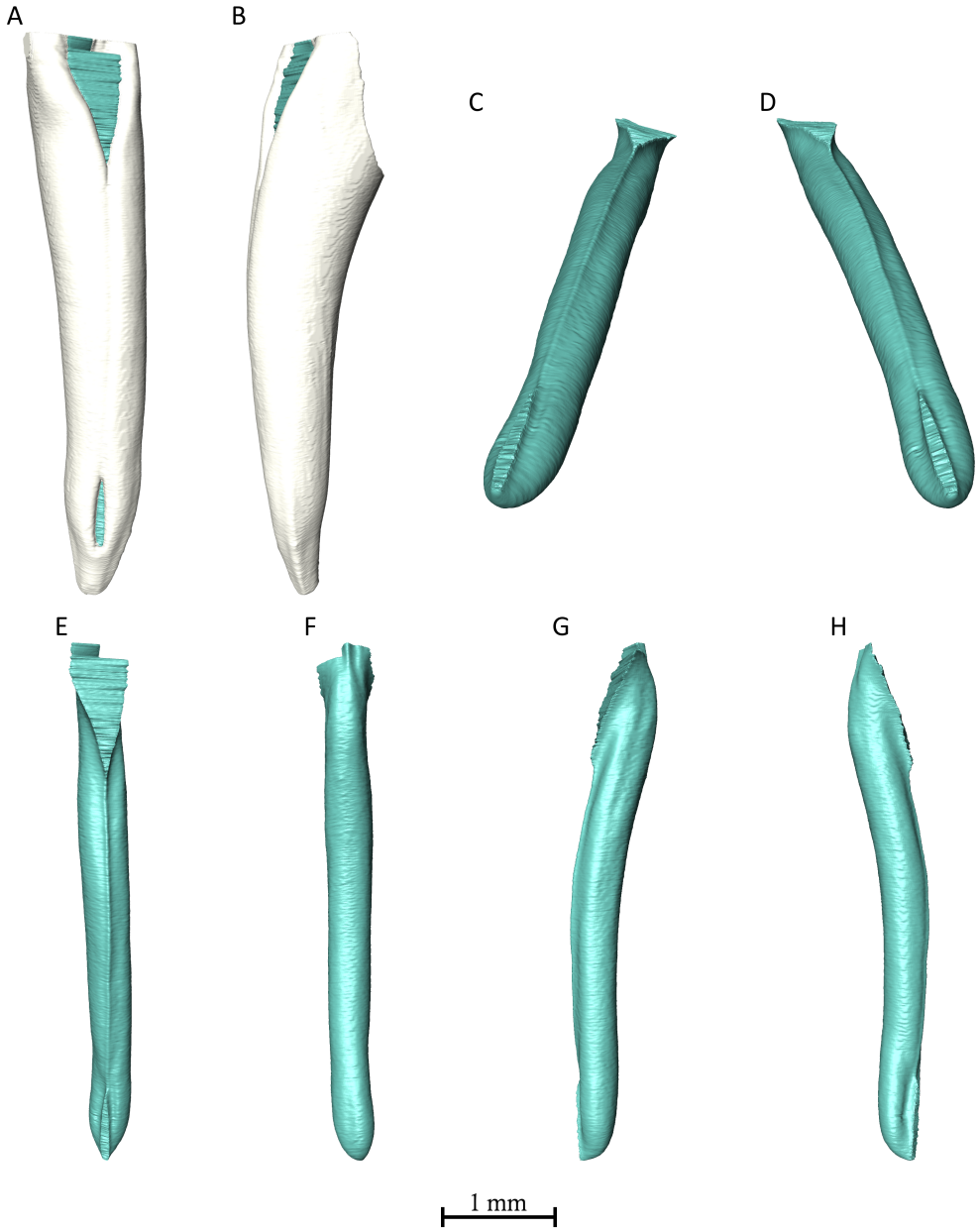


**Figure 75:** (*Naja siamensis*) 3D reconstruction of fang sia1. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1. The fang tip is broken.

