Extending the Schwartz-Saltikov Algorithm to Visualize Three-Dimensional Particle Distributions

Bennjamin Snyder
The College of Wooster
benn.snyder@gmail.com

Ruth Steinhour
The College of Wooster
rsteinhour13@wooster.edu

Joshua Thomas
The College of Wooster
josh.e.thomas11@gmail.com

Denise Byrnes
The College of Wooster
dbyrnes@wooster.edu

John David
Virginia Military Institute
davidja@vmi.edu

ABSTRACT

Data collected from two-dimensional images of three-dimensional material samples does not always provide sufficient information about the three-dimensional characteristics of particles in the sample. We extend the Schwartz-Saltikov algorithm [2, 9] to relate the two-dimensional distribution of particle diameters to their three-dimensional distribution. We implement the extended algorithm as a plug-in to an existing image analysis tool. The plug-in automates many of the tasks required to preprocess images for analysis. The plug-in displays the data collected during the analysis phase and provides a three-dimensional visualization of the distributed particles.

1. INTRODUCTION

Introducing additives to chemical compounds can improve the physical properties of the compounds, enhancing elasticity, toughening efficiency, and impact resistance [7, 8] for example. By examining the three-dimensional structure of a blended chemical we can observe the correlation between additive distribution and desirable characteristics.

Analysis of additive distribution involves taking microscopic (scanning electron microscopy or transmission electron microscopy) cross-sections of compounds, producing two-dimensional slices less than 100 nm in thickness. In the slice, additives appear as particles.

We extend the Schwartz-Salktikov algorithm, as presented by Corte and Leibler [1] to produce a three-dimensional visualization of particle distribution using two-dimensional particle image information.

2. SCHWARTZ-SALTIKOV ALGORITHM

The purpose of the Schwartz-Saltikov algorithm is to predict the three-dimensional distribution in particle diameter based on the two-dimensional distribution in particle diameter.

The algorithm corrects for the phenomenon known as the cross-section effect.

2.1 Cross-Section Effect

Consider a spherical object such as an orange. If we slice the orange horizontally, the cut exactly through the center of the orange has the orange's correct diameter. Any other horizontal cut produces a diameter smaller than the actual diameter. The cross-section effect is depicted in Figure 1.

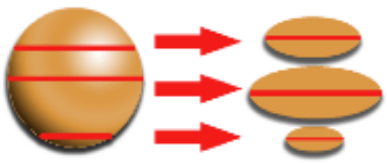

Figure 1: The Cross-Section Effect

The middle circle slice is the actual diameter.

For our purposes, the two-dimensional sample images are taken from cross-sections of a polymer blend. The samples contain various particles that result from the cross-sectioning process. The particles are shown in white in Figure 2. Because the particles are positioned in three-dimensions, they do not all lie in the same horizontal plane.

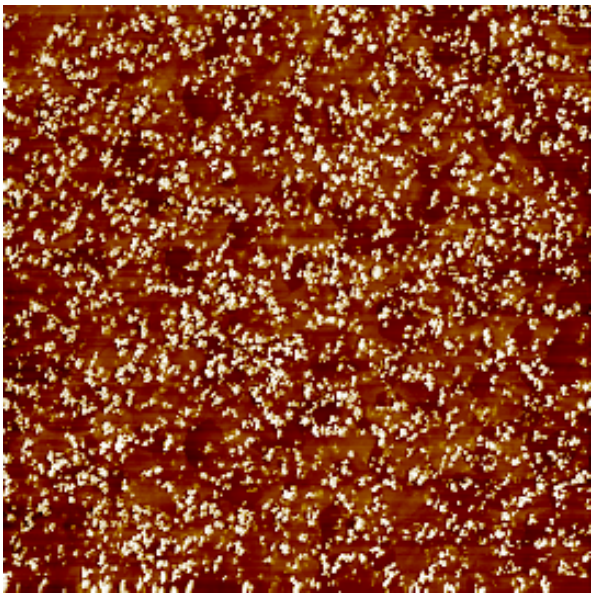

Figure 2: Example Cross-sectioned Image

Thus when the cross-section is taken, the resulting particle diameters can vary significantly. It is possible than none of the particles are sliced through their centers, meaning that none of the particles in the cross-section have the correct diameter.

