Internal Report 2012-2013-08 June 2013

# Universiteit Leiden
# Opleiding Informatica

Infinite

3D

Worlds

Mark Hoekveen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

**Abstract**

Current trends in Computer Graphics are towards larger 3D environments. Traditional methods for generating 3D surfaces impose limitations on the size of the generated environment. Methods are proposed to let a user interact with a procedurally generated virtual environment that seems infinite from the user's perspective. The proposed solution involves stitching together separate height-maps in a seamless manner to create a larg er, possibly infinite surface. We will consider joining these heightmaps using techniques normally used for 2D graphics. We consider simple blending, feather blending and Radial Basis Function(RBF) interpolation. The results are shown and compared for these different methods, both as heightmap and as 3D render.

# Contents

1 Introduction 3
1.1 Proposed solution . . . 3

2 Methods 4
2.1 Midpoint Displacement Algorithm . . . 4
2.2 Simple Blending . . . 5
2.3 Feathering . . . 5
2.4 Radial Basis Function Interpolation . . . 6
2.4.1 RBF Interpolation Basics . . . 6
2.4.2 Gaussian . . . 7
2.4.3 Inverse Multiquadratic . . . 8
2.4.4 RBF Interpolation on heightmaps . . . 8

3 Results 9
3.1 Midpoint Displacement Joining . . . 9
3.2 Blending . . . 9
3.3 Feathering . . . 10
3.4 RBF Interpolation . . . 10
3.5 Comparison . . . 11

4 Conclusion 13
4.1 Discussion . . . 14
4.2 Further work . . . 14

References 15

# 1 Introduction

The trend for modern 3D graphics applications, mainly in computer games, is towards larger worlds. More games no longer confine the player to a small space but provide a landmass the size of a small country. This constitutes a need for a simple and effective method to create large environments using relatively small amounts of storage space. Because of these storage limits, any generated virtual environment is limited in size. This limitation is so severe that it becomes very hard to render surfaces with reasonable detail for a large scale. To render the surface of a planet about the same size as earth with one sample per square meter we need $0.5 * 10^{15}$ samples. Assuming each sample is stored as a byte, the surface data would need 0.5 petabytes of storage, excluding any storage needed for texture information or placed objects. This amount is too large for any practical use in applications on today's home computers. Proposed is a solution to let a user interact with a procedurally generated virtual environment that seems infinite from the user's perspective.

## 1.1 Proposed solution

There are already several algorithms capable of generating a landscape of finite size, but these algorithms are dependent on the fact that the generated environment is finite in size. However, we can generate multiple tiles of a large but finite size and join these together as a mosaic as shown in Figure 1. The user frustrum (visible field of view) must not reach

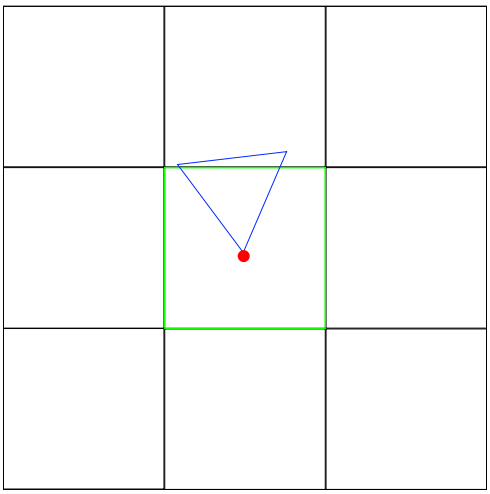


***Figure 1:*** *A 3 by 3 grid of seperate tiles. The red dot is the user, with the cone being the user's view frustrum, which must not reach beyond the boundaries of this 3 by 3 grid.*

beyond the boundaries of the current height-map view, such that the user will not be able to see the edge of it's neighbouring tiles. As the user moves outside of a tile, the tile he steps to becomes the new centre of the 3 by 3 grid and the new 3 by 3 grid will have to be calculated. The old tiles are then deleted, in order to free up memory. As these generated height-maps are separate, we can expect very visible seams. Therefore, we need to have a way to join two completely separate surfaces together to a single seamless plane as shown in Figure 2. This method imposes several conditions that both the heightmap generation and joining algorithm need to comply to. For instance, the same set of 9 heightmaps must be generated for the same position in the global heightmap grid every time. We

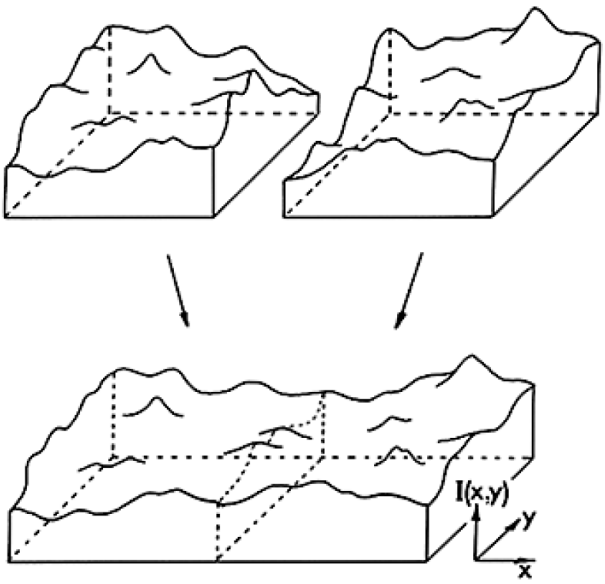



***Figure 2:*** *We can view two heightmaps as two surfaces above the (x,y) plane. We need to merge two surfaces in a seamless manner. From the paper by Peter J. Burt and Edward H. Adelson[1]*

