

[1.]

P4 /

Introduction Project 4

My work has taken me so far to this point, but I do not want to take the research done in Project 3 further in this regard, to free the research from focusing on only one media content. Having tested the possibilities of a designed media content in three different contexts, the potential, however, seems convincing, but this would have to be taken further in research beyond the scope of that which I have defined as my own. In Project 4, a number of exemplary codes were provided which created a dynamic space. The outcome of these codes are neither a form of art or communication there function is testing my researched system further. In comparison to the designed media content presented in the previous Project 3 I do not have the intention to create an art work or a form of communication with these codes. I want to hand over the design of the media content to other designers or artists in order to focus on curating supplied media content. To do so, I have written these sets of different codes in VB script/Rhino script to possibly allow artists or designers to work with the *Spatial Dynamic Media System* and explore new possibilities. My intention at this point is the testing of the system with written codes.

To cover different ranges of possible media content the following codes were offered:

- A code to create a moving surface delivered through media content generated from digital input taken from the environment. With this proposed code the research offers artists the opportunity to use sensors as a method of capturing environment activity. Sensors here imply all kinds of recording devices that can measure temperature, wind speed, movement and humidity among others.ⁱ
- A zone based on an image or a movie clip. Based on the colour information stored in an image, a code could use this information to generate a moving surface based on the movement within an array of images or a movie clip. Every image has an array of pixels in X and Y

ⁱ The function of sensors and the possibilities of sensors in architecture are based on a research using *Smart Materials and Technologies* by Michelle Addington and Daniel Schodek

direction that define its size and store colour information. The size of the image could be used to define the size of the zone in X and Y direction. The Z direction would then be driven by the colour. In the background discussion I talked about creating an amalgam of form and image, the code has been written with this interest in mind.

- A surface based on a scanned 3D form. Similar to the first step of the designed media content described in chapter [3.4.3.] a surface can be scanned in, such as a face. This scanned in surface is then displayed with light points, each light point being a different colour depending on its position in space. The previous Project 3 had a designed media content based on a 3D scan. The code written in this test wants to explore if it is possible to create a light point surface based on a scanned 3D form.

The general outcome of the tests conducted so far shows that it is possible to ‘inject’ information into the system to generate a moving surface in real-time. They can be used as tools for artists and designers utilising the *Spatial Dynamic Media System*, as described above, regarding tests such as the visualisation of information or the amalgam of form and image. In addition, the tests also proved propositions made in Chapter 3. Here the research discusses a multilayered surface and a decay function for a surface and the significance of the possibilities for architecture, as per the theoretical discussion in the background discussion, but the tests in this Chapter 4 **Project 4: Effect and use – Working as a curator for different media content** have demonstrated that it is possible to create such surfaces. Therefore, test series 5 and test series 6 investigates anomalies of surface in an intangible zone – the possibility to create a multilayered surface and the option of applying a decay function.

- To create a multilayered surface two types of discrete data stored in two remote Excel™ sheets are used and played simultaneously. These two data sheets create two surfaces, coloured differently. When both surfaces overlap their colour changes to a third colour. Here two surfaces can be perceived at one time, overlapping but still visible due to their incorporeal nature, and furthermore the two surfaces generate a third form or surface depending on the input of the two original surfaces. This has been

discussed as a possible significance in the theoretical framework and now been tested.

- The Plug-in for decay function also demonstrates a proposition made in the theoretical framework. A code has been written which will simulate a surface decay by giving single LEDs a brightness that varies from 0% till 100%. This creates the outcome of an after-effect of the façade that has just been, or will be in the future when the zone modifies itself through movement from its present state.

All these scripts have been loaded into and presented via Rhino.

The presented scripts can be seen as a first step to what is possible by creating a system such as the *Spatial Dynamic Media System*, and how artists and designers can be involved in the design of a surface. All the codes function as a tool set which can be developed further via outsourced ideas and media content.

[2.]
P4 /
Test Series I:
Creating a surface based
on an Excel datasheet
input

[2.1]
Introduction

To give artists a first tool to work with, a translation of data from an Excel sheet into a surface defined by light points should be scripted. The idea would be to create a program where the artist could use any kind of information translated into an Excel sheet and have this information displayed on the façade. Due to having no explicit data to be shown and not wanting to provide any meaningful content in this exercise, Excel will provide results with random data.

