Interpol, Routines for Interpolation

Alan Louis Scheinine, Senior Researcher, CRS4

CRS4
Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna
Sesta Strada, Ovest
Zona Industriale Macchiareddu
09010 Uta (Cagliari) Italy

E-mail: scheinin@crs4.it

Contents

List of Figures

List of Tables

1 Introduction

A set of C++ classes have been written for finding an interpolated scalar value at any point on a rectangular grid in one, two or three dimensions. The grid is considered to be composed of pixels or voxels and the known values are considered to be defined in one of two ways, either 1) the values of a continuous field measured at the center of each pixel or voxel, or 2) the integrated (averaged) values of the field for each pixel or voxel. (The classes include a method for interpolation using known values defined at arbitrary positions rather than on a rectangular grid, but this part of the package has not been tested.)

This packages's implementation of interpolation could be of general use, but the motivation has been for segmentation of medical images, hence, some design choices have been made for that specific application. Rather than converting the entire field to a set of coefficients of basis functions, a local conversion is done for a given point. Though not as efficient as a global conversion, an "as needed" conversion is useful for generating a two-dimensional slice of a three-dimensional grid at an arbitrary angle, since the number of values (points or pixels) of the slice is small in comparison to the number of values of the original three-dimensional grid. Another application is the calculation of forces on a set of points that define a balloon surface imbedded in three dimensions. For the specific set of basis functions used in this package, subroutines have been written to explicitly encode the integral and the directional derivatives.

Two types of basis functions have been implemented: 1) a Taylor expansion around a point and 2) a set of Gaussian functions, one per pixel/voxel implemented using b-splines.

For a given point, the set of positions for which the values are used as input to the interpolation is defined by a "stencil". The stencils are small. For example, in three dimensions the stencils are either 3-cubed, or slightly large with either 6 or 30 points added from the next outer layer. For a Taylor expansion based on a cube of 27 stencil sites, there are 27 polynomials and all but the constant term become zero at the center of the cube. For this kind of interpolation, a global conversion would not be practical because the numerical representation increases by a factor of 27 or more. For interpolation using Gaussian basis functions, each Gaussian is centered on a site of the stencil. For this latter kind of interpolation, a global conversion would not increase the size of the numerical representation.

These C++ classes should be useful for using interpolation in the following cases:

- slicing a volume,
- changing the aspect ratio, for example, changing the step size in the z direction so that it is equal to the step size of the x and y directions,
- finding forces for active contours or active surfaces.

The author hopes that these higher level algorithms can remain intact even if the basis functions and stencils of the interpolation technique are changed. The interpolation technique has not been studied with respect to distortions that might be inherent in X-ray tomography or magnetic resonance imaging. In particular, it is assumed that the value of each voxel is the average of the actual "material" within that voxel. If the imaging procedure created a more defocused result than just averaging within a voxel, then deconvolution would be necessary and a larger stencil would be justified. Instead, it is assumed that any necessary deconvolution is done during data acquisition. With such an assumption, a large stencil would create averaging that is not appropriate when the medically important information concerns abrupt boundaries.

2 The Mathematics of Interpolation

Suppose that the graylevel G at a point $\vec{x}$ of an image is given by the sum of m functions

$$G(\vec{x}) = \sum_{i=1}^{m} f_i(\vec{x}) \times C_i \ , \tag{1}$$

where the graylevel at point $\vec{x}$ is an unknown, to be determined by interpolation. If we have n known graylevels where $n \geq m$ then we can derive the set of coefficients $\vec{C}$ using the equation

$$\vec{C} = \left(\mathrm{M}^t\mathrm{M}\right)^{-1}\mathrm{M}^t\vec{G} \tag{2}$$

where we define

$$\vec{G} \equiv (G_1, G_2, \ldots, G_n) \tag{3}$$

with the definition

$$G_j \equiv G(\vec{x}_j) \ . \tag{4}$$

The matrix M is defined by

$$\mathrm{M}_{ij} \equiv f_i(\vec{x}_j) \tag{5}$$

where, for this particular implementation, the functions $f(\vec{x})$ are either terms of a Taylor expansion in $\vec{x}$ or are approximations to Gaussian functions. When the number of points of known graylevels is equal to the number of functions that are summed to derive an interpolated graylevel, then the matrix M is square and the coefficients of the functions can be derived from

$$\vec{C} = \mathrm{M}^{-1}\vec{G} \tag{6}$$

For this implementation, the classes called ArbitrarySites and ArbitraryMatrix have been written for the case of more known points than terms in the sum and when the positions of the known graylevel points are arbitrary. However, these classes have not been tested. The rest of this document deals with that case in which the number of coefficients is equal to the number of known graylevels, that is, Eq. 6. For a rectangular grid in which the known values are always at the same positions, the matrix $f_i(\vec{x}_j)$ is constant. Its inverse needs to be taken only once.

Suppose that the known values are not graylevels at a point, but rather, graylevels of pixels or voxels (the term "box" will be used to indicate either a pixel or a voxel) where the value for a given box is the average of the intensity (or material density) within the box. In that case we can write

$$\vec{C} = \widetilde{\mathrm{M}}^{-1}\vec{\mathcal{G}} \tag{7}$$

where $\vec{\mathcal{G}}$ is the graylevel of a box and the matrix $\widetilde{\mathrm{M}}$ is given by

$$\widetilde{\mathrm{M}}_{ij} \equiv \int_{box_j} f_i(\vec{x}_j)\, d\vec{x}_j \ . \tag{8}$$

The coefficients $\vec{C}$ are the same as in Eq. 6 for graylevels defined at points. So for either a grid of values or a grid of boxes (the latter being typical of a medical image) we derive a set of coefficients for the underlying functions. The interpolated value for a point is given by Eq. 1 whereas the interpolated value for a box (pixel or voxel) is given by Eq. 9.

$$\int_{box} G(\vec{x})\, d\vec{x} = \sum_{i=1}^{m} \int_{box} f_i(\vec{x})\, d\vec{x} \times C_i \ , \tag{9}$$

For the functions implemented in this package, there are class methods for the integral over a rectangular region, as well as, class methods for the gradient. So the user can easily derive either M^{-1} or $\widetilde{\mathrm{M}}^{-1}$ for input that is a grid of points or a grid of boxes, to arrive at the set of coefficients $\vec{C}$. Then using the set $\vec{C}$ the user can easily derive a value at an arbitrary point or the average value for an arbitrary box. Going from boxes to boxes would be the procedure used for creating a more refined grid. Going from boxes to points, then going from points to boxes would be the procedure for rotating a dataset or for slicing a dataset at an arbitrary angle. The integration of the underlying functions for a given box has only been implemented when the box is aligned with the grid. When a new grid is created that is at an angle relative to the original grid, the interpolation must derive values for points on the new grid, then the points are converted to average, integrated values for pixels or voxels. On the other hand, the conversion from underlying functions to pixels or voxels does not need to be done if the segmentation algorithm is based on calls to methods of classes of this package for obtaining the value at a point or the gradient at a point; for example, the methods of active contours or active surfaces.

The set of positions of the known values is a called a stencil. The stencils implemented in this package are shown in Sec. A.

3 Regriding

The C++ class RegridBrick has been defined for changing the pixel or voxel size, that is, RegridBrick contains methods for refining or coarsening a grid. (Every reference in this section to "pixels" also applies to the three-dimensional case of voxels.) The user can either change the number of steps in any direction, or change the pixel size in any direction. If the number of steps is changed then the pixel size is adjusted accordingly to maintain the same overall size as the original. If the pixel size is changed, then the number of steps is adjusted but the overall size may change slightly. The interpolation in this case is from pixel to the underlying functions then back to pixel. In other words, the image is assumed to represent intensity or material densities integrated over the pixel. The procedure can best be described by referring to Fig. 3.1.

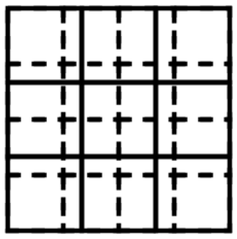


Figure 3.1: Two pixel sizes for regriding. One grid is indicated by solid lines and the other grid is indicated by dashed lines.