will discuss the pros and cons of each method seperately in the next sections. This paper is mainly concerned with the seamless joining of the heightmaps in order to be able to achieve this 3 by 3 grid.
There has already been extensive research into the seamless joining of two images, mainly for use in stitching panoramic photos together [1][2]. One of the more commonly used methods is using the cut-path algorithm to find an optimal seam[3], among others. However, these methods have a limitation that makes them unsuitable for the purpose of this research. These algorithms assume that there is overlap between the images, which is not the case with heightmaps. However, the nature of heightmaps allows the application of algorithms that are unsuited for use in general seamless joining. Heightmaps do not have specific recognisable traits or features that need to be preserved, as long as the algorithm itself does not generate any artefacts. This might allow methods that are less succesful for conventional images to give better results for this application.

# 2 Methods

## 2.1 Midpoint Displacement Algorithm

As the basis for the proposed joining methods is the two-dimensional Midpoint Displacement Algorithm[4], or MDA, to generate our tiles. The algorithm starts with four randomly initialised corners for a square, which are then subdivided into four sub-squares. The newly generated middle co-ordinate is then the average of the four corners displaced with a factor dependent on the size of the square being divided, such that the larger variations occur early. The rest of the newly generated co-ordinates just get the average of their neighbours. This process is illustrated in Figure 3 and Algorithm 1. This algorithm creates a realistic looking heightmap. However, there are visible artefacts in a characteristic plus-shape due to the nature of creating the heightmap in a grid, and the algorithm has been described as flawed[5]. While there are some better algorithms available, this

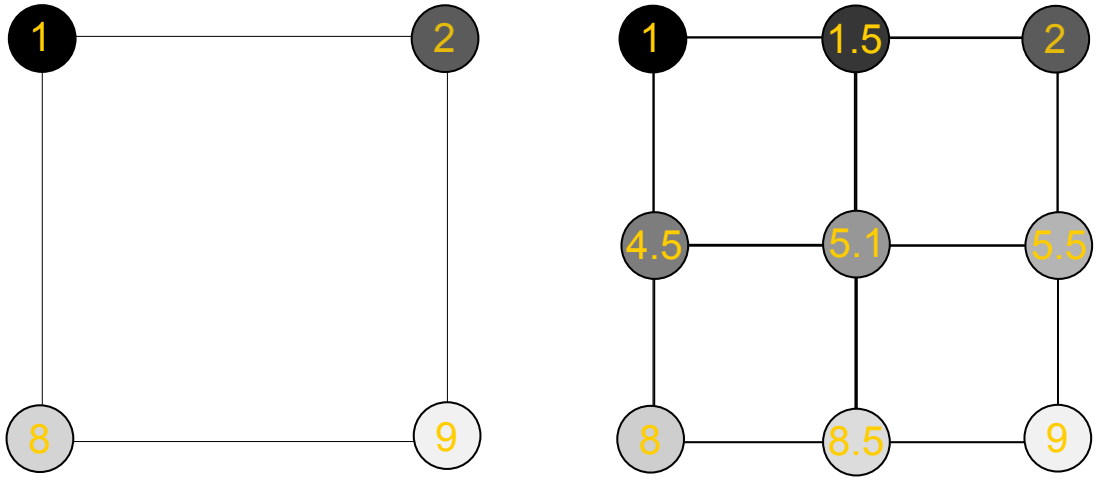



***Figure 3:*** *One step of the MDA algorithm on a square with corner values 1, 2, 8, 9. The displacement in the middle is 0.1 here. Color is used as a rough indication of the pixel value in a gray-scale heightmap.*

algorithm suffices as a quick and easy method to generate a heightmap for the purposes of this research.

## 2.2 Simple Blending

The first proposed method does not only use the described MDA, but also an extremely simplified version of it, which subdivides a square into four sub-square where it adds or subtracts a small random value to all the values there. This generates a heightmap with extremely noisy terrain and very noticable square artefacts. The generation algorithm is illustrated in Figure 4. When tiles generated by this random algorithm are placed together, the seams are not directly noticeable, if only because there are already large internal seams and artefacts in the images themselves.
We then apply a simple blending algorithm to these images in which every pixel becomes the average value of all the pixels in a square neighbourhood of a certain size. With a neighbourhood size of sufficient size, this will smooth out the noisy nature of the heightmap.

## 2.3 Feathering

Feathering an image is a technique that is both widely known and widely used in consumer software[6] and various other computer graphics applications[7]. It involves alpha-blending an edge of the image such that the outer pixels of the image are entirely invisible (i.e. opacity is zero) and increasing in opacity as they go towards the center of the image. This creates the typical feather effect that the technique is named after. When two feathered images are overlapped in their feathered parts, this results in feather blending. Feather blending creates a smooth transition between two images.
The heightmap grid is then adjusted such that the tiles overlap in their feathered edges like shown in Figure 5. Because of the overlap, this means that the grid becomes smaller with respect to a non-blended version.