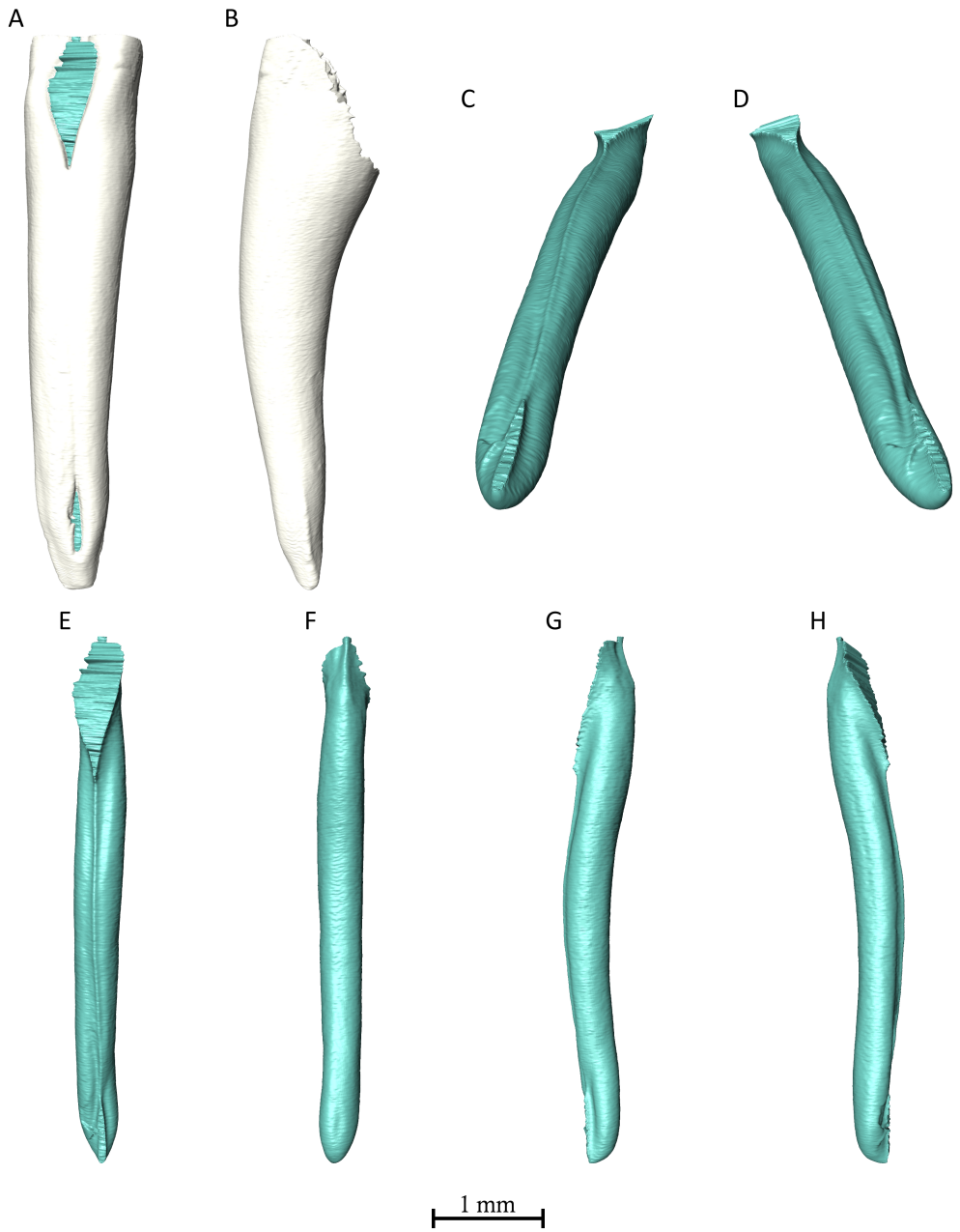


**Figure 76:** (*Naja siamensis*) 3D reconstruction of fang sia2. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1. The fang tip is broken.