2.2 Algorithm

The algorithm works by using the two dimensional distribution in particle diameter to predict the three dimensional distribution.

The 2-D distribution vector is determined by binning the particles based on their two-dimensional diameter measurements. The i^{th} entry in the distribution vector indicates the number of particles per unit area that have a diameter in between $i*\Delta$ and $(i+1)*\Delta$, where Δ is the specified bin size.

The algorithm uses the 2-D measurements to predict actual particle diameter in three dimensions. The j^{th} entry in the 3-D distribution vector is the number of particles per unit volume that have a diameter between $j*\Delta$ and $(j+1)*\Delta$.

The Schwartz-Saltikov algorithm relates the two-dimensional distribution in particle diameter to the three-dimensional distribution in particle diameter with a transition matrix:

2-D Distribution Vector = Transition Matrix * 3-D Distribution Vector [1].

The transition matrix TM is defined by:

$$TM = H * I + \Delta * \left(\sqrt{(j+1)^2 - (i^2)} - \sqrt{j^2 - i^2} \right)$$

if $i \leq j$ [1]. EQ 1

H is the thickness of the cross-section and I is the identity matrix [1]. This produces an upper-triangular matrix, so we use back substitution to solve for the three-dimensional distribution.

3. IMPLEMENTATION

In this section, we describe the design and implementation of our 3D Particle plug-in.

3.1 ImageJ

ImageJ is a free, open source image-editing and analysis tool that is developed and maintained by the National Institutes of Health [5]. The tool is written in the Java programming language. ImageJ can open many types of images, make modifications to the images (cropping, paint brush tool, and so on) and apply filters to the images.

ImageJ allows customization through the use of extensions: macros and plug-ins. These extensions provide the ability to automate preexisting tasks, create new ImageJ tools (macros), and create entirely new functionality for the application [6]. These features allow us to implement the Schwartz-Saltikov algorithm.

3.2 OpenGL

OpenGL is ``the industry's most widely used and supported 2D and 3D graphics application programming interface" [4]. To paraphrase, OpenGL is an open source platform that supports features ranging from rendering to texture mapping. OpenGL's diverse capabilities allow it to be used for many different applications, such as ``broadcasting, CAD/CAM/CAE, entertainment, [...], and display[ing] incredibly compelling 2D and 3D graphics" [4]. In our case, OpenGL is combined with the Java programming language (JOGL) to produce three-dimensional renderings of particle distributions for our 3D Particle plug-in.

3.3 Plug-in Features

The flowchart in Figure 3 depicts the execution flow of our plug-in.

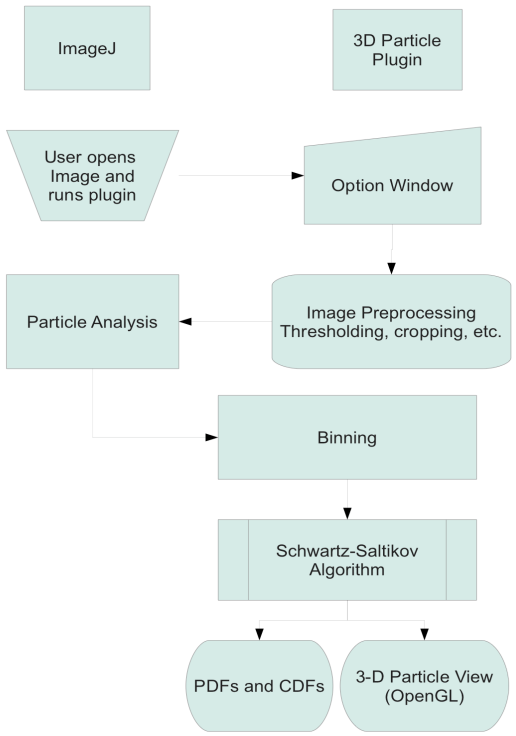

Figure 3: Execution Flow of the 3D Particle Plug-in

The first feature in the plug-in is a user-selectable, automated options window (Figure 4) that simplifies the manual tasks a user performs prior to image analysis. This allows the user to select the editing tasks appropriate for the image they are working on. The automation process removes the time-consuming manual navigation of menus and options required to edit and preprocess an image in ImageJ.