**Algorithm 1** Midpoint Displacement Algorithm

**function** MDA(width, height, roughness)
  globally defined matrix $m$ of size $(width, height)$
  globally defined roughness $r$
  Initialise corners $c1, c2, c3, c4$ to a random value
  DivideGrid(0, 0, width, height, c1, c2, c3, c4)
**end function**
**function** DIVIDEGRID($x, y, w, h, c1, c2, c3, c4$)
  $m := \frac{c1+c2+c3+c4}{4}$
  **if** $w > 2$ **then**
    calculate $d$ ▷ d = displacement
    $m := m + d * r$
    $e1 := \frac{c1+c2}{2}$
    $e2 := \frac{c2+c3}{2}$
    $e3 := \frac{c3+c4}{2}$
    $e4 := \frac{c4+c1}{2}$
    DivideGrid$(x, y, \frac{w}{2}, \frac{h}{2}, c1, e1, m, e4)$
    DivideGrid$(x + \frac{w}{2}, y, \frac{w}{2}, \frac{h}{2}, e1, c2, e2, m)$
    DivideGrid$(x + \frac{w}{2}, y + \frac{h}{2}, \frac{w}{2}, \frac{h}{2}, m, e2, c3, e3)$
    DivideGrid$(x, y + \frac{h}{2}, \frac{w}{2}, \frac{h}{2}, e4, m, e3, c4)$
  **else**
    m(x, y) := m
  **end if**
**end function**

## 2.4 Radial Basis Function Interpolation

A radial basis function is a function $\phi(x, c) = \phi(||x - c||)$ where $x$ and $c$ are points in space with c called the centre. This means that the radial basis function is only dependant on the distance between the two points. A Radial Basis Function, or RBF has various applications in machine learning[8] and time series prediction[9], but a RBF can also be used for image reconstruction[10] or a function approximation. This is used in Computer Graphics to recreate smooth surfaces from a limited set of points on the surface of a 3D object[11]. Since our heightmap is a surface on the (x, y) plane, we can also use RBF Interpolation to give an approximation for our surface.

### 2.4.1 RBF Interpolation Basics

To understand the method used, it is important to understand some of the basics of RBF interpolation.

Given a set $S$ of $n$ samples on a function $f(x)$, an approximation of that function is given of the form

$$y(x) = \sum_{i=0}^{n} w_i \phi(||x - x_i||)$$

The weights $w$ can be approximated by solving a linear system $Aw = b$ where $A = [\phi(||x_i - x_j||)], i = 0..n, j = 0..n$ and $b = [v(x_i)], i = 0..n$ where $v(x)$ is the evaluation

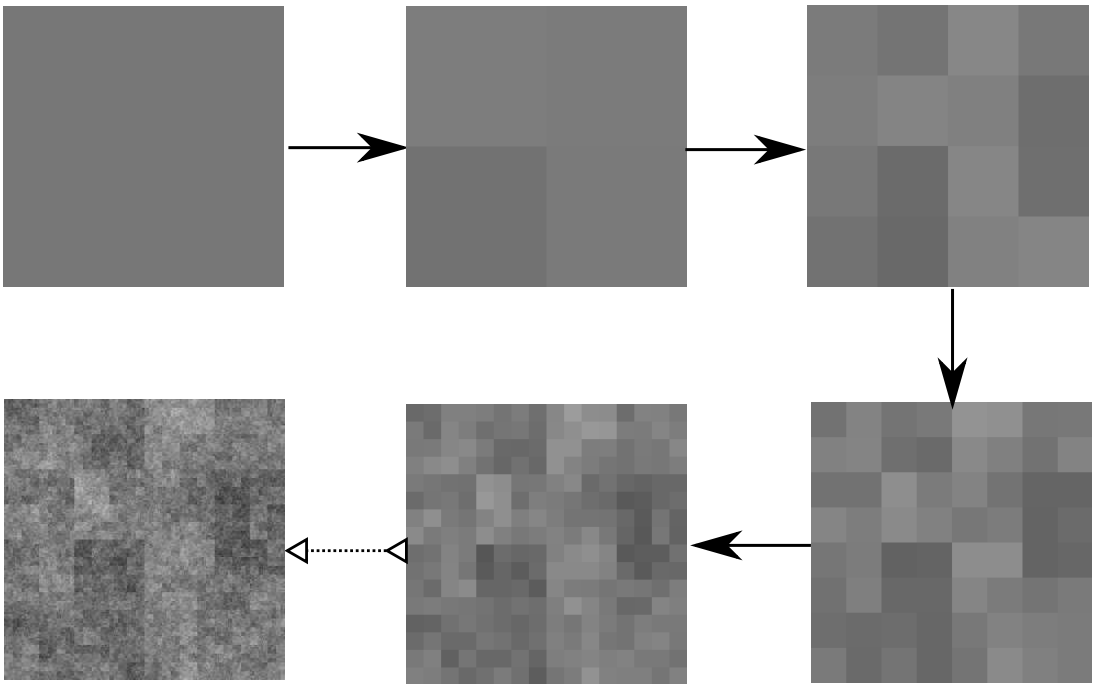


***Figure 4:*** *Like in the Midpoint Displacement Algorithm, this algorithm recursively subdivides a square, only this simply adds or subtracts a random value each time.*

function, i.e. the height of the sample.
For this project, we consider two different basis functions: the Gaussian distance and the inverse multiquadratic.