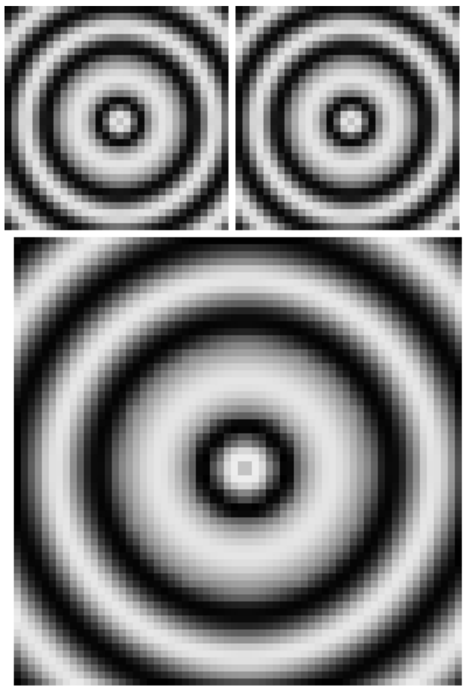
The two grids are indicated by the lattice of solid lines and the lattice of dashed lines. Whether going from a larger grid to the smaller grid, or vice versa, a given pixel of the new grid can contain parts of several pixels of the original grid. For each pixel of the new grid, all rectangular components that are intersections between a pixel of the old grid and a given pixel of the new grid undergo the interpolation procedure. The value of the new pixel is the average weighted by area of the interpolated values of the rectangular components.

A test program was written in which an image was refined then coarsened using the methods of the class $\mathrm{RegridBrick}$, then the difference was measured between the original and final image. The test was done for grids of one, two and three dimensions. For the three dimension case, the aspect ratio of the original voxels was either $1:1:1$ or $1.5:0.8:1$. The number of grid steps for the original image was either 8, 16, or 32. The refinement was either $0.5:0.5:0.5$ or $1/3.25:1/4.75:1/6.5$. The graylevels were floating point numbers.

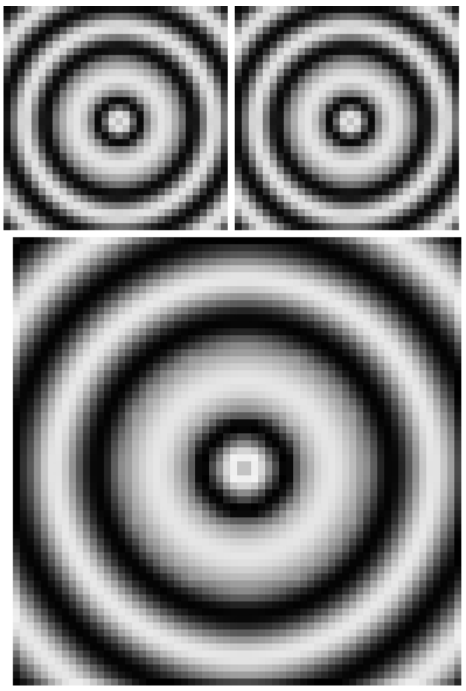
The interpolation was done with various stencils and basis functions. See Sec. A for a description of the stencils. See Sec. B and Sec. C for a description of the basis functions. In every case, when the refinement was a factor of 0.5, the difference between the original and final grids was neglible, on the order of one part per billion. For the case in which the grid was refined by the factors 1/3.25, 1/4.75, 1/6.5 in the x, y, and z directions, respectively, the average difference between the original and final image was a factor of 0.001 to 0.005 of the average value of the field. Though larger, this error is relatively small: on the order of one graylevel step for an 8-bit image. The error was larger, on the order of one percent, for the unrealistic case of using Gaussian basis functions that do not correctly deal with the edges of the stencil. Though the differences between the original and final grids were small, the images of the intermediate, refined grids showed some differences that depended on the stencil and basis functions. The results for a refinement of 1/2 are shown in Fig. 3.2, Fig. 3.3 and Fig. 3.4. The results for a refinement of 1/3.25 and 1/4.75 in the x and y directions are shown in Fig. 3.5, Fig. 3.6 and Fig. 3.7. The images were only made for the two-dimensional cases.

Comparing Fig. 3.2, a stencil of 9 sites, with Fig. 3.3, a stencil of 13 sites, we see that the larger stencil gives a slightly more uniform refinement. The Fig. 3.4, in which a stencil of 21 sites was used, does not show a Taylor expansion because one was not implemented for 21 terms in two dimensions. For these figures, a refinement was a factor of 1/2. When using Gaussian basis functions, the functions at the edges of the stencil need to adjust for the fact that the set of Gaussians are spatially truncated (the special case is handled by a sum of a two of Gaussians offset from each other by one lattice site). If this is not done, the consequence, aliasing from the original pixels, can be seen in Fig. 3.2(c), Fig. 3.3(c) and Fig. 3.4(b).

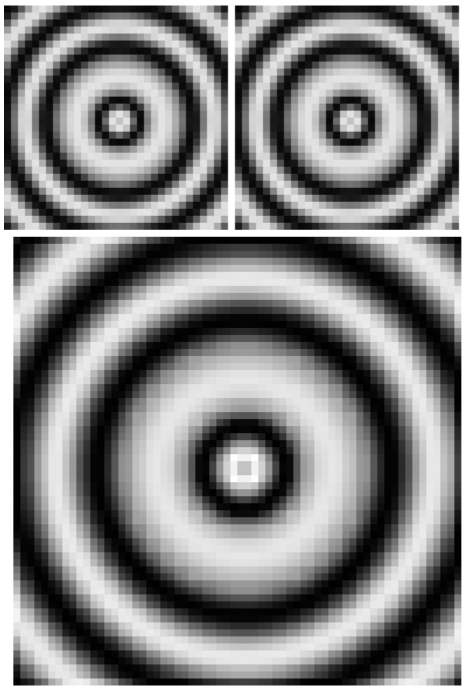
Comparing Fig. 3.5, Fig. 3.6 and Fig. 3.7 we see that a larger stencil gives a smoother intermediate, refined image. Of coarse, the last image in each set



(a)

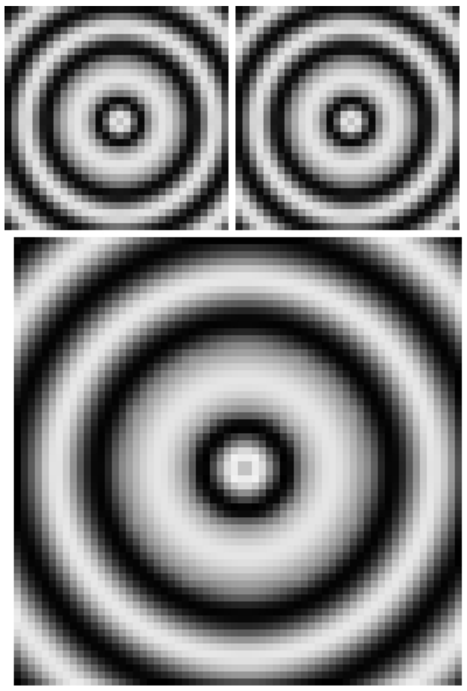


(b)

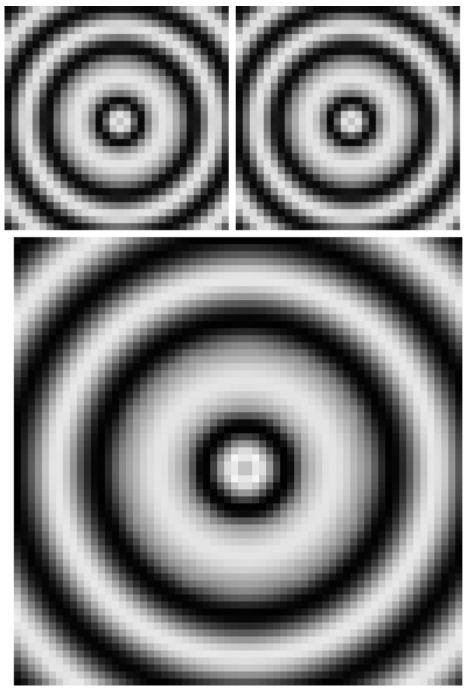


(c)