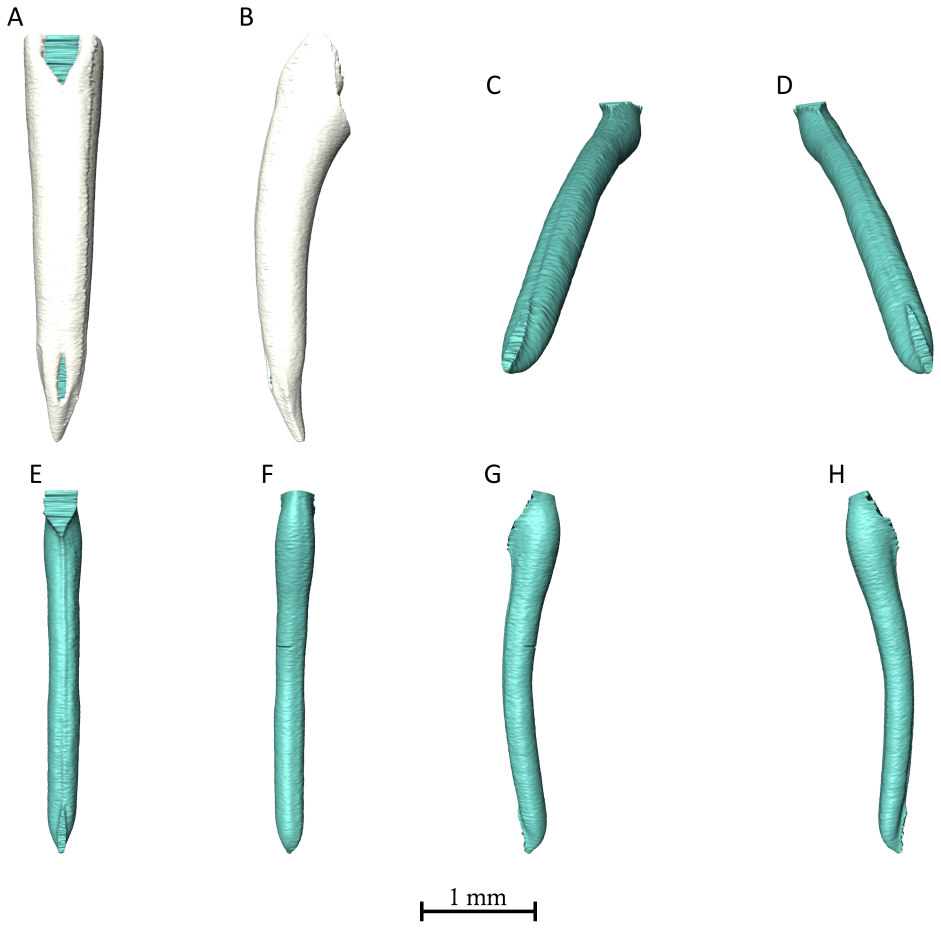


**Figure 77:** (*Naja siamensis*) 3D reconstruction of fang sia3. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

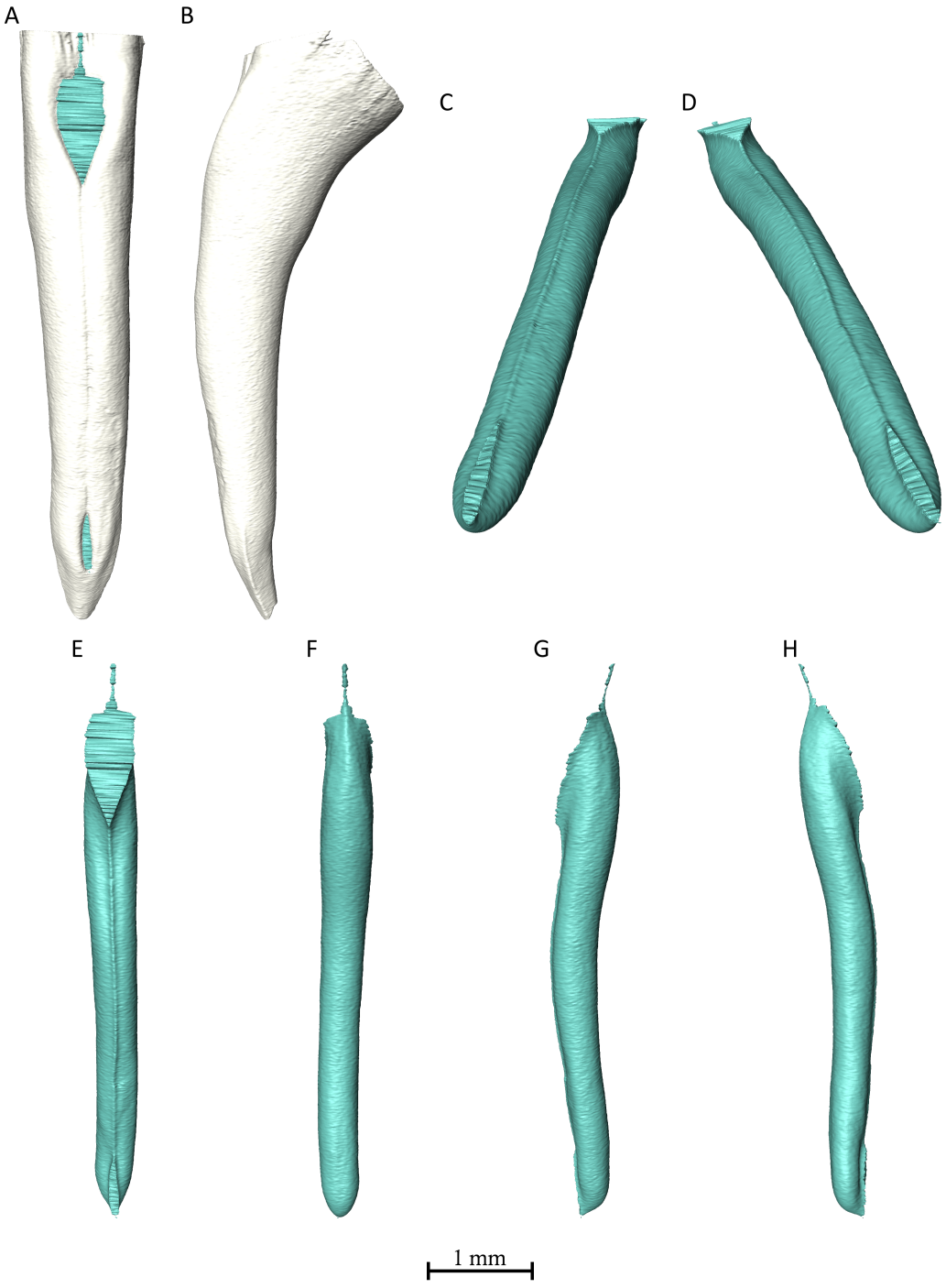


**Figure 78:** (*Naja siamensis*) 3D reconstruction of fang sia4. Cyan = venom canal. As the venom canal is a hollow structure it is depicted as a negative cast. Structures like the ridges described in this chapter are seen as depressions in the "solid" canal. A, E: frontal view, B, G: side view (left), C, D: inclined view, F: back view, H: side view (right). The reconstructed surface area is smoothed. Scale is about 15 : 1.

