

**Simulation:**  
**Lichtstreuung an Oberflächen mit Defekten**  
**Dr. Stefan Bosse**  
**13.10.2003**

---

## Eindimensionale Oberflächen

---

Zunächst sollten Methoden entwickelt werden, mit denen die Lichtstreuung an Oberflächen, die durch eine eindimensionale Oberflächenfunktion beschrieben werden, simulieren zu können, um dann die erarbeiteten Modelle und die dahinter stehende Numerik auf Oberflächen mit einer zweidimensionalen Oberflächenfunktion übertragen zu können.

Oberflächen mit einer gaußförmigen Korrelationsfunktion wurden ausgewählt, da diese realen Oberflächen in ihren Parametern und Eigenschaften nahe liegen.

---

## Mathematische Modellierung

---

### Oberfläche

Die Oberfläche soll durch  $N$  Höhenwerte  $Z_n$  in der  $x$ -Koordinate diskretisiert werden.

Dazu werden zunächst  $N+M$  Zufallszahlen  $u_i$  mit einer gaußförmigen Wahrscheinlichkeitsverteilung erzeugt.

Die Höhenverteilung  $Z_n$  der Oberfläche mit  $n=1\dots N$  erhält man dann mittels der  $M$  Gewichte  $w_l$  mit Gleichung (1) durch einen gewichteten Mittelungsprozeß, der sogenannten Moving-Average-Methode [OGI91]:

$$Z_n = \sum_{l=-M/2}^{M/2} w_l u_{n+l} \quad (1)$$

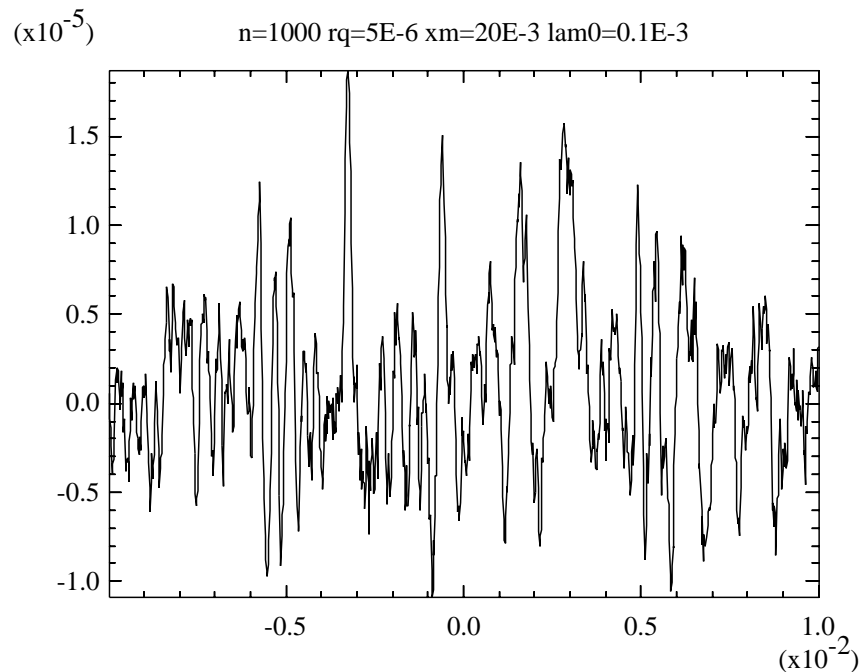
Die Gewichte werden dann mit einer Gaußfunktion nach Gleichung (2) berechnet, die eine gaußförmige Autokorrelation der Oberflächenwerte  $Z_n$  erzeugt:

$$w_l = w_0 \frac{1}{\sqrt{\lambda_0}} e^{-2(l\Delta x)^2 / \lambda_0^2} \quad (2)$$

mit  $l=-M/2\dots M/2$  und  $\Delta x=X/N$ .  $X$  ist die Ausdehnung der Oberfläche in  $x$ -Richtung. Der Parameter  $\lambda_0$  wird als Oberflächenwellenlänge bezeichnet, und gibt die räumliche Kohärenz (hier in  $x$ -Richtung) der Oberfläche an.

Eine Beispiel für eine mit dieser Methode erzeugte Oberfläche zeigt **Fig. 1** mit einem Rauheitswert von  $R_q = 5$  nm und einer Oberflächenwellenlänge  $\lambda_0 = 100$   $\mu\text{m}$ .

Fig. 1



*Eine synthetisch erzeugte Oberfläche, diskretisiert mit 1000 Datenpunkten und einer Oberflächenwellenlänge von  $\lambda_0=100 \mu\text{m}$ .*

## Lichtstreuung

Ziel ist die Berechnung des von der synthetisch generierten Oberflächen erzeugten Streulichtfeldes entlang einer Bildlinie, deren Normale mit der Oberflächennormale einen Winkel  $\Theta$  einschließt, und eine Breite  $L$  besitzt. Dazu werden folgende Näherungen durchgeführt:

- Die Oberfläche wird mit monochromatischen Lichtwellen konstanter Phase, d.h. ebene Wellen, beleuchtet.
- Von jedem Oberflächenpunkt gehen Kugelwellen aus.
- Mehrfachreflexion an Oberflächenelementen und "Abschattungen" werden vernachlässigt.
- Die Polarisation der beleuchtenden Lichtwellen werden vernachlässigt, ebenso Polarisationsänderungen durch die Lichtstreuung und Phasensprünge.

Allgemeine Formulierung des elektrischen Feldes von Kugelwellen:

$$E(t,r) = E(r) e^{-i\omega t + ikr} \quad (3)$$

mit

$$E(r) = \frac{E_0}{r}, k = \frac{2\pi}{\lambda}, \quad (4)$$

sowie von ebenen Wellen:

$$\mathbf{E}(t, \mathbf{r}) = \mathbf{E}_0(\mathbf{r}) e^{-i\omega t + i\mathbf{k}\mathbf{r} + i\Phi_0} \quad (5)$$

Die Beleuchtung der Oberfläche mit ebenen Wellen führt direkt auf den Gangunterschied der Lichtwellen zwischen zwei Oberflächenpunkten  $P_S(x_i, z_i)$  und  $P_S(x_j, z_j)$ :

$$\xi_{i,j} = (z_i - z_j) \cos(\Theta) + (x_i - x_j) \sin(\Theta) \quad (6)$$

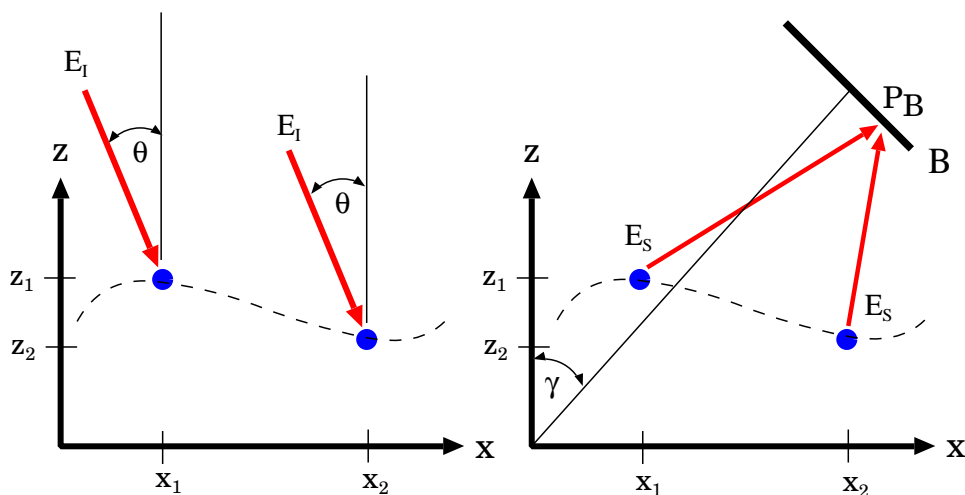
was einer Phasendifferenz

$$\Delta \Phi_{i,j} = k \xi_{i,j} \quad (7)$$

entspricht. Der Winkel  $\Theta$  ist dabei der Winkel zwischen der Ausbreitungsrichtung der beleuchtenden Lichtwellen und der Oberflächennormalen, wie in **Fig. 2** dargestellt ist.

