IT 10 217

Examensarbete 15 hp
May 2010

Synthetic 3D Pap smear nucleus generation

Sandra Gómez

Institutionen för informationsteknologi
Department of Information Technology

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Synthetic 3D Pap smear nucleus generation

Sandra Gomez

In this project we present a 3D Pap smear cell nucleus generator. The shape and the texture are the important features for a realistic synthetic nucleus. For the first one, the shape, a deformed distance transform is used in order to generate deformed spheres. For the second one, the texture, a pseudorandom noise algorithm, Perlin noise, is applied to the shape in order to generate the most realistic texture of a cell. As a result, we obtain synthetic 3D cell nuclei as they appear in Pap smear tests.

Handledare: Patrik Malm
Ämnesgranskare: Patrik Malm
Examinator: Anders Jansson
IT 10 021
Tryckt av: Reprocentralen ITC

Contents

1 INTRODUCTION

Computer graphics have many different applications, not only for entertainment, videogames, animation and films but also in the scientific community. Computer graphics is used for data visualization and simulation, which makes analysis of the data easier. Another field is data simulation or synthetic data generation which can be very useful for developing and validating algorithms.

While medical image simulation software has been under development since the 1980s, until recently the complexity of the procedures and long computation times have limited the realism and accuracy of artificially generated images. Improvements in computational systems have facilitated simulations that were previously infeasible. Advancements in processor architecture, increases in speed and amount of memory, and development of large storage systems have enabled computers to be used for increasingly complex problems. The use of distributed systems and technologies provide unparalleled computational capabilities [10].

When developing algorithms, one of the most difficult tasks is to know the ground truth of the data. The acquisition of ground truth is a laborious task that usually requires the assistance of several experts and a good amount of time. In addition, this process is often very subjective and different experts may have different views on what could be considered a correct result.

The goal of synthetic data generation is to solve this problem. Using synthetic data the ground truth is known *a priori*. Synthetic data can be created in large numbers, in a short time, with suitable variation of the model parameters, making it possible to draw statistically sound conclusions regarding the relative performance of different methods.

Currently, cancer is a big problem in the population. One of the most common cancers, is the cervical cancer which is the second most deadly cancer among women.

The Papanicolau test (also called PAP-smear) is a screening test used in gynecology to detect premalignant and malignant (cancerous) processes in the ectocervix. Using a spatula or a brush the area known as the transformation zone of the uterin cervix is scraped in order to obtain endocervical and ectocervical cells. The collected material is then smeared onto a glass slide. The sample is then fixated and stained to enhance the contrast between nucleus and cytoplasm [11,12].

The goal of this project is to mimic cells as they appear in Pap smear. There are many previous studies in two-dimensional synthetic cell image generation, e. g. [1, 2, 3]. They follow the same general process: generate the shape of the cell, then apply different noise algorithms and create a realistic image with a population of cells.

This project is an extension of [1] in three dimensions. A three-dimensional model of the nuclei can give us a more realistic representation since the depth in the synthetic image becomes more natural.

2 BACKGROUND

2.1 Preliminary concepts

2.1.1 Uniform distribution

In probability theory and statistics, the continuous uniform distribution is a family of probability distributions such that for each member of the family, all intervals of the same length on the distribution's support are equally probable [13].

The support is defined by the two parameters, *a* and *b,* which are its minimum and maximum values, as seen in Figure 1. The distribution is often abbreviated *U(a,b)*.

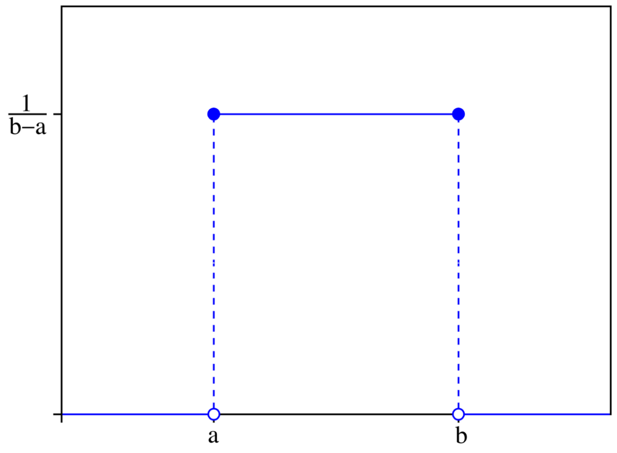

Fig. 1. Probability density function of the continuous uniform distribution.

2.1.2 Fourier series

A Fourier series is an expansion of a periodic function *f(x)* in terms of an infinite sum of sines and cosines. Fourier series make use of the orthogonality relationships of the sine and cosine functions. The computation and study of Fourier series is known as harmonic analysis and is extremely useful as a way to break up an *arbitrary* periodic function into a set of simple terms that can be plugged in, solved individually, and then recombined to obtain the solution to the original problem or an approximation to it to whatever accuracy is desired or practical [14].

Using the method for a generalized Fourier series [21], the usual Fourier series involving sines and cosines is obtained by taking $f_1(x)=cos(x)$ and $f_2(x)=sin(x)$. Since these functions form a complete orthogonal system over [-π, π], the Fourier series of a function is given by:

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx), \qquad (1)$$

where

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)dx \tag{2}$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx \tag{3}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx \tag{4}$$

and $n=1,2,3,\ldots$Note that the coefficient of the constant term a_0 has been written in a special form compared to the form for a generalized Fourier series in order to preserve symmetry with the definitions a_n and b_n.