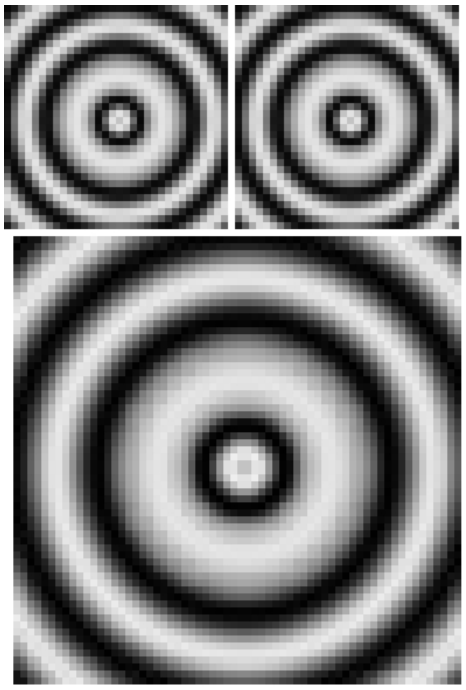
Figure 3.2: Refinement using a 3 by 3 stencil. The images at the top of each group of three show the original image and the final image. The final image is nearly identical to the original image. The lower image of each set shows a refinement by a factor of 1/2. (a) Taylor expansion basis. (b) Gaussian basis with correct treatment of edges. (c) Gaussian basis without edge correction.



(a)

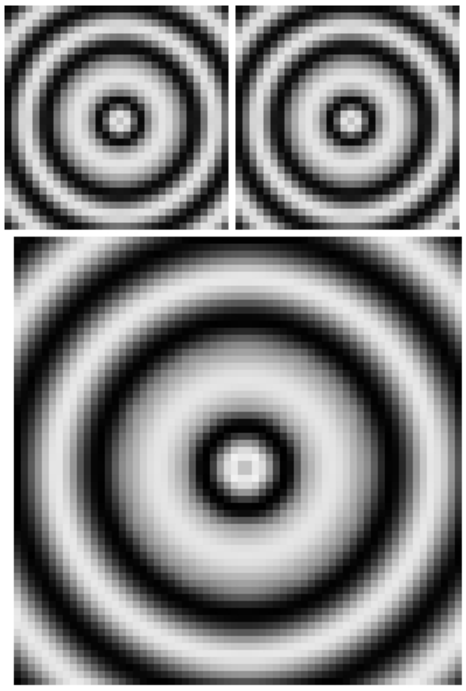


(b)



(c)

Figure 3.3: Refinement using a 13-site stencil, which is a 3 by 3 stencil plus four sites along the axes. The images at the top of each group of three show the original image and the final image. (The final image is nearly identical to the original image.) The lower image of each set shows a refinement by a factor of 1/2. (a) Taylor expansion basis. (b) Gaussian basis with correct treatment of edges. (c) Gaussian basis without edge correction.


(a)

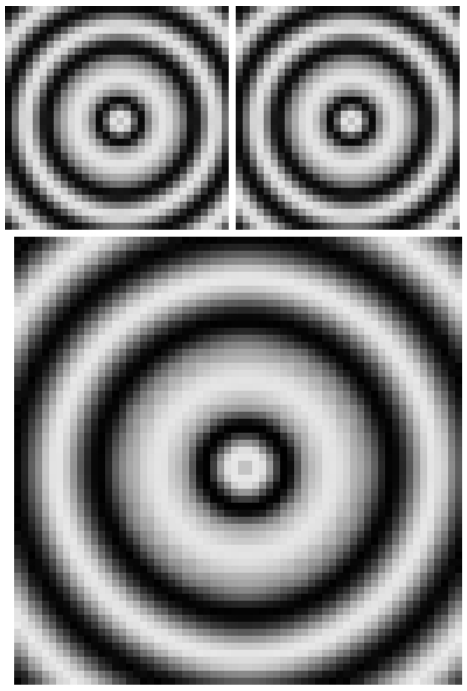
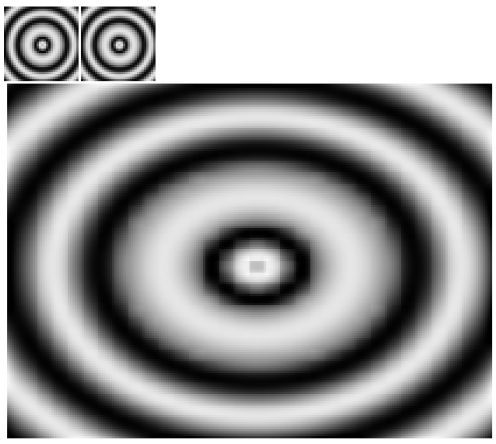
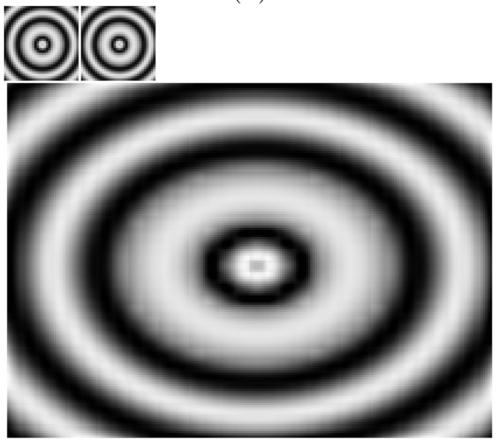

(b)

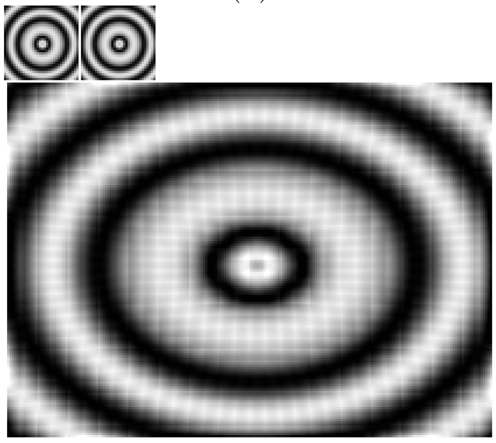
Figure 3.4: Refinement using a 21-site stencil, which is a 5 by 5 stencil without the corners. The images at the top of each group of three show the original image and the final image. (The final image is nearly identical to the original image.) The lower image of each set shows a refinement by a factor of 1/2. For 21 terms, a Taylor expansion was not implemented. (a) Gaussian basis with correct treatment of edges. (b) Gaussian basis without edge correction.



(a)

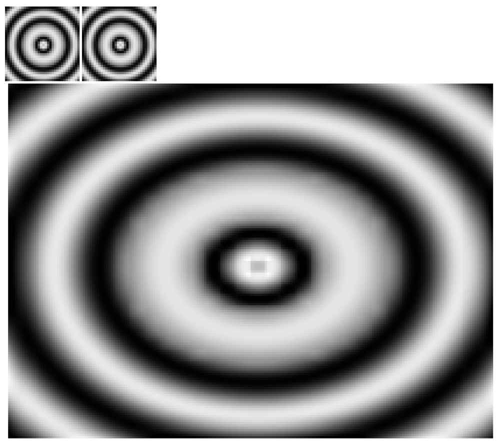


(b)

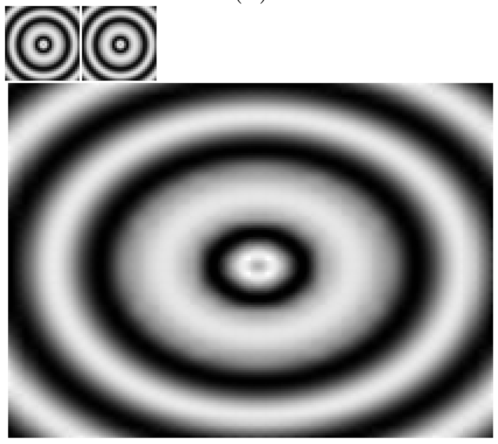


(c)

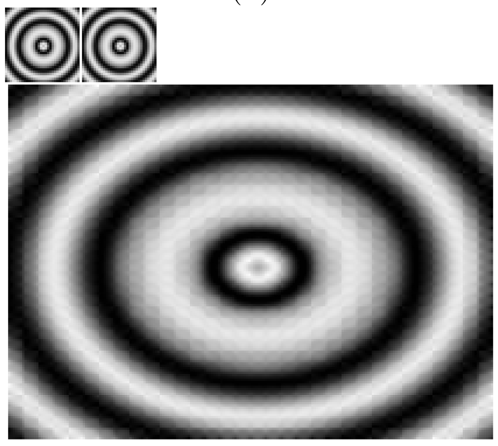
Figure 3.5: Refinement using a 3 by 3 stencil. The images at the top of each group of three show the original image and the final image. The final image is nearly identical to the original image. The lower image of each set shows a refinement by a factor of $x/3.25$ and $y/4.75$. (a) Taylor expansion basis. (b) Gaussian basis with correct treatment of edges. (c) Gaussian basis without edge correction.