Die von den Oberflächenpunkten ausgehenden Kugelwellen sollen sich in einem Punkt  $P_B(x_p, z_p)$  des Streulichtbildes, welches sich in einem Abstand  $R$  vom Koordinatenursprung befinden soll, und dessen Normalenrichtung einen Winkel  $\gamma$  mit der Oberflächennormalen bildet, überlagern, wie in **Fig. 2** veranschaulicht wird. Das Streulichtbild wird dabei ebenfalls diskretisiert.

**Fig. 2**



*Die in der Simulation verwendete Streugeometrie.*

Das elektrische Feld am Ort  $P_B(j)$  ergibt sich dann aus einer Summation der Phasenbeiträge aller gestreuten Lichtwellen  $E_S$  mit:

$$E_B(j) = \frac{1}{N} \sum_{i=1}^N E_0(i) \frac{1}{r_{ij}} e^{i(k r_{ij} + k \xi_{1,i})} \quad (8)$$

mit

$$r_{ij} = \sqrt{(x_b(j) - x_s(i))^2 + (z_b(j) - z_s(i))^2} \quad (9)$$

Ebene Wellen treten in der Strahtaille eines Gaußstrahls auf, der z.B. durch einen Laser erzeugt wird, der in der Grundmode  $TEM_{00}$  arbeitet. Um der transversal gaußförmigen Intensitätsverteilung eines solchen Strahls gerecht zu werden, wird die Feldamplitude  $E_0(i)$  durch eine Gaußfunktion in Abhängigkeit von der  $x$ -Koordinate ersetzt, so daß gilt:

$$E_0(i) = e^{-x_i^2 / (2 \cdot \sigma^2)} \quad (10)$$

mit einer  $\sigma$ -Breite von

$$\sigma = 0.6 (x_N - x_1) \quad (11)$$

Eine Verbesserung des Modells läßt sich erzielen, wenn man einen ortsabhängigen Reflexionskoeffizienten  $R(x)$  in die Berechnung des Streulichts einbezieht **[OG191]**. Dieser Reflexionskoeffizient setzt sich aus folgenden Anteilen zusammen:

- einem materialcharakteristischen Parameter,
- dem Einfallswinkel- und Ausfallswinkel,
- und dem Oberflächengradienten  $dz/dx$ .

Dieser Reflexionskoeffizient beeinflußt die Amplitude des gestreuten elektrischen Feldes, so daß man für die Amplitude schreiben kann:

$$E_0^{\mathfrak{R}}(x) = E_0(x) \left( a \frac{dz}{dx} - c \right) \quad (12)$$

mit

$$a = \sin(\theta)(1 - R_0) + \sin(\gamma)(1 + R_0) \quad (13)$$

$$c = \cos(\theta)(1 + R_0) - \cos(\gamma)(1 - R_0) \quad (14)$$

## Randbedingungen

Wie aus der Beugungstheorie zu erwarten ist, und sich durch erste Simulationsergebnisse auch bestätigte, hat die Wahl des Diskretisierungsintervalls  $\Delta x$  der Oberfläche entscheidenden Einfluß auf die Aussagefähigkeit und Qualität der Ergebnisse. Das Intervall  $\Delta x$  muß dabei deutlich kleiner als die verwendete Lichtwellenlänge gewählt werden. Ansonsten kommt es zu Beugungserscheinungen, die durch das Diskretisierungsgitter der Oberfläche entstehen.

---

## Numerische Simulation

---

Die im folgenden aufgezeigten Simulationsergebnisse wurden mit dem vom BSSLAB neu entwickelten VAMLAB System [VLA02] programmiert und berechnet. VAMLAB basiert auf dem Vorgängersystem PsiLAB [PSI98]. VAMLAB wurde mit der funktionalen Programmiersprache OCaml [OCA02] implementiert. Dabei wird der ML-Code in Byte-Code übersetzt, und von einer virtuellen Maschine ausgeführt.

Im ersten Schritt wurden alle Berechnungsalgorithmen direkt in ML programmiert. Da der Rechenaufwand bereits bei eindimensionalen Oberflächen nicht unerheblich ist, wurden aus Effizienzgründen die Kernalgorithmen, nach dem sie getestet und validiert wurden, in Fortran umgeschrieben und an das VAMLAB-System in binärer Form gebunden.

---

Source code: surf.ml

Mathematische Modellierung

---

```
(*
** Surface parameterization
*)

let surf_rq z =
  let n = dim2 z in

  if (Array.length n = 1) then
  begin
    let n = n.(0) in

    let rq = ref 0.0 in
    for i = 1 to n
    do
      let zv = (Genarray.get z [|i|]) in
      rq := !rq + zv**2.0;
    done;
    rq := sqrt (!rq / (float_of_int n));
    !rq
  end
  else
    failwith "rq: dim not supported"

let surf_ra z =
  let n = dim2 z in

  if (Array.length n = 1) then
  begin
    let n = n.(0) in

    let ra = ref 0.0 in
    for i = 1 to n
    do
      let zv = (Genarray.get z [|i|]) in
      ra := !ra + (abs zv);
    done;
    ra := !ra / (float_of_int n);
```

```

        !ra
    end
    else
        failwith "ra: dim not supported"

(*
** Surface generation
*)

(*
** Moving Average method with gaussian autocorrelation
** behaviour.
*)

let surfma_gen ~xm ~rq ~n ~lam0 =

    (*
    ** Number (-0+) of points for moving average procedure
    *)

    let wn = int_of_float (
        lam0 / xm * (float_of_int n)
    )
    in

    let dx = xm / (float_of_int n) in
    let fn2 = (float_of_int n) / 2.0 in

    (*
    ** The uncorrelated random variables u_n
    *)

    let unity = fmatrix ~dim:[n+wn] () in
    let un = (fun x -> (Rand.snorm ())) ||<< unity in

    (*
    ** Calculate the weights
    *)

    let xw = fmatrix ~fill:[-(float_of_int wn)/2.0;
        1.0;
        (float_of_int wn)/2.0] () in

    let w0' = 1.0 in

    let zw = (fun l ->
        w0'/(sqrt lam0)*
        (exp ( (-2.0*(l*dx)**2.0)/(lam0**2.0) ))
    )

```

```

) ||<< xw in

(*
** Apply the wights to the random field
*)

let x = fmatrix ~dim:[n] ~fill:[0.0] () in
let z = fmatrix ~dim:[n] ~fill:[0.0] () in

for i = 1 to n
do
  x.{i} <- ((float_of_int i)-fn2)/fn2*xm;
  for j = 0 to wn
  do
    z.{i} <- z.{i} +
              zw.{j+1} * un.{i+j};
  done
done;

(*
** Scale the surface to rq
*)

let rq' = surf_rq z in

z =<< (( fun z -> z * rq / rq' ),z,[]);
x,z

```

---

Source code: [scatt.ml](#)

**Mathematische Modellierung**

```

let rad_of_deg g =
  (2.0*C.pi*g/360.0)

let gauss ~x ~w =
  let y0 = 0.0 in
  let ym = 1.0 in
  let sigm = 0.3 * (w/2.0) in
  (
    y0 +
    ym *
    exp (
      (-1.0/(2.0*(sigm**2.0)))*
      x**2.0
    )
  )

let b_plane ~l ~gam ~r ~n =
  let gam = rad_of_deg gam in

  let gam' = atan (1 / (2.0*r)) in
  let r' = 1 / (2.0 * (sin gam')) in