2.1.3 Hermite cubic splines

In the mathematical subfield of numerical analysis, a Hermite spline is a third-degree spline curve where each polynomial of the spline is in Hermite form. The Hermite form consists of two control points and two control tangents for each polynomial [20].

On the unit interval (0,1), given a starting point p_0 at $t=0$ and an ending point p_1 with starting tangent m_0 at $t=0$ and ending tangent m_1 at $t=1$, the polynomial can be defined by:

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 - 3t^2)p_1 + (t^3 - t^2)m_1 \tag{5}$$

where $t \in [0,1]$.

2.2 Shape generation

There are several methods to generate spherical shapes. Here three approaches of them are described: parametric model, layered approach and distance based model.

2.2.1 Parametric model

The parametric model is the basic representation of a sphere: it calculates the Cartesian coordinates from the spherical coordinates theta (θ) and phi (ϕ). Figure 2 shows the correspondence with these coordinates and the sphere.

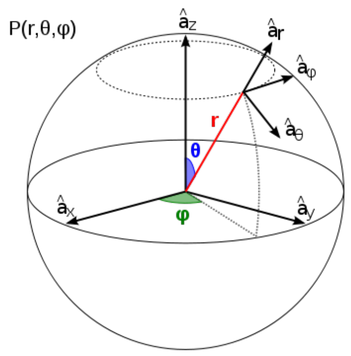

Fig. 2. Spherical coordinates of a sphere.

The parameterized equation of a sphere, in which the parameters x_0, y_0, z_0 are the center points of the sphere, is given by:

$$x = x_0 + r\sin\theta\cos\varphi \tag{6}$$
$$y = y_0 + r\sin\theta\sin\varphi \tag{7}$$
$$z = z_0 + r\cos\theta, \tag{8}$$

where

$$\left(0 \le \varphi \le 2\pi \text{ and } 0 \le \theta \le \pi\right) \tag{9}$$

2.2.2 Layered approach

This approximation consists in build a sphere from layers of two-dimensional circles, as seen in Figure 3. It is important to realize that the circles have to be placed equally separate along the size of the radius. To calculate this increment between the circles, the initial point is necessary. Equations 10 and 11 show how to calculate the initial point and the height increment.

$$init = \frac{r}{nCircles}x2 \tag{10}$$

$$\Delta height = ceil\left(\frac{r}{\left(\frac{nCircles}{2}\right)}\right) \tag{11}$$

Once the initial position and the height increment are known, the circles can be generated.

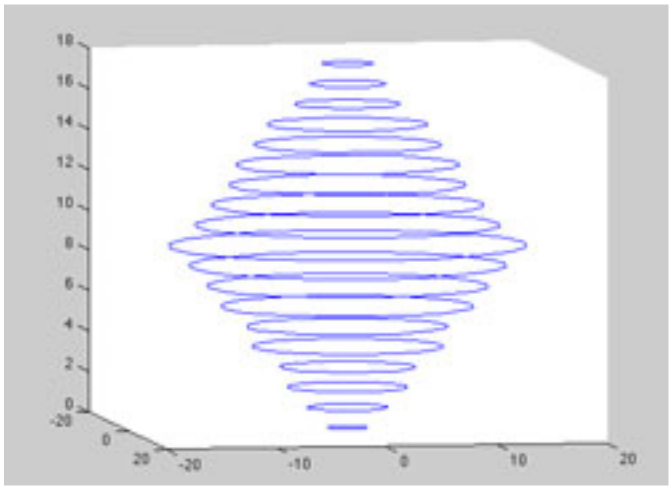

Fig. 3. An example of a sphere drawn using the layered approach technique.

2.2.3 Distance based model

This representation uses the Euclidean distance transform to generate the sphere. The Euclidean distance between two points (x_0,y_0,z_0) and (x_1,y_1,z_1) is given by:

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \tag{12}$$

This algorithm generates, from a binary image with a initial point set to *0* and the rest of the matrix points set to *1*, a new matrix in which each position *(x,y,z)* has the Euclidean distance to the initial point (x_0,y_0,z_0). Figure 4 shows an example of this algorithm in two dimensions.

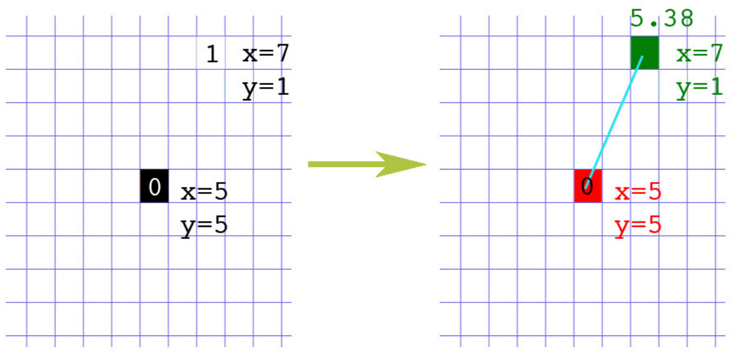

Fig. 4. At the left, original binary matrix (all the white boxes are set to *1*). At the right the result of apply the distance transform to the matrix (all the boxes that were white before now have the Euclidean distance from the center point *(5,5)*).

Now with the matrix full of the Euclidean distance to the center point, a sphere with the desired radius is obtained by thresholding the matrix with the required radius. In the example from the Figure 4, applying a threshold with a value of 4 generates a circle of radius 4.