# E Circumferences and areas of venom canals

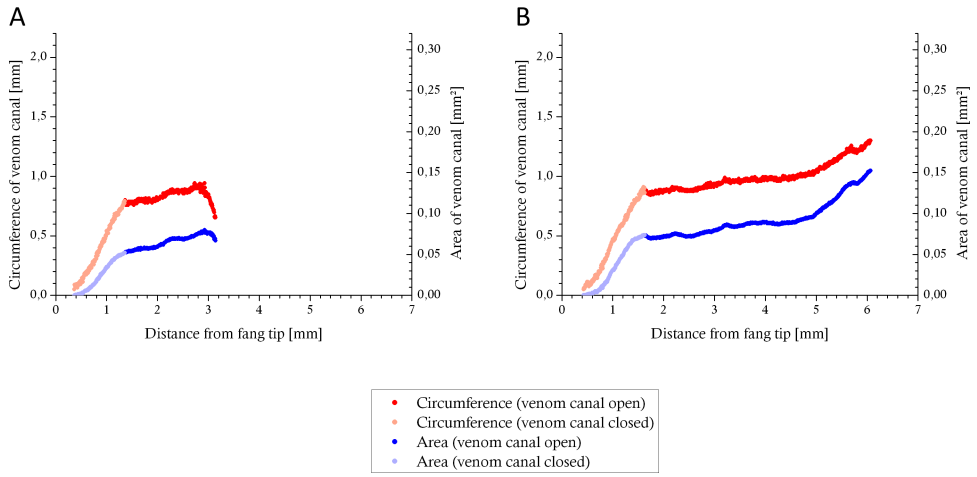


**Figure 79:** Circumferences and areas of venom canals of *Dendroaspis angusticeps*. A: ang1, B: ang2.

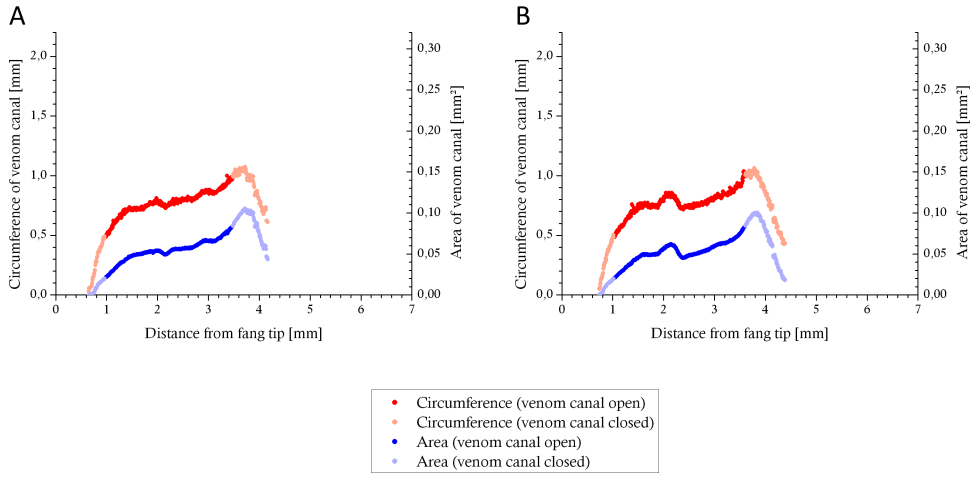


**Figure 80:** Circumferences and areas of venom canals of *Hemachatus haemachatus*. A: hae1, B: hae2.

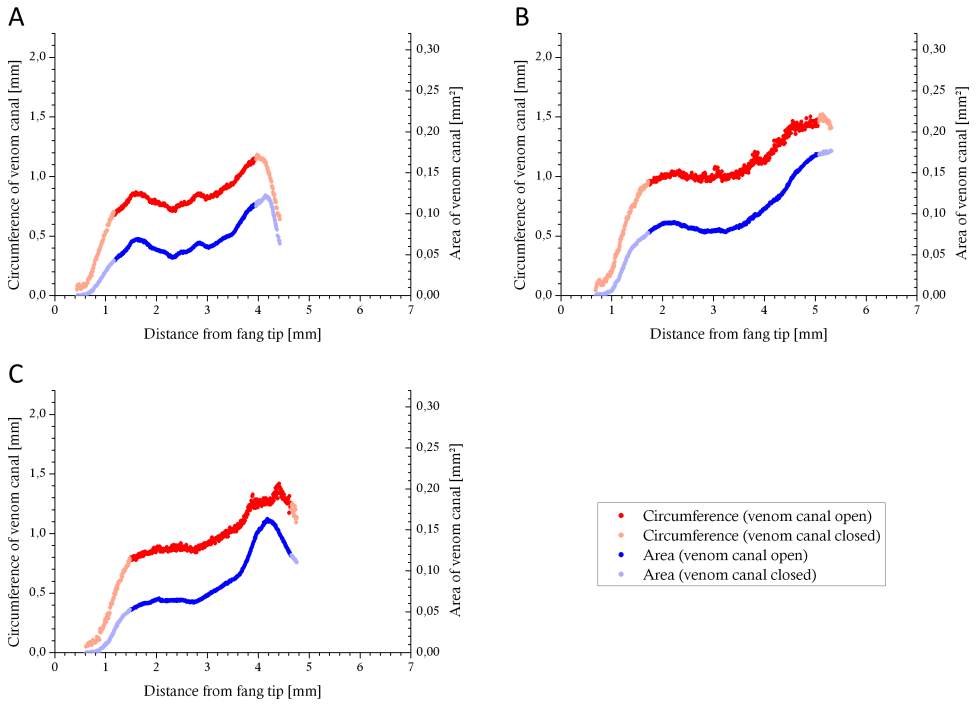