### 2.4.2 Gaussian

A Gaussian function is a function of the form

$$f(x) = ae^{-\frac{(x-b)^2}{2\sigma^2}}$$

When graphed, this gives a characteristic bell curve, where the parameters $a, b$ determine the height and position of the curve. And $\sigma$ is the standard deviation, i.e. controls the width of the curve. We will use a special case of this function where

$$a = 1, b = 0, \sigma = \frac{1}{\epsilon\sqrt{2}}$$

yielding the simplified function

$$\phi(x, c) = e^{-(\epsilon||x-c||)^2}$$

for some value of $\epsilon$ that determines the width of the curve. A smaller value for $\epsilon$ yields a wider bell. Various values for $\epsilon$ are plotted in Figure 6a. We need to choose $\epsilon$ such that the function is not yet zero for the distance between grid points, but is also not very high. As the curve quickly approaches zero, it is important to know how far away your samples are such that you can take a good value for epsilon. This also constraints this function to interpolations on a more or less evenly distributed grid.

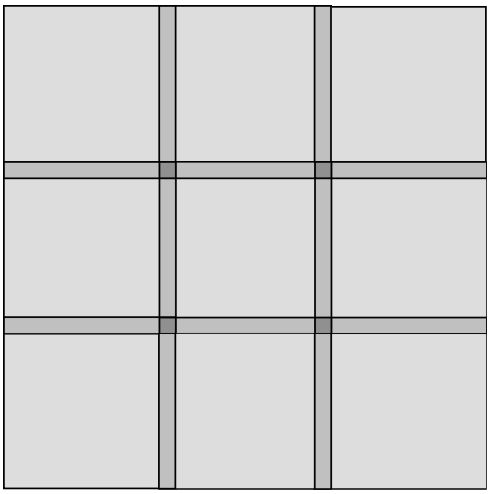
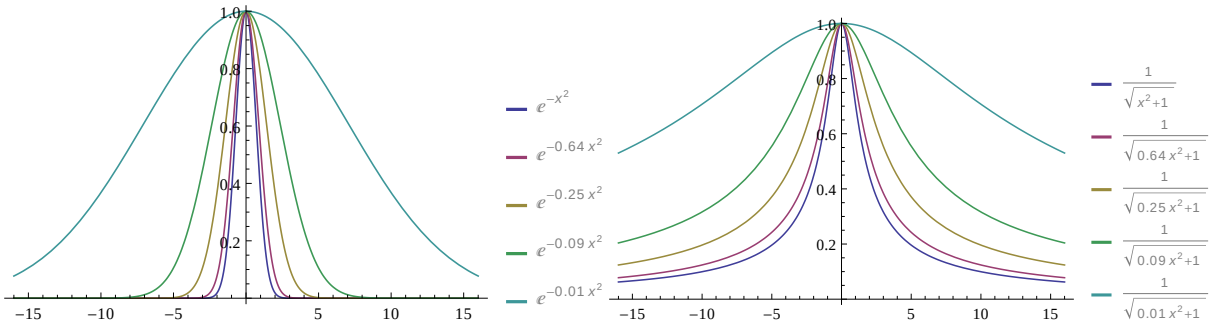


***Figure 5:*** *The nine tiles in the grid are overlapped on the parts that are feathered.*




***(a)*** *Gaussian* ***(b)*** *Inverse Multiquadratic*

***Figure 6:*** *The Gaussian and Inverse Multiquadratic Functions plotted for various values of* $\epsilon$*. Here* $x = ||x - c||$*.*

### 2.4.3 Inverse Multiquadratic

The Inverse Multiquadratic function is a function of the form:

$$\phi(r) = \frac{1}{\sqrt{1 + (\epsilon r)^2}}$$

The plot of this function resembles the Gaussian function, where $\epsilon$ controls, like in the Gaussian function, the width of the curve. However, this function nears 0 much less quickly, making it more versatile in terms of distribution of the samples. This function is plotted in Figure 6b with several different values for $\epsilon$. As visible in Figure 6b, the value of the function approaches zero much less quickly. There is still some choosing required for $\epsilon$, but the larger margin for error allows the random selection of samples from the image.

### 2.4.4 RBF Interpolation on heightmaps

Our heightmap consists of a large number of samples in the form of pixels. However, in order to get a more regular surface it is much more useful to select a number of samples from these pixels and use them to interpolate. We will use both an even method of selecting

the samples and a randomly selected set of samples. Because the inverse multiquadratic function offers more versatility in terms of distance between samples, we will use that RBF on the randomly selected set. The Gaussian function requires a more even distribution to fine-tune the value for $\epsilon$, and we will use an evenly distributed set of samples for this. An outline of the algorithm used to interpolate a heightmap from a heightmap $m$ is given in Algorithm 2.

**Algorithm 2** Radial Basis Function Interpolation Algorithm

$s \leftarrow$n samples selected from m
**for** $i = 0 \rightarrow n$ **do**
    $values(i) := v(s_i)$
    **for** $j = 0 \rightarrow n$ **do**
        $A(i, j) := \phi(||s_i - s_j||)$
    **end for**
**end for**
Solve $Aw = values$ to get $w$
**for all** points x in m **do**
    $m(x) = \sum_{i=0}^{n} w_i \phi(||x - s_i||)$
**end for**