2.3 Binarization

Rasterization or rastersation (also binarization) is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format [23]. In this project there have been studied different rasterization methods: Bresenham, MidPoint, a seed fill algorithm and ScanLine algorithm.

2.3.1 Bresenham algorithm

The Bresenham line algorithm is an algorithm that determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. [16]

Consider drawing a line on a raster grid where we restrict the allowable slopes of the line to the range $0 \leq m \leq 1$.

If we further restrict the line-drawing routine so that it always increments x as it plots, it becomes clear that, having plotted a point at *(x,y)*, the routine has a severely limited range of options as to where it may put the next point on the line:

- It may plot the point *(x+1,y)*, or:
- It may plot the point *(x+1,y+1)*.

So, working in the first positive octant of the plane, line drawing becomes a matter of deciding between two possibilities at each step.

Figure 5 shows draw a diagram of the basic Bresenham algorithm [17].

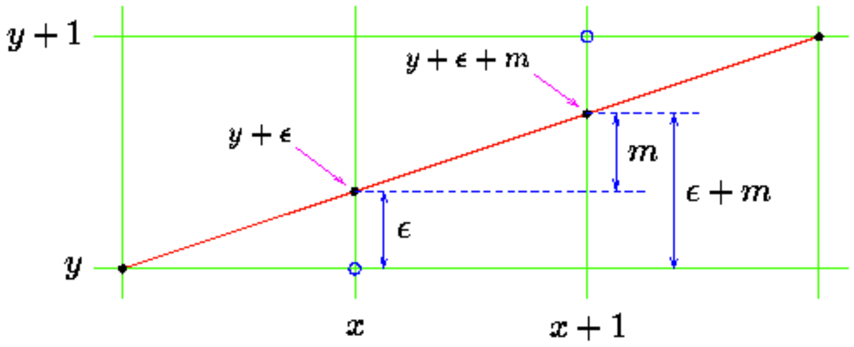

Fig. 5. Diagram of the situation which the plotting program finds itself in having plotted *(x,y)*.

This algorithm browses the matrix from left to right on *x*, and chooses which pixel has to be plotted depending on the value of ε+*m*. If this value is closer to *y*+1 then (*x,y+1)* is plotted if, on the contrary, ε+*m* is closer to *y*, then the pixel (*x,y)* is plotted.

The extension of this algorithm in three dimensions is just to decide a plane of reference, e.g. plane *xy*, and then apply the basic 2D algorithm on *xz* and *yz*, as they were images.

2.3.2 MidPoint algorithm

Mid Point is an extension of Bresenham algorithm [18]. It is a fast and efficient process of conversion an image into binary due to the integer operations. It is readily applicable to any type of geometric element.

Considering the Figure 6, which represents a certain step in the algorithm process, the blue point P: *(x,y)* is already painted. Next step is to chose which pixel will be plotted. There are two choices, E: *(x+1,y)* and NE: *(x+1,y+1)*. The decision function consists in calculate the mid point M between E and NE as follows:

$$M = \left(x + 1, y + \tfrac{1}{2}\right) \tag{13}$$

Then, if the intersection point Q of the real line with the vertical line connecting E and NE is below M, the next pixel is E. Otherwise the next pixel is NE.

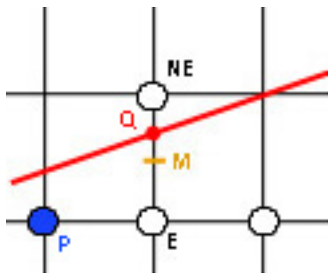

Fig. 6. One step of the MidPoint algorithm. The blue point is already plotted and the next step is to decide which is the next plotted pixel.

2.3.3 Seed fill algorithm

A seed fill algorithm, also called Flow fill algorithm, is used to fill areas of an image or a volume. This algorithm is very common in the image analysis but it can be expanded into three dimensions.

The algorithm in [19] is a 3D seed fill algorithm. It starts with a stack containing only one entity: the *(x,y,z)* coordinates of the seed. Upon the completion of an iterative procedure, which is indicated by the stack being empty, it returns the number of the filled voxels. Basically, the iterative procedure consists of four parts:

1. Pops up the top entity of the stack and makes it the current voxel.
2. Fills the voxel and its left and right span in dimension x, until the boundary voxels are met at the extremes of the respective spans.
3. Saves the extremes as x_{left} and x_{right}.
4. Within the span between x_{left} and x_{right}, checks whether the four face neighbor scan lines (front and back in dimension y, top and bottom in dimension z) contain only boundary voxels or filled voxels; if it is not true to a scan line between x_{left} and and x_{right}, takes the right extreme of each sub-span as a new seed and pushes it onto the stack.

The algorithm skips the holes inside a solid model and the concave parts of its boundaries.

The Figure 7 shows an example of how this algorithm works in two dimensions.