```

```

let x1      = r' * (sin (gam-gam')) in
let z1      = r' * (cos (gam-gam')) in

let x2      = r' * (sin (gam+gam')) in
let z2      = r' * (cos (gam+gam')) in

let vec t =
  let x = x1 + t * (x2-x1) in
  let z = z1 + t * (z2-z1) in
  x,z
in

let x = fmatrix ~dim:[n] () in
let z = fmatrix ~dim:[n] () in

let tn = 1.0 / (float_of_int n) in
for i = 1 to n
do
  let t = (float_of_int i) * tn in
  let x',z' = vec t in
  x.{i} <- x';
  z.{i} <- z';
done;
x,z

let scatt ~xs ~zs ~xb ~zb ~theta ~lam =
  let theta = rad_of_deg theta in

  let dx = xs.{2} - xs.{1} in
  let zeta i =
    (zs.{i}-zs.{1}) * (cos theta) +
    ((float_of_int (i-1))*dx) * (sin theta)
  in

  let nb = dim2 xb in
  let ns = dim2 xs in

  let swidth = xs.{ns.(0)} - xs.{1} in

  let es = cmatrix ~dim:[nb.(0)] ~fill:[0.0,0.0] () in

  let phi0 = 0.0 in
  let k = 2.0*C.pi / lam in

  let e0 = (fun x -> gauss ~x:x ~w:swidth) ||<< xs in

  for i = 1 to nb.(0)
  do
    for j = 1 to ns.(0)

```

```

do

  let rj = sqrt ((xb.{i}-xs.{j})**2.0 + (zb.{i}-zs.{j})**2.0) in
  let earg = k*(rj + (zeta j)) in

  es.{i} <- es.{i} +
            (e0.{j},0.0) * (cexp (0.0,earg))
            / (rj,0.0);

done;

done;

let sqn = (float_of_int ns.(0)) in
let ins = fmatrix ~dim:[nb.(0)] ~fill:[0.0] () in

for i = 1 to nb.(0)
do
  ins.{i} <- ((cabs es.{i})/(sqn)) ** 2.0;
done;
ins

```

---

## Ergebnisse

---

Abbildungen **Fig. 3** bis **Fig. 7** zeigen Simulationsergebnisse mit folgenden gemeinsamen Parametern:

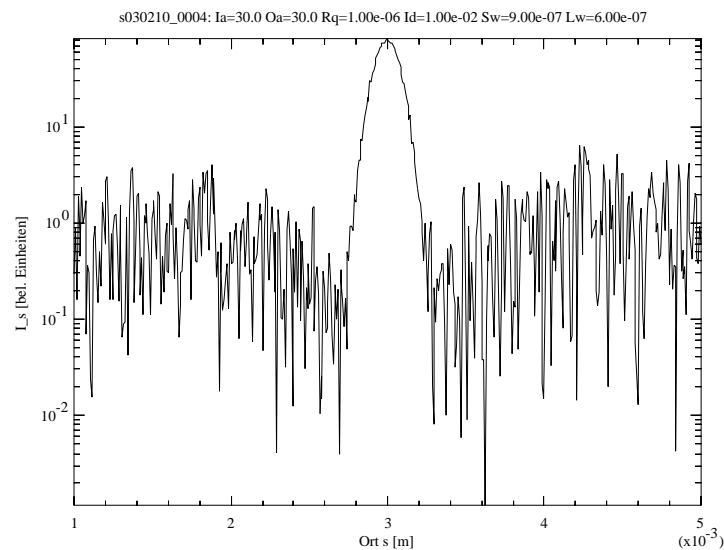
Tab. 1

Oberflächenwellenlänge: $\lambda_S=0.9 \mu\text{m}$	Lichtwellenlänge: $\lambda_I=600 \text{ nm}$
Rauheitswert: $R_Q=1.0 \mu\text{m}$	Detektorbreite: $D=4 \text{ mm}$
Anzahl der Oberflächen- elemente: $N_S=10000$	Anzahl der Detektor- elemente: $N_B=512$

Es ist jeweils die Intensität  $I_s$  in der Detektorebene in Abhängigkeit vom Ort der Detektorelemente  $s$  entlang der Detektorachse aufgetragen, wobei  $s=1 \text{ mm}$  dem äußersten linken Pixel, und  $s=5 \text{ mm}$  dem äußersten rechten Detektorelement entspricht. Der Reflexionskoeffizient wurde hier noch nicht berücksichtigt.

---

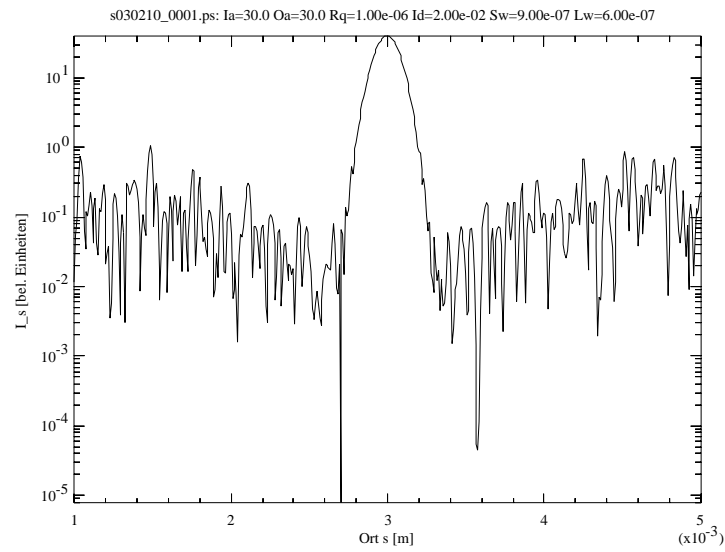
**Fig. 3**



*Intensität in der Detektorebene in Abhängigkeit vom Ort der Detektorelemente:  
 $\theta=30^\circ$ ,  $\gamma=30^\circ$ , Abstand Detektor  $L=10 \text{ mm}$*

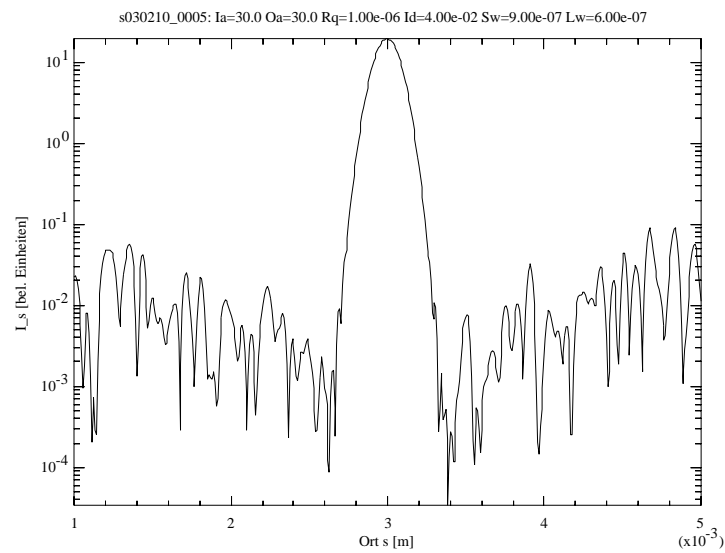
---

Fig. 4



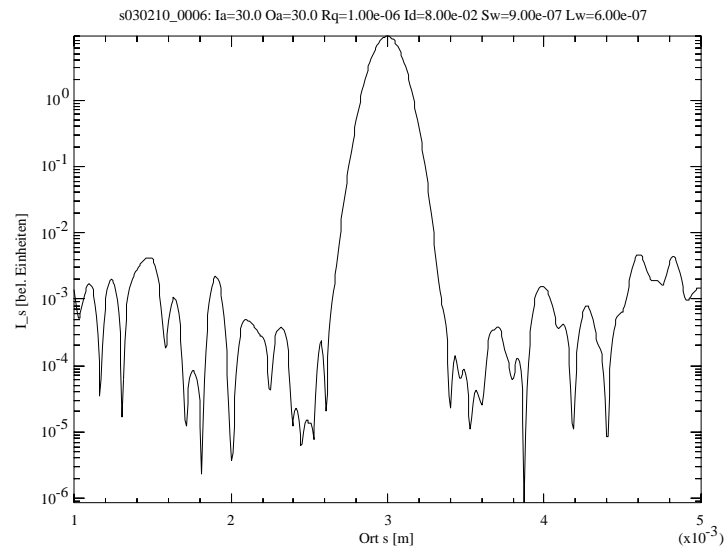
*Intensität in der Detektorebene in Abhängigkeit vom Ort der Detektorelemente:  
 $\theta=30^\circ$ ,  $\gamma=30^\circ$ , Abstand Detektor  $L=20$  mm*

