Applied Microscopy
Korean Society of Microscopy

pISSN 2287-5123 · eISSN 2287-4445
http://dx.doi.org/10.9729/AM.2015.45.2.101

Technical Report

DigitalMicrograph Script Source Listing for a Geometric Phase Analysis

Kyou-Hyun Kim*

Advanced Process and Materials R&BD Group, Incheon Regional Division, Korea Institute of Industrial Technology, Incheon 406-840, Korea

*Correspondence to:
Kim KH,
Tel: +82-32-850-0431
Fax: +82-32-850-0430
E-mail: khkim1308@kitech.re.kr

Received April 17, 2015
Revised June 25, 2015
Accepted June 25, 2015

Numerous digital image analysis techniques have been developed with regard to transmission electron microscopy (TEM) with the help of programming. DigitalMicrograph (DM, Gatan Inc., USA), which is installed on most TEMs as operational software, includes a script language to develop customized software for image analysis. Based on the DM script language, this work provides a script source listing for quantitative strain measurements based on a geometric phase analysis.

Key Words: DigitalMicrograph script, Transmission electron microscopy, Quantitative strain measurement, Geometric phase analysis

INTRODUCTION

Transmission electron microscopy (TEM) is a powerful tool for analyzing a broad range of materials with a variety of analysis techniques. Conventional TEM analysis techniques have been improved continuously since TEM was developed in 1930s. One recent advance is the incorporation of aberration correctors, which drastically enhances the resolution of TEM. On the other hand, numerous data analysis techniques based on post-image processing have also been introduced in an effort to obtain more information from raw TEM data, which is scarcely distinguishable by empirical methods or manual calculations (Buseck et al., 1988; Reetz et al., 2000; Zuo et al., 2003; Huang et al., 2009; Kim et al., 2013a).

Several tools, such as Matlab, C/C++ and Python, can be used to develop software for advanced image processing from TEM images. DigitalMicrograph (DM, Gatan Inc., USA) also provides a scripting language called DM script for developing data analysis software for TEM. The syntax of DM is very similar to that of C/C++. Unlike other programming languages, however, DM script simultaneously serves as a compiler to create an object file, as a linker to create an executable file, and as a runtime environment. More precisely, DM works as not a low-level language but as an interpreter for customized scripts. For this reason, its computation speed is relatively slow. Nevertheless, DM script provides a convenient programming environment with a variety of functions to manipulate images and to access the TEM instrument for customized operations. Several analysis techniques have already been reported based on the DM script (Kim & Zuo, 2013; Kim et al., 2015). Kim and colleagues recently introduced new algorithms to quantify the symmetry recorded in convergent beam electron diffraction (CBED) patterns and to perform scanning CBED for symmetry mapping (Kim et al., 2012; Kim, 2013; Kim et al., 2013b; Kim & Zuo, 2013). Also, the DM script database contains a considerable number of applications for customized software scripting using the DM script language (FELMI-ZFE [Internet]).

Using the advantages of DM script, this work provides a DM script source listing for quantitative strain measurements. Quantitative strain, or displacement measurement, is a widely used image processing technique for materials. Algorithms for strain mapping are mainly classified into two groups in terms of real and reciprocal space. The Peak Pairs algorithm (PPA) calculates the local displacement using the intensity maximum of the peaks between each pair of neighboring lattice points

This work was supported by a research grant from the Korea Institute of Industrial Technology (KITECH).

in real space (Galindo et al., 2007). Another algorithm known as the geometric phase analysis was proposed by Hÿtch et al. (1998). Geometric phase analysis (GPA) is based on phase retrieval calculated in the Fourier (reciprocal) space. While the two techniques are complementary to each other, PPA can fail to calculate the local displacement with a complex lattice structure (Galindo et al., 2007). In addition, GPA is created relatively easily with DM script language considering the computation process of the DM script. As mentioned earlier, DM script only works as an interpreter, unlike other low-level programming languages. It therefore requires a significant amount of computation time for a complex loop structure. The PPA algorithm requires a pixel-by-pixel operation to find neighboring lattice points such that the computation time increases rapidly, even for a small two-dimensional image. A dynamic-link library (dll) file may need to be created using C++ in order to create the script for PPA on DM. In contrast, the DM script has the basic functions for basic Fourier/inverse Fourier transform and image processing, which are the major functions for the GPA algorithm. This results in a reasonable computation time reduction on DM. In this report, a DM script source listing for GPA is created and listed in the Appendix (online only at http://www.appmicro.org) to be used directly on DM.

THEORY

Hÿtch et al. (1997, 1998) proposed a means of quantifying local displacement and strain fields from a high-resolution (HR) image recorded using TEM. This section will briefly introduce the fundamental theory of GPA as proposed by Hÿtch et al. (1997, 1998). More details about the GPA can be found in the literature (Hÿtch et al., 1998).

The image intensity at position $\boldsymbol{r}$ can be expressed as follows,

$$I(\boldsymbol{r})=\sum_g H_g(\boldsymbol{r})\exp\{2\pi i\boldsymbol{g}\cdot\boldsymbol{r}\}, \quad (1)$$

where $\boldsymbol{g}$ is the Bragg position and $\boldsymbol{r}$ is the position vector of intensity at a pixel (x, y) in an image. The Fourier coefficient, $H_g(\boldsymbol{r})$, is then described as

$$H_g(\boldsymbol{r})=A_g(\boldsymbol{r})\exp\{iP_g(\boldsymbol{r})\}, \quad (2)$$

where the amplitude $A_g(\boldsymbol{r})$ and phase $P_g(\boldsymbol{r})$ respectively represent the contrast level and lateral position of the fringes in the raw image. The inverse Fourier transform of equation (2) results in the complex image $H'_g(\boldsymbol{r})$:

$$H'_g(\boldsymbol{r})=H_g(\boldsymbol{r})\{2\pi i\boldsymbol{g}\cdot\boldsymbol{r}\} \quad (3)$$

Equation 2 is substituted into equation (3) for $H_g(\boldsymbol{r})$.

$$H'_g(\boldsymbol{r})=A_g(\boldsymbol{r})\{2\pi i\boldsymbol{g}\cdot\boldsymbol{r}+iP_g(\boldsymbol{r})\} \quad (4)$$

Equation (4) is then used to calculate the Bragg-filtered image intensity, amplitude, phase, and raw-phase images from the original image in the following manner,

$$\text{Bragg-filtered image intensity: } B_g(\boldsymbol{r})=2\text{Real}[H'_g(\boldsymbol{r})] \quad (5)$$

$$\text{Amplitude image: } A_g(\boldsymbol{r})=\text{Mod}[H'_g(\boldsymbol{r})] \quad (6)$$

$$\text{Phase image: } P_g(\boldsymbol{r})=\text{Phase}[H'_g(\boldsymbol{r})]-2\pi\boldsymbol{g}\cdot\boldsymbol{r} \quad (7)$$

$$\text{Raw-phase image: } P_g(\boldsymbol{r})=\text{Phase}[H'_g(\boldsymbol{r})], \quad (8)$$

where Real, Mod and Phase, respectively denote the real part, modulus and phase of $[H'_g(\boldsymbol{r})]$. The two-dimensional displacement, $u(\boldsymbol{r})$, is then calculated from equation (7) as follows:

$$u(\boldsymbol{r})=-\frac{1}{2\pi}[P_{g_1}(\boldsymbol{r})\boldsymbol{a}_1+P_{g_2}(\boldsymbol{r})\boldsymbol{a}_2 \quad (9)$$

In this equation, the vectors $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ correspond to the lattice vectors in real space as defined by the reciprocal lattice vectors of $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ ($\boldsymbol{g}_i\cdot\boldsymbol{a}_j=\boldsymbol{\delta}_{ij}$). The strain and rotation tensors are finally given by the derivatives of the two-dimensional displacement, $u(\boldsymbol{r})$,

$$=\varepsilon=\frac{1}{2}\{e+e^T\} \quad (10)$$

$$=\omega=\frac{1}{2}\{e-e^T\}, \quad (11)$$

where $e=\begin{pmatrix}e_{xx} & e_{xy}\\ e_{yx} & e_{yy}\end{pmatrix}=\begin{pmatrix}\frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y}\\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y}\end{pmatrix}=-\frac{1}{2\pi}\begin{pmatrix}a_{1x} & a_{2x}\\ a_{1y} & a_{2y}\end{pmatrix}\begin{pmatrix}\frac{\partial P_{g1}}{\partial x} & \frac{\partial P_{g1}}{\partial y}\\ \frac{\partial P_{g2}}{\partial x} & \frac{\partial P_{g2}}{\partial y}\end{pmatrix}$.

THE PROGRAM

Fig. 1 shows the GPA software installed on DM. In the Appendix, the script source listing can be found and can simply be installed on DM included in Gatan Microscopy Suite (GMS, version 2.11, 32-bit architecture; Gatan Inc.). GMS can be freely downloaded from Gatan's website.