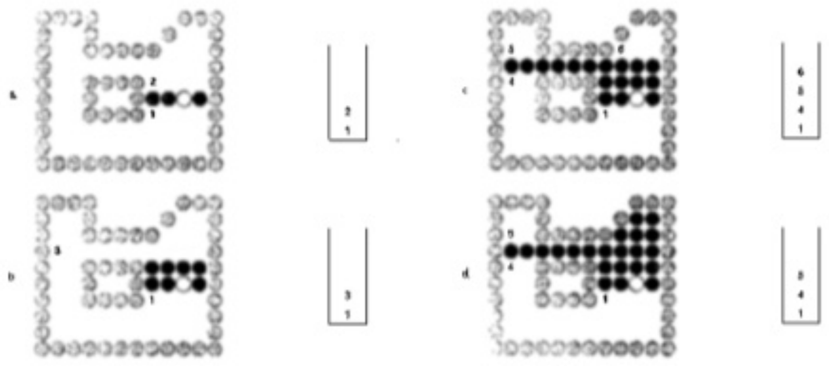

Fig. 7. Example of a seed algorithm in two dimensions in a polygon with an intern hole.

2.3.4 Scan-line algorithm

Scan-line rendering is an algorithm for visible surface determination, in 3D computer graphics, that works on row-by-row basis rather than polygon-by-polygon basis or pixel-by-pixel basis. All of the polygons to be rendered are first sorted by the top *y* coordinate at which they first appear, then each row or scan line of the image is computed using the intersection of a scan line with the polygons on the front of the sorted list, while the sorted list is updated to discard no-longer-visible polygons as the active scan line is advanced down the picture.

The usual method starts with edges of projected polygons inserted into buckets, one per scan-line; the raster maintains an active edge table (AET). Entries maintain sort links, X coordinates, gradients, and references to the polygons they bound, as shown in Figure 8. To rasterize the next scan-line, the edges no longer relevant are removed; new edges from the current scan-lines' Y-bucket are added, inserted sorted by X coordinate. The active edge table entries have X and other parameter information incremented. Active edge table entries are maintained in an X-sorted list by bubble-sort, effecting a change when two edges cross. After updating edges, the active edge table is traversed in X order to emit only the visible spans, maintaining a Z-sorted active Span table, inserting and deleting the surfaces when edges are crossed.

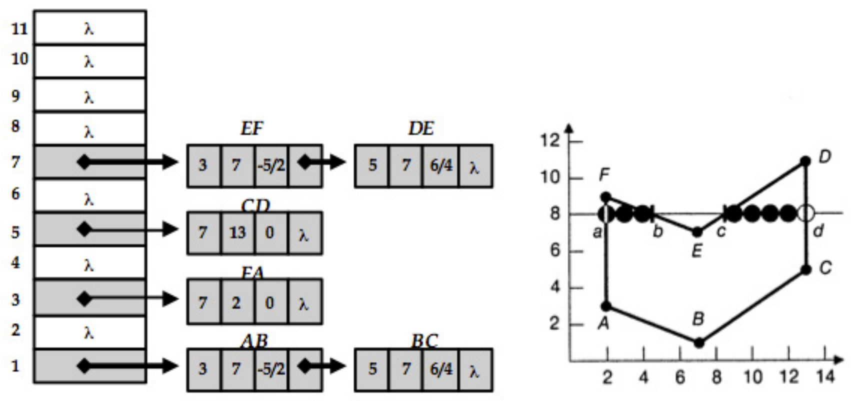

Fig. 8. Example of a scan-line algorithm application to a polygon.

2.4 Texture generation

Creating a texture is basically adding noise to an image or a volume. On of the most typical textures generator is the Perlin noise.

2.4.1 Perlin noise

Perlin noise is a procedural texture primitive, used by visual effects artists to increase the appearance of realism in computer graphics. This is a type of gradient noise. The function has a pseudo-random appearance, yet all of its visual details are the same size (see image). This property allows it to be readily controllable; multiple scaled copies of Perlin noise can be inserted into mathematical expressions to create a great variety of procedural textures. Synthetic texture using Perlin noise is often used in CGI to make computer-generated objects appear more natural, by imitating the controlled random appearance of textures of nature.

The initial implementation, first used in 1983 and first published in 1985 (Perlin 1985) [5, 7], defined noise at any point (x, y, z) by using the following algorithm:

1 At each point in space (i, j, k) that has integer coordinates, assign a value of zero and a pseudo-random gradient that is hashed from (i, j, k).

2 Define the coordinates of (x, y, z) as an integer value plus a fractional remainder: $(x, y, z) = (i + u, j + v, k + w)$. Consider the eight corners of the unit cube surrounding this point: $(i, j, k), (i + 1, j, k), \dots (i + 1, j + 1, k + 1)$.

3 Fit a Hermite spline through these eight points, and evaluate this spline at (x, y, z), using u, v, and w as interpolants. If we use a table lookup to predefine the Hermite cubic blending function $3t^2 - 2t^2$, then this interpolation requires only seven scalar linear interpolations: for example, four in x, followed by two in y, followed by one in z, as shown in Figure 9.

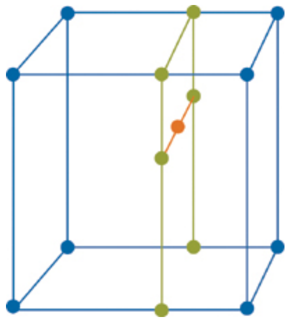

Fig. 9. Steps required to interpolate a value from eight samples in a regular 3D lattice.