The RAND function in Excel “Returns an evenly distributed random real number greater than or equal to 0 and less than 1. A new random real number is returned every time the worksheet is calculated.”ⁱ Based on this function a surface of no meaning can be produced as a test of the program and to analyse the outcome.

[2.2]
Providing a script
to translate an Excel
data sheet into a
surface

[2.2.1]
Script

The following script has been generated to achieve the translation of Excel data into a surface:

```
' CREATES A MATRIX OF POINTS BASED ON AN EXCEL DATA SHEET  
  
' Script for Rhino 3.0  
' Written by M .Hank Haeusler 13.01.2007  
' Matthias.Haeusler@ems.rmit.edu.au  
  
' DESCRIPTION: creates a matrix of points based on an image txt file  
' STATUS: working  
' TO DO: For this script to run, Rhino3 needs to be running first.  
  
' *****  
  
Public Sub ImportPoints()  
' *****  
' start of rhino connection  
    Dim RhinoApp As Object  
    Set RhinoApp = CreateObject("Rhino3. Application")
```

ⁱ Microsoft Excel Help text for RAND

```

    Dim RhinoScript As Object
    Set RhinoScript = RhinoApp.GetScriptObject()
    Dim arrObjObjects
' end of rhino connection
' *****

' *****

' SURFACE VARIABLES
    Uvalue = 10 'surface height - number of rows
    Vvalue = 10 'surface length - number of columns
    'arrCount = Array(Uvalue, Vvalue)
    'arrDegree = Array(2, 1)
' SPHERE VARIABLES
    myradius = 10
' END VARIABLES

' Loop for each sheet in the Excel book.
    Dim Loop1(99) 'has to equal 0 + Uvalue*Vvalue

' For j = 1 To Sheets.Count
    For j = 1 To 40

' EnableRedraw - switch of screen to start / finish script
        RhinoScript.EnableRedraw (False)

' Removes all objects.
        arrObjObjects = RhinoScript.AllObjects()
        If IsArray(arrObjObjects) Then
            RhinoScript.DeleteObjects (arrObjObjects)
        End If

' Loops to get XYZ from Excel spreadsheet
        For i = 0 To UBound(Loop1)
            pointX = Sheets(j).Cells(i + 1, 1).Value
            pointY = Sheets(j).Cells(i + 1, 2).Value
            pointZ = Sheets(j).Cells(i + 1, 7).Value
            Loop1(i) = Array(pointX, pointY, pointZ)

' Create a point
            ' RhinoScript.addpoint Array(pointX, pointY, pointZ) SWITCHED
OFF

                Next

' creating a surface of points from Excel
            'mySrf = RhinoScript.AddSrfPtGrid(arrCount, Loop1,
arrDegree)SWITCHED OFF

' creating sphere at each point with a function
            myspheres Loop1, myradius

' EnableRedraw - switch of screen to start / finish script
            RhinoScript.EnableRedraw (True)

                Next

' Tells that script is finished
            MsgBox ("Script Finish")

End Sub

' *****
Function myspheres(centres, radius)
' Function for Rhino 3.0
' Written by M. Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