A fast Fourier transform or power spectrum image is initially calculated from an HR image in order to measure the local strain, rotation and displacement. For the power spectrum image, the software provides a "Gaussian edge smoothing" function to make spots more distinguishable in the calculated power spectrum. From the power spectrum, any two non-linear $\boldsymbol{g}$-vectors can be chosen using two virtual apertures (the oval annotations in DM). Fig. 2 shows an example of

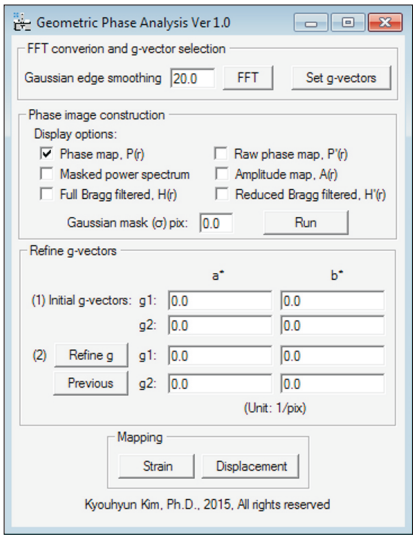

Fig. 1. Dialogue box for the geometric phase analysis software installed on DigitalMicrograph (Gatan Inc., USA).

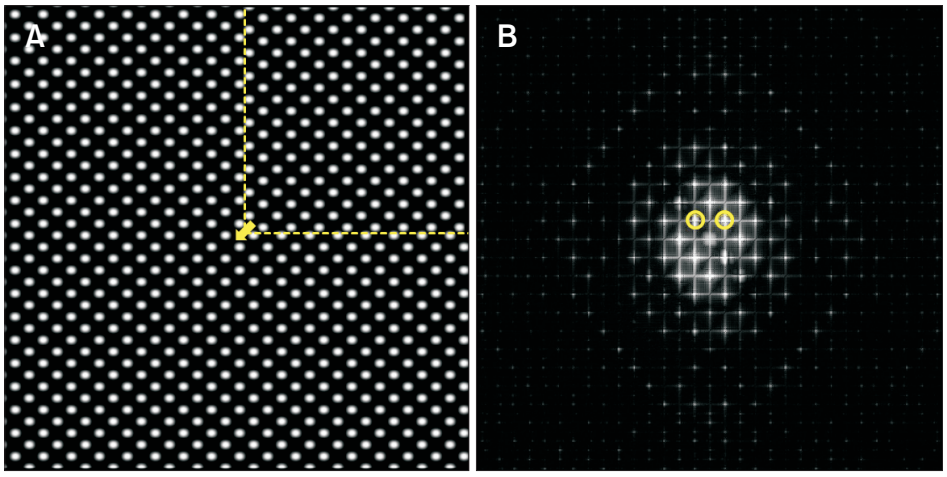

Fig. 2. (A) Simulated high-resolution image for geometric phase analysis. (B) Gaussian-edge-smoothed power spectrum calculated from Fig. 2A.

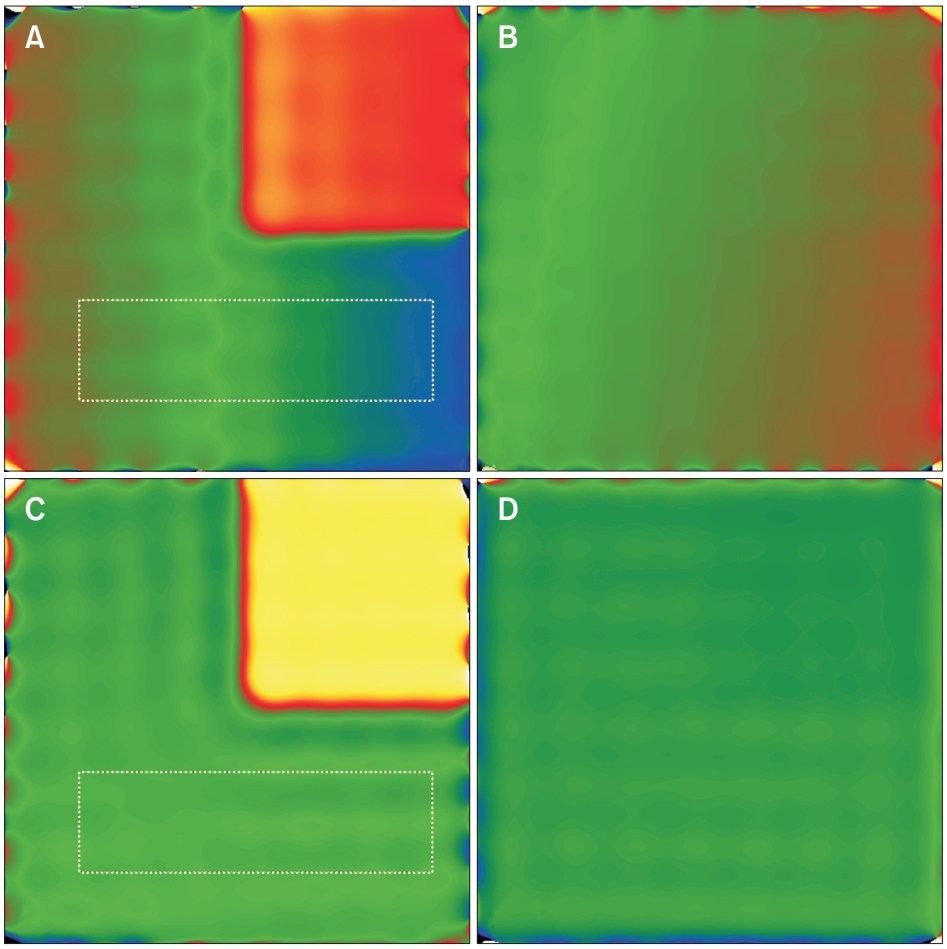

Fig. 3. (A, B) Calculated P(r) image for two g-vectors. (C, D) P(r) images with the refined g-vectors. A region-of-interest box automatically appears on one of the P(r) images to refine the selected g-vectors.

a simulated HR image and Fig. 2B is the power spectrum calculated from Fig. 2A with a Gaussian edge smoothing value of 50. In order to determine two $\boldsymbol{g}$-vectors, two non-linear spots are selected with the oval annotations (yellow circles) as shown in Fig. 2B and are then stored in the software using the "Set $\boldsymbol{g}$-vectors" function. In this process, two reciprocal lattice vectors are determined by the distance from the center of the power spectrum to the intensity maximum of the peaks. The intensity maximum can be calculated with sub-pixel accuracy based on the center of mass of the intensity of the peaks (the CenterOfMass subroutine in the script source; please see the appendix).

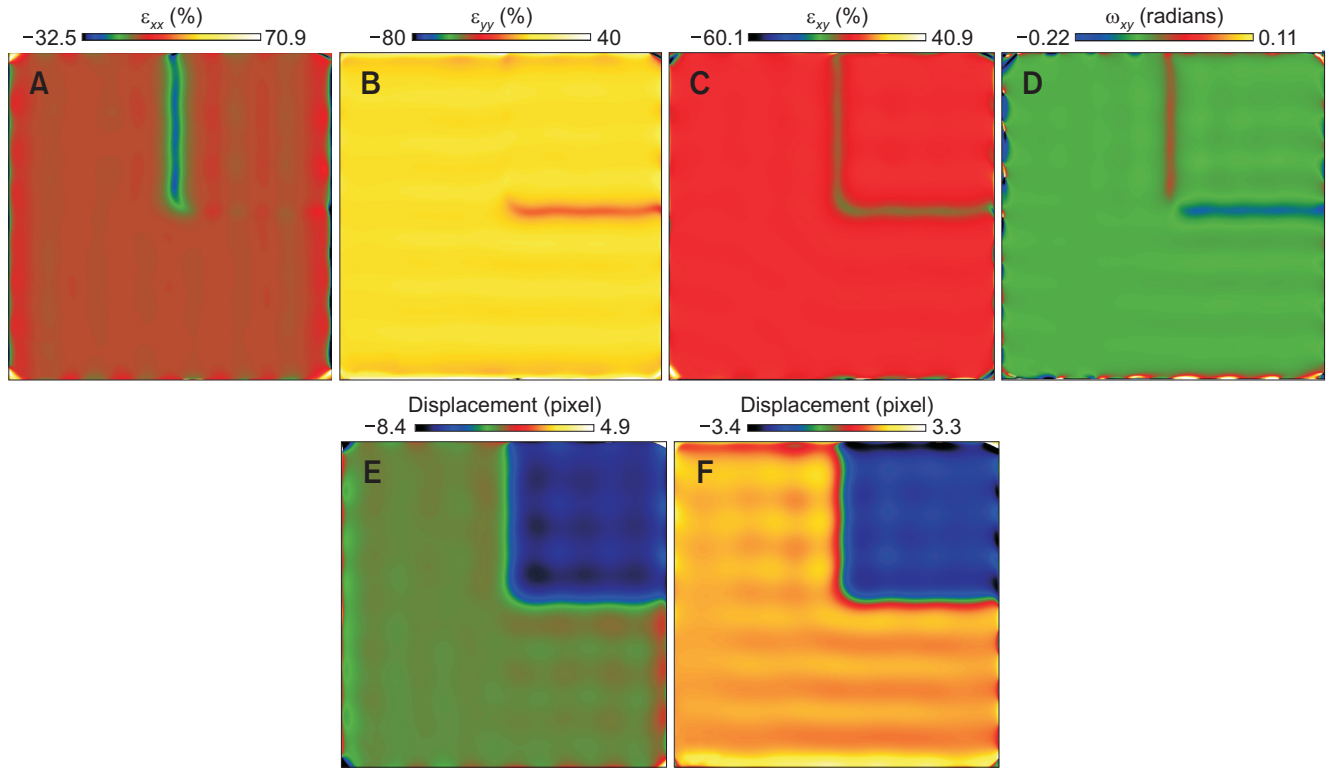

Fig. 4. Calculated strain maps for ε_{xx} (A), ε_{yy} (B), and ε_{xy} (C); rotation map for ω_{xy} (D); two dimensional-displacement u(r) along x (E) and y (F).