However, this model has deficiencies in the Hermite function used and in the permutation table [6]. The new Hermite function is given by:

$$6t^5 - 15t^4 + 10t^3 \tag{14}$$

On the other hand, the permutation table has not to be random, so it is substituted by:

$$(1,1,0),(-1,1,0),(1,-1,0),(-1,-1,0),$$
$$(1,0,1),(-1,0,1),(1,0,-1),(-1,0,-1),$$
$$(0,1,1),(0,-1,1),(0,1,-1),(0,-1,-1),$$
$$(1,1,0),(-1,1,0),(0,-1,1),(0,-1,-1). \tag{15}$$

2.5 Real nuclei images

As we said in the introduction, the goal of this project is to generate synthetic cells as they appear in Pap smear. Figure 10 shows a few examples of real Pap smear cells. These images highlight the key features that this project is trying to achieve.

As seen on the Figure 10.3.a, nuclei have a similar roundish shape of varying size. It is important that the generator can create varied shapes and sizes by modifying parameters.

Another relevant feature is the texture, which provides the final appearance of the nucleus cell. On the images shown in the figure, there are darkness differences on the cells but, in general, the texture is quite homogeneous. The importance of this feature is because the main information about a cell is in the texture, so it must be as realistic as possible.

The last key feature is the chromatin spots present in the nuclei. In Figure 10 there are different nucleus with these blobs, depending on the contrast they can be more or less visible. What matters is that they are present in each nucleus to add realism.

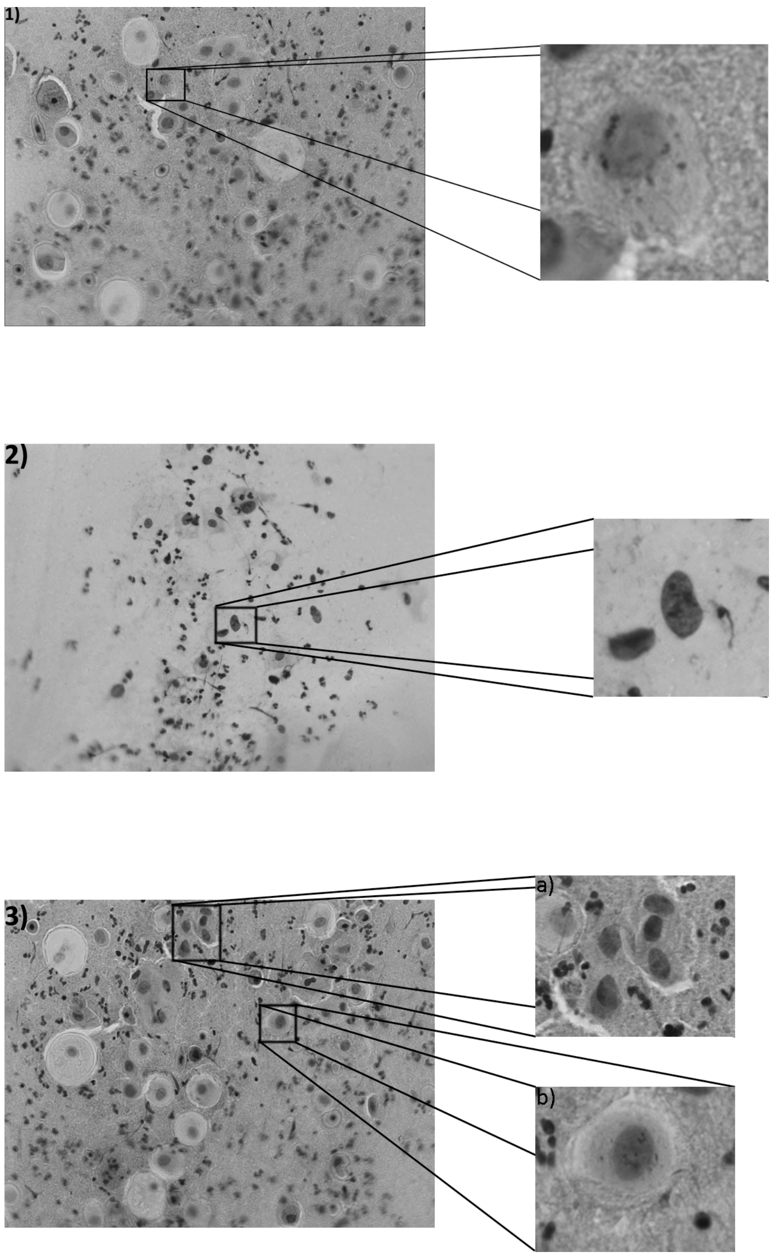

Fig. 10. Three examples of different real Pap smear cells. The image 1) shows a nucleus with few internal blobs; 2) shows another nucleus with a different shape, a darker contrast and some internal blobs, and 3) shows several nucleus: a) shows different shapes of the nucleus and b) shows another nucleus with littler blobs.

3 MATERIAL AND METHODS

3.1 Development platform

This project has been developed using the MATLAB r2009b programming environment and the DIPimage toolbox.

3.2 Shape generation

In the previous work on cell generation in two dimensions [1, 2, 3], a cell nucleus is modelled from a basic circle, due to its roundish form. Then, in accordance with this representation, a cell nucleus can be modelled as a sphere in three dimensions.

For selecting the appropriate shape method generation, we considered several features. First of all, the method must be able to be parameterized; it means that it can be modified with different parameters values. These parameters allow us to deform the sphere for generating the desired shape. This shape must be continuous and uniform, so the type of the deformation is very important in the model.

Furthermore, the method cannot be slow; the generation must be fast in order to obtain synthetic images faster than real images. The point is to avoid slow algorithms for the shape generation, and in this case, these are the binarization algorithms. 3D rasterization algorithms are too slow because they have to fill the volume inside, and it requires many loops. Therefore, the binarization method is critical for the shape generation.