(a)

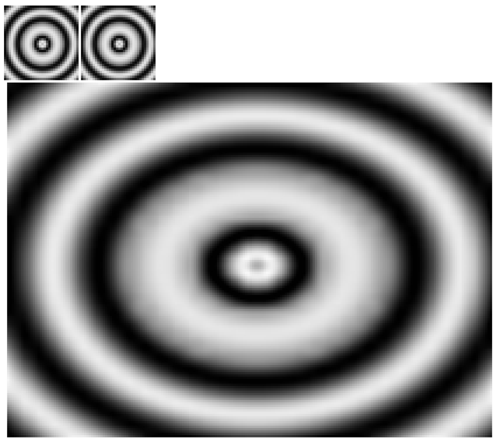


(b)

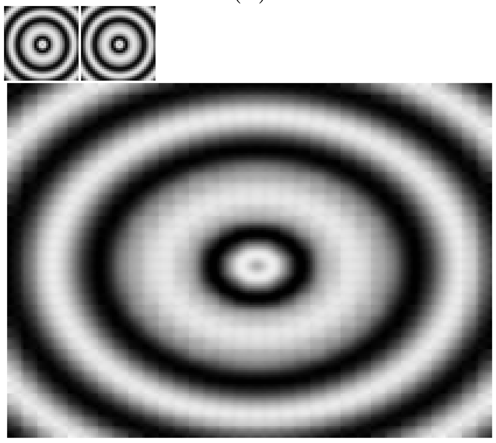


(c)

Figure 3.6: Refinement using a 13-site stencil, which is a 3 by 3 stencil plus four sites along the axes. The images at the top of each group of three show the original image and the final image. (The final image is nearly identical to the original image.) The lower image of each set shows a refinement by a factor of $x/3.25$ and $y/4.75$. (a) Taylor expansion basis. (b) Gaussian basis with correct treatment of edges. (c) Gaussian basis without edge correction.



(a)



(b)

Figure 3.7: Refinement using a 21-site stencil, which is a 5 by 5 stencil without the corners. The images at the top of each group of three show the original image and the final image. (The final image is nearly identical to the original image.) The lower image of each set shows a refinement by a factor of $x/3.25$ and $y/4.75$. For 21 terms, a Taylor expansion was not implemented. (a) Gaussian basis with correct treatment of edges. (b) Gaussian basis without edge correction.

shows the consequence of not correctly treating the edges when using Gaussian basis functions and is not a proposed interpolation method.

It is not necessarily the case that the smoother refined image, which does not show aliasing of the original coarser pixels, is the better technique. Perhaps, the more the smoothing, the greater the uncertainty of the position of abrupt boundaries.

4 Rotation Test

To test the interpolation computer program and to compare the different basis functions and stencils, a portion of a medical image was rotated by in five steps, an angle of $360/5$ degrees per step, then compared to itself. The image was small, $50 \times 50 \times 50$, so it cannot be considered a representative case. Nonetheless, the results are informative. The rotation was done for all axes in the set $(x = -1, 01, y = -1, 0, 1, z = -1, 0, 1)$ except, of course, $(0, 0, 0)$. For each step, the voxel graylevels were used to derive the coefficients of the underlying basis functions then this information was used to reconstruct a grid in which the voxel graylevels were the integrated averages. The graylevels were floating point numbers.

Four techniques were used for the construction of new voxels, though only two of the techniques were expected to give correct results. The other two techniques were tried for comparison purposes and for curiosity. The first and most primitive technique is to map the center of each new voxel to a point on the previous grid and use the graylevel found at the point, that is, no interpolation. The second technique is to map the center of each new voxel to a point on the previous grid, then use interpolation to find a graylevel for that point on the previous grid, then use that value for the voxel on the new grid. This technique is also wrong because on the new grid point values are assigned to voxels.

One fact of fundamental importance is that the voxels of the new grid do not have angular alignment with the voxels of the previous grid, so that it is not possible to find an integrated graylevel on the previous grid using the Interpol package. The information carried over from the previous grid to the new grid must be point values. The third technique uses the points as determined by the second technique then uses interpolation on the new grid to find voxel values. In general, this technique works well, though the quality depends on the stencil and basis functions. The fourth technique is over-sampling. From the previous grid, eight interpolated point values are found for each voxel of the new grid, points uniformly spaced in the voxel. Then, as in the third technique, the points are used to find basis function coefficients that are then used to derive voxel values. The techniques are summarized in Table 4.1.

For a rotation around the z-axis, images of the central xy-plane were written to files. Some of these images are shown in the following figures. The first technique is shown in Fig. 4.8. The second, third and fourth techniques for a 27-site stencil using a Taylor expansion basis are shown in Fig. 4.9. The second, third and fourth techniques for a 27-site stencil using a Gaussian weight basis are shown in Fig. 4.10. The second, third and fourth techniques for a 33-site stencil using a Taylor expansion basis are shown in Fig. 4.11. The second, third and fourth techniques for a 33-site stencil using a Gaussian weight basis are shown in Fig. 4.12. The basis composed

Interpolation Techniques for Rotation Mapping	
(1)	Value at voxel of previous lattice used for new voxel.
(2)	Interpolated value at point of previous lattice used for new voxel.
(3)	Points found in (2) interpolated on new lattice for new voxel values.
(4)	Technique (3) with over-sampling.

Table 4.1: Because the previous and new grids are not angularly aligned, the mapping is point to point. For the first three techniques, the center point of a new voxel is mapped back to a point on the previous grid, whereas, for the fourth technique a number $2^{dimension}$ points are mapped back for each new voxel. For the first technique, the voxel value at the backwards-mapped point is used as the new voxel value. For the second technique, the value of the point on the previous grid is calculated using interpolation. In the case that the values on the new grid represent integrated averages, the third technique is needed, in which, the points found with the second technique are used to find coefficients of basis functions on the new grid from which voxel values are obtained. The fourth technique is similar to the third, except that more points are mapped so that the basis functions on the new grid are derived from a more dense set of points.

of Guassian functions will be called the “Gaussian weight” basis.

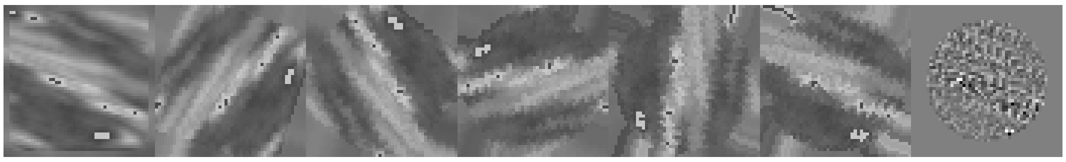
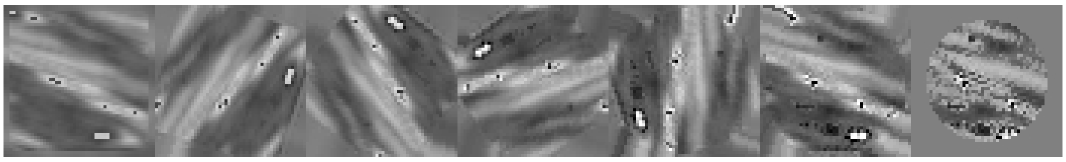
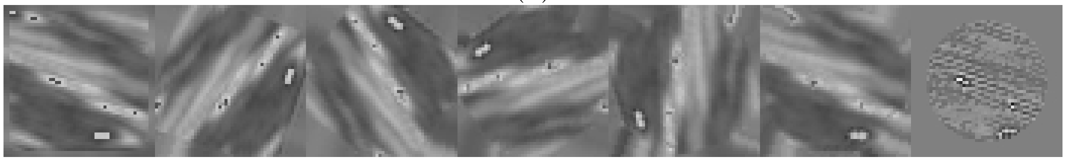


Figure 4.8: Five rotation steps and difference between final and original 50-cubed grid, central xy-plane. Mapping from new voxel center to a point in the previous grid and using that voxel value for the value of the new voxel.