# 3 Results

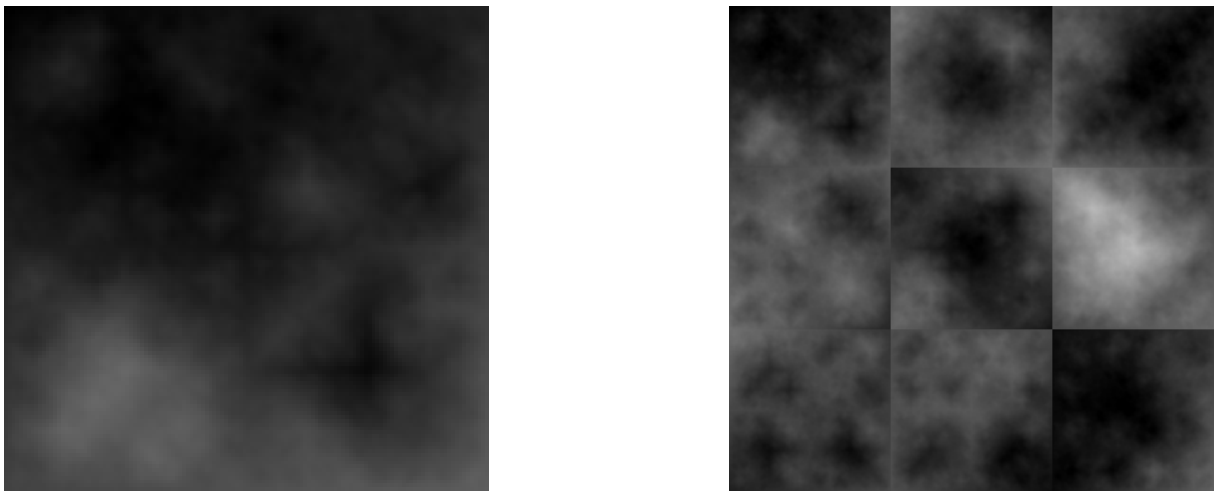
I have written a program implementing the methods described in previous sections. This program gives output in the form of a grayscale representation of the heightmap and a real-time rendered version of these heightmaps in OpenGL.

## 3.1 Midpoint Displacement Joining

At the basis of all these results are nine heightmaps generated by the previously introduced Midpoint Displacement Algorithm. Like stated before and as visible in Figure 7a, the algorithm produces visible creases in a characteristic plus-shape. Unless otherwise specified, the result in Figure 7b will be used as a basis for the seamless joining algorithms in further sections. The size of the individual tiles is 128 pixels by 128 pixels for all these examples.

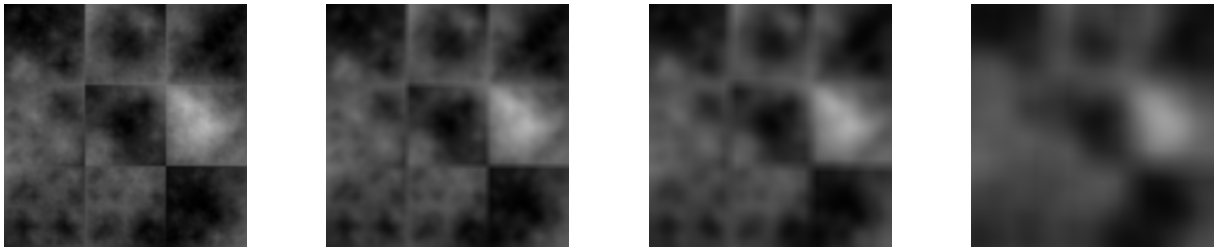
## 3.2 Blending

Results for the blending algorithm are shown in Figures 8 and 9. As visible from those images, the visible seaming decreases with higher neighbourhood sizes, but so does the amount of detail present. Removal of detail is a positive thing in a very noisy heightmap such as in Figure 9, but is not desirable when the original heightmap is more realistic in nature such as in Figure 8. Increasing the size of the neighbourhood even further yields an even more seamless image, but eventually results in an extremely smooth gradient.



***(a)*** *A single tile* ***(b)*** *Nine joined tiles*

***Figure 7:*** *Heightmaps generated with the Midpoint Displacement Algorithm*



***(a)*** *nbh = 2* ***(b)*** *nbh = 5* ***(c)*** *nbh = 10* ***(d)*** *nbh = 25*

***Figure 8:*** *Blending algorithm applied to Figure 7b, with various neighbourhood sizes.*

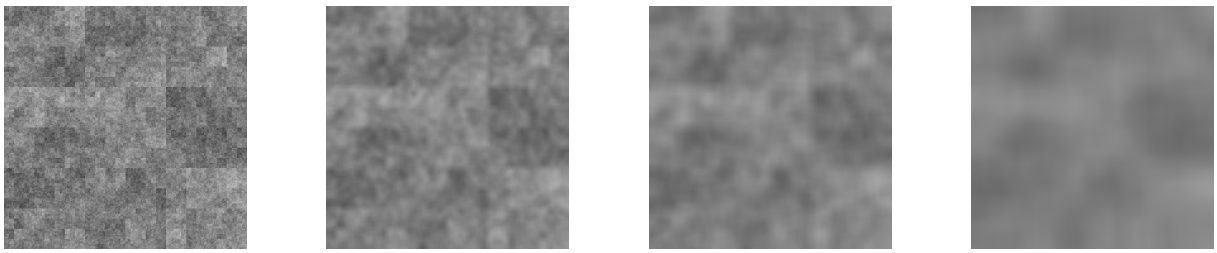
## 3.3 Feathering

The results for featherblending are shown in Figure 10. As visible from these images, when using a sufficiently large feathering size $f$ the seams are no longer visible. Smaller values for $f$ result in an image with more detail but also more visible seams. As a result of the feathering there exists an edge around the joined image that is feathered out to black. This edge should be cropped, or at least not in reach of the user view frustrum.