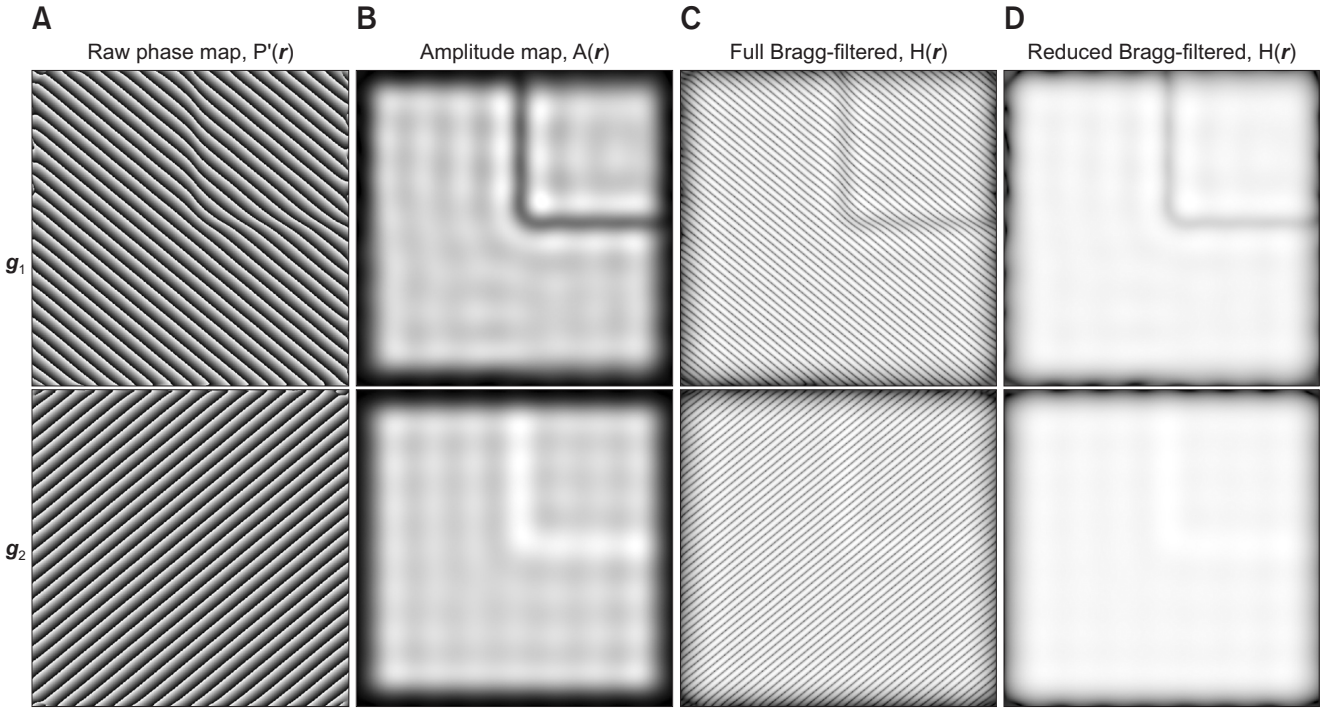

Fig. 5. Raw phase map P'($\boldsymbol{r}$) (A), amplitude map A($\boldsymbol{r}$) (B), full Bragg-filtered map H($\boldsymbol{r}$) (C), and reduced Bragg-filtered H($\boldsymbol{r}$) (D) for the lattice reciprocal vectors of $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$.

Once two ***g***-vectors are selected, a phase map, P(***r***), can be generated with the "Run" button in the phase image construction menu. Fig. 3 shows the generated P(***r***) images for the selected ***g***-vectors. From the first calculated P(***r***) images, the selected ***g***-vectors can be refined. As shown in Fig. 3A, a region-of-interest tool automatically appears on one of the P(***r***) images. The user then needs to choose a proper reference area and then refine the *g*-vectors using the "Refine-g" button. New P(***r***) images are then reloaded with the refined *g*-vectors using the "Run" button in the phase image construction menu. These procedures should be repeated until the ***g***-vectors have the same values. Fig. 3C and D show the final P(***r***) images after several refinement procedures.

Based on the refined P(***r***) images, the local strain (ε), rotation (ω) can be calculated using the "strain" button in the mapping menu (see Fig. 1). Fig. 4A-D show the calculated strain maps of ε_x (Fig. 4A), ε_y (Fig. 4B), ε_{xy} (Fig. 4C) and the rotation matrix ω_{xy} (Fig. 4D). The local strain fields are observed in the calculated strain maps. A user can also obtain the map of $u(\boldsymbol{r})$ by using the "displacement" button in the mapping menu. The two-dimensional displacements, $u(\boldsymbol{r})$, are used to calculate the strain maps with equation (9) to (11). Fig. 4E and F display $u(\boldsymbol{r})$ for the x and y direction, respectively. The unit of $u(\boldsymbol{r})$ is determined by the unit of used images. Thus, Fig. 4E and F are represented based on the unit of pixel because the lattice vectors calculated from the simulated HR image have the unit of pixel.

Other images of the raw phase image, masked power spectrum, amplitude map, and full/reduced Bragg-filtered images can be also obtained from the software as shown in Fig. 5. From the raw phase image, the local reciprocal lattice ***g***-vectors are directly calculated by differentiation: $\nabla P'_g(\boldsymbol{r})=2\pi \boldsymbol{g}(\boldsymbol{r})$. This is then used to calculate the phase image $P_g(\boldsymbol{r})$ by subtracting $2\pi \boldsymbol{g}\cdot\boldsymbol{r}$. The corresponding algorithm can be found in the Appendix (please see the "//Phase image, P(r)"). While the amplitude map and full/reduced Bragg-filtered images are not directly related to the calculation of strain map, they can be applied to image an antiphase domain boundary (Hÿtch, 1997) or dislocations (Hÿtch et al., 2003). In order to obtain the images shown in Fig. 5, a user needs to check "display options" in the phase image construction menu.

CONFLICT OF INTEREST

No potential conflict of interest relevant to this article was reported.

REFERENCES

Buseck P R, Epelboin Y, and Rimsky A (1988) Signal processing of high-resolution transmission electron microscope images using Fourier transforms. *Acta Cryst. A* **44**, 975-986.

FELMI-ZFE: DigitalMicrograph™ script database [Internet]. Available from: http://portal.tugraz.at/portal/page/portal/felmi/DM-Script.

Galindo P L, Kret S, Sanchez A M, Laval J, Yáñez A, Pizarro J, Guerrero E, Ben T, and Molina S I (2007) The peak pairs algorithm for strain mapping from HRTEM images. *Ultramicroscopy* **107**, 1186-1193.

Hÿtch M J (1997) Geometric phase analysis of high resolution electron microscope images. *Scanning Microscopy* **11**, 53-66.

Hÿtch M J, Pataux J L, and Penisson J M (2003) Measurement of the displacement field of dislocations to 0.03 Å by electron microscopy. *Nature* **423**, 270-273.

Hÿtch M J, Snoeck E, and Kilaas R (1998) Quantitative measurement of displacement and strain fields from HREM micrographs. *Ultramicroscopy* **74**, 131-146.

Huang W J, Zuo J M, Jiang B, Kwon K W, and Shim M S (2009) Sub-angstrom-resolution diffractive imaging of single nanocrystals. *Nature Phys.* **5**, 129-133.

Kim H, Meng Y, Rouviére J L, Isheim D, Seidman D N, and Zuo J M. (2013a) Atomic resolution mapping of interfacial intermixing and segregation in InAs/GaSb superlattices: a correlative study. *J. Appl. Phys.* **113**, 103511.

Kim K H (2013) Local symmetry and polarization in relaxor-based ferroelectric crystals, Materials Science and Engineering, (University of Illinois at Urbana-Champaign, Champaign).

Kim K H, Payne D A, and Zuo J M (2012) Symmetry of piezoelectric (1-x) $Pb(Mg_{1/3}Nb_{2/3})O_3$-x$PbTiO_3$ (x=0.31) single crystal at different length scales in the morphotropic phase boundary region. *Phys. Rev. B* **86**, 184113.

Kim K H, Payne D A, and Zuo J M (2013b) Determination of fluctuations in local symmetry and measurement by convergent beam electron diffraction: applications to a relaxor-based ferroelectric crystal after thermal annealing. *J. Appl. Cryst.* **46**, 1331-1337.

Kim K H, Xing H, Zuo J M, Zhang P, and Wang H (2015) TEM based high resolution and low-dose scanning electron nanodiffraction technique for nanostructure imaging and analysis. *Micron* **71**, 39-45.

Kim K H and Zuo J M (2013) Symmetry quantification and mapping using convergent beam electron diffraction. *Ultramicroscopy* **124**, 71-76.