Fig. 5



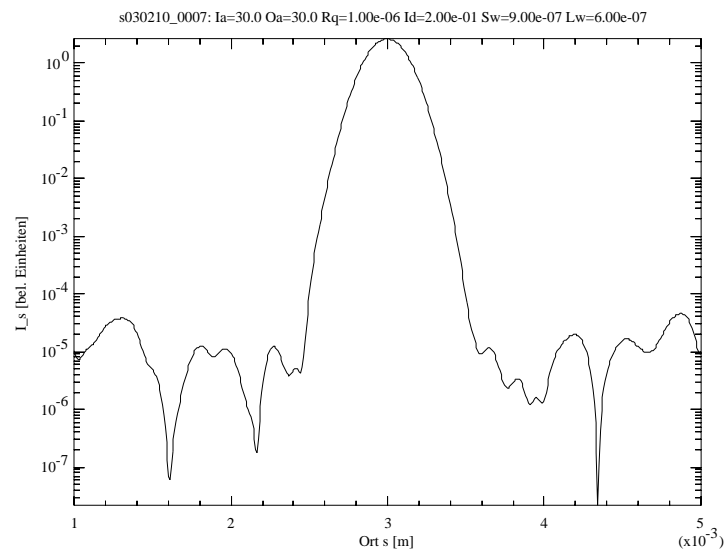
*Intensität in der Detektorebene in Abhängigkeit vom Ort der Detektorelemente:  
 $\theta=30^\circ$ ,  $\gamma=30^\circ$ , Abstand Detektor  $L=40$  mm*

Fig. 6



*Intensität in der Detektorebene in Abhängigkeit vom Ort der Detektorelemente:  
 $\theta=30^\circ$ ,  $\gamma=30^\circ$ , Abstand Detektor  $L=80$  mm*

Fig. 7



*Intensität in der Detektorebene in Abhängigkeit vom Ort der Detektorelemente:  
 $\theta=30^\circ$ ,  $\gamma=30^\circ$ , Abstand Detektor  $L=200$  mm*



---

## Optimierte Implementierung mit VAMLAB

---

Bei der zuvor gezeigten Implementierung der Numerik in OCaml zeigte sich, daß bereits bei eindimensionalen Oberflächen eine erhebliche Rechenleitung und Zeit benötigt wird. Um zunächst die vorhandenen Ressourcen optimal nutzen zu können, wurden die Implementierung optimiert. Dazu wurden alle rechenintensiven Funktionen in Fortran-Routinen ausgelagert, und über entsprechende Schnittstellen an das vorhandene VAMLAB-System angekoppelt, so daß auch weiterhin eine benutzerfreundliche Arbeitsumgebung vorhanden ist.

Die Rechenzeit konnte durch diese Maßnahme um den Faktor 4 reduziert werden.

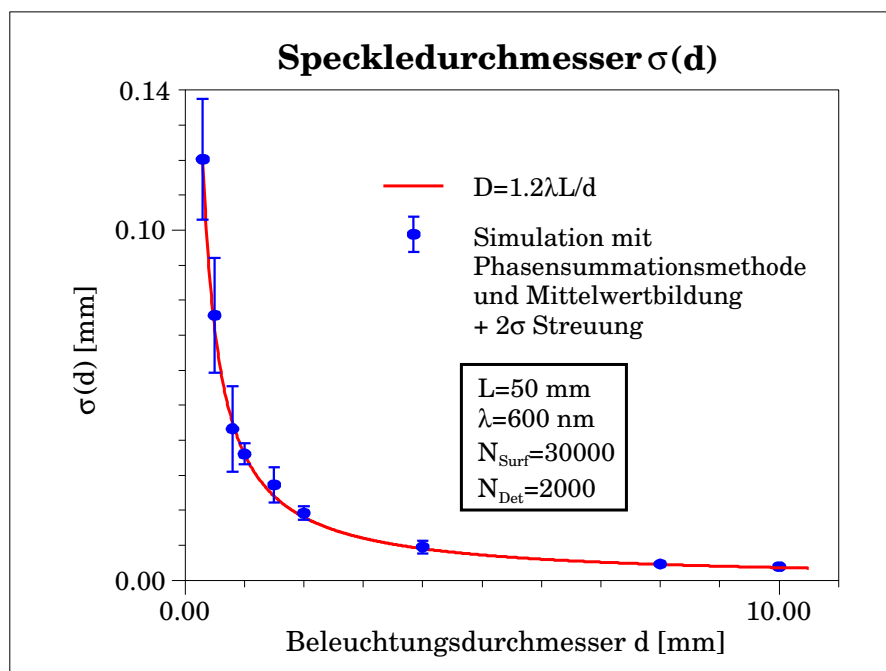
---

## Ergebnisse

---

**Fig. 8** zeigt die Berechnung des Speckledurchmessers einer rauhen Oberfläche mit  $R_q=1\mu\text{m}$  in Abhängigkeit vom Beleuchtungsdurchmesser ( $1/e$ ).

Fig. 8



*Berechnung des Speckledurchmessers von simulierten Streulichtbildern in Abhängigkeit vom Beleuchtungsdurchmesser*

---

Source code: surf.ml

**Optimierte Implementierung mit VAMLAB**

(\*

```

** Version 1.0
*)

open Math
open Matrix
open Callgate

external surflrq : ('a, 'b, 'c) Matrix.t -> float =
  "surflrq"

let surf_rq z =
  let n = mat_dim z in

  let rq:float =
    if (Array.length n = 1) then
      call_1 "surflrq" z
    else
      failwith "rq: dim not supported"
  in
  rq

external surflra : ('a, 'b, 'c) Matrix.t -> float =
  "surflra"

let surf_ra z =
  let n = mat_dim z in

  let ra:float =
    if (Array.length n = 1) then
      call_1 "surflra" z
    else
      failwith "ra: dim not supported"
  in
  ra

external surflmagen : ('a, 'b, 'c) Matrix.t *
  ('a, 'b, 'c) Matrix.t *
  ('a, 'b, 'c) Matrix.t *
  ('a, 'b, 'c) Matrix.t *
  float * float * float -> unit
  = "surflmagen"

(*
** Surface generation
*)

(*
** Moving Average method with gaussian autocorrelation
** behaviour.
*)

let surfma_gen ~dm ~rq ~n ~lam =

```

```

let dm = dm / 2.0 in

(*
** Number (-0+) of points for moving average procedure
*)

let wn' = int_of_float (
    lam / dm * (float_of_int n)
)
in
let wn = if (wn' mod 2) = 0 then
    (wn'+1)
else
    wn'
in

print_string ("Using "^(string_of_int wn)^" points for MA procedure.");
print_newline ();

(*
** The uncorrelated random variables u_n
*)

let unmat = fmatrix ~dim:[n+wn] () in

(*
** The weights
*)

let wgmatt = fmatrix ~dim:[wn] () in

(*
** The z(x) matrices
*)

let xmat = fmatrix ~dim:[n] () in
let zmat = fmatrix ~dim:[n] () in

ignore(call_1 "surflmagen" (xmat,zmat,unmat,wgmatt,dm,rq,lam));

wgmatt,xmat,zmat

```

---

**Source code:** surf.f

**Optimierte Implementierung mit VAMLAB**

---

```

C      Version 1.0
C
C      Surface parameterization
C
C
C      DOUBLE PRECISION FUNCTION SURF1RQ(MAT,N)
C      =====
C
C      DOUBLE PRECISION MAT(N),RQ,ZV
C      INTEGER I,N