**Figure 81:** Circumferences and areas of venom canals of *Naja kaouthia*. A: kao1, B: kao2, C: kao3.

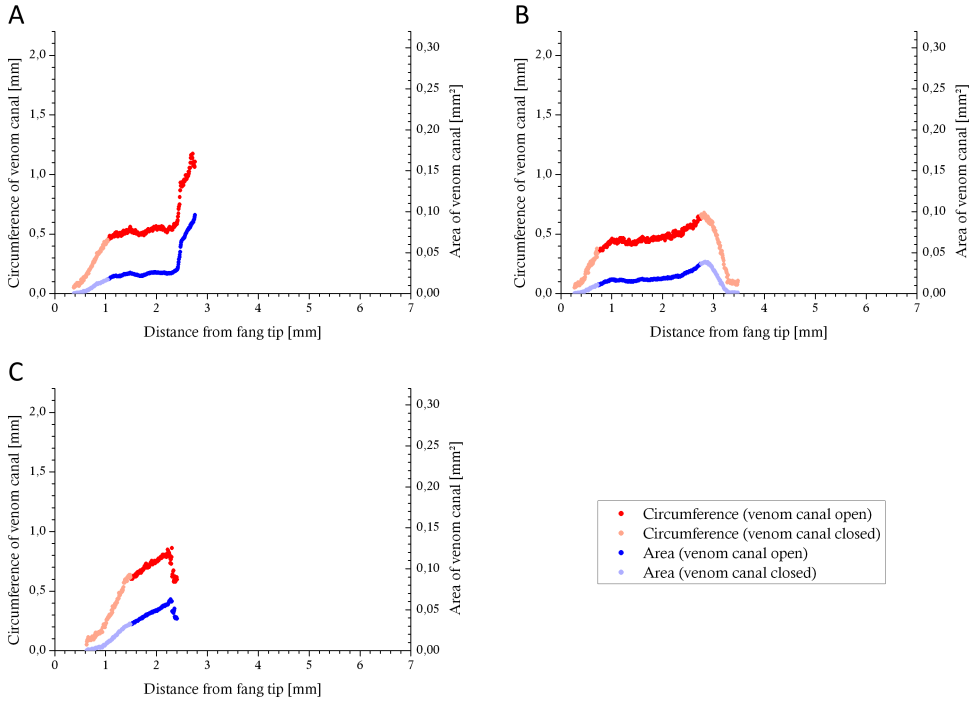


**Figure 82:** Circumferences and areas of venom canals of *Naja melanoleuca* and *Naja haje*. A: mel1, B: mel2, C: haj1. The sharp increase in area and circumference of mel1 (A) at about 2.5 mm is an artifact due to the fang being broken (compare figure 61 on page 211).