3.3 Texture generation

When using an algorithm to generate the texture of the nucleus, it has to be considered the runtime of the algorithm. The computational cost is the critical feature of the texture generation and likeness with real images.

3.4 Visualization

For the visualization of the results we used three different methods: the MATLAB command *surf* and two MATLAB modules for 3D plotting, *plot3D* and *vol3d* [22]. An example of each method is shown in Figures 11, 12 and 13, respectively.

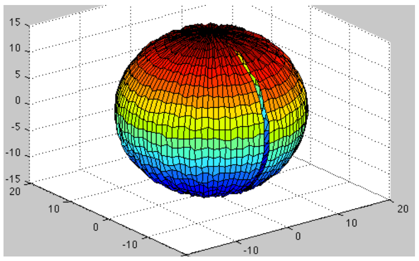

Fig. 11. An example of using the command *surf*.

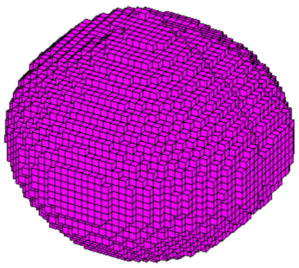

Fig. 12. An example of using the command *plot3D*.

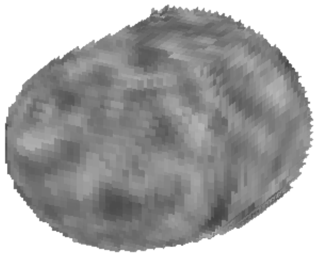

Fig. 13. An example of using the command *vol3d*.

4 RESULTS

The synthetic image generation presented in this project is a sequential process: first creating a plane model with the desired shape and then adding the features that make it realistic. This method can be divided in two sub processes: shape generation and texture generation, as the flowchart shows in Figure 14.

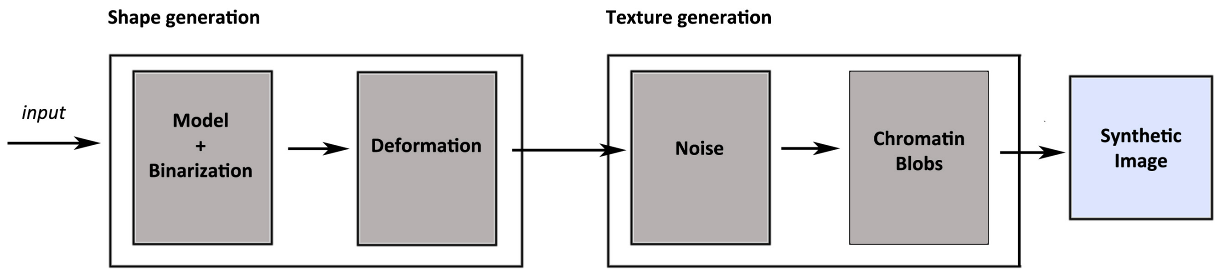

Fig. 14. Flowchart describing the simulation process.

4.1 Shape generation method

From the several representation models of a sphere, we used the distance based model *(see section 2.2.3)*, based on the low computational cost, and because it avoids all the problems with the binarization algorithms.

Since the sphere can be seen as circles, the deformation can be applied to these circles, forgetting about the third dimension. Therefore, according to the method for creating variance on the shape of a circle in [1], we also used Fourier series to deform the basic form of a circle. As the sphere was not in the parametric model *(see section 2.2.1)*, the results of the Fourier series had been applied in a look-up table, referenced by the angle φ of the sphere (from 1 to 360), as seen in Figure 2, hence the deformation was equally applied along the sphere. Figure 15 presents the final shape with the deformation applied to each circle of the sphere.

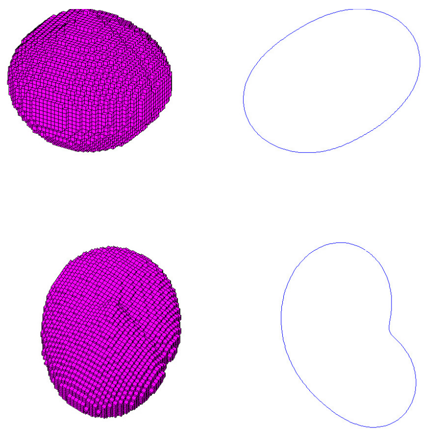

Fig. 15. Representation of deformed spheres using the distance transform model. At the right, the binary 3D model, and at the left the deformation in 2D applied to each circle of the sphere.

4.2 Texture generation method

According to the previous work on synthetic Pap smear cells [1], we used the same well-known texture algorithm but in three dimensions: Perlin noise *(see section 2.4)*.

In the first place, in order to generate the base model, a Perlin noise smoothed function was applied on each point *(x,y,z)* of the matrix as follows:

$$\sum_{i=1}^{n} \left| \frac{noise(2^i x, 2^i y, 2^i z)}{2^i} \right|, \tag{16}$$

where n controls how much the texture smoothes and *noise(x,y,z)* is the Perlin noise generation function. Figure 16 shows some examples of the variation produced in the texture by changing the value n in the smoothing function.