## 3.4 RBF Interpolation

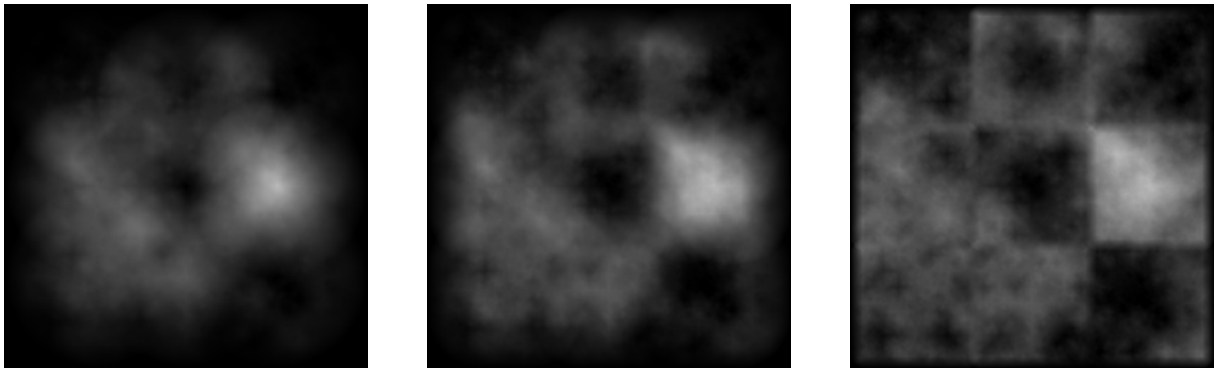
The results for RBF Interpolation are shown in Figure 11. For both the Gaussian function and the inverse multiquadratic, the chosen samples and the interpolated result are shown. For the choice of epsilon for the Gaussian function the function is rewritten such that for the maximal distance $d$ between two neighbouring grid points, the value of the function would be $c$ such that we get the formula

$$e^{-(\epsilon d)^2} = c$$



***(a)*** *nbh = 0* ***(b)*** *nbh = 5* ***(c)*** *nbh = 10* ***(d)*** *nbh = 25*

***Figure 9:*** *Blending algorithm applied to a more noisy heightmap, with various neighbourhood sizes.*



***(a)*** $f = \frac{128}{2}$ ***(b)*** $f = \frac{128}{3}$ ***(c)*** $f = \frac{128}{10}$

***Figure 10:*** *Featherjoining demonstrated on Figure 7b with various feather sizes f*

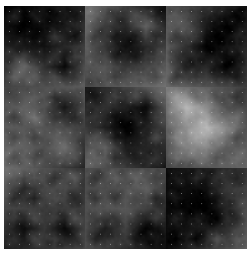
where $d$ is a constant, so we can derive

$$\epsilon = \sqrt{\frac{ln(c)}{-(d)^2}}$$

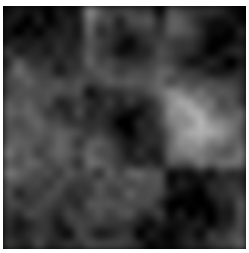
A reasonable choice for $c$ is 0.2 or 0.1. The choice for $\epsilon$ for the inverse multiquadratic function requires less fine tuning.
From Figure 11 is visible that both the Gaussian and inverse multiquadratic functions yield results that reduce seams, but also reduce noise/detail considerably. The very nature of these radial basis functions enforces a smooth surface and the RBF is most suitable to remove noisy elements from surfaces[12], even when that is sometimes unwanted.

## 3.5 Comparison

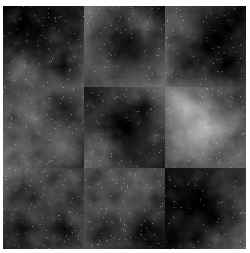
While the heightmap offers a nice data structure to work with, allowing the application of algorithms developed for 2D graphics to our surface, and also allowing human eyes to judge the surface in a quick and easy way, the best way to judge a surface is to see its rendered counterpart. The renders shown in this section are screencaps taken from a real-time 3D application using only extremely simple shading and lighting. As an example the basic joined version as shown in Figure 7b is rendered in Figure 12.
Figure 13a is the rendered result of a feathered heightmap. The outer feathered edge is quite visible here. The outer edge will have to be cropped. As the feathered image is



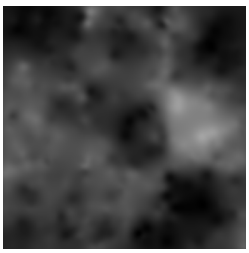
***(a)*** *Evenly selected samples*



***(b)*** *Gaussian interpolated*



***(c)*** *Randomly selected samples*



***(d)*** *Inverse Multiquadratic interpolated*

***Figure 11:*** *RBF Interpolation demonstrated. For both selection methods, 576 samples were selected from 16384 pixels indicated as white pixels in these images.*