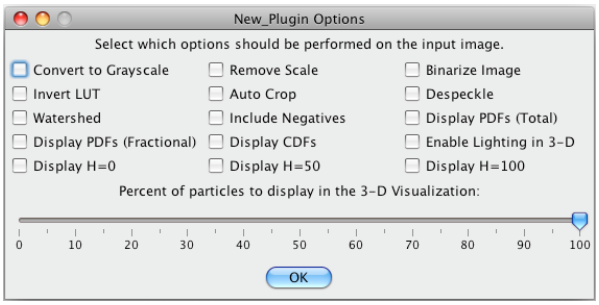

Figure 4: Automated Tasks in 3D Particle Plug-in

The plug-in automatically executes these tasks for the user (with prompts for manual input when necessary).

After completion of preprocessing options, the plug-in hands off the preprocessed image to ImageJ for particle identification. ImageJ identifies particles using an edge detection algorithm. It also identifies the x-y location and area of each particle.

Once the identification phase completes, the particles are returned to the plug-in to be placed in bins based on the particle's size. The binned particles are then automatically analyzed using the Schwartz-Saltikov algorithm.

The second feature of the plug-in produces a histogram and graphs of relevant data

These graphs are a Probability Density Function (PDF) and a Cumulative Distribution Function (CDF).

The histogram gives the number of particles with a diameter between $j*\Delta$ and $(j+1)*\Delta$. These values correspond to the values stored in the 3-D distribution vector obtained through the application of the Schwartz-Saltikov algorithm. The Probability Density

Function shows the same information, but expresses the number of particles per unit volume as a percentage of total particles per unit volume.

The Cumulative Distribution Function shows the percentage of particles that have a diameter less than or equal to a specified value.

The final feature of the plug-in produces a 3-D visualization of particle distribution. This feature is implemented using JOGL.

3.4 Density and Distribution Functions

3.4.1 Probability Density Function

The histogram and Probability Density Functions are shown in Figures 6 and 7 respectively, for the sample image in Figure 5.

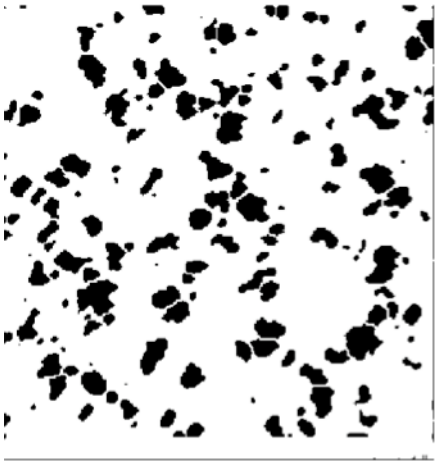

Figure 5: Sample Image

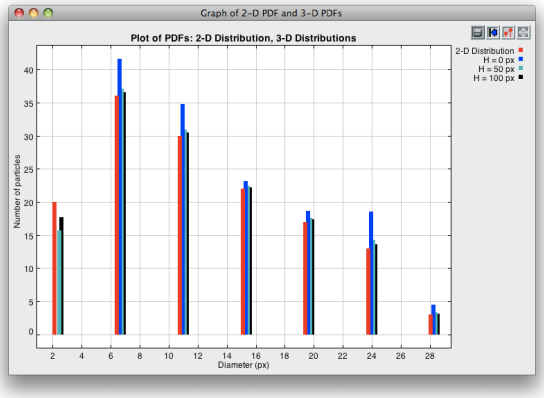

Figure 6: Histogram produced from a sample image. H represents the thickness of cross-sections in three-dimensions.