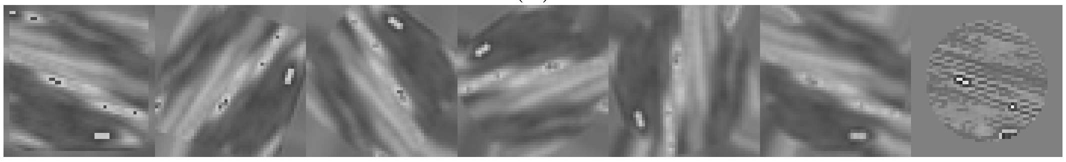
In the figures, the final frame of each strip is the difference between the final image and the original image. The differences are exaggerated in order to be visible. The cases labeled (a) in the figures (the second technique) are not expected to work correctly because voxel values on the new grid are assigned directly from point values on the new grid. The second technique is shown in order to emphasis the need to convert the mapped points to intergrated average values using interpolation on the new grid, as is done for the third and fourth techniques.



(a)

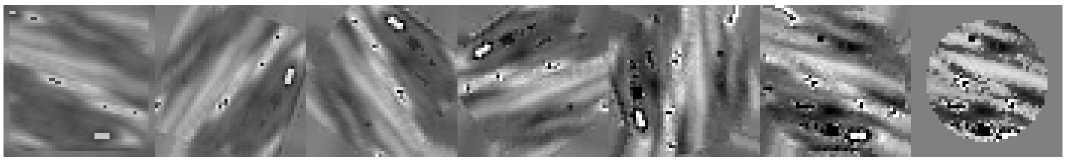


(b)

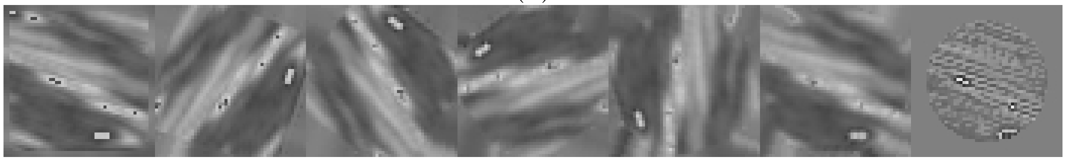


(c)

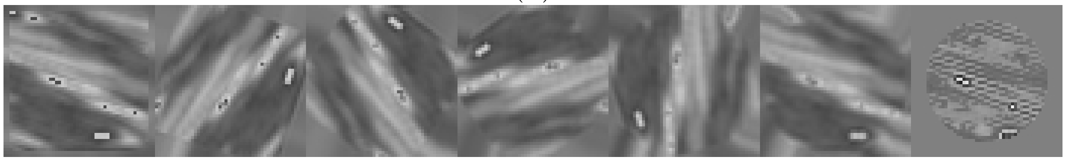
Figure 4.9: Five rotation steps and difference between final and original 50-cubed grid, central xy-plane; stencil $3 \times 3 \times 3$ and Taylor expansion. (a) Using an interpolated value of the previous grid for the new voxel. (b) Interpolated values of the previous grid interpoled on the new grid for new voxel values. (c) Same as case (b) but with a factor of 8 over-sampling.



(a)

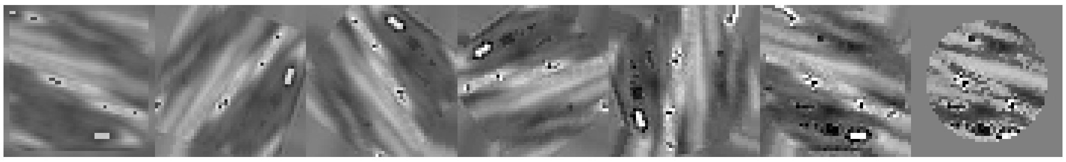


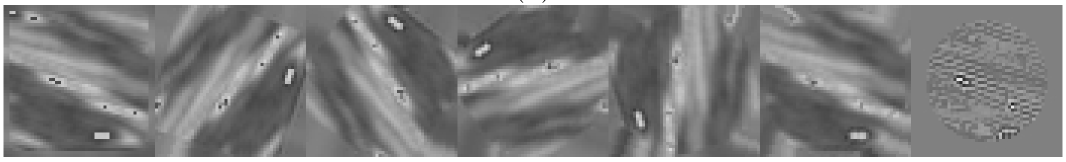
(b)



(c)

Figure 4.10: Five rotation steps and difference between final and original 50-cubed grid, central xy-plane; stencil $3 \times 3 \times 3$ and Gaussian basis functions. (a) Using an interpolated value of the previous grid for the new voxel. (b) Interpolated values of the previous grid interpoled on the new grid for new voxel values. (c) Same as case (b) but with a factor of 8 over-sampling.


(a)


(b)

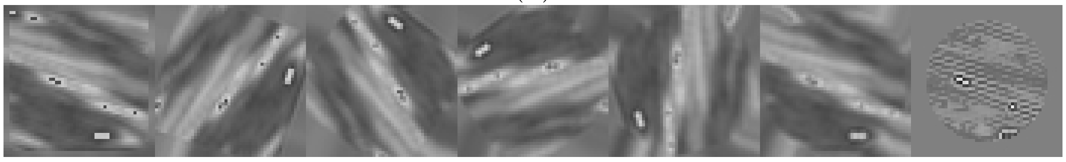
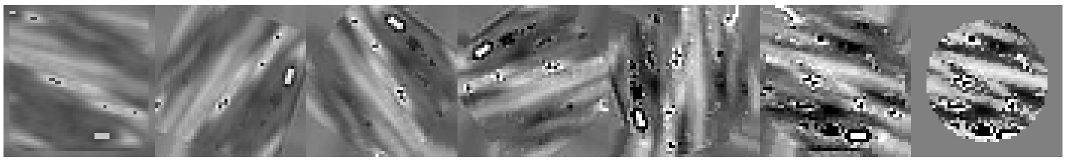
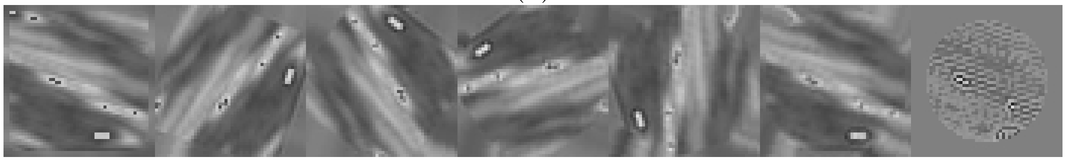

(c)

Figure 4.11: Five rotation steps and difference between final and original 50-cubed grid, central xy-plane; 33-site stencil and Taylor expansion. (a) Using an interpolated value of the previous grid for the new voxel. (b) Interpolated values of the previous grid interpoled on the new grid for new voxel values. (c) Same as case (b) but with a factor of 8 over-sampling.


(a)


(b)

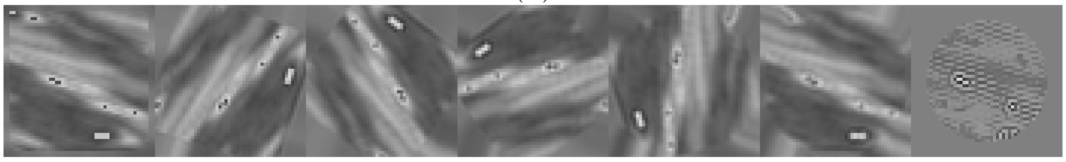

(c)

Figure 4.12: Five rotation steps and difference between final and original 50-cubed grid, central xy-plane; 33-site stencil and Gaussian basis functions. (a) Using an interpolated value of the previous grid for the new voxel. (b) Interpolated values of the previous grid interpoled on the new grid for new voxel values. (c) Same as case (b) but with a factor of 8 over-sampling.