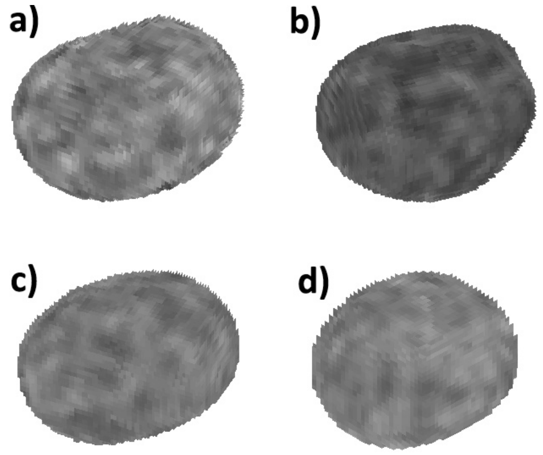

Fig. 16. Different nuclei textures applying a) the noise function without smoothing, b) the smoothed noise function (Eq. 17) where $n=4$, c) the smoothed noise function where $n=6$, and d) the smoothed noise function where $n=10$.

Once the textured base model was generated, the chromatin blobs were added to this texture. The position of these blobs was chosen randomly and their shapes generated in the same way. Figure 17 shows few examples of the base texture with the added blobs.

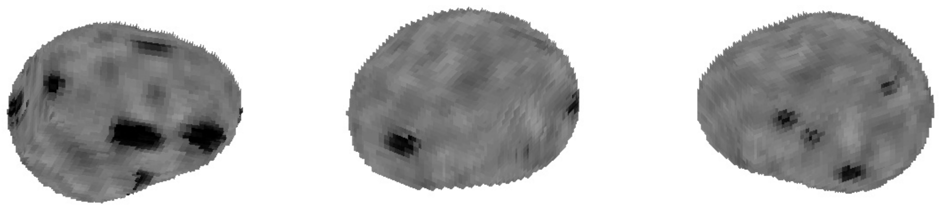

Fig. 17. Some examples of textured nuclei with different sizes of the chromatin blobs: *(from left to right)* size 6x8, size 4x4 and size 3x3.

4.3 Final images

As a result of the generation process, we obtain different nuclei models by changing some parameters. Figure 18 shows these final images and some slices of them so we can compare the synthetic images with the real ones.

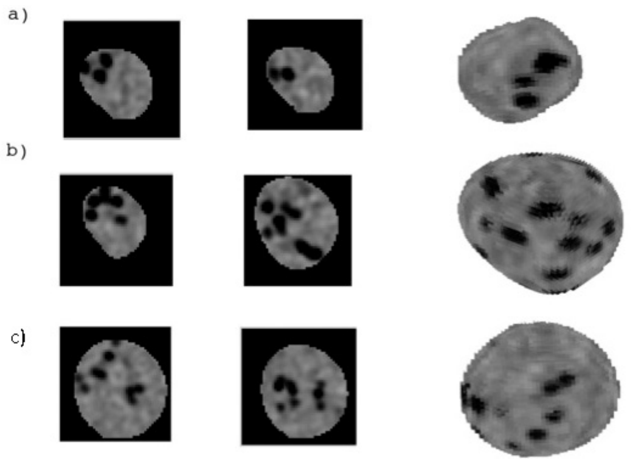

Fig. 18. Final images of synthetic nuclei. At the left, 2D slices from the 3D volume, at the right, the volume rendered with a 50x50x50 resolution: a) radius 25, number of blobs 100 and size of the blobs 7x6, b) radius 36, number of blobs 150 and size of the blobs 5x5, and c) radius 35, number of blobs 100 and size of the blobs 5x5.

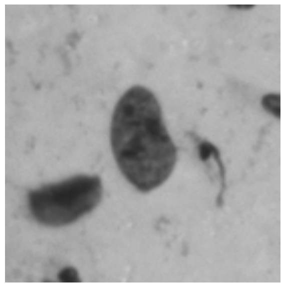

Fig. 19. Image of a real cell nucleus.

5 DISCUSSION

5.1 Shape generation

The shape generation process depends on the representation model for the sphere chosen. At the beginning of the project and attending to the previous work in 2D synthetic image generation [1,2, 3], the parametric model was chosen *(see section 2.2.1)*. With this model, the deformation of the sphere was only applying a variation to the radius of the sphere, but it did not produce the expected result, as seen in Figure 20. This modification in the radius was not continuous and made abrupt changes in the sphere model.

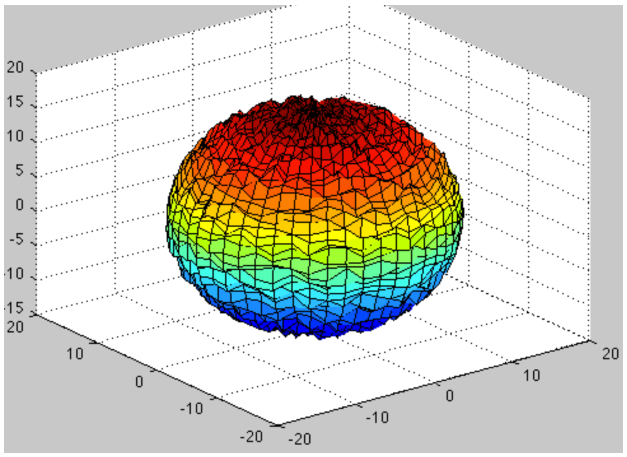

Fig. 20. Representation of a deformed sphere using the parametric model.