```

```

EXTERNAL DMATGET1
DOUBLE PRECISION DMATGET1
RQ = 0.0

DO I = 1,N
    ZV = DMATGET1(MAT,I,N)
    RQ = RQ + ZV*ZV
ENDDO
SURF1RQ = SQRT(RQ / N)
RETURN
END

```

```

DOUBLE PRECISION FUNCTION SURF1RA(MAT,N)
=====

```

C

```

DOUBLE PRECISION MAT(N),RA,ZV
INTEGER I,N
EXTERNAL DMATGET1
DOUBLE PRECISION DMATGET1
RA = 0.0

DO I = 1,N
    ZV = DMATGET1(MAT,I,N)
    RA = RA + ABS(ZV)
ENDDO
SURF1RA = RA / N
RETURN
END

```

C

Surface generation

C

```

MAT: Z(N) surface height matrix
UNMAT: U(N+M) unity random matrix
WGMAT: WGMAT(WN) weight matrix
XMAT: XMAT(N) surface location matrix
DM: Width of surface
RQ: preferred roughness parameter
WN: Number of averaged points
LAM: surface wavelength (correlation length)

```

C

```

SUBROUTINE SURF1MAGEN(XMAT,ZMAT,UNMAT,WGMAT,N,DM,RQ,WN,LAM)
=====

```

C

```

DOUBLE PRECISION ZMAT(N),UNMAT(N+WN),WGMAT(WN),XMAT(N)
DOUBLE PRECISION RQ,DM,LAM,DX,DM,V,Z,W0,E,T1,T2,T3,FN2,RQ2
INTEGER I,J,N,WN
EXTERNAL DMATSET1,DMATGET1,SNORM
REAL SNORM
DOUBLE PRECISION DMATGET1
DOUBLE PRECISION SURF1RQ

DX = DM / N

```

C

```

C      The uncorrelated random variables u_n
C
DO I =1,N+WN
    V = SNORM()
    CALL DMATSET1(UNMAT,I,N+WN,V)
ENDDO

C
C      Calculate the weights
C

W0 = 1.0
V = -(WN/2.0)+0.5
DO I = 1,WN
    T1 = -2.0*(V*DX)**2
    T2 = LAM**2
    E = EXP (T1/T2)
    Z = W0/(SQRT (LAM)) * E
    CALL DMATSET1(WGMAT,I,WN,Z)
    V = V + 1.0
ENDDO

C
C      Apply the weights to the random field
C

FN2 = N/2.0

DO I = 1,N
    CALL DMATSET1 (XMAT,I,N, (I-FN2)/FN2*DM)
    DO J = 0,WN-1
        T1 = DMATGET1(ZMAT,I,N)
        T2 = DMATGET1(WGMAT,J+1,WN)
        T3 = DMATGET1(UNMAT,I+J,N+WN)
        CALL DMATSET1 (ZMAT,I,N, T1+T2*T3)
    ENDDO
ENDDO

C
C      Scale the surface to rq
C

RQ2 = SURF1RQ(ZMAT,N)

DO I = 1,N
    T1 = DMATGET1(ZMAT,I,N)
    T2 = T1 * RQ/RQ2
    CALL DMATSET1(ZMAT,I,N,T2)
ENDDO
RETURN
END

```

---

Source code: surf\_ext.c

Optimierte Implementierung mit VAMLAB

---

/\*

```

** Version: 1.0
*/

#include <stddef.h>
#include <stdarg.h>
#include <string.h>

#include "caml/alloc.h"
#include "caml/custom.h"
#include "caml/fail.h"
/*#include "intext.h"*/
#include "caml/memory.h"
#include "caml/mlvalues.h"

#include "matrix.h"
#include "f2c.h"
#include "matrix_f.h"

extern
double real surflrq_(double real *mat,integer *n);
extern
double real surflra_(double real *mat,integer *n);

int surflmagen_(
    double real *xmat, double real *zmat, double real *unmat, double real *wgmat,
    integer *n,
    double real *dm, double real *rq,
    integer *wn,
    double real *lam);

CAMLprim value
surflrq(value vb)
{
    Matrix_dim1(vb,n);
    Matrix_double(vb,mat);
    double real rq;

    rq = surflrq_(mat,&n);

    return (copy_double(rq));
};

CAMLprim value
surflra(value vb)
{
    Matrix_dim1(vb,n);
    Matrix_double(vb,mat);
    double real ra;

    ra = surflra_(mat,&n);

    return (copy_double(ra));
};

```

```

};

/*
** Args:
**   (xmat,zmat,unmat,wgmat,dm,rq,lam) tuple
**/

CAMLprim value
surflmagen(value vt)
{
    Matrix_dim1(Field(vt,0),n);
    Matrix_dim1(Field(vt,3),wn);

    Matrix_double(Field(vt,0),xmat);
    Matrix_double(Field(vt,1),zmat);
    Matrix_double(Field(vt,2),unmat);
    Matrix_double(Field(vt,3),wgmat);

    doublereal dm=Double_val(Field(vt,4));
    doublereal rq=Double_val(Field(vt,5));
    doublereal lam=Double_val(Field(vt,6));

    surflmagen_(xmat,zmat,unmat,wgmat,
                &n,&dm,&rq,&wn,&lam);

    return Val_unit;
};

```

---

**Source code:** `scatt.ml`

**Optimierte Implementierung mit VAMLAB**

---

```

open Math
open Complex
open Matrix
open Matrix_high
open Const

let rad_of_deg g =
    (2.0*C.pi*g/360.0)

let gauss ~x ~w =
    let y0 = 0.0 in
    let ym = 1.0 in
    let sigm = 0.3 * (w/2.0) in
    (
        y0 +
        ym *
        exp (
            (-1.0/(2.0*(sigm**2.0)))*
            x**2.0
        )
    )

let b_plane ~l ~gam ~r ~n =

```

```

let gam = rad_of_deg gam in

let gam'   = atan (1 / (2.0*r)) in
let r'     = 1 / (2.0 * (sin gam')) in

let x1     = r' * (sin (gam-gam')) in
let z1     = r' * (cos (gam-gam')) in

let x2     = r' * (sin (gam+gam')) in
let z2     = r' * (cos (gam+gam')) in

let vec t =
  let x = x1 + t * (x2-x1) in
  let z = z1 + t * (z2-z1) in
  x,z
in

let x = fmatrix ~dim:[n] () in
let z = fmatrix ~dim:[n] () in

let tn = 1.0 / (float_of_int n) in
for i = 1 to n
do
  let t = (float_of_int i) * tn in
  let x',z' = vec t in
  x.{i} <- x';
  z.{i} <- z';
done;
x,z

let scatt ~xs ~zs ~xb ~zb ~theta ~lam =
  let theta = rad_of_deg theta in

  let dx = xs.{2} - xs.{1} in
  let zeta i =
    (zs.{i}-zs.{1}) * (cos theta) +
    ((float_of_int (i-1))*dx) * (sin theta)
  in

  let nb = mat_dim xb in
  let ns = mat_dim xs in

  let swidth = xs.{ns.(0)} - xs.{1} in

  let es = cmatrix ~dim:[nb.(0)] ~fill:[0.0,0.0] () in

  let phi0 = 0.0 in
  let k = 2.0*C.pi / lam in

  let e0 = (fun x -> gauss ~x:x ~w:swidth) ||<< xs in

```

```

for i = 1 to nb.(0)
do

  for j = 1 to ns.(0)
  do

    let rj = sqrt ((xb.{i}-xs.{j})**2.0 + (zb.{i}-zs.{j})**2.0) in
    let earg = k*(rj + (zeta j)) in

    es.{i} <- es.{i} +
              (e0.{j},0.0) * (cexp (0.0,earg))
              / (rj,0.0);

  done;
done;

let sqn = (float_of_int ns.(0)) in
let ins = fmatrix ~dim:[nb.(0)] ~fill:[0.0] () in

for i = 1 to nb.(0)
do
  ins.{i} <- ((cabs es.{i})/(sqn)) ** 2.0;
done;
ins,es,e0