For this particular image, we see that the fourth technique of over-sampling is not better than the third technique. Moreover, we can see that carrying over one point per voxel is enough to represent the information contained in the image. In general, the 33-site stencil gives better results than the 27-site stencil. Moreover, though it may not be visible on the printed page, the Taylor expansion basis gives a slight smoothing of details whereas the Gaussian weight basis is very slightly more crisp. For a Gaussian weight basis, a 57-site stencil is also available. The results are not shown because the difference from a 33-site stencil is barely visible.

For all the figures presented thus far in this section, the Gaussian weight basis has a thicker Gaussian at the edges. See Sec. C for an explanation. For the sake of comparison, the Gaussian basis that does not correct for the edges is shown in Fig. 4.13. The figure shows the third and fourth technique. We see that the over-sampling used in the fourth technique compensates for the error in the choice of basis functions.

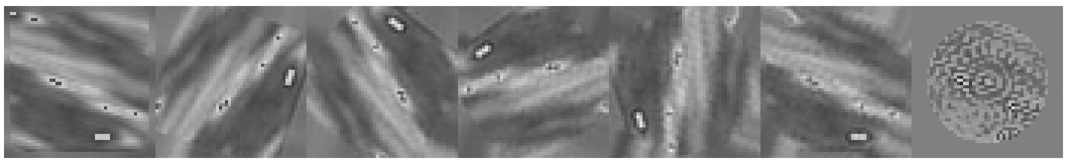
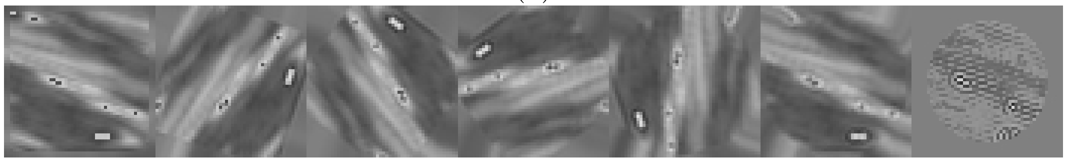
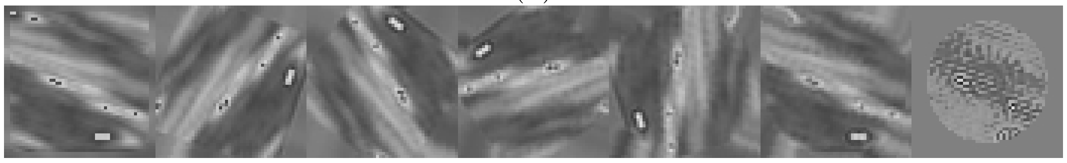
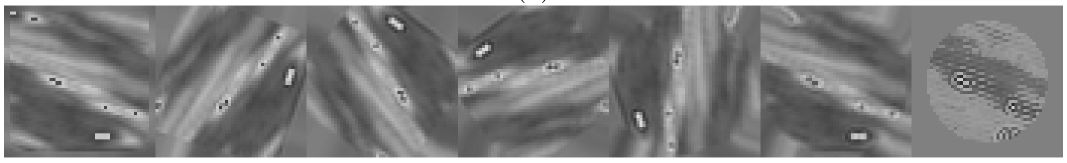

(a)

(b)

(c)

(2)

Figure 4.13: Five rotation steps and difference between final and original 50-cubed grid, central xy-plane; 33-site stencil and 57-site stencil using Gaussian basis functions that do not correctly handle the edges. (a) 33-site stencil and technique 3. (b) 33-site stencil and technique 4. (c) 57-site stencil and technique 3. (d) 57-site stencil and technique 4.

The figures show rotation around one axis. The results for rotation around 26

different axes in shown in Table 4.2.

The rankings from Table 4.2 should be considered approximate because they describe just one case and because the difference between the fluctuation value is very small between neighboring entries. Techniques 1 and 2 (see Table 4.1) are not expected to give good results and in fact those cases are all grouped at the bottom of the list. We see that the stencils with a larger number of sites (stencils 2 and 3) give better interpolation accuracy than stencil 1. In addition, we see that over-sampling, technique 4, is more accurate than technique 3. The basis 3, which does not have the correct basis function for the edges of the stencil, has cases near the top of the table. However, we see the problem with basis 3 when looking at the average difference, which is often large for basis 3.

Interpolation Accuracy for Rotation				
Fluctuation	Average	Stencil	Basis	Technique
0.134	0.00285	3	2	4
0.136	0.00933	2	3	4
0.138	0.07431	3	3	4
0.141	0.01860	2	2	4
0.144	0.00385	3	2	3
0.148	0.06747	3	3	3
0.149	0.01280	2	2	3
0.157	0.03073	2	3	3
0.159	0.00036	2	1	3
0.160	0.00016	2	1	4
0.170	0.00027	1	2	4
0.172	0.00035	1	1	3
0.177	0.00015	1	1	4
0.179	0.00019	1	2	3
0.228	0.23680	1	3	3
0.248	0.26362	1	3	4
0.293	0.00083	1	1	2
0.347	0.00161	1	1	1
0.372	0.00101	2	1	2
0.462	0.00081	1	2	2
0.685	0.00725	2	2	2
0.695	0.03442	1	3	2
0.748	0.00360	3	2	2
0.823	0.03596	2	3	2
0.871	0.01379	3	3	2

Table 4.2: Rotations of five steps with results averaged over 26 different axes of rotation. The first column, *Fluctuation*, is the root mean square (RMS) of the difference of the original and final fields after the average difference is removed. This value is used to rank the interpolation techniques. The second column, *Average*, is the difference averaged over all voxels. As an average over rotation axes, the absolute value of the average difference is used. The values for both columns are divided by the RMS value of the original field. The column *Stencil* indicates (1) 27-site stencil, (2) 33-site stencil, (3) 57-site stencil. The column *Basis* indicates (1) Taylor expansion, (2) Gaussian weights with edge correction, (3) Gaussian weights without edge correction. The number in the column *Technique* refers to the techniques described in Table 4.1.

A Stencil Sites

A stencil can have different step lengths for different directions. In particular, the size is the pixel or voxel size. For intensities measured at points, the center of each box is the position of the point. Not all stencil patterns are available in the Interpol package, the ones that have been implemented are shown in the following figures.

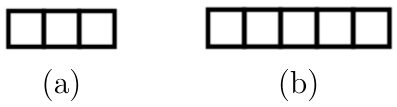


Figure A.14: One-dimensional stencils. (a) Three sites. (b) Five sites.

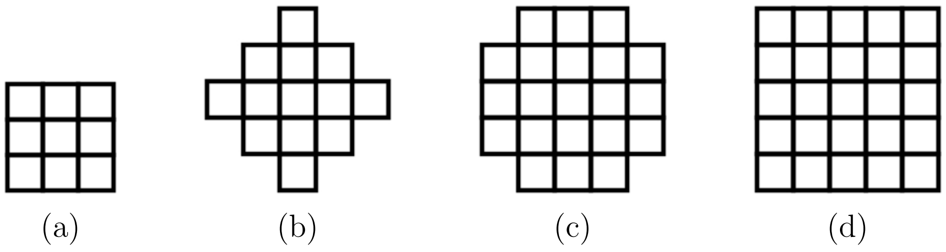


Figure A.15: Two-dimensional stencils. (a) 9 sites. (b) 13 sites. (c) 21 sites. (d) 25 sites.