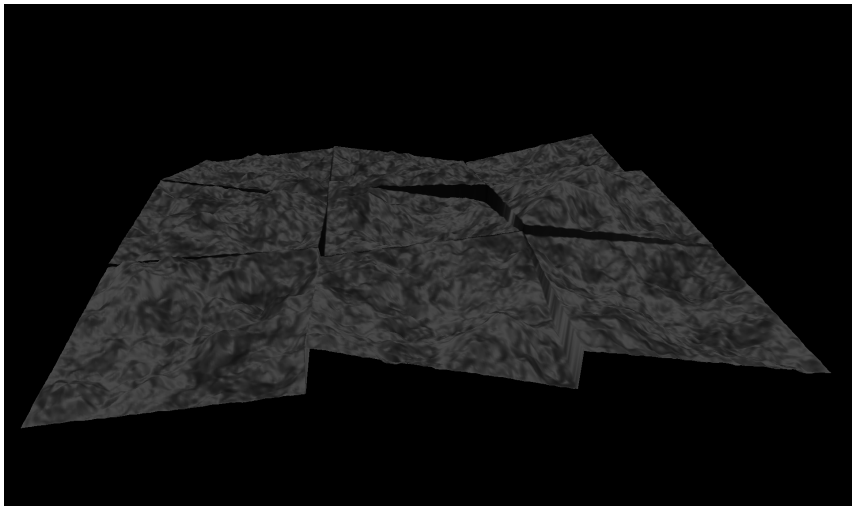
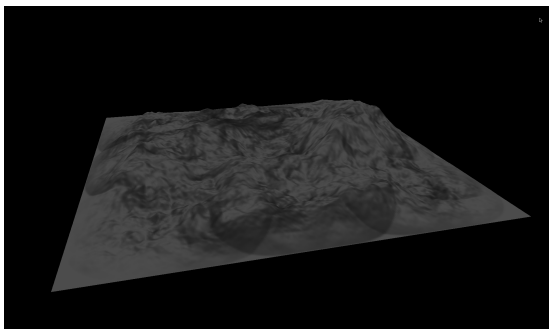
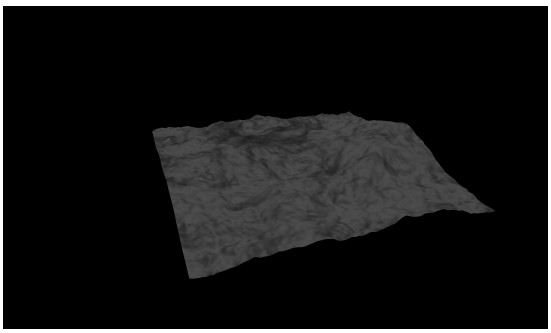


***Figure 12:*** *A simple render of Figure 7b*

already smaller in size, due to the overlapping of the tiles, this will yield an even smaller surface area. Something that will have to be compensated by using a larger tile size. Visible in the render more so than in the heightmap, we can see that there is only a very slight amount of loss of detail in the feathered result.
The simple blended rendered heightmaps look different from the heightmaps themselves, shown in Figure 14. The shading reveals a great amount of artefacts. In Figure 14b there is a texture that almost reminds of a canvas that was not obvious from just the heightmap. In Figure 14a we can see that the edges between tiles have gotten bigger, but are still very visible. While the result of 14b is not very poor, the result still yields a seemingly low resolution image as a result of the blending. Like with the feathered result this can be corrected by choosing a larger size for the tiles.
Shown in Figure 15 are the rendered results for the RBF interpolation. As visible in both Figure 15a and 15b, the result is much smoother than with other methods. While this smoothness is not always desired, neither is it a neccesarily bad quality. The landscape looks like a silk blanket was pulled over a normally rugged landscape. This smoothness could be mitigated by applying any one of a number of noise filters.

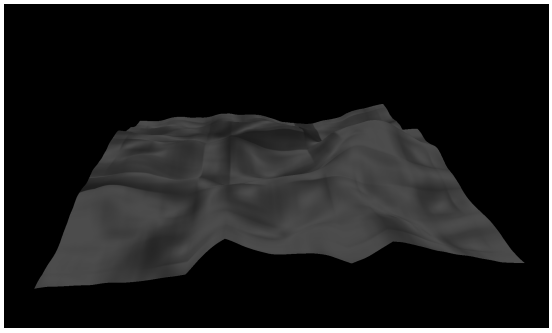


***(a)*** *Uncropped*

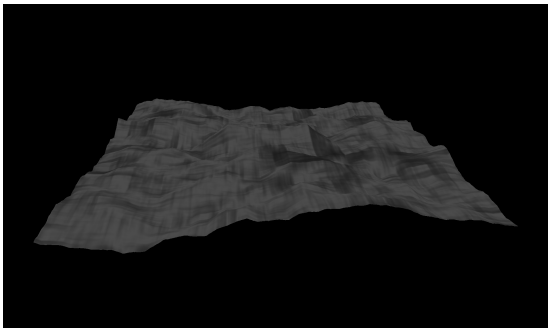


***(b)*** *Cropped*

***Figure 13:*** *A render of 10b, both 'as is' and cropped.*



***(a)*** *MDA*



***(b)*** *Simplified MDA*

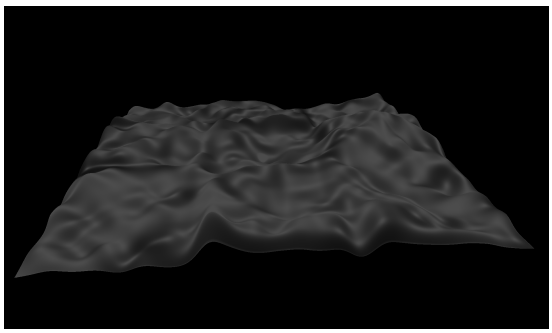
***Figure 14:*** *Renders of heightmaps generated with simple blending with a neighbourhoodsize of* 15