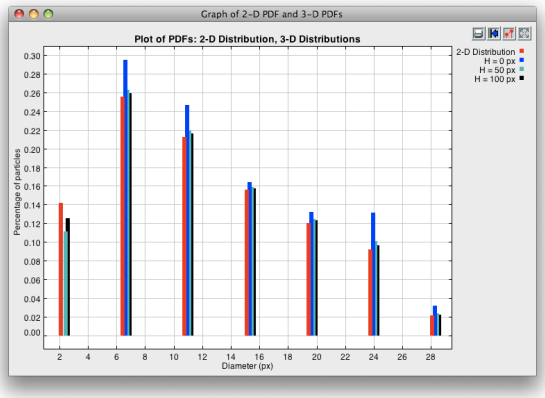

Figure 7: PDF produced from a sample image. H represents the thickness of cross-sections in three-dimensions.

3.4.2 Cumulative Distribution Function

The 3D Particle plug-in also includes an option to display a Cumulative Distribution Function (CDF) of the analysis. Figure 8 is a CDF for the sample image in Figure 5.

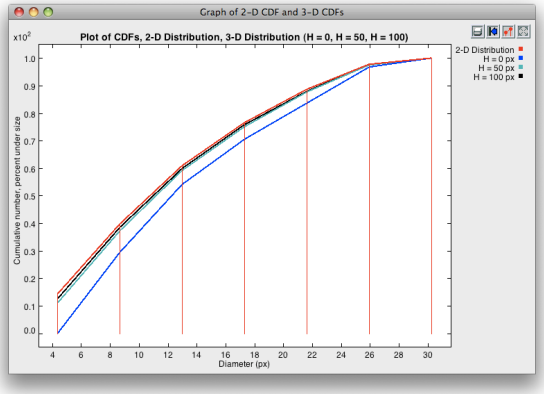

Figure 8: CDF produced from a sample image. H represents the thickness of cross-sections in three-dimensions.

3.5 3-D Visualization

We use each particle's center of mass to place the particle in the xy-plane. The particle dispersion in the xy-plane should match the particle dispersion in both the xz- and yz-planes. Therefore, we use the distribution in particle distance in the xy-plane to determine the particle's depth (z-location). We place the

particles in z such that the particles have the same dispersion in all three planes.

3.5.1 Distribution in Particle Distance

For each particle, we measure the distance between the particle and every other particle and store this information in a vector. For all of our sample images, this data fits a normal distribution curve as shown in Figure 9.

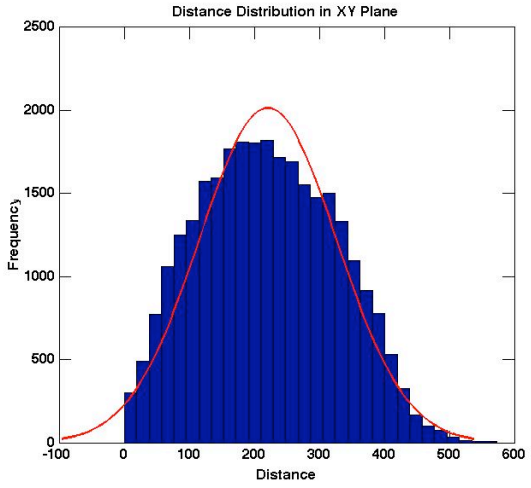

Figure 9: The x-axis displays the distance in pixels and the y-axis indicates the total frequency that the corresponding distance occurs between all particles in the sample image.

One can use the mean and standard deviation of the particle distance data to predict particle depth. This is accomplished by minimizing the following equation.

$$J(\vec{z}) = (\mu_{xz} - \mu_{xy})^2 + (\mu_{yz} - \mu_{xy})^2 + (\sigma_{xz} - \sigma_{xy})^2 + (\sigma_{yz} - \sigma_{xy})^2 \quad \text{EQ 2}$$

In this equation, μ is the mean of the particle distance in the associated plane and σ is the standard deviation. The particles' z-coordinates are stored in vector **z**. This equation uses μ and σ in the xy-plane as a reference and compares it to μ and σ in the xz- and yz-planes. By minimizing this function, we determine a vector **z** such that μ and σ of the particle distance distribution in all three planes are roughly equivalent.