Reetz M T, Maase M, Schilling T, and Tesche B (2000) Computer image processing of transmission electron micrograph pictures as a fast and reliable tool to analyze the size of nanoparticles. *J. Phys. Chem. B* **104**, 8779-8781.

Zuo J M, Vartanyants I, Gao M, Zhang R, and Nagahara L A (2003) Atomic resolution imaging of a carbon nanotube from diffraction intensities. *Science* **300**, 1419-1421.

Appendix. Script source for the geometric phase analysis

```
//Commercial use is not allowed.
//Kyouhyun Kim, Ph.D. (2015)
//Variables
Image Front
Number F_xSize, F_ySize, TempPSID
Number BraggFilteredID, SingleFilteredID
Number OriginX, OriginY, ScaleX, ScaleY, CalibFormat
Number PSOriginX, PSOriginY, PSScaleX, PSScaleY
Number NoOfRois
Image RoiInfo:=RealImage("", 4, 4, 4)
Image CalInfo:=RealImage("", 4, 5, 2)
Number CenMassX, CenMassY
String Units, IntensityUnit
Number IntensityOrigin, IntensityScale
Number ReferenceRoiID
Image SingleFiltered, BraggFiltered, P_raw, Phase_r, Phase_complex
Image SingleMask, DoubleMask, MaskedPS, B_r, A_r
Image SingleSaved, P_rawSaved, PhaseSaved, PhaseComplexSaved
Image PhaseImagesID:=RealImage("", 4, 1, 2)
Number RefinedProcedure=0
Image PreviousG:=RealImage("", 4, 2, 2)
Image DxDyImage
Number DefinedGx1, DefinedGy1, DefinedGx2, DefinedGy2

//Variables for TagGroup
TagGroup MaskPSOption, AmpMapOption, FullFFTOption, ReducedFFTOption, RawPhaseOption, PhaseMapOption, GaussianOption, SmoothingEdge
TagGroup InitialGxField, InitialGyField, InitialGx2Field, InitialGy2Field
TagGroup RefinedGxField, RefinedGyField, RefinedGx2Field, RefinedGy2Field
TagGroup DefinedGxField, DefinedGyField, DefinedGx2Field, DefinedGy2Field
Number DefaultMask=0, DefaultAmp=0, DefaultFull=0, DefaultReduced=0, DefaultRawPhase=0, DefaultPhase=1, Sigma
Number InitialG, Smoothing=20

//********Subroutines**********************
Number AnnotInfo(Image Template, Number &Cen_X, Number &Cen_Y, Number &R)
{
	Number i=0, NoOfAnnot, AnnotID, TypeOfAnnot
	NoOfAnnot=CountAnnotations(Template)
	Number MaskT, MaskL, MaskB, MaskR
	While (i<NoOfAnnot)
	{
		AnnotID=GetNthAnnotationID(Template, i)
		TypeOFAnnot=AnnotationType(Template, AnnotID)
		If(TypeOfAnnot==6)
		{
			GetAnnotationRect(Template, AnnotID, MaskT, MaskL, MaskB, MaskR)
			i++
		}
		Else
		{
			i++
		}
	}
	Cen_X=0.5*(MaskL+MaskR)
	Cen_Y=0.5*(MaskT+MaskB)
	R=abs(MaskB-MaskT)*0.5
}

Image CreateOvalMask(Image MaskTemplate, Number Cen_X, Number Cen_Y, Number Radius)
{
	If(sigma!=0)
	{
		MaskTemplate=Tert(SQRT((Cen_X-icol)**2+(Cen_y-irow)**2)<=radius, (1/(2*PI()*Sigma**2))*exp(-((Cen_X-icol)**2+(Cen_Y-irow)**2)/(2*Sigma**2)), 0)
		Image Temp=Modulus(MaskTemplate)
		Number TempMax=Max(Temp)
		MaskTemplate=complex(Real(MaskTemplate)/TempMax, Imaginary(MaskTemplate)/TempMax)
		DeleteImage(Temp)
	}

	Else If(Sigma==0)
	{
		MaskTemplate=Tert(SQRT((Cen_X-icol)**2+(Cen_y-irow)**2)<=radius, 1, 0)
	}

	Return MaskTemplate
}

Image ComplexImageMult(Image ImgA, Image ImgB)
{
	Image ResultImg
	ResultImg=ImgA
	ResultImg=complex((Real(ImgA)*Real(ImgB)-Imaginary(ImgA)*Imaginary(ImgB)), (Real(ImgA)*Imaginary(ImgB)+Imaginary(ImgA)*Real(ImgB)))
	return ResultImg
}

Image ComplexImageSum(Image ComplexA, Image ComplexB)
{
	Image ResultImage
	ResultImage=complex((Real(ComplexA)+Real(ComplexB)), (Imaginary(ComplexA)+Imaginary(ComplexB)))
	Return ResultImage
}

Image ComplexImageSubtract(Image ComplexA, Image ComplexB)
{
	Image ResultImage
	ResultImage=complex((Real(ComplexA)-Real(ComplexB)), (Imaginary(ComplexA)-Imaginary(ComplexB)))
	Return ResultImage
}

Image ComplexImageDivision(Image ComplexA, Image ComplexB)
{
	Image ResultImage
	Image a=1/(Real(ComplexB)**2+Imaginary(ComplexB)**2)
	Image b=Real(ComplexA)*Real(ComplexB)+Imaginary(ComplexA)*Imaginary(ComplexB)
	Image c=-(Real(ComplexA)*Imaginary(ComplexB)-Real(ComplexB)*Imaginary(ComplexA))
	ResultImage=complex(a*b, a*c)
	DeleteImage(a)
	DeleteImage(b)
	DeleteImage(c)
	Return ResultImage
}

Image ImgMod(Image ComplexImg)
```

```
{
	Number x_Size, y_Size
	Get2DSize(ComplexImg, x_Size, y_Size)
	Image ResultImage=RealImage("", 4, x_Size, y_Size)
	ResultImage=SQRT(Real(ComplexImg)**2+Imaginary(ComplexImg)**2)
	Return ResultImage
}

Image ComplexDot(Image ComplexA, Image ComplexB)
{
	Image ResultImage
	ResultImage=Real(ComplexA)*Real(ComplexB)+Imaginary(ComplexA)*Imaginary(ComplexB)
	Return ResultImage
}

Void CenterOfMass(Image Img, Number Cen_X, Number Cen_Y, Number &CenX, Number &CenY)
{
	Image Temp:=RealImage("", 4, 3, 3)
	Temp[0, 0, 3, 3]=Img[Cen_Y-1, Cen_X-1, Cen_Y+2, Cen_X+2]
	Number i, j, xx, yy
	While(j<3)
	{
		While(i<3)
		{
			xx=xx+GetPixel(Temp, i, j)*i
			yy=yy+GetPixel(Temp, i, j)*j
			i++
		}
		i=0
		j++
	}
	j=0
	CenX=xx/Sum(temp)+Cen_X-1
	CenY=yy/Sum(temp)+Cen_Y-1
	DeleteImage(Temp)
}

Void GetVariables(Number Row)
{
	Number Cen_X, Cen_Y, Radius, TempMaxX, TempMaxY, AdjX, AdjY, Gx, Gy
	Image Temp:=GetImageFromID(TempPSID)
	Max(ImgMod(Temp), TempMaxX, TempMaxY)

	Cen_X=0.5*(RoiInfo.GetPixel(1, row)+RoiInfo.GetPixel(3, row))
	Cen_Y=0.5*(RoiInfo.GetPixel(0, row)+RoiInfo.GetPixel(2, row))
	Radius=abs(RoiInfo.GetPixel(0, row)-RoiInfo.GetPixel(2, row))*0.5
	CreateTextAnnotation(Temp, Cen_Y-Radius*1.5, Cen_X-Radius*1.5, ""+(Row+1))
	Image SingleMask:=ComplexImage("", 8, F_xSize, F_ySize)
	SingleMask=CreateOvalMask(SingleMask, Cen_X, Cen_Y, Radius)

	Image ModOfTemp
	ModOFTemp=ImgMod(ComplexImageMult(Temp, SingleMask))
	Max(ModOfTemp, AdjX, AdjY)
	CenterOfMass(ModOfTemp, AdjX, AdjY, CenMassX, CenMassY)

	Gx=(CenMassX-TempMaxX)/F_xSize
	Gy=-(CenMassY-TempMaxY)/F_ySize

	CalInfo.SetPixel(0, row, AdjX)
	CalInfo.SetPixel(1, row, AdjY)
	CalInfo.SetPixel(2, row, Gx)
	CalInfo.SetPixel(3, row, Gy)
	CalInfo.SetPixel(4, row, Radius)

	DeleteImage(singleMask)
	DeleteImage(ModOfTemp)
}

Image ImageDerivative(Image Source, String xy)
{
	Number xSize, ySize
	Get2DSize(Source, xSize, ySize)
	Image ResultImage:=RealImage("", 4, xSize, ySize)

	If (xy=="dx")
	{
		Image Temp:=RealImage("", 4, xSize, ySize)
		Temp[0, 0, ySize, xSize-1]=Source[0, 1, ySize, xSize]
		Image TempRow=2*(Source[0, xSize-1, ySize, xSize])-Source[0, xSize-2, ySize, xSize-1]
		Temp[0, xSize-1, ySize, xSize]=TempRow
		ResultImage=(Temp-Source)
		DeleteImage(Temp)
		DeleteImage(TempRow)
	}

	If (xy=="dy")
	{
		Image Temp:=RealImage("", 4, xSize, ySize)
		Temp[0, 0, ySize-1, xSize]=Source[1, 0, ySize, xSize]
		Image TempCol=2*(Source[ySize-1, 0, ySize, xSize])-Source[ySize-2, 0, ySize-1, xSize]
		Temp[ySize-1, 0, ySize, xSize]=TempCol