```

**Source code:** `scatt.f`

**Optimierte Implementierung mit VAMLAB**

```

C
C      Version 1.01
C

DOUBLE PRECISION FUNCTION RADOFDEG(G)
DOUBLE PRECISION G
DOUBLE PRECISION PI
DATA PI / 3.14159265358979324D0 /
RADOFDEG=2.0*PI*G/360.0
RETURN
END

DOUBLE PRECISION FUNCTION GAUSS(X,W)
DOUBLE PRECISION X,W,T1,T2,SIG,Y0,YM
Y0=0.0
YM=1.0
SIG=0.3*(W/2.0)
T1=-1.0/(2.0*(SIG**2))
T2=X**2
GAUSS = Y0 + YM * EXP(T1*T2)
RETURN
END

C
C      Calculate the image plane.
C      Args:

```

```

C          XB: XB(NB)
C          ZB: ZB(NB)
C          WID: Image width
C          GAM: Observation angle
C          R: Image distance
C          NB: Number of image points
C
C
SUBROUTINE BPLANE(XB,ZB,NB,WID,GAM,R)
DOUBLE PRECISION XB(NB),ZB(NB),GAM,R,WID
DOUBLE PRECISION GAM2,R2,X1,X2,Z1,Z2,TN,XV,ZV,TV
INTEGER NB,I
DOUBLE PRECISION RADOFDEG

GAM = RADOFDEG(GAM)
GAM2 = ATAN(WID/(2.0*R))
R2 = WID / (2.0 * SIN(GAM2))
X1 = R2 * SIN(GAM-GAM2)
Z1 = R2 * COS(GAM-GAM2)
X2 = R2 * SIN(GAM+GAM2)
Z2 = R2 * COS(GAM+GAM2)

TN = 1.0 / NB

DO I=1,NB
  TV = I*TN
  XV = X1 + TV * (X2-X1)
  ZV = Z1 + TV * (Z2-Z1)
  CALL DMATSET1(XB,I,NB,XV)
  CALL DMATSET1(ZB,I,NB,ZV)
ENDDO
RETURN
END

C
C          Initialize and calculate the electrical field amplitudes
C
C          => Surface gradient, incident and observation angle
C          dependent reflection coefficient controls E0
C          => Gaussian transversal distribution
C
C          sin(thet)*dz/dx+cos(thet)
C
C          THET: incident angle
C          GAMM: observation angle
C          R0: reflection coefficient

SUBROUTINE REFLEC(XS,ZS,E0,NS,THET,GAMM,R0)
DOUBLE PRECISION XS(NS),ZS(NS),E0(NS)
DOUBLE PRECISION THET,GAMM

DOUBLE PRECISION DX,DZ,V,SW,R0
DOUBLE PRECISION Q1,Q2,Q3,Q4
INTEGER NS,I

```

```

DOUBLE PRECISION RADOFDEG
DOUBLE PRECISION DMATGET1
DOUBLE PRECISION GAUSS

THET = RADOFDEG(THET)
GAMM = RADOFDEG(GAMM)
DX = DMATGET1(XS,2,NS)-DMATGET1(XS,1,NS)
SW = DMATGET1(XS,NS,NS)-DMATGET1(XS,1,NS)

WRITE (*,*) THET,GAMM,R0

Q1=SIN(THET)
Q2=COS(THET)
Q3=SIN(GAMM)
Q4=COS(GAMM)

DO I = 1,NS
  V = GAUSS(DMATGET1(XS,I,NS),SW)
  IF (I .EQ. NS) THEN
    DZ = 1E-11
  ELSE
    DZ = (DMATGET1(ZS,I+1,NS)-DMATGET1(ZS,I,NS))
  END IF
  V=V*(Q1*DZ/DX+Q2)
  V=V*(DZ/DX*(Q1*(1.0-R0)+Q3*(1.0+R0))-(Q4*(1+R0)-Q2*(1-R0)))
  CALL DMATSET1(E0,I,NS,V)
ENDDO

RETURN
END

C
C Calculate the scattered light field from the surface points
C (XS,ZS) for all image points (XB,ZB). THET is the illumination
C angle.
C ARgs:
C XS: XS(NS)
C ZS: ZS(NS)
C XB: XB(NB)
C ZB: ZB(NB)
C ES: ES(NB) NI
C E0: E0(NS) NI
C IS: IS(NB) NI
C THET: Illumination angle
C LAM: Light wavelength
C
C
SUBROUTINE SCATT(XS,ZS,XB,ZB,ES,E0,IS,NS,NB,THET,LAM)
DOUBLE PRECISION XS(NS),ZS(NS),XB(NB),ZB(NB),E0(NS),IS(NB)
DOUBLE PRECISION THET,LAM
DOUBLE COMPLEX ES(NB)

DOUBLE PRECISION DX,SW,PHI0,K,V,T1,T2,T3,T4,RJ,Z1,ZETA,EARG
DOUBLE COMPLEX CV,C1,C2,C3,C4
INTEGER NS,NB,I,J

```

```

DOUBLE PRECISION RADOFDEG
DOUBLE PRECISION DMATGET1
DOUBLE COMPLEX CMATGET1
DOUBLE PRECISION PI
DATA PI / 3.14159265358979324D0 /

THET = RADOFDEG(THET)
DX = DMATGET1(XS,2,NS)-DMATGET1(XS,1,NS)
SW = DMATGET1(XS,NS,NS)-DMATGET1(XS,1,NS)
PHI0 = 0.0
K = 2.0*PI/LAM

Z1 = DMATGET1(ZS,1,NS)

DO I = 1,NB
  DO J = 1,NS
    T1 = DMATGET1(XB,I,NB)
    T2 = DMATGET1(ZB,I,NB)
    T3 = DMATGET1(XS,J,NS)
    T4 = DMATGET1(ZS,J,NS)

    RJ = SQRT((T1-T3)**2 + (T2-T4)**2)
    ZETA = (T4-Z1)*COS(THET)+((J-1)*DX)*SIN(THET)
    EARG = K*(RJ+ZETA)

    C1 = CMATGET1(ES,I,NB)
    C2 = ZEXP(CMPLX(0.0,EARG))
    C3 = CMPLX(RJ,0.0)

    C4 = CMPLX(DMATGET1(E0,J,NS),0.0)
    CV = C1 + C4 * C2 / C3

    CALL CMATSET1(ES,I,NB,CV)
  ENDDO
ENDDO

DO I = 1,NB
  V = (ZABS(CMATGET1(ES,I,NB))/NS)**2
  CALL DMATSET1(IS,I,NB,V)
ENDDO
RETURN
END

```

---

**Source code:** `scatt_ext.c`

**Optimierte Implementierung mit VAMLAB**

---

```

#include <stddef.h>
#include <stdarg.h>
#include <string.h>

#include "caml/alloc.h"
#include "caml/custom.h"

```

```

#include "caml/fail.h"
/*#include "intext.h"*/
#include "caml/memory.h"
#include "caml/mlvalues.h"

#include "matrix.h"
#include "f2c.h"
#include "matrix_f.h"

extern
double real gauss_(double real *x,double real *w);

CAMLprim value
gauss_ext(value vx,value vw)
{
    double real g;
    double real x=Double_val(vx);
    double real w=Double_val(vw);
    g=gauss_(&x,&w);
    return (copy_double(g));
};

extern
int bplane_(
    double real *xb,
    double real *zb,
    integer *nb,
    double real *wid,
    double real *gam,
    double real *r__);

/*
** Args:
** (xbmat,zbmat,wid,gam,r)
*/

CAMLprim value
bplane_ext(value vt)
{
    double real wid = Double_val(Field(vt,2));
    double real gam = Double_val(Field(vt,3));
    double real r = Double_val(Field(vt,4));
    Matrix_dim1(Field(vt,0),nb);
    Matrix_dim1(Field(vt,1),nb2);
    Matrix_double(Field(vt,0),xb);
    Matrix_double(Field(vt,1),zb);

    if (nb != nb2)
        failwith ("bplane: xb and zb of different size!\n");
    bplane_(xb,zb,&nb,&wid,&gam,&r);
    return Val_unit;
};