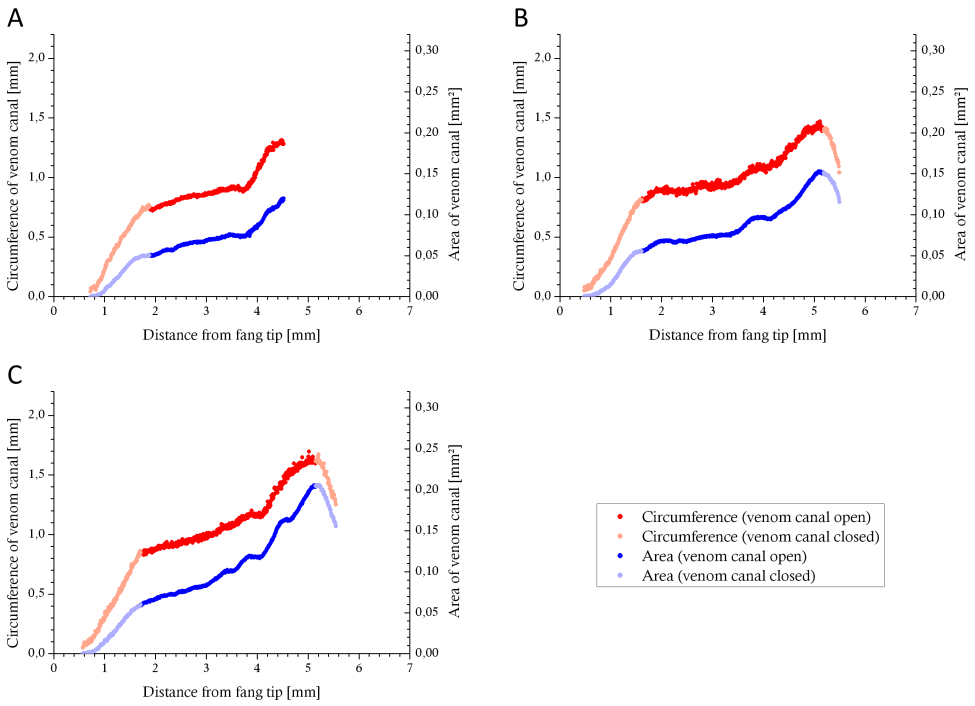


**Figure 83:** Circumferences and areas of venom canals of *Naja naja*. A: naj1, B: naj2, C: naj3.

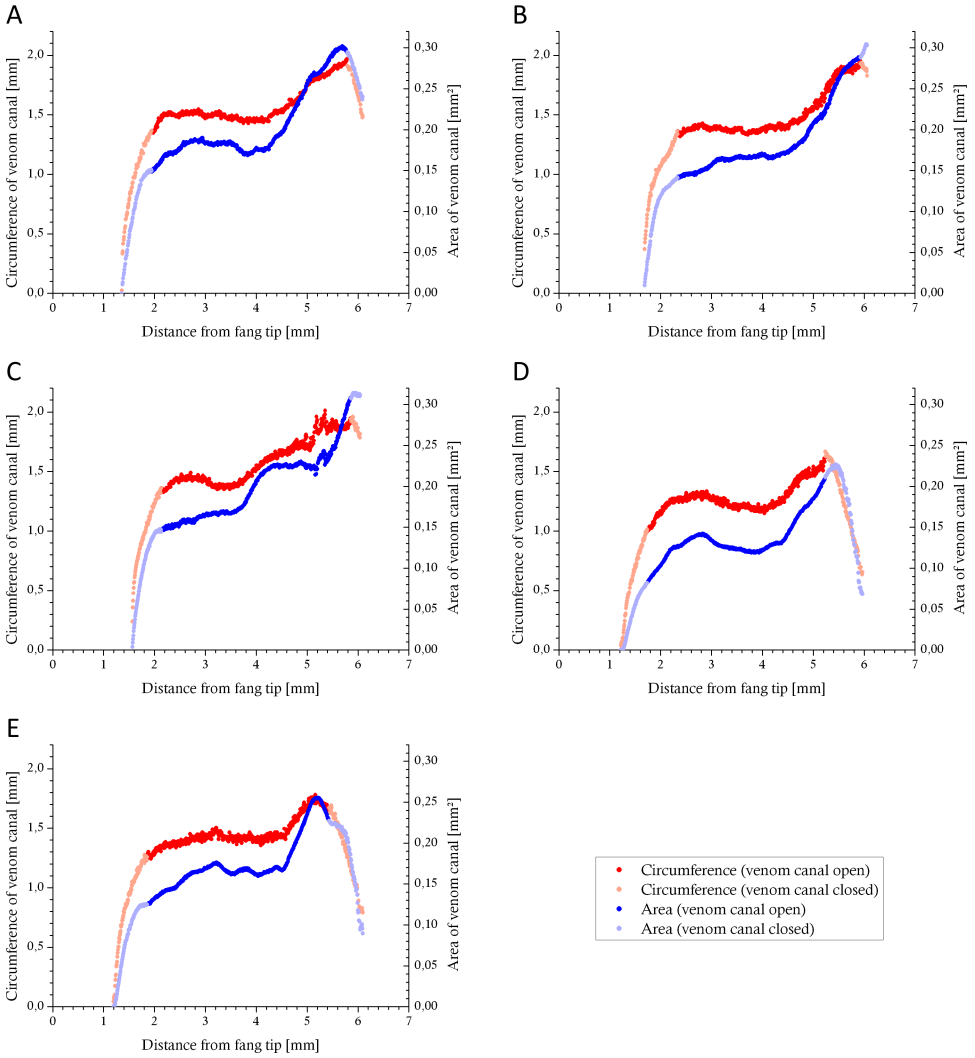


**Figure 84:** Circumferences and areas of venom canals of *Naja nigricollis*. A: nig1, B: nig2, C: nig3, D: nig4, E: nig5.