		ResultImage=(Source-Temp)
		DeleteImage(Temp)
		DeleteImage(TempCol)
	}
	Return ResultImage
}

Image Gradient(Image Source, String XX)
{
	Number xSize, ySize
	Get2DSize(Source, xSize, ySize)
	Image ResultImage:=ComplexImage("", 8, xSize, ySize)

	Image Temp:=ComplexImage("", 8, xSize, ySize)

	If(XX=="dx")
	{
		Temp[0, 0, ySize, xSize-1]=Source[0, 1, ySize, xSize]
		Image TempRow:=ComplexImage("", 8, 1, ySize)
		Image TempA=Real(Source[0, xSize-1, ySize, xSize])
		Image TempB=Source[0, xSize-2, ySize, xSize-1]
		TempRow=ComplexImageSubtract(2*TempA, TempB)
		Temp[0, xSize-1, ySize, xSize]=TempRow[0, 0, ySize, 1]
		ResultImage=ComplexImageSubtract(Temp, Source)
		DeleteImage(TempRow)
		DeleteImage(TempA)
		DeleteImage(TempB)
	}
```

```
		If(XX=="dy")
		{
			Temp[0, 0, ySize-1, xSize]=Source[1, 0, ySize, xSize]
			Image TempCol:=ComplexImage("", 8, xSize, 1)
			Image TempA=Real(Source[ySize-1, 0, ySize, xSize])
			Image TempB=Source[ySize-2, 0, ySize-1, xSize]
			TempCol=ComplexImageSubtract(TempA, TempB)
			Temp[ySize-1, 0, ySize, xSize]=TempCol[0, 0, 1, xSize]
			ResultImage=ComplexImageSubtract(Source, Temp)
			ResultImage=Complex(Real(ResultImage), Imaginary(ResultImage))

			DeleteImage(TempCol)
			DeleteImage(TempA)
			DeleteImage(TempB)
		}
		Return ResultImage
}


Image PhaseImgNormalization(Image Img)
{
		Return -Pi()*(sgn(Img)-1)+Img
}

Image PhaseImgNormalizationB(Image Img)
{
		Number xx, yy
		Max(abs(Img), xx, yy)
		If(Sgn(GetPixel(Img, xx, yy))==1)
		{
			Return Img-Pi()*(Sgn(Img)+1)
		}

		If(Sgn(GetPixel(Img, xx, yy))==-1)
		{
			Return -Pi()*(sgn(Img)-1)+Img
		}
}

Image OriginAdjust(Image ComplexImg) //Pixel address needs to be converted to a real cartesian coordinates
{
		ComplexImg=complex(Real(complexImg), -Imaginary(ComplexImg))
		Return ComplexImg
}

Number RefineG(Image Img)
{
		Number xSize, ySize, GG
		Get2DSize(Img, xSize, ySize)
		Image Temp=RealImage("1", 4, xSize, ySize)
		Image TempB=RealImage("2", 4, xSize, ySize)
		Image TempC=RealImage("3", 4, xSize, ySize)

		Number TempMin, TempMax
		TempMin=Min(Trunc(Img))
		TempMax=Max(Trunc(Img))

		If(TempMax==0)
		{
			If(abs(TempMin)>TempMax)
			{
				Temp=Trunc(Img)
				Temp=Temp/abs(TempMin)
				Temp=Trunc(Temp)
				Temp=Temp*2*Pi()
				TempB=Img-Temp
				Return GG=sum(TempB)/(xSize*ySize)/(2*Pi())
			}
			Else
			{
				OKDialog("Check RefineG function.")
				exit(0)
			}
			DeleteImage(Temp)
			DeleteImage(TempB)
			DeleteImage(TempC)
		}

		Else
		{
			Temp=sgn(Trunc(Img))
			TempB=(Temp-1)/2
			TempC=((Temp-TempB)*Trunc(Img))/max(trunc(img))
			Img=Img+(TempB+TempC)*-2*Pi()
			Return GG=sum(Img)/(xSize*ySize)/(2*Pi())
		}
		DeleteImage(Temp)
		DeleteImage(TempB)
		DeleteImage(TempC)
}

Image GaussianEdgeSmoothing(Image Source, Number Nedge)
{
		If(Nedge!=0)
		{
			Number top, left, bottom, right, width, height, SizeXX,SizeYY,sizeX,sizeY

			Source.GetSize(sizeX,sizeY)
			Source.GetSelection(top, left, bottom, right)

			// get the size of the image:
			SizeYY = SizeY/2;
			SizeXX = SizeX/2;

			if (Nedge > SizeXX/2)
			{
				Nedge=SizeXX/2
			}
			if (Nedge > SizeYY/2)
			{
				Nedge=SizeYY/2
			}

			Image XX = ExprSize(sizeX,SizeY,SizeXX-abs(icol-SizeXX))/Nedge
			Image YY = ExprSize(sizeX,SizeY,SizeYY-abs(irow-SizeYY))/Nedge

			Source *= tert(XX>1,1,0.5*(1-cos(Pi()*XX)))*tert(YY>1,1,0.5*(1-cos(Pi()*YY)))
			deleteimage(XX)
```

```
                        deleteimage(YY)
        }
        Else
        {
                        Source=Source
        }
                        Return Source
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////        Main funtions start here        /////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////


class CreateGPADialog : uiframe
{
        void M_FFT(object self)
        {
                Front=GetFrontImage()
                Get2DSize(Front, F_xSize, F_ySize)
                DLGGetValue(SmoothingEdge, Smoothing)
                Front=GaussianEdgeSmoothing(Front, Smoothing)
                Image TempPS=ComplexImage("FFT", 8, F_xSize, F_ySize)
                TempPS=RealFFT(Front)
                PSScaleX=1/(ScaleX*F_xSize)
                PSScaleY=1/(ScaleY*F_ySize)
                PSOriginX=-0.5*(F_xSize+1)*PSScaleX
                PSOriginY=-0.5*(F_ySize+1)*PSScaleY
                SetName(TempPS, "FFT")
                TempPS.ShowImage()
                SetWindowSize(TempPS, 512, 512)
                TempPSID=GetImageID(TempPS)
                Front.DeleteImage()
        }
        void M_SetGVector(object self)
        {
                NoOfRois=0
                Image TempPS:=GetImageFromID(TempPSID)
                Number i=0, Row=0, AnnotID, TypeOFAnnot
                Number NoOfAnnot=CountAnnotations(TempPS)
                While (i<NoOfAnnot)
                {
                        AnnotID=GetNthAnnotationID(TempPS, i)
                        TypeOFAnnot=AnnotationType(TempPS, AnnotID)
                        If(TypeOfAnnot==6)
                        {
                                Number MaskT, MaskL, MaskB, MaskR
                                GetAnnotationRect(TempPS, AnnotID, MaskT, MaskL, MaskB, MaskR)
                                RoiInfo.SetPixel(0, row, MaskT)
                                RoiInfo.SetPixel(1, row, MaskL)
                                RoiInfo.SetPixel(2, row, MaskB)
                                RoiInfo.SetPixel(3, row, MaskR)
                                row++
                                i++
                                NoOfRois++
                        }
                        Else
                        {
                                i++
                        }
                }

                i=0

                While(i<NoOfRois)
                {
                        GetVariables(i)
                        i++
                }

                self.LookUpElement("#InitialGx1").DLGValue(CalInfo.GetPixel(2, 0))
                self.LookUpElement("#InitialGy1").DLGValue(CalInfo.GetPixel(3, 0))
                If(NoOfRois>1)
                {
                        self.LookUpElement("#InitialGx2").DLGValue(CalInfo.GetPixel(2, 1))
                        self.LookUpElement("#InitialGy2").DLGValue(CalInfo.GetPixel(3, 1))
                }
                RefinedProcedure=0
        }

        void M_Filtering(object self)
        {
                DLGGetValue(MaskPSOption, DefaultMask)
                DLGGetValue(AmpMapOption, DefaultAmp)
                DLGGetValue(FullFFTOption, DefaultFull)