Furthermore, the binarization (or rasterization) *(see section 2.3)* process from this model has a high computational cost. Raster conversion algorithms as Bresenham or MidPoint algorithm work well on images but on volumes, they do not generate a full model, they just convert the boundaries. So, in order to obtain a full volume, a seed fill algorithm *(see section 2.3.3)* has to be used, and this is what increases the computational cost. There are another algorithms with a low computational cost, for instance, Scan-line algorithm, but due to the problem with the shape and the binarization cost, this model representation was discarded.

The next representation model studied was the layered approach *(see section 2.2.2)*. This model, as seen in Figure 3, generated a new problem: create a volume from the layered sphere. This problem was complicated to solve and we found the distance based model, which seemed better for our project. So, this model was also discarded.

Finally, we found the solution for all the problems seen above: the distance based model *(see section 2.2.3)*. This model avoids the problem with the binary conversion because, as seen in *2.2.3*, just applying a threshold generates the binary matrix. Another problem solved was the variation of the shape, using Fourier series *(see section 2.1.2)* to generate the desired shape of the sphere in two dimensions. This shape was the perturbation applied to the sphere to deform it. Figure 15 presents two examples of the final shape model and the 2D model generated for deform the sphere.

5.2 Texture generation

In the texture generation process, *Perlin noise* function was chosen from the beginning to simulate the nucleus texture, because it is a well-known algorithm for generating textures. As said before, Equation 16 is used to smooth the noise in order to obtain a better approximation to the real nuclei texture. Figure 16 shows four examples of the application of this smoothing function with different settings. As seen, the noise function without smoothing seems very random and not realistic.

The last step on the process is to add the chromatin blobs to the textured model. The size and the amount of blobs can be controlled. Figure 17 shows a few examples of the model with different blob sizes.

Finally, several images of the final result, the synthetic cell nucleus, are showed in Figure 18. In order to show how the chromatin blobs are distributed into the nucleus, we show some two-dimensional slices of the model. Therefore we can compare these images with a real nucleus as seen in Figure 19 and observe the similarity between them.

As a future work, the model can be improved by adding different aberrations to the model, adding more realistic features that can help to develop more kind of algorithms. The application can be improved as well by using another algorithms with less computational cost to obtain a faster generator In addition, it could have more parameters to modify in order to generate more variable models

6 ACKNOWLEDEGMENTS

I am really grateful for the cooperation and interest of Patrik Malm, supervisor of the project. Without his collaboration and his help this project would not be possible. Thanks also to the CBA department from the Uppsala Universitet for providing me a place to work and to help Patrik and me to solve some problems of the project.

7 REFERENCES

[1] P. Malm, A. Brun and E. Bengtsson, "PAPSYNTH: Simulated bright-field images of cervical smears".

[2] A. Lehmussola, P. Ruusuvuori, J. Selinummi, H. Huttunen, and O. Yli-Harja, "Computational framework for simulating fluorescence microscope images with cell populations", *IEEE Transactions on Medical Imaging*, vol. 26, no. 7 pp. 1010-1016, Jul 2007.

[3] A. Lehmussola, J. Selinummi, P. Ruusuvuori, A. Niemisto, and O. Yli-Harja, "Simuating fluorescent microscope images of cell populations", in *Proceedings of the 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Piscataway, NJ, USA, 2005.

[4] A. Huisman, L. S. Ploeger, H. F. J. Dullens, N. Poulin, W. E. Grizzle and P. J. van Diest, "Development of 3D chromatin texture analysis using confocal laser scanning microscopy", *Cellular Oncology*, vol. 27, pp. 335-345, 2005.

[5] K. Perlin, "An image synthetizer", in *SIGGRAPH'85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques,* New York, NY, USA, Jul 1985, vol. 19, pp. 287-296.

[6] K. Perlin, "Improving noise", In *Proceedings of SIGGRAPH 2002*, Ed Computer Graphics Proceedings, Annual Conference Series, ACM, 2002, 681–682.

[7] K. Perlin, "Making noise", http://www.noisemachine.com/talk1/, 2000.

[8] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin and M. Zwicker, "State of the art in procedural noise functions", in *Eurographics STAR program 2010*, 2010.

[9] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, "Texturing & modeling, a procedural approach", Second Edition, pp. 64-80, July 1998.

[10] http://www.cis.rit.edu/research/ultrasound/projects/generationSyntheticMedicalImages.html

[11] H.K. Grohs and O.A.N. Husain, Eds., Automated Cervi- cal Cancer Screening, Igaku-Shoin Medical Publishers, Inc., 1994.

[12] WHO, Comprehensive cervical cancer control: A guide to essential practice, WHO Press, 2006.

[13] http://en.wikipedia.org/wiki/Uniform_distribution_(continuous)

[14] http://mathworld.wolfram.com/FourierSeries.html

[15] http://sabia.tic.udc.es/gc/teoria/pto_medio_lineas/Inicio.html

[16] http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

[17] http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

[18] A. Weitzenfeld, "Gráfica: Línea", http://www.docstoc.com/docs/273281/Graficos-Lineas-Circulos, pp. 7-10.

[19] L. Feng, and S. H. Soon, "An effective 3D seed fill algorithm", *Computers and graphics,* vol. 22, issue 5, pp. 641-644, Oct, 1998.

[20] http://en.wikipedia.org/wiki/Cubic_Hermite_spline

[21] http://mathworld.wolfram.com/GeneralizedFourierSeries.html

[22] http://www.mathworks.com/matlabcentral/fileexchange/4927-vol3d-m-vol3dtool-m

[23] http://en.wikipedia.org/wiki/Rasterisation