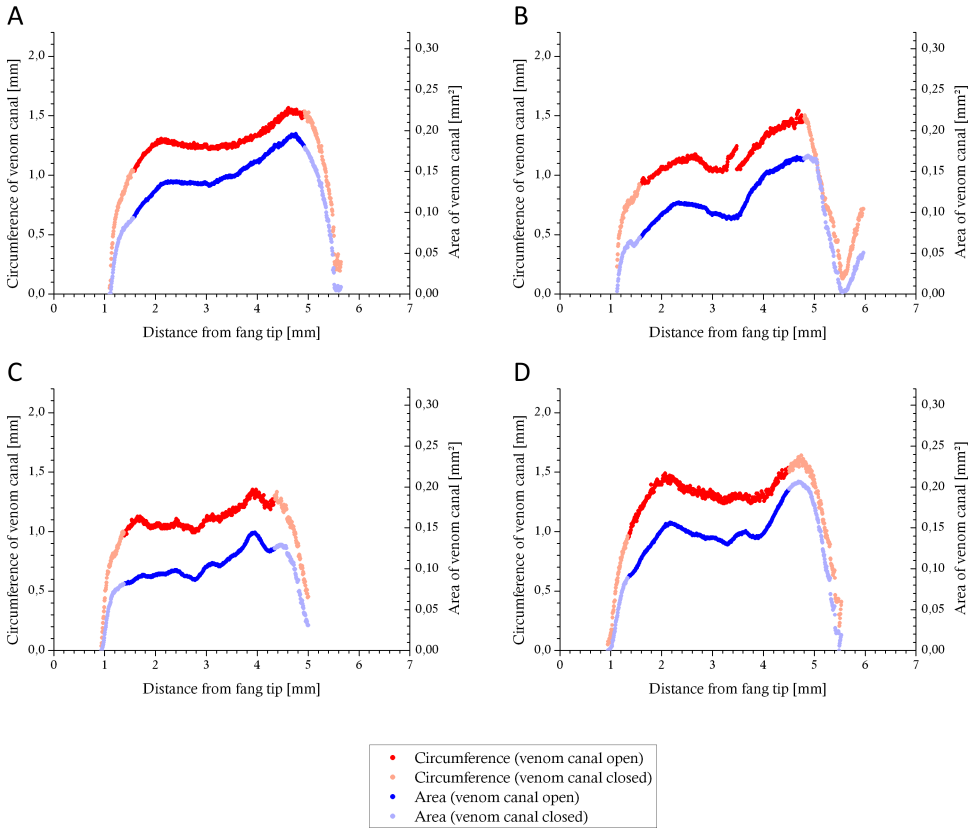


**Figure 85:** Circumferences and areas of venom canals of *Naja pallida*. A: pal1, B: pal2, C: pal3, D: pal4.

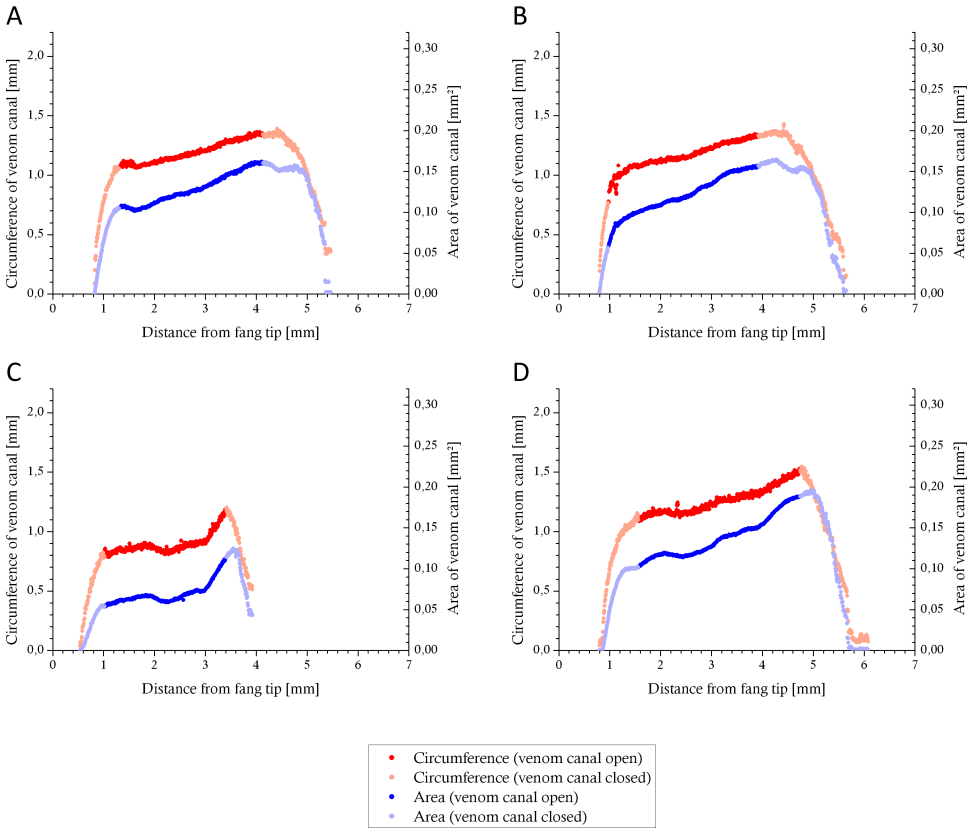


**Figure 86:** Circumferences and areas of venom canals of *Naja siamensis*. A: sia1, B: sia2, C: sia3, D: sia4.

# F Exemplary eccentricities

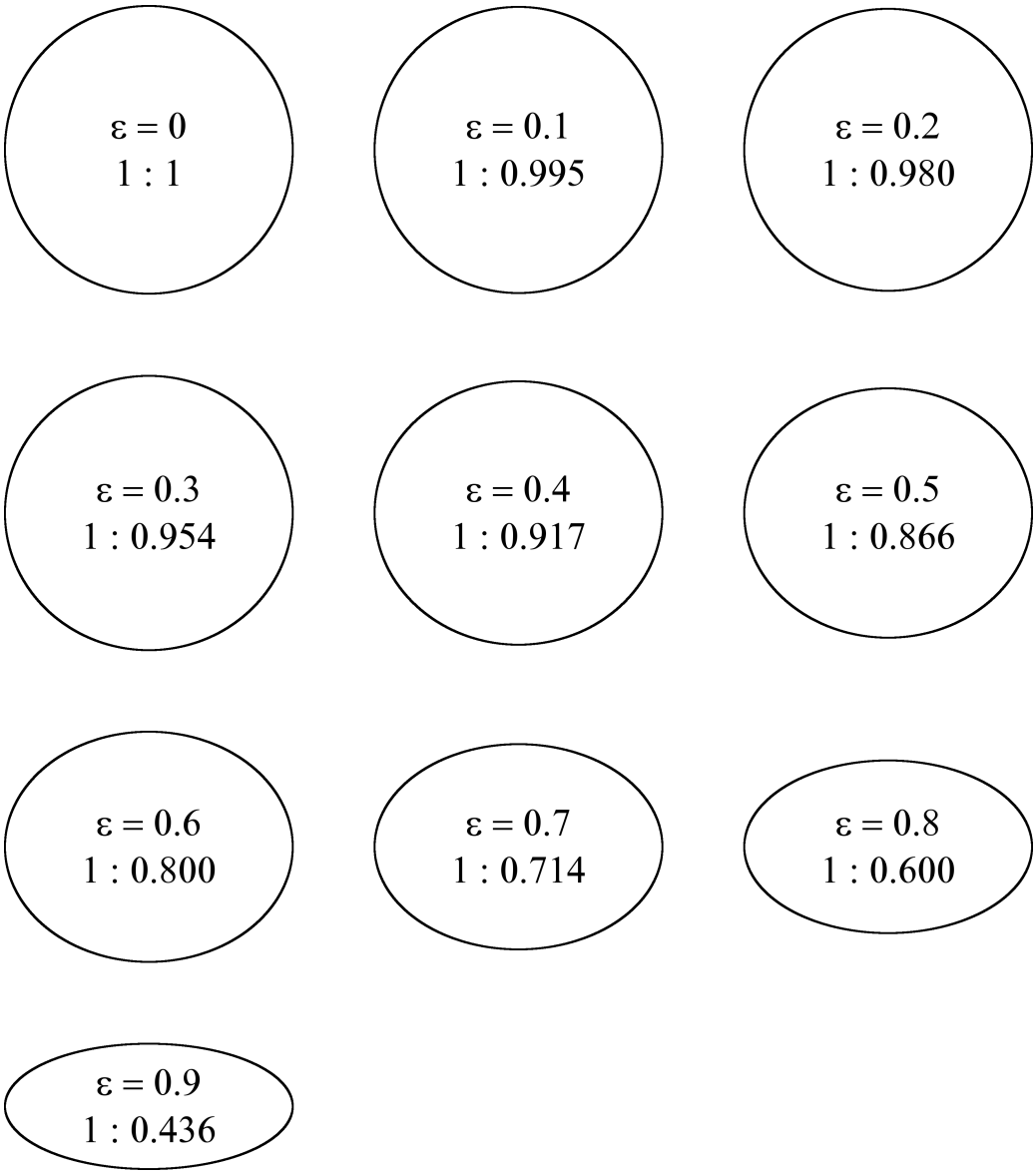


**Figure 87:** Exemplary numerical eccentricities. For each ellipse the numerical eccentricity ε is given. The line below gives the ratio between the major axis and the minor axis.

# Acknowledgements

I like to thank Prof. Dr. Bleckmann and Dr. Guido Westhoff for the possibility to work on this topic and their trust in me that I could handle the cobras. My thanks go to Prof. Dr. Wolfgang Böhme for being the second reviewer of my thesis.

My gratitude goes to all the people of the department, who made my stay there an enjoyable time. Special thanks go to Tobias Kohl for his help with the snakes and sharing his knowledge of them. Slava Braun was a great help in taking care of cobras and I would like to thank him for that. I am also thankful to Marlene Spinner and Stéphanie de Pury for their critical proof reading. Maik Dobiey helped with taking photos of the spray patterns and Fons de Mey explained the microtomograph in Antwerp to me.