```

```

'DESCRIPTION: creates a sphere
'STATUS: working
'TO DO: add parameters required in the () out of sub main to run
'*****
' start of rhino connection
Dim RhinoApp As Object
Set RhinoApp = CreateObject("Rhino3. Application")
Dim RhinoScript As Object
Set RhinoScript = RhinoApp.GetScriptObject()
RhinoApp.Visible = True
'end of rhino connection
'*****
' Loop for each point.
For j = 0 To UBound(centres)
myspheroid = RhinoScript.AddSphere(centres(j), radius)
myred = centres(j)(2)

'Variables for colour arrangement
Dim mycolourzero, mycolour02, mycolour03, mycolour04,
mycolour05
Dim mycolour06, mycolour07, mycolour08, mycolour09, mycolour10
Dim TempColor
Dim limit1, limit2, limit3, limit4, limit5, limit6

'Colour definition for 10 different rows
mycolourzero = RGB(0, 0, 0)
mycolour01 = RGB(195, 32, 217)
mycolour02 = RGB(0, 0, 255)
mycolour03 = RGB(0, 132, 239)
mycolour04 = RGB(0, 240, 144)
mycolour05 = RGB(167, 252, 0)
mycolour06 = RGB(238, 246, 0)
mycolour07 = RGB(240, 202, 0)
mycolour08 = RGB(229, 114, 0)
mycolour09 = RGB(239, 41, 0)
mycolour10 = RGB(239, 59, 59)

'gets Z value from coordinate of sphere
tempcoord = centres(j)
tempz = tempcoord(2)

If (tempz = 0) Then RhinoScript.ObjectColor myspheroid,
mycolourzero
If (tempz = 100) Then RhinoScript.ObjectColor myspheroid,
mycolour01
If (tempz = 200) Then RhinoScript.ObjectColor myspheroid,
mycolour02
If (tempz = 300) Then RhinoScript.ObjectColor myspheroid,
mycolour03
If (tempz = 400) Then RhinoScript.ObjectColor myspheroid,
mycolour04
If (tempz = 500) Then RhinoScript.ObjectColor myspheroid,
mycolour05
If (tempz = 600) Then RhinoScript.ObjectColor myspheroid,
mycolour06
If (tempz = 700) Then RhinoScript.ObjectColor myspheroid,
mycolour07
If (tempz = 800) Then RhinoScript.ObjectColor myspheroid,
mycolour08
If (tempz = 900) Then RhinoScript.ObjectColor myspheroid,
mycolour09
If (tempz = 1000) Then RhinoScript.ObjectColor myspheroid,
mycolour10
Next

```

End Functi on

Certain lines have been switched off, having been written in the process of creating the script in helping to work with a methodology and then switched on after reaching the next.

These are the figures that have been used based on the random function explained earlier. The first column stands for the value generating the X coordinate, the second row for the value generating the Y coordinate and the third one the Z value. To create a surface as it could be perceived when using the system the values in column 3 have to be rounded due to having no position in a Z direction, for example a value of 755 will be rounded to 800, a figure that can be displayed in the 3D grid.

Row defining position Z	X-Value	Y-Value	Z-value	Rounded
	0	0	755	800
	100	0	184	200
	200	0	829	800
	300	0	673	700
	400	0	228	200
	500	0	164	200
	600	0	797	800
	700	0	239	200
	800	0	954	1000
	900	0	673	700
	0	100	259	300
	100	100	430	400
	200	100	808	800
	300	100	796	800
	400	100	602	600
			

This data continues until a surface with the dimensions 10 light points in X direction, 4 light points in Y direction and 10 light points in Z direction could be generated.

[2.2.2]
Rendering of surface

When running this script the following surface appears (**Fig. 1 - Fig. 3: Images of the visualisation of script**).

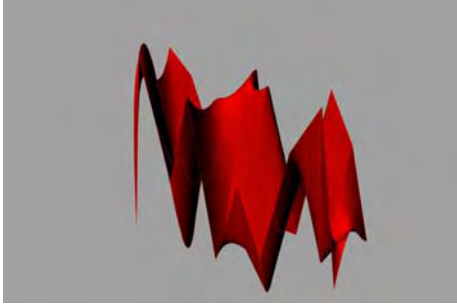


Fig. 1: Images of the surface created by the script as a first step



Fig. 2: This surface then gets translated into light points only



Fig. 3: Light points then gets displayed in different colours regarding on their position on the Z axis

One can see that picture 1 creates a highly irregular surface. Due to the random function of Excel it is difficult to see more than an assembly of light points in picture 2. Since these images are only for visualisation of the script, this does not matter.

[2.3] Testing the script with an input

[2.3.1] Water level of water reservoirs Melbourne

A collection of data for testing the script has been taken from the water levels of the dams around Melbourne.

“Melbourne gets its water supply by 9 different catchments located around Melbourne. Most of the catchments are located high up in the Yarra Ranges, north east of Melbourne. Traditionally these reservoirs were built to meet increasing demand for water, spurred on by population growth, dry spells and the inevitable drought. The last and largest of Melbourne's water storages, the Thomson Reservoir, was completed in 1984.”¹

The water levels of these dams are published daily on the homepage of Melbourne Water and the local newspaper. The data used for this research has been taken from the homepage of Melbourne Waterⁱ, a homepage which details the water levels and several actions to preserve water and the water system in general. The water levels in the archives have been stored weekly and the levels from 5. January 2006 – 12 October 2006 have been used to create 40 sheets in Excel as an input for the script.