                DLGGetValue(ReducedFFTOption, DefaultReduced)
                DLGGetValue(RawPhaseOption, DefaultRawPhase)
                DLGGetValue(PhaseMapOption, DefaultPhase)

                SingleSaved:=ComplexImage("", 8, F_xSize*NoOfRois, F_ySize)
                P_rawSaved:=RealImage("", 4, F_xSize*NoOfRois, F_ySize)
                PhaseSaved:=RealImage("", 4, F_xSize*NoOfRois, F_ySize)
                PhaseComplexSaved:=ComplexImage("", 8, F_xSize*NoOfRois, F_ySize)

                Number i=0
                While(i<NoOfRois)
                {
                        SingleMask:=ComplexImage((i+1)+": Filtering Mask", 8, F_xSize, F_ySize)
                        DoubleMask:=ComplexImage((i+1)+": Filtering Mask", 8, F_xSize, F_ySize)
                        MaskedPS:=ComplexImage((i+1)+": Masked FFT", 8, F_xSize, F_ySize)
                        B_r:=RealImage((i+1)+": Intensity, B(r)", 4, F_xSize, F_ySize)
                        A_r:=RealImage((i+1)+": Amplitude, A(r)", 4, F_xSize, F_ySize)
                        P_raw:=RealImage((i+1)+": Raw phase image, P'(r)", 4, F_xSize, F_ySize)

                        If(RefinedProcedure==0)
                        {
                                Phase_r:=RealImage((i+1)+": Phase image, P(r)", 4, F_xSize, F_ySize)
                        }
                        Else If(RefinedProcedure==1)
                        {
                                Phase_r:=GetImageFromID(GetPixel(PhaseImagesID, 0, i))
                        }
```

```
BraggFiltered:=ComplexImage((i+1)+": Bragg filtered for +-g, H(r)", 8, F_xSize, F_ySize) //H(r)
SingleFiltered:=ComplexImage((i+1)+": Bragg filtered for g, H'(r)", 8, F_xSize, F_ySize) //H'(r)

Image TempPS:=GetImageFromID(TempPSID)

MaskedPS=TempPS
DLGGetValue(GaussianOption, Sigma)
SingleMask=CreateOvalMask(SingleMask, CalInfo.GetPixel(0, i), CalInfo.GetPixel(1, i), CalInfo.GetPixel(4, i))
DoubleMask=ComplexImageSum(SingleMask, Rotate(SingleMask, Pi()))

//Bragg filtered for +-g, H(r)
BraggFiltered=IFFT(ComplexImageMult(MaskedPS, DoubleMask))
BraggFiltered=OriginAdjust(BraggFiltered)

//Bragg filtered for g, H'(r)
SingleFiltered=IFFT(ComplexImageMult(MaskedPS, SingleMask))
SingleFiltered=complex(Real(SingleFiltered), -Imaginary(SingleFiltered))
SingleSaved[0, F_xSize*i, F_ySize, F_xSize*(i+1)]=SingleFiltered[0, 0, F_ySize, F_xSize]

//Intensity, B(r)
B_r=2*Real(SingleFiltered)

//Amplitude, A(r)
A_r=ImgMod(SingleFiltered)

//Raw phase image, P'(r)
P_raw=Phase(SingleFiltered)
P_rawSaved[0, F_xSize*i, F_ySize, F_xSize*(i+1)]=P_raw[0, 0, F_ySize, F_xSize]

//Phase image, P(r)
Image r_vector:=ComplexImage((i+1)+": r vector", 8, F_xSize, F_ySize)
r_vector=complex(iCol, -iRow)
Image g_VectorImg:=ComplexImage("", 8, F_xSize, F_ySize)
g_VectorImg=complex(CalInfo.GetPixel(2, i), CalInfo.GetPixel(3, i))

Image GdotR:=ComplexImage((i+1)+": GdotR", 8, F_xSize, F_ySize)
GdotR=exp(2*Pi()*ComplexDot(r_vector, g_vectorImg))
Image Temp=Phase(SingleFiltered)+2*Pi()-2*Pi()*ComplexDot(r_vector, g_vectorImg)
Phase_complex:=ComplexImage((i+1)+" Complex Phase image, P(r)", 8, F_xSize, F_ySize)
Phase_complex=exp(complex(0, Temp))
PhaseComplexSaved[0, F_xSize*i, F_ySize, F_xSize*(i+1)]=Phase_complex[0, 0, F_ySize, F_xSize]
Phase_r=Phase(Phase_complex)
PhaseSaved[0, F_xSize*i, F_ySize, F_xSize*(i+1)]=Phase_r[0, 0, F_ySize, F_xSize]

If(DefaultMask==1)
{
		Image DisplayMaskedPS:=ComplexImage((i+1)+": Masked PS", 8, F_xSize, F_ySize)
		If(DefaultFull==0)
		{
				DisplayMaskedPS=ComplexImageMult(MaskedPS, SingleMask)
		}
		Else
		{
				DisplayMaskedPS=ComplexImageMult(MaskedPS, DoubleMask)
		}
		DisplayMaskedPS.ShowImage()
		SetWindowSize(DisplayMaskedPS, 256, 256)
}
Else If(DefaultMask=0)
{
		SingleMask.DeleteImage()
		DoubleMask.DeleteImage()
		MaskedPS.DeleteImage()
}

If(DefaultAmp==1)
{
		A_r.ShowImage()
		SetWindowSize(A_r, 256, 256)
}
Else If(DefaultAmp==0)
{
		A_r.DeleteImage()
}

If(DefaultFull==1)
{
		BraggFiltered=modulus(BraggFiltered)
		BraggFiltered.ShowImage()
		SetWindowSize(BraggFiltered, 256, 256)
}
Else If(DefaultFull==0)
{
		BraggFiltered.DeleteImage()
}

If(DefaultReduced==1)
{
		SingleFiltered.ShowImage()
		SetWindowSize(SingleFiltered, 256, 256)
}

If(DefaultRawPhase==1)
{
		P_raw.UpdateImage()
		P_raw.ShowImage()
		SetWindowSize(P_raw, 256, 256)
}
Else If(DefaultRawPhase==0)
{
		P_raw.DeleteImage()
}

If(DefaultPhase==1)
{
		SetColorMode(Phase_r, 4)
		SetLimits(Phase_r, -Pi(), Pi())
		If(RefinedProcedure==1)
		{
				Phase_r.UpdateImage()
		}
		Else
		{
				Phase_r.ShowImage()
		}
		SetPixel(PhaseImagesID, 0, i, GetImageID(Phase_r))
		SetWindowSize(Phase_r, 256, 256)
}
```

```
		i++
		}
		If(RefinedProcedure==0)
		{
			ImageDisplay ImgDisp
			ImgDisp=GetImageFromID(GetPixel(PhaseImagesID, 0, 0)).ImageGetImageDisplay(0)
			ROI ReferenceRoi=NewRoi()
			ReferenceRoi.RoiSetRectangle(F_ySize*0.25, F_xSize*0.25, F_xSize*0.75, F_xSize*0.75)
			ImgDisp.ImageDisplayAddRoi(ReferenceROI)
			ReferenceRoiID=ReferenceRoi.ROIGetID()
		}
}

void M_Refine(object self)
{
		Number i=0
		Image TempRawPhase:=RealImage("", 4, F_xSize, F_ySize)

		While(i<NoOfRois)
		{
			TempRawPhase[0, 0, F_xSize, F_ySize]=P_rawSaved[0, F_xSize*i, F_ySize, F_xSize*(i+1)]

			Roi ReferenceRoi=GetRoiFromID(ReferenceROIID)
			Number RoiT, RoiL, RoiB, RoiR
			ReferenceRoi.RoiGetRectangle(RoiT, RoiL, RoiB, RoiR)

			// New start
			Image RefRegion:=RealImage("Phase", 4, RoiR-RoiL, RoiB-RoiT)
			Image TempA:=RealImage("", 4, RoiR-RoiL, RoiB-RoiT)
			Image TempB:=RealImage("", 4, RoiR-RoiL, RoiB-RoiT)

			Number AvgGx, AvgGy, DeltaGx, DeltaGy

			RefRegion[0, 0, RoiB-RoiT, RoiR-RoiL]=TempRawPhase[RoiT, RoiL, RoiB, RoiR]

			AvgGx=RefineG(ImageDerivative(PhaseImgNormalization(RefRegion), "dx"))
			DeltaGx=AvgGx-CalInfo.GetPixel(2, i)
			SetPixel(PreviousG, 0, i, DeltaGx)
			SetPixel(CalInfo, 2, i, AvgGx)
			SetPixel(CalInfo, 0, i, AvgGx*F_xSize+0.5*F_xSize)

			AvgGy=RefineG(ImageDerivative(PhaseImgNormalization(RefRegion), "dy"))
			DeltaGy=AvgGy-CalInfo.GetPixel(3, i)
			SetPixel(PreviousG, 1, i, DeltaGy)
			SetPixel(CalInfo, 3, i, AvgGy)
			SetPixel(CalInfo, 1, i, -AvgGy*F_ySize+0.5*F_ySize)
			i++
		}
		RefinedProcedure=1

		self.LookUpElement("#RefinedGx1").DLGValue(CalInfo.GetPixel(2, 0))
		self.LookUpElement("#RefinedGy1").DLGValue(CalInfo.GetPixel(3, 0))
		If(NoOfRois>1)
		{
			self.LookUpElement("#RefinedGx2").DLGValue(CalInfo.GetPixel(2, 1))
			self.LookUpElement("#RefinedGy2").DLGValue(CalInfo.GetPixel(3, 1))