3.5.2 Optimization

The authors investigated optimization routines to improve the run time by minimizing EQ 2. The methods investigated were Nelder-Mead and Steepest Descent. Both methods require an initial iterate for **z**. To determine this iterate, we consider the particle locations in the xy-plane. First, we compute the mean and standard deviation of the particles' x-coordinates and y-coordinates. Then, we average the two means and we average the two standard deviations. We use these averaged values and a random number generator to assign random z-coordinates that fit the average mean and standard deviation of the xy-coordinates. This step is performed ten times, generating ten possible initial iterates. The **z** that returns the lowest value of Equation 2 is used as the initial iterate for optimization.

Both optimization methods run slowly and yield only a slight improvement from the initial iterate. We use Matlab's fminsearch function to perform Nelder-Mead. Nelder-Mead uses a simplex of N+1 points to find the minimum of a function in N-dimensions. This runs slowly because some of our sample images have several hundred particles, resulting in a large simplex.

We use Kelley's implementation of the steepest descent method [3]. We determine the gradient using a finite difference method. Steepest Descent also runs slowly because the function's gradient is small.

4. RESULTS

In the final implementation of the 3D Particles plug-in, we do not implement either optimization technique discussed in section *3.5.2*. We implement the procedure to calculate an initial iterate for optimization and then use that iterate for the z-coordinates in the three-dimensional rendering. We recommend this method, as the optimizations take several hours and yield only slightly

better results. Table 1 summarizes the percent improvement attained by finding **z** through optimization compared to using the initial iterate.

Table 1: Percent improvement attained by finding z through Nelder-Mead the optimization for sample image in Figure 5.

Value	Percent Improvement
μ_{xz}	0.2880
μ_{yz}	−0.2290
σ_{xz}	2.1846
σ_{yz}	2.4602

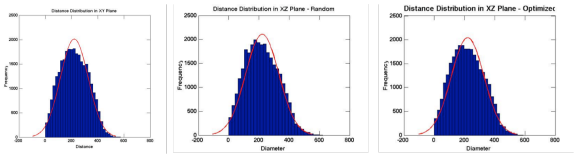

Figure 10: These graphs show the distribution results using the image in Figure 5. The left graph displays the distance distribution on the x-y plane. The center graph displays the distance distribution in the xz-plane using the randomly generated iterate. The graph on the right displays the distance distribution in the xz-plane using z values found through optimization.

We see both quantitatively and visually that the optimization produces only a slight improvement. These results are roughly consistent across all of the sample images we analyzed. As such, we feel that the added computations and runtime do not produce a significant increase in accuracy for their costs.

4.1. Java OpenGL Rendering

Equations 1 and 2 in sections *2.2* and *3.5* are applied to an OpenGL canvas using JOGL (Java OpenGL). Each particle is placed into the three-dimensional canvas using its center of mass for its (x,y) location. We take the z-coordinates for the particles from the initial iterate as described in section *3.5*. We give each particle its diameter from the two-dimensional image. We then compute α, the average growth rate of the particles from two dimensions to three dimensions.

The growth rate is computed by taking the sum of the values in each bin of the three-dimensional distribution of a given thickness and dividing by the sum of the values in each bin of the two-dimensional distribution. This ratio is our growth factor, α. Each particle's diameter is then multiplied by α. This results in diameters that are scaled by the distribution of particle sizes in three dimensions.

Figures 11 and 12 show the three-dimensional rendering of various sample images.