```

```

extern
int reflec_(
    doublereal *xs, doublereal*zs,
    doublereal *e0,
    integer *ns,
    doublereal *thet, doublereal *gamm, doublereal *r0);

/*
** Args:
** (xsmat,zsmat,e0mat,thet,gamm,r0)
*/

CAMLprim value
reflec_ext(value vt)
{
    doublereal thet = Double_val(Field(vt,3));
    doublereal gamm = Double_val(Field(vt,4));
    doublereal r0 = Double_val(Field(vt,5));

    Matrix_dim1(Field(vt,0),ns);
    Matrix_dim1(Field(vt,1),ns2);
    Matrix_double(Field(vt,0),xs);
    Matrix_double(Field(vt,1),zs);

    Matrix_dim1(Field(vt,2),ne0);
    Matrix_double(Field(vt,2),e0);

    if(ns != ns2)
        failwith ("scatt_ext: xs and zs of different size");
    if(ne0 != ns)
        failwith ("scatt_ext: e0 of wrong size");

    reflec_(xs,zs,e0,&ns,&thet,&gamm,&r0);
    return Val_unit;
};

extern
int scatt_(
    doublereal *xs, doublereal*zs, doublereal*xb, doublereal*zb,
    doublecomplex *es,
    doublereal *e0, doublereal *is,
    integer *ns, integer *nb,
    doublereal *thet, doublereal*lam);

/*
** Args:
** (xsmat,zsmat,xbmat,zbmat,esmat,e0mat,ismat,thet,lam)
*/

CAMLprim value
scatt_ext(value vt)
{
    doublereal thet = Double_val(Field(vt,7));

```

```

doublereal lam = Double_val(Field(vt,8));

Matrix_dim1(Field(vt,0),ns);
Matrix_dim1(Field(vt,1),ns2);
Matrix_double(Field(vt,0),xs);
Matrix_double(Field(vt,1),zs);
Matrix_dim1(Field(vt,2),nb);
Matrix_dim1(Field(vt,3),nb2);
Matrix_double(Field(vt,2),xb);
Matrix_double(Field(vt,3),zb);

Matrix_dim1(Field(vt,4),nes);
Matrix_dim1(Field(vt,5),ne0);
Matrix_dim1(Field(vt,6),nis);
Matrix_complex(Field(vt,4),es);
Matrix_double(Field(vt,5),e0);
Matrix_double(Field(vt,6),is);

if(ns != ns2)
    failwith ("scatt_ext: xs and zs of different size");
if(nb != nb2)
    failwith ("scatt_ext: xb and zb of different size");
if(nes != nb)
    failwith ("scatt_ext: es of wrong size");
if(ne0 != ns)
    failwith ("scatt_ext: e0 of wrong size");
if(nis != nb)
    failwith ("scatt_ext: is of wrong size");

scatt_(xs,zs,xb,zb,es,e0,is,&ns,&nb,&thet,&lam);
return Val_unit;
};

```

---

## Ergebnisse mit Reflexionskoeffizient

---

Wird der Reflexionskoeffizient der Oberfläche berücksichtigt, zeigt sich eine deutlich bessere Übereinstimmung der berechneten Streulichtverteilungen mit experimentellen Messungen, insbesondere die Abhängigkeit der Streulichtverteilung von der Oberflächenrauheit.

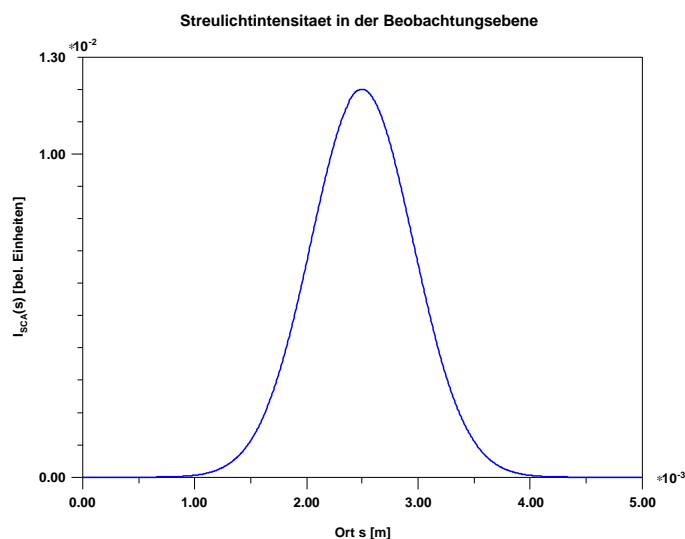
$$E_0 = E_0 \left( \frac{dz}{dx} (\sin(\theta)(1 - R_0) + \sin(\gamma)(1 + R_0)) - (\cos(\gamma)(1 + R_0) - \cos(\theta)(1 - R_0)) \right) \quad (15)$$

mit  $R_0$  als generalisierter Reflexionskoeffizient des Oberflächenmaterials.

Verwendete Parameter: Lichtwellenlänge  $\lambda=600$  nm, Einfallswinkel  $\theta, \gamma=30^\circ$ , Beleuchtungsdurchmesser 0.5 mm, Bildabstand 80 mm, Oberflächenwellenlänge 30  $\mu\text{m}$ . Die Oberflächenrauheit wurde variiert.

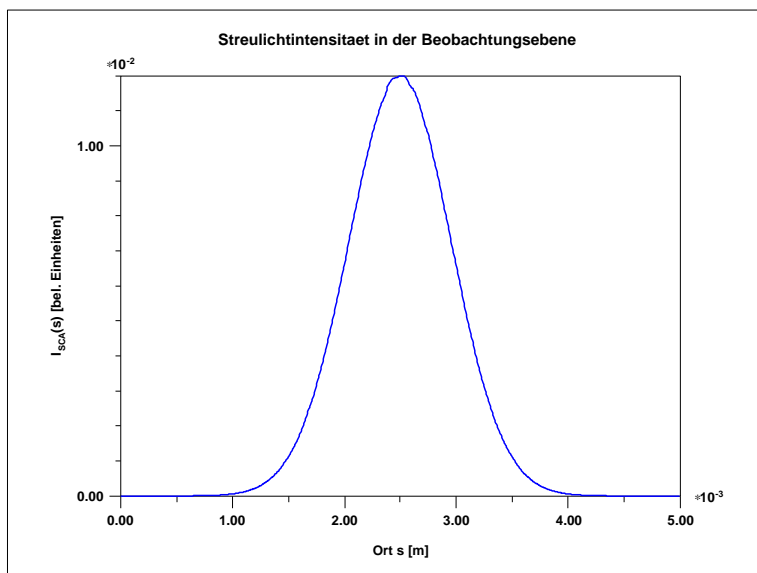
**Fig. 9** bis **Fig. 13** zeigen die berechneten Streulichtverteilungen in Abhängigkeit von der Oberflächenrauheit. Ab einer Rauheit von  $R_Q=100$  nm zeigt sich eine deutliche Zunahme des diffusen Streulichts. Qualitativ zeigt sich eine Übereinstimmung mit experimentellen Messungen an rauen Oberflächen.