		}
		TempRawPhase.DeleteImage()
}

Void M_Previous(object self)
{
		Number i=0
		While(i<NoOfRois)
		{
			SetPixel(CalInfo, 2, i, CalInfo.GetPixel(2, i)-GetPixel(PreviousG, 0, i))
			SetPixel(CalInfo, 3, i, CalInfo.GetPixel(3, i)-GetPixel(PreviousG, 1, i))
			i++
		}

		self.LookUpElement("#RefinedGx1").DLGValue(CalInfo.GetPixel(2, 0))
		self.LookUpElement("#RefinedGy1").DLGValue(CalInfo.GetPixel(3, 0))
		If(NoOfRois>1)
		{
			self.LookUpElement("#RefinedGx2").DLGValue(CalInfo.GetPixel(2, 1))
			self.LookUpElement("#RefinedGy2").DLGValue(CalInfo.GetPixel(3, 1))
		}
}

Void M_Strain(object self)
{
		Image IGVect:=RealImage("Inver g-vector", 4, 2, 2)
		Number Gax, Gay, Gbx, Gby
		Gax=CalInfo.GetPixel(2, 0)
		Gay=CalInfo.GetPixel(3, 0)
		Gbx=CalInfo.GetPixel(2, 1)
		Gby=CalInfo.GetPixel(3, 1)
		SetPixel(IGVect, 0, 0, Gax)
		SetPixel(IGVect, 1, 0, Gay)
		SetPixel(IGVect, 0, 1, Gbx)
		SetPixel(IGVect, 1, 1, Gby)
		IGVect=MatrixInverse(IGVect)

		Image TempRef1:=ComplexImage("TempRef1", 8, F_xSize, F_ySize)
		Image TempRef2:=ComplexImage("TempRef2", 8, F_xSize, F_ySize)
		TempRef1[0, 0, F_xSize, F_ySize]=PhaseComplexSaved[0, 0, F_ySize, F_xSize]
		TempRef2[0, 0, F_xSize, F_ySize]=PhaseComplexSaved[0, F_xSize, F_ySize, F_xSize*2]

		Image P1dx:=RealImage("P1dx", 4, F_xSize, F_ySize)
		Image P1dy:=RealImage("P1dy", 4, F_xSize, F_ySize)
		Image TempX1:=ComplexImage("dx_1", 8, F_xSize, F_ySize)
		Image TempY1:=ComplexImage("dy_1", 8, F_xSize, F_ySize)
		TempX1=Gradient(TempRef1, "dx")
		P1dx=Imaginary(ComplexImageMult(complex(Real(TempRef1), -Imaginary(TempRef1)), TempX1))
		TempY1=Gradient(TempRef1, "dy")
		P1dy=Imaginary(ComplexImageMult(complex(Real(TempRef1), -Imaginary(TempRef1)), TempY1))
		TempX1.DeleteImage()
		TempY1.DeleteImage()

		Image P2dx:=RealImage("P2dx", 4, F_xSize, F_ySize)
		Image P2dy:=RealImage("P2dy", 4, F_xSize, F_ySize)
		Image TempX2:=ComplexImage("dx_2", 8, F_xSize, F_ySize)
		Image TempY2:=ComplexImage("dy_2", 8, F_xSize, F_ySize)
		TempX2=Gradient(TempRef2, "dx")
		P2dx=Imaginary(ComplexImageMult(complex(Real(TempRef2), -Imaginary(TempRef2)), TempX2))
		TempY2=Gradient(TempRef2, "dy")
		P2dy=Imaginary(ComplexImageMult(complex(Real(TempRef2), -Imaginary(TempRef2)), TempY2))
		TempX2.DeleteImage()
		TempY2.DeleteImage()
```

```
				Image exx:=RealImage("e_xx", 4, F_xSize, F_ySize)
				Image eyy:=RealImage("e_yy", 4, F_xSize, F_ySize)
				Image exy:=RealImage("e_xy", 4, F_xSize, F_ySize)

				exx=(IGVect.GetPixel(0, 0)*P1dx+IGVect.GetPixel(1, 0)*P2dx)/(-2*PI())*100
				eyy=(IGVect.GetPixel(0, 1)*P1dy+IGVect.GetPixel(1, 1)*P2dy)/(-2*PI())*100
				exy=50*(IGVect.GetPixel(0, 0)*P1dy+IGVect.GetPixel(1, 0)*P2dy+IGVect.GetPixel(0, 1)*P1dx+IGVect.GetPixel(1, 1)*P2dx)/(-2*PI())
				exx.ShowImage()
				SetColorMode(exx, 4)
				SetWindowSize(exx, 512, 512)
				eyy.ShowImage()
				SetColorMode(eyy, 4)
				SetWindowSize(eyy, 512, 512)
				exy.ShowImage()
				SetColorMode(exy, 4)
				SetWindowSize(exy, 512, 512)

				Image Wxy:=RealImage("Rotation matrix, W_xy", 4, F_xSize, F_ySize)
				Wxy=0.5*(IGVect.GetPixel(0, 0)*P1dy+IGVect.GetPixel(1, 0)*P2dy-IGVect.GetPixel(0, 1)*P1dx-IGVect.GetPixel(1, 1)*P2dx)/(-2*PI())
				Wxy.ShowImage()
				SetColorMode(Wxy, 4)
				SetWindowSize(Wxy, 512, 512)

				P1dx.DeleteImage()
				P1dy.DeleteImage()
				P2dx.DeleteImage()
				P2dy.DeleteImage()
		}

		Void M_Displacement(object self)
		{
				Image IGVect:=RealImage("Inver g-vector", 4, 2, 2)
				Number Gax, Gay, Gbx, Gby
				Gax=CalInfo.GetPixel(2, 0)
				Gay=CalInfo.GetPixel(3, 0)
				Gbx=CalInfo.GetPixel(2, 1)
				Gby=CalInfo.GetPixel(3, 1)
				SetPixel(IGVect, 0, 0, Gax)
				SetPixel(IGVect, 1, 0, Gay)
				SetPixel(IGVect, 0, 1, Gbx)
				SetPixel(IGVect, 1, 1, Gby)
				IGVect=MatrixInverse(IGVect)

				Image TempRef1:=RealImage("TempRef1", 8, F_xSize, F_ySize)
				Image TempRef2:=RealImage("TempRef2", 8, F_xSize, F_ySize)
				TempRef1[0, 0, F_xSize, F_ySize]=PhaseSaved[0, 0, F_ySize, F_xSize]
				TempRef2[0, 0, F_xSize, F_ySize]=PhaseSaved[0, F_xSize, F_ySize, F_xSize*2]

				Image ux:=RealImage("u_x", 4, F_xSize, F_ySize)
				Image uy:=RealImage("u_y", 4, F_xSize, F_ySize)

				ux=(IGVect.GetPixel(0, 0)*TempRef1+IGVect.GetPixel(1, 0)*TempRef2)/(-2*PI())
				uy=(IGVect.GetPixel(0, 1)*TempRef1+IGVect.GetPixel(1, 1)*TempRef2)/(-2*PI())

				ux.ShowImage()
				uy.ShowImage()

				TempRef1.DeleteImage()
				TempRef2.DeleteImage()
		}

void abouttoclosedocument(object self)
		{
				// Source and save the dialog position

				number xpos, ypos
				documentwindow dialogwindow=getframewindow(self)
				windowgetframeposition(dialogwindow, xpos, ypos)

				setpersistentnumbernote("Dialog x position",xpos)
				setpersistentnumbernote("Dialog y position",ypos)
		}

////////////////////////////////////////////////
//						Create a dialogue box						//
////////////////////////////////////////////////
TagGroup Box0(object self)
{
		TagGroup Box0_items
		TagGroup Box0=dlgcreatebox("FFT converion and g-vector selection", box0_items)
		Box0.DLGInternalPadding(5, 8)
		Box0.DLGExternalPadding(5, 5)

		TagGroup SmoothLabel=DLGCreateLabel("Gaussian edge smoothing")
		SmoothLabel.DLGExternalPadding(0, 0)
		SmoothingEdge=DLGCreateRealField(Smoothing, 7, 0)
		SmoothingEdge.DLGExternalPadding(0, 0)
		TagGroup FFTImage=DLGCreatePushButton("    FFT    ", "M_FFT")
		FFTImage.DLGExternalPadding(0, 0, 0, 10)

		TagGroup SetGVector=DLGCreatePushButton("    Set g-vectors    ", "M_SetGVector").DLGIdentifier("#SetGvector")
		SetGVector.DLGExternalPadding(0, 0, 0, 0)
		TagGroup Box0_Group1=DLGGroupItems(SmoothLabel, SmoothingEdge, FFTImage, SetGVector)
		Box0_Group1.DLGTableLayout(4, 1, 0)
		Box0_Items.DLGAddElement(Box0_Group1)

		Return Box0
}

TagGroup Box1(object self)
{
		TagGroup Box1_items
		TagGroup Box1=dlgcreatebox("Phase image construction", box1_items)
		Box1.DLGExternalPadding(6, 0, 0, 0)
		Box1.DLGInternalPadding(7, 3)

		TagGroup OptionLists=DLGCreateLabel("Display options:")
		OptionLists.DLGExternalPadding(0, 5, 0, 230)
		Box1_Items.DLGAddElement(OptionLists)

		PhaseMapOption=DLGCreateCheckBox("Phase map, P(r)", DefaultPhase)
		PhaseMapOption.DLGExternalPadding(0, 25, 0, 20)
		RawPhaseOption=DLGCreateCheckBox("Raw phase map, P'(r)", DefaultRawPhase)
		RawPhaseOption.DLGExternalPadding(0, -35, 0, 0)
		TagGroup Box1_Group2=DLGGroupItems(PhaseMapOption, RawPhaseOption)
		Box1_Group2.DLGExternalPadding(3, 0, 0, 0)
		Box1_Group2.DLGTableLayout(2, 1, 0)
		Box1_Items.DLGAddElement(Box1_Group2)