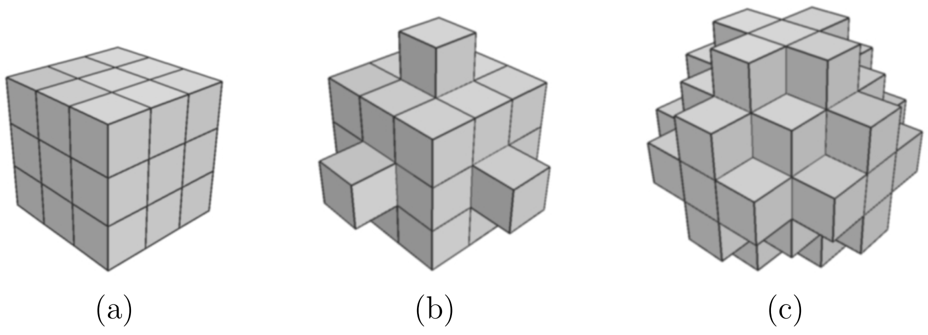


Figure A.16: Three-dimensional stencils. (a) 27 sites. (b) 33 sites. (c) 57 sites.

B Taylor Expansion Basis Functions

Given a set of points (a stencil) with graylevel values defined for the points, the values can be fitted using a Taylor expansion around the center point of the stencil. The following polynomial terms were chosen.

3 terms, one dimension

$$1, x, x^2 \tag{10}$$

5 terms, one dimension

$$1, x, x^2, x^3, x^4 \tag{11}$$

9 terms, two dimensions

$$(1, x, x^2) \times (1, y, y^2) \tag{12}$$

13 terms, two dimensions

$$(1, x, x^2) \times (1, y, y^2) \text{ and } x^3, y^3, x^4, y^4 \tag{13}$$

27 terms, three dimensions

$$(1, x, x^2) \times (1, y, y^2) \times (1, z, z^2) \tag{14}$$

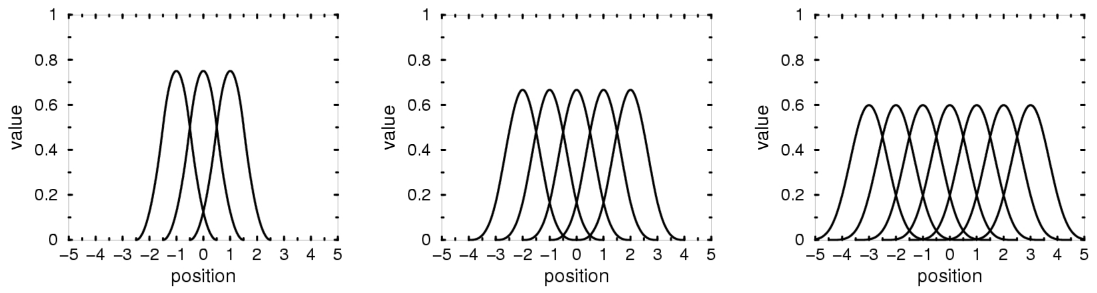
33 terms, three dimensions

$$(1, x, x^2) \times (1, y, y^2) \times (1, z, z^2) \text{ and } x^3, y^3, z^3, x^4, y^4, z^4 \tag{15}$$

The corresponding stencils are shown in Sec. A. The Interpol package has functions to calculate an n-vector of these terms for a given point, where n is the number of polynomial terms; to calculate the d n-vectors that are the derivative of these terms, where d is the dimension, and an n-vector of the integral of these terms for a given bounding box.

C Bspline Functions

Rather than using a sum of polynomial terms centered on the middle of a pixel, an interpolation can be made by using the same function for every site, as long as the functions overlap where the interpolation is being done. Several functional forms are considered in [1]. Based on their evaluation, this author chose a b-spline approximation to a Gaussian function. The second, third and fourth order b-spline approximations are shown in Fig. C.17. The figures are for one dimension. Higher dimensional functions are the product of the one-dimensional functions.



(a) (b) (c)

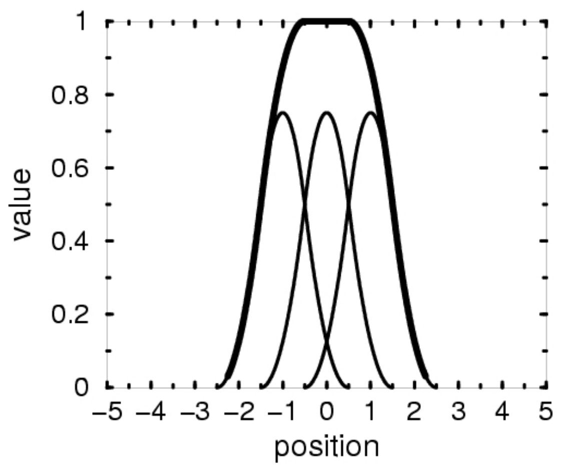
Figure C.17: Second, third and fourth order b-spline approximations to a Gaussian function. The zone in which an interpolation is made is the range from -0.5 to +0.5. There is one curve for each site. the sites are centered on the integer values of the x-axis. The curves shown are those that are non-zero within the zone of interpolation. (a) Second order. (b) Third order. (c) Fourth order.

An advantage of Gaussian weights is that the importance of a given pixel changes gradually as the point being interpolated moves from one pixel center to another pixel center. This should give continuous higher derivatives for the interpolated values.

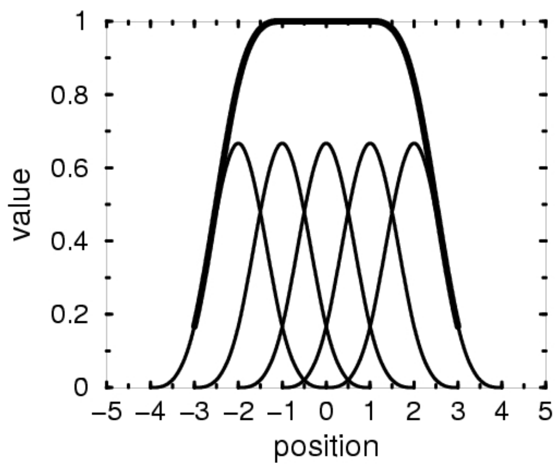
It would be possible to convert an entire image from a set of graylevel values to a set of coefficients for the Gaussian weight functions in which there is one function at each site. The Interpol **package takes a different approach by calculating the coefficents when needed for each point interpolated. The approach of the** Interpol **package is less efficient and is due to the fact that the first technique considered was a Taylor expansion for which there can be as many as 33 coefficients at each site. That is, a global conversion to interpolation coefficients would increase the data set size by as much as 27 or 33 for a three-dimensional image.**

Due to that fact that the set of Gaussian weights does not extend over the entire image, the groups shown in Fig. C.17 give poor results. To understand why, consider a uniform graylevel. That should be approximated by a set of equal

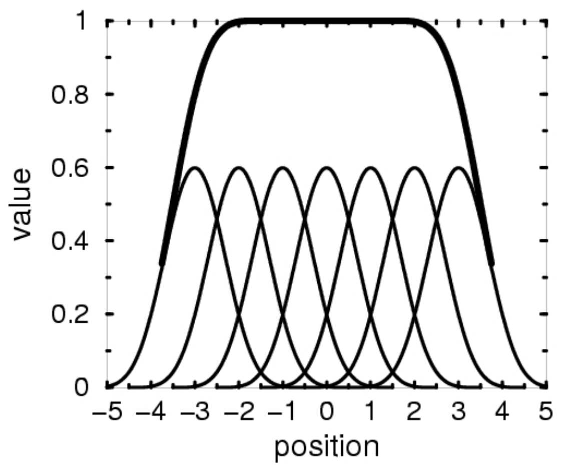
Gaussian weights. But the sum of equal Gaussian weights is not flat over the entire stencil, as can be seen in Fig. C.18.