**Fig. 9**



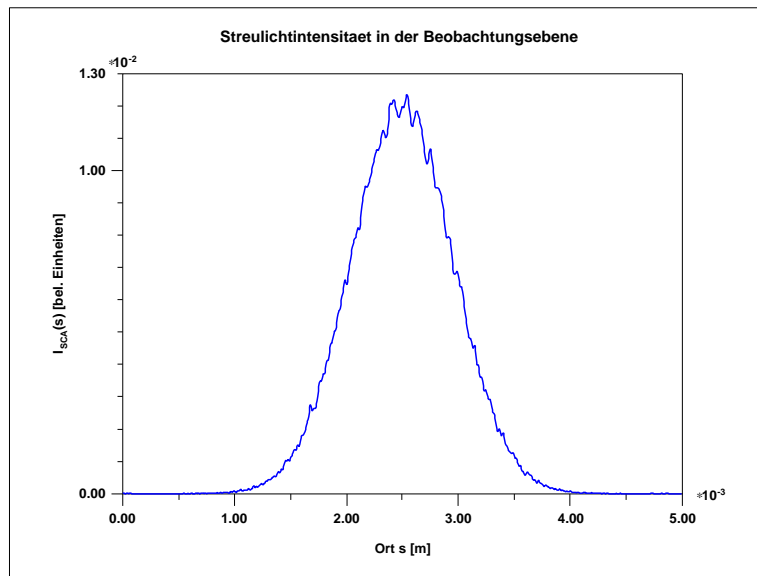
*Berechnete Streulichtintensität mit  $R_Q=1$  nm.*

**Fig. 10**



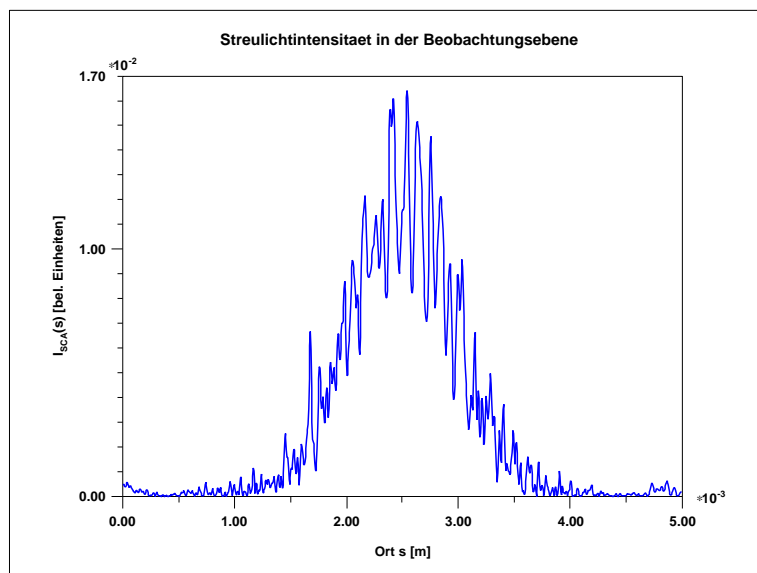
*Berechnete Streulichtintensität mit  $R_Q=10$  nm.*

Fig. 11



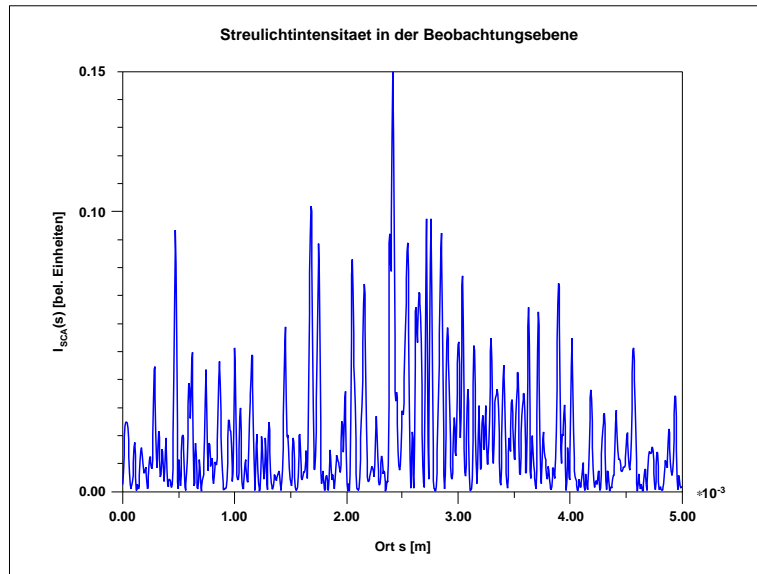
*Berechnete Streulichtintensität mit  $R_Q=100$  nm.*

Fig. 12



*Berechnete Streulichtintensität mit  $R_Q=1000$  nm.*

Fig. 13



Berechnete Streulichtintensität mit  $R_0=10000$  nm.

Source code: scatt.f

Ergebnisse mit Reflexionskoeffizient

```
C      Only changes from scatt.f:
C
C      Initialize and calculate the electrical field amplitudes
C
C      => Surface gradient, incident and observation angle
C      dependent reflection coefficient controls E0
C      => Gaussian transversal distribution
C
C      sin(thet)*dz/dx+cos(thet)
C
C      THET: incident angle
C      GAMM: observation angle
C      R0: reflection coefficient

      SUBROUTINE REFLEC(XS,ZS,E0,NS,THET,GAMM,R0)
      DOUBLE PRECISION XS(NS),ZS(NS),E0(NS)
      DOUBLE PRECISION THET,GAMM

      DOUBLE PRECISION DX,DZ,V,SW,R0
      DOUBLE PRECISION Q1,Q2,Q3,Q4
      INTEGER NS,I

      DOUBLE PRECISION RADOFDEG
      DOUBLE PRECISION DMATGET1
      DOUBLE PRECISION GAUSS
```

```
THET = RADOFDEG(THET)
GAMM = RADOFDEG(GAMM)
DX = DMATGET1(XS,2,NS)-DMATGET1(XS,1,NS)
SW = DMATGET1(XS,NS,NS)-DMATGET1(XS,1,NS)
```

```
WRITE (*,*) THET,GAMM,R0
```

```
Q1=SIN(THET)
Q2=COS(THET)
Q3=SIN(GAMM)
Q4=COS(GAMM)
```

```
DO I = 1,NS
```

```
  V = GAUSS(DMATGET1(XS,I,NS),SW)
```

```
  IF (I .EQ. NS) THEN
```

```
    DZ = 1E-11
```

```
  ELSE
```

```
    DZ = (DMATGET1(ZS,I+1,NS)-DMATGET1(ZS,I,NS))
```

```
  END IF
```

```
    V=V*(Q1*DZ/DX+Q2)
```

```
    V=V*(DZ/DX*(Q1*(1.0-R0)+Q3*(1.0+R0))-(Q4*(1+R0)-Q2*(1-R0)))
```

```
    CALL DMATSET1(E0,I,NS,V)
```

```
ENDDO
```

```
RETURN
```

```
END
```

C



---

## Literaturverzeichnis

---

[OGI91]

*Theory of wave scattering from random rough surfaces*

J.A. Ogilvy, 1991, IOP

[PSI98]

*PsiLAB: Scientific and numeric research software environment*

Stefan Bosse, <http://psilab.sourceforge.net>

[VLA02]

*VAMLAB: Distributed Scientific and numeric research software environment*

Stefan Bosse, <http://www.bsslabs.de>

[OCA02]

*OCAML: The Caml language*

INRIA, <http://caml.inria.fr>

## Table of Content

Eindimensionale Oberflächen. . . . .	2
Mathematische Modellierung . . . . .	2
Oberfläche . . . . .	2
Lichtstreuung . . . . .	3
Randbedingungen . . . . .	5
Numerische Simulation . . . . .	6
Source code: surf.ml . . . . .	6
Source code: scatt.ml. . . . .	8
Ergebnisse . . . . .	11
Optimierte Implementierung mit VAMLAB . . . . .	15
Ergebnisse . . . . .	15
Source code: surf.ml . . . . .	15
Source code: surf.f . . . . .	17
Source code: surf_ext.c . . . . .	19
Source code: scatt.ml. . . . .	21
Source code: scatt.f . . . . .	23
Source code: scatt_ext.c . . . . .	26
Ergebnisse mit Reflexionskoeffizient . . . . .	29
Source code: scatt.f . . . . .	32
Zweidimensionale Oberflächen. . . . .	34
Literaturverzeichnis . . . . .	35