An example of how this data has been delivered is illustrated in the image below which represents the water level at the 24. January 2007. The Data sheet is attached in Chapter 6 Appendix: Appendix 2 –Tables.

One can see all 9 Reservoirs (Thomson, Upper Valley, O'Shannassy, Maroondah, Sugarloaf, Yan Yean, Greenvale, Silvan, and Cardinia) and the total water level. The percentage in % of these 10 figures has been used to run the script.

[2.3.2] Rendering of surface

The following images illustrate the outcome of the script test when applying the water level input. All images have been taken straight from Rhino by taking screenshots while the script was running.

ⁱ Information from <http://www.melbournewater.com.au/>

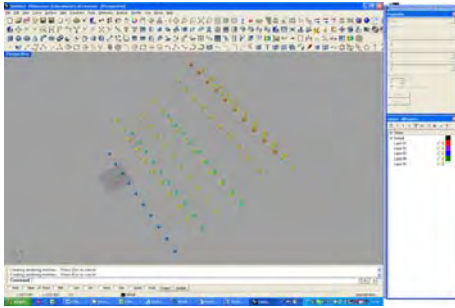


Fig. 4: Screenshot in Rhino which illustrates the environment the scripts is running at

The Image sequence P_2.3.2.B – E shows four zone arrangements as a result of the input data from a certain week out of the 40 weeks possible.

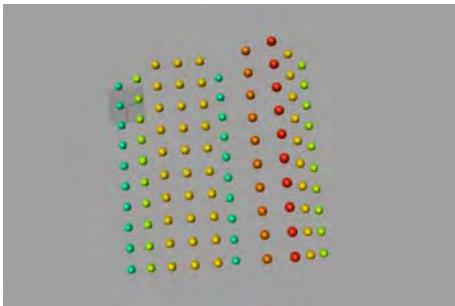


Fig. 5: Screenshot in Rhino of representation of information of water level at week 7

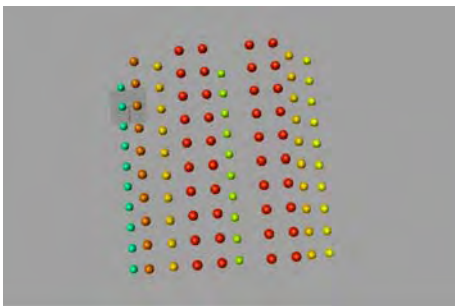


Fig. 6: Screenshot in Rhino, shows water level in week 18

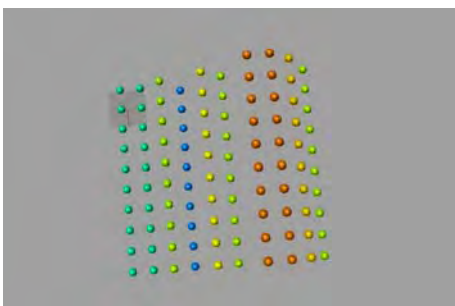


Fig. 7: Screenshot in Rhino in week 27

[3.]
P4 /

Test Series II:
Creating a surface
based on a movie clip

[3.1]
Introduction

[3.1.1]
Research question

The test should investigate another possibility in how media content could be presented. Here my research wants to offer the possibility for designers or artists to translate the movement represented in a movie into the movement of a surface. This translation should not be seen as a direct translation of the moving 2D image, ie. a person walking from left to right on a screen into the same image but in 3D, but as a translation of the movement of pixels by their change of colour - colour information into a spatial movement.

Each moment in a movie, a certain pixel has different colour information. If pixel 'A' at position $X = 0$ and $Y = 0$ has at time $T = 0$ sec. the colour $RGB = 0, 0, 0$ for black and the movie has 25 frames per second it would be possible that the RGB colour information would change 25 times per second. This movement should be expressed by a spatial movement with the use of my researched system.

At present video artists are exploring possibilities such as creating different atmospheres within a movie when using different lights or colours, but so far none have made a movie considering how the effects of colours, light, movement and narrative will influence a surface. This is what my research wants to provide to artists when being the curator for different media contents.

[3.2]
Tests with function
Heightfield from
Image

[3.2.1]
Introduction

The test is based on a function offered by the modelling software RHINO where one has the possibility to import an image into Rhino and alter the 2D image into a 3D surface. What the function does is analyse the colour of the image and give each colour a certain depth. The produced surface is therefore a result of what colours have been used in the image and how they have been arranged. One has the option to import *.bmp, *.tga, *.jpg, *.pcx, *.png, *.tif, *.tiff files into the function.