# 4 Conclusion

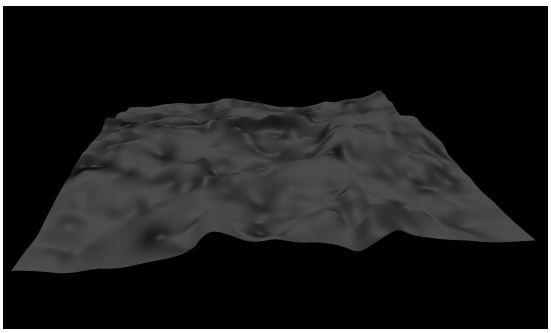
The proposed algorithms yield a relatively easy, but by no means perfect way to create a surface that is infinite from the perspective of the user. A selection of algorithms ranging from relatively simple to slightly mathematically complex were proposed. While there is extensive research done in the area of merging two or more images, these methods often require the images to have some overlap. The methods for the joining of heightmaps involve merging images that have no overlap. However, since we don't need to use generic images or photographs, we can apply methods that would ruin the detail in a typical photo but work quite well for heightmaps.

As visible from the previous section, the results for each method are very different. The feathering method has the advantage of maintaining the noisy nature of the original tiles, making it useful for direct implementation in a 3D application. The results for the blending and RBF interpolation methods require some more post-processing if they are to be used as a realistic heightmap for a more rugged environment. Their smoother surfaces might make them more suitable for heightmaps in a sandy desert environment or a smoothed underwater landscape.

It is also worth noting that artefacts that are visible from the perspective of a heightmap are not directly visible in a rendered version. An example here is the plus-shaped artefact that appears as a result of the MDA generation. While several plus shaped artefacts are



***(a)*** *Gaussian, Figure 11b*



***(b)*** *Inverse Multiquadratic. Figure 11d*

***Figure 15:*** *Renders of heightmaps using RBF interpolation as shown in Figure 11*

visible on the heightmap in Figure 7b, these are not immediately visible when viewing the rendered result in Figure 12. This underlines the importance of viewing the results as a render as well as a heightmap.

## 4.1 Discussion

The original purpose of the proposed algorithms is for use in real-time 3D applications. The idea is to generate these tiles and merge them seamlessly 'on-the-fly'. Both the feathering method and the blending method are well-suited for this, but the RBF interpolation algorithm is not quite so fast. The examples shown were of size 128 by 128 pixels, but heightmaps for large environments might well be several times larger. Because of the way the RBF interpolation is implemented for $n$ samples the required matrix to solve will require $n * n$ elements. And every pixel that needs to be interpolated will require a summation of $n$ expressions. The storage space and computational requirements quickly become excessive with larger sample sizes.
The other problem with defining an environment in this manner lies with the restriction it poses on world designers. Because the landscape is not pre-defined, any man-made structures that are desired for an application will also have to be procedurally placed. If this environment has to be interactive there will also need to be a system that stores changes users make to the environment.

## 4.2 Further work

As previously stated, the RBF interpolation is not fast enough for real-time application in the implementation I have given. So an obvious step would be research into optimisation of this method.
The given set of algorithms is also in no way complete. There exists a number of interpolation methods that might seem less suitable for heightmaps, such as bilinear interpolation, that are still worth researching.

# References


[1] P. Burt and E. Adelson. A multiresolution spline with application to image mosaics. In *ACM Transactions on Graphics*, volume 2, pages 217–236, New York, NY, USA, 1983. ACM.

[2] Y. Xiong and K. Pulli. Fast and High-Quality Image Blending on Mobile Phones. In *2010 7th IEEE Consumer Communications and Networking Conference*, pages 1–5. IEEE, January 2010.

[3] N. Gracias, M. Mahoor, S. Negahdaripour, and A. Gleason. Fast image blending using watersheds and graph cuts. In *Image and Vision Computing*, volume 27, pages 597–607, Newton, MA, USA, April 2009. Butterworth-Heinemann.

[4] A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. In *Communications of the ACM*, volume 25, pages 371–384, New York, NY, USA, June 1982. ACM Press.

[5] G. Miller. The definition and rendering of terrain maps. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 39–48, New York, NY, USA, 1986. ACM.

[6] A. Burton. Building panoramic images in the gimp. In *Linux Journal*, volume 2004, pages 8–, Houston, TX, March 2004. Belltown Media.

[7] R. Szeliski. Image alignment and stitching: a tutorial. In *Foundations and Trends in Computer Graphics and Vision*, volume 2, pages 1–104, Hanover, MA, USA, January 2006. Now Publishers Inc.

[8] S. Chen, C. F.N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. In *IEEE Transactions on Neural Networks*, volume 2, pages 302–309, Piscataway, NJ, USA, March 1991. IEEE Press.

[9] R. Jones, Y. Lee, C. Barnes, G. Flake, K. Lee, P. Lewis, and S. Qian. Function approximation and time series prediction with neural networks. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 649–665. IEEE, 1990.

[10] V. Skala. Radial basis functions interpolation and applications: an incremental approach. In *Proceedings of the 4th international conference on Applied mathematics, simulation, modelling*, ASM'10, pages 209–213, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[11] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[12] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth surface reconstruction from noisy range data. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive*

*techniques in Australasia and South East Asia*, pages 119–126, New York, NY, USA, February 2003. ACM.