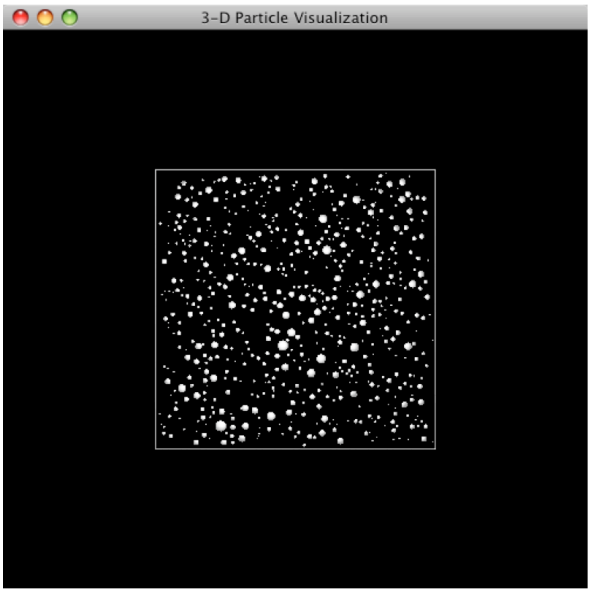

Figure 11: Top down view of three-dimensional particle distribution.

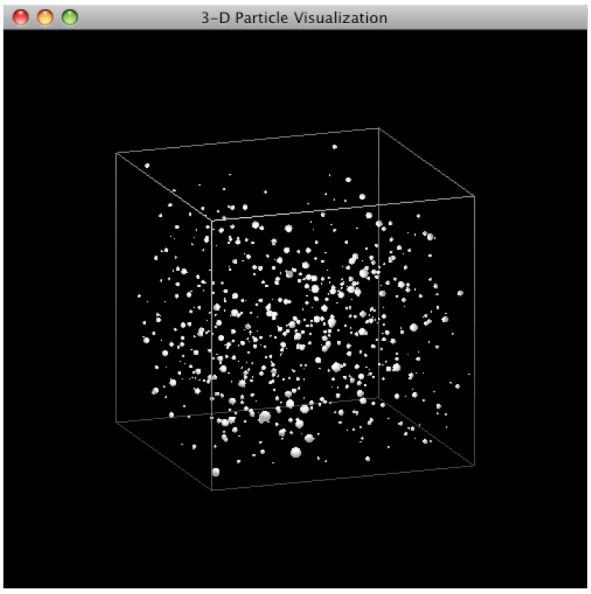

Figure 12: Default angle view of the three-dimensional particle distribution.

5. ACKNOWLEDGMENTS

We acknowledge the Applied Mathematics Research Experience program at The College of Wooster for supporting this project during the summer of 2011. We also acknowledge The Goodyear Tire & Rubber Company, especially Craig Burkhart and George Papakonstantopo, R&D Associates, for supporting this project.

6. REFERENCES

[1] Corte L, Leibler L. *Analysis of Polymer Blend Morphologies from Transmission Electron Micrographs*. Polymer 2005:46:6360-368.

[2] Giumelli AK, Militzer M, Hawbolt EB. *Analysis of the Austenite Grain Size Distribution in Plain Carbon Steels*. ISIJ International 1999; 39:271-80.

[3] Kelly CT. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics. SIAM 2011. http://www.siam.org/books/kelley/fr18/index.php.

[4] Khronos Group. OpenGL Overview 2011. http://www.khronos.org/opengl.

[5] National Institutes of Health. ImageJ. http://rsbweb.nih.gov/ij/.

[6] National Institutes of Health. ImageJ Features. http://rsbweb.nih.gov/ij/features/features.html.

[7] Paul DR, Barlow JW. *Polymer Blends (or Alloys).* Macromol Sci-Part C 1980; 109-68.

[8] Ruzette A-V, Leibler L. *Block Copolymers in Tomorrow's Plastics.* Nat Mater 2005; 4:19-30.

[9] Saltikov SA. *The Determination of the Size Distribution of Particles in an Opaque Material from a Measurement of the Size Distribution of their Sections*. Proceedings of the Second International Congress for Stereology 1967. p. 163-73.