```

```
		MaskPSOption=DLGCreateCheckBox("Masked power spectrum", DefaultMask)
		MaskPSOption.DLGExternalPadding(0, 20, 0, 15)
		AmpMapOption=DLGCreateCheckBox("Amplitude map, A(r)", DefaultAmp)
		AmpMapOption.DLGExternalPadding(0, 0, 0, 15)
		TagGroup Box1_Group3=DLGGroupItems(MaskPSOption, AmpMapOption)
		Box1_Group3.DLGExternalPadding(4, 0, 0, 0)
		Box1_Group3.DLGTableLayout(2, 1, 0)
		Box1_Items.DLGAddElement(Box1_Group3)

		FullFFTOption=DLGCreateCheckBox("Full Bragg filtered, H(r)", DefaultFull)
		FullFFTOption.DLGExternalPadding(0, -10, 0, 11)
		ReducedFFTOption=DLGCreateCheckBox("Reduced Bragg filtered, H'(r)", DefaultReduced)
		ReducedFFTOption.DLGExternalPadding(0, -16, 0, 1)
		TagGroup Box1_Group4=DLGGroupItems(FullFFTOption, ReducedFFTOption)
		Box1_Group4.DLGExternalPadding(4, 0, 0, 0)
		Box1_Group4.DLGTableLayout(2, 1, 0)
		Box1_Items.DLGAddElement(Box1_Group4)

		TagGroup GLabel=DLGCreateLabel("Gaussian mask (σ) pix: ")
		GLabel.DLGExternalPadding(0, 0, 0, 0)
		GaussianOption=DLGCreateRealField(Sigma, 5, 4)
		GaussianOption.DLGExternalPadding(0, 0, 0, 20)
		TagGroup Filtering=DLGCreatePushButton("          Run          ", "M_Filtering").DLGIdentifier("#Filtering")
		Filtering.DLGExternalPadding(0, 0)
		TagGroup Box1_Group1=DLGGroupItems(GLabel, GaussianOption, Filtering)
		Box1_Group3.DLGExternalPadding(4, 0, 0, 0)
		Box1_Group1.DLGTableLayout(3, 1, 0)
		Box1_Items.DLGAddElement(Box1_Group1)

		Return Box1
}

TagGroup Box2(object self)
{
		TagGroup Box2_items
		TagGroup Box2=dlgcreatebox("Refine g-vectors", box2_items)
		Box2.DLGInternalPadding(8, 8)
		Box2.DLGExternalPadding(4, 0, 0, 0)

		TagGroup InitialLabel=DLGCreateLabel("                    a*                                    b*")
		InitialLabel.DLGExternalPadding(0, -80, 0, 0)
		Box2_Items.DLGAddElement(InitialLAbel)

		TagGroup InitialLabel2A=DLGCreateLabel("(1) Initial g-vectors:")
		InitialLabel2A.DLGExternalPadding(0, 0)
		TagGroup InitialLabel2=DLGCreateLabel("g1: ")
		InitialLabel2.DLGExternalPadding(0, 0)
		InitialGxField=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#InitialGx1")
		InitialGxField.DLGExternalPadding(0, 0)
		InitialGyField=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#InitialGy1")
		InitialGyField.DLGExternalPadding(0, 0)
		TagGroup Box2_Group1=DLGGroupItems(InitialLabel2A, InitialLabel2, InitialGxField, InitialGyField)
		Box2_Group1.DLGTableLayout(4, 1, 0)
		Box2_Group1.DLGExternalPadding(0, 0, 0, 0)
		Box2_Items.DLGAddElement(Box2_Group1)

		TagGroup InitialLabel3=DLGCreateLabel("g2: ")
		InitialLabel3.DLGExternalPadding(0, 0)
		InitialGx2Field=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#InitialGx2")
		InitialGx2Field.DLGExternalPadding(0, 0)
		InitialGy2Field=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#InitialGy2")
		InitialGy2Field.DLGExternalPadding(0, 0)
		TagGroup Box2_Group2=DLGGroupItems(InitialLabel3, InitialGx2Field, InitialGy2Field)
		Box2_Group2.DLGTableLayout(3, 1, 0)
		Box2_Group2.DLGExternalPadding(3, -96, 0, 0)
		Box2_Items.DLGAddElement(Box2_Group2)

		//
		TagGroup RefineLabel1=DLGCreateLabel("(2)")
		RefineLabel1.DLGExternalPadding(0, -1, 0, 0)
		TagGroup RefineButton=DLGCreatePushButton("    Refine g    ", "M_Refine")
		RefineButton.DLGExternalPadding(0, 0, 0, 8)
		TagGroup RefineLabel2=DLGCreateLabel(" g1: ")
		RefineLabel2.DLGExternalPadding(0, 7, 0, 0)
		RefinedGxField=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#RefinedGx1")
		RefinedGxField.DLGExternalPadding(0, 0)
		TagGroup Box2_Group3A=DLGGroupItems(RefineLabel1, RefineButton, RefineLabel2, RefinedGxField)
		Box2_Group3A.DLGTableLayout(4, 1, 0)
		Box2_Group3A.DLGExternalPadding(0, 0, 0, 0)
		RefinedGyField=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#RefinedGy1")
		RefinedGyField.DLGExternalPadding(0, 0)
		TagGroup Box2_Group3=DLGGroupItems(Box2_Group3A, RefinedGyField)
		Box2_Group3.DLGTableLayout(2, 1, 0)
		Box2_Group3.DLGExternalPadding(0, 0, 0, 0)
		Box2_Items.DLGAddElement(Box2_Group3)

		TagGroup PreviousButton=DLGCreatePushButton("    Previous    ", "M_Previous")
		PreviousButton.DLGExternalPadding(0, -20, 0, 0)
		TagGroup RefineLabel3=DLGCreateLabel("    g2: ")
		RefineLabel3.DLGExternalPadding(0, 2, 0, 0)
		RefinedGx2Field=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#RefinedGx2")
		RefinedGx2Field.DLGExternalPadding(0, 0)
		RefinedGy2Field=DLGCreateRealField(InitialG, 18, 10).DLGIdentifier("#RefinedGy2")
		RefinedGy2Field.DLGExternalPadding(0, 0)
		TagGroup Box2_Group4=DLGGroupItems(PreviousButton, RefineLabel3, RefinedGx2Field, RefinedGy2Field)
		Box2_Group4.DLGTableLayout(4, 1, 0)
		Box2_Group4.DLGExternalPadding(3, 0, 0, 0)
		Box2_Items.DLGAddElement(Box2_Group4)

		TagGroup UnitInfo=DLGCreateLabel("(Unit: 1/pix)")
		UnitInfo.DLGExternalPadding(2, 0, 0, -120)
		Box2_Items.DLGAddElement(UnitInfo)
		Return Box2
}

TagGroup Box3(object self)
{
		TagGroup Box3_items
		TagGroup Box3=dlgcreatebox("Mapping", box3_items)
		Box3.DLGInternalPadding(8, 8)
		Box3.DLGExternalPadding(4, 0, 0, 0)

		TagGroup StrainField=DLGCreatePushButton("        Strain        ", "M_Strain")
		StrainField.DLGExternalPadding(0, 0, 0, 0)
		TagGroup DisplacementField=DLGCreatePushButton("    Displacement ", "M_Displacement")
		DisplacementField.DLGExternalPadding(0, 0, 0, 0)
		TagGroup Box3_Group1=DLGGroupItems(StrainField, DisplacementField)
		Box3_Group1.DLGTableLayout(2, 1, 0)
```

```
		Box3_Items.DLGAddElement(Box3_Group1)

		Return Box3
}


taggroup MyInfo(object self)
{
		Taggroup MyInfo=dlgcreatelabel("Kyouhyun Kim, Ph.D., 2015, All rights reserved\n")
		MyInfo.dlgexternalpadding(0,0)
		return MyInfo
}

TagGroup CreateDialog(object self)
{
		TagGroup dialog_items;
		TagGroup dialog = DLGCreateDialog("GPA analysis", dialog_items)

		// Call each buttons
		Dialog_items.DLGAddElement(Self.Box0());
		Dialog_items.DLGAddElement(Self.Box1());
		Dialog_items.DLGAddElement(Self.Box2());
		Dialog_items.DLGAddElement(Self.Box3());
	Dialog_items.DLGAddElement(Self.MyInfo())

		Return Dialog
}

		CreateGPADialog(object self)
		{
			Self.Super.Init(Self.CreateDialog())

			number xpos, ypos
			getpersistentnumbernote("Dialog x position",xpos)
			getpersistentnumbernote("Dialog y position",ypos)

			self.display("Geometric Phase Analysis Ver 1.0")
			documentwindow dialogwin=getdocumentwindow(0)

			if(xpos!=0 && ypos!=0)
			{
				windowsetframeposition(dialogwin, xpos, ypos)
			}
		}
		~CreateGPADialog(object self)
		{
		}
}
Alloc(CreateGPADialog)
```