(a)



(b)

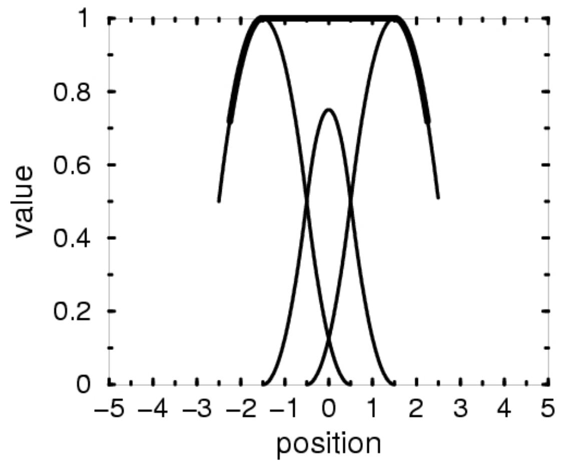


(c)

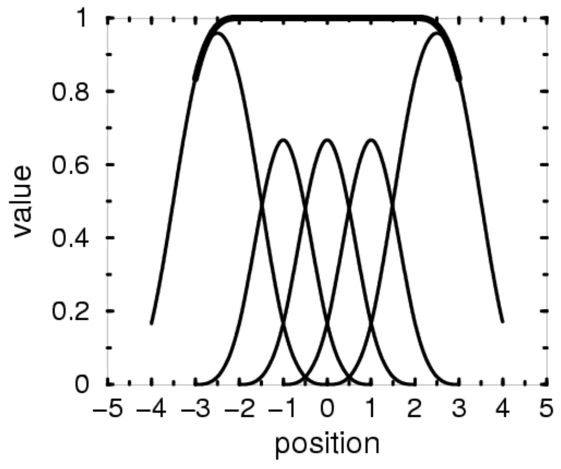
Figure C.18: Second, third and fourth order b-spline approximations to a Gaussian function. The darker curve above the others is the sum. For the second-order b-spline approximation there are three sites and three curves. All three sites cover the range from -1.5 to +1.5 whereas the flat region of the sum is less wide. The same problem can be seen for the third-order and fourth-order bsplines. (a) Second order. (b) Third order. (c) Fourth order.

At the extremes, the sum of the Gaussian weights is not equal to one, as can be seen in Fig. C.18. If the graylevels are said to be an integrated average over a box, then the coefficients at the extremes needs to be larger than the inside coefficients in order to fit the data of a uniform gray level. But in that case, interpolated values will be variable whereas they should be constant. In other words, fitting the known values will force a gradual curve in the values as a function of position, whereas the

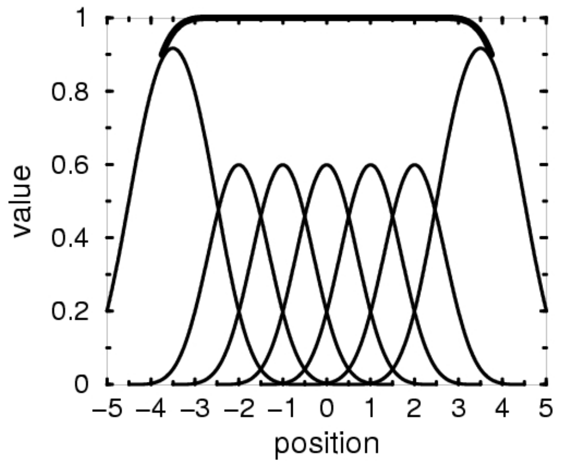
values should be flat. The solution is to use a different function at the edge: the original Gaussian plus a Gaussian one step further. This is shown in Fig. C.19.



(a)



(b)



(c)

Figure C.19: Second, third and fourth order b-spline approximations to a Gaussian weight used for all sites except at the extremes. At the extremes the sum of two Gaussians is used. The darker curve above the others is the sum. The flat region extends beyond the stencil sites used as input to the interpolation. (a) Second order. (b) Third order. (c) Fourth order.

The Gaussian weights have the magnitudes shown in Fig. C.19. The consequence is that a set of coefficients for the Gaussian weights for which all coefficients have the same value, will fit the data for either a stencil of points or a stencil of integrated averages over boxes; since for equal coefficients the sum of these Gaussian weights is uniform over the entire width of the stencil. Numerical experiments have shown that this technique of choosing the Gaussian weights works well for interpolation for non-constant graylevels.

Though the technique of choosing different Gaussian weight functions at the extremes of a stencil makes the Interpol package somewhat complex, the main goal of the package is to provide methods for higher level transformations such as changing the scale, rotating or slicing an image. For the higher level methods, the user simply indicates with a flag what type of basis is to be used. The details of the implementation of the weight functions are restricted to a few classes. A different interpolation weight function (set of functions) could be implemented by changing just a few classes if it proved better for a certain type of image, and in some cases the coefficients of the weight fuction might not be conveniently calculated globally.

With regard to calculating Gaussian weight coefficients as needed, rather than globally as an initial step, there are two additional considerations. First of all, for a given point (or pixel) in which an interpolated value is to be calculated, the polynomial terms need to be calculated for that point (or pixel), that is, the factors $f_i(\vec{x})$ or $\int_{box} f_i(\vec{x})\, d\vec{x}$ in Eq. 1 or Eq. 9 respectively. Only the coefficients of the polynomial terms can be pre-calculated. The coefficients are found by multiplying a matrix by the graylevel values on sites (or pixels) indicated by a stencil. The matrix is small, for a stencil of n sites the matrix is $n \times n$. Moreover, for a uniformly spaced grid the matrix is constant and only needs to calculated once. In fact, it is stored unrolled as a linear array. So pre-calculating the coefficients only eliminates a small part of the interpolation calculation for a given point (or pixel). A second consideration is that pre-calculating the coefficients means that all of the Gaussian weight functions are equal, one for each site. In that case, in order to eliminate "edge effects," more Gaussian weight functions need to be used, so the stencil is larger. But for images with abrupt boundaries, a larger stencil may cause in inappropriately large amount of smoothing.

D Subpixel Accuracy, a Personal Perspective

My first experience with segmentation of images involved the carotid artery. The technique of active contours was used in which a closed contour was fitted to the data of each two-dimensional slice of a three-dimensional dataset. The program used was GSNAKE [2]. The goal of the project was to study the fluid dynamics of blood, for example, to see if arterial plaques tended to form where the pressure was unusually high or unusually low, or where the speed of flow was unusually high or low. For the initial stages of that project, an archetype bifurcation of the carotid artery was sufficient. The GSNAKE program tended place points at discrete positions corresponding to the pixel positions and the contour had a tendency to jump between being just inside or just outside the region in which the graylevel changed abruptly. This problem was solved by smoothing the contour at a later stage. The postprocessing that smoothed the contours and smoothed the reconstructed arterial surface did not impede the use of the reconstruction as an archetype. However, for other types of medical image segmentation such as surgical planning, a postprocessing stage of smoothing that does not make reference to the original data may be an unacceptable source of error.

The algorithms developed in this package have not been studied with reference to machine (CAT, MRI) attributes, such as, comparing interpolated fine-grid images generated by this package to the object actually scanned. On the other hand, if it turns out to be the case that interpolation with higher order than linear does "see inside" the pixel structure better than linear interpolation, then interpolation can increase the accuracy of even simple algorithms. For example, if a finer grid was generated by interpolation, then even the errors of GSNAKE would be acceptable because the discretization jitter of the contour would be small in comparison to the pixels of the original data. To use another example, a segmentation program has been developed at CRS4 that uses linear interpolation to generate two-dimensional slices of three-dimensional datasets at arbitrary angles in order to use a two-dimensional segmentation technique on a slice that is tranverse to a given section of an artery. If the dataset used for this segmentation was a refined grid, interpolated from the original data, then the linear interpolation for slicing would not be a significant source of error.

For three-dimensional medical images, the step size in the z direction is often larger than the step size for the x and y directions. From one slice to the next, the contour of an artery can vary greatly near a bifurcation and bridging the gap between contours can be difficult. It would be interesting to see if reducing the step size in the z direction using interpolation can result in a set of contours that change more gradually without introducing errors such as excessive smoothing of abrupt boundaries.

Acknowledgments

This work has been partially supported by the Sardinian Regional Authorities.

References

[1] P. Thévenaz, T. Blu and M. Unser, "Interpolation Revisited," *IEEE Trans. Medical Imaging*, **vol. 19, no. 7, pp. 739-758, July 2000.**

[2] Kok F Lai Information Technology Institute, Singapore S. Chan, C.W. Ngo, E.L. Ang and K.W. Ong School of Applied Science, Nanyang Technological University, Singapore, "GSNAKE Reference Manual," Sept. 1995.