When following the program it first asks for the first corner and the size. It is therefore possible to define the size of the image, which is important because that will reflect later on the size of the screen available.

After defining the size the program asks for information about the number of sample points and height.

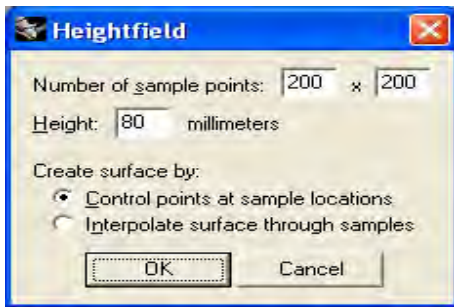


Fig. 8: Program interface Rhino to create Heightfield function

The number of sample points defines the angle from one level to another, and the higher the number the closer the angle gets to 90°, as better one pixel or colour information could get separated in 3D from another.

The height enables defining the depth of the surface, ie. when choosing 80 mm the surface will have a depth of 80 mm, and through this function the depth of the surface created with the LED sticks could be defined, as well as the number of colours existing in the movie.

If for example each single frame of a movie has 8 different colours per image, then each colour will stand for one depth level, so the total height of the surface will be a multiplication of 8 and the surface will have 8 depth level, with each depth level displayed by one LED within the LED stick.

[3.2.2]
**Foundation studies
with Heightfield
function I**

To gain a better understanding of the Heightfield function and to optimise it for the translation of a moving image to a moving surface the following tests were conducted.

Firstly a white square was imported to Rhino with the Heightfield function to prove if an image with only one colour would create an even surface. This did happen - when importing the white square one could create a flat surface. See images below.



Fig. 9: White square, which will be altered in Rhino by the Heightfield function

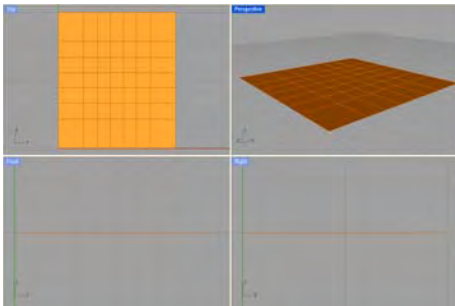


Fig. 10: Result white square imported to Rhino altered with Heightfield function

As a next step a black and white image was imported, and here the number of sample points was tested. One could achieve the best results with a number of 200 sample points, which guaranteed, as mentioned earlier, the 90° angle between the surfaces.

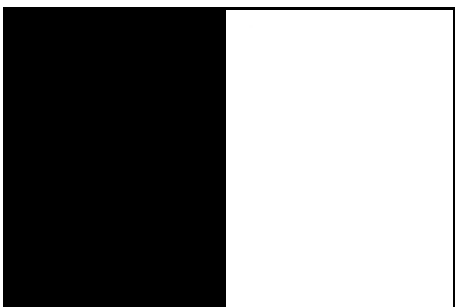


Fig. 11: Black and white square, which will be altered in Rhino by the Heightfield function

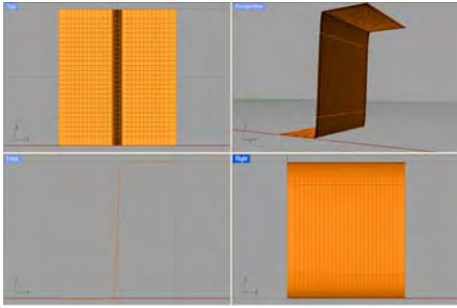


Fig. 12: Result Black And White Square imported to Rhino, altered with Heightfield function

To test it further the square was divided into four pieces.

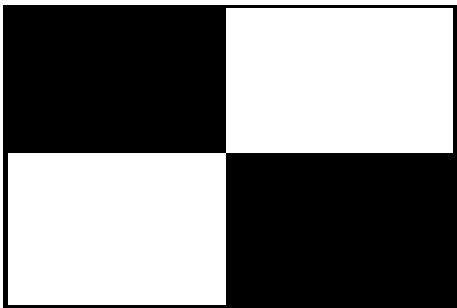


Fig. 13: Black and white square four pieces, the base for the next test of the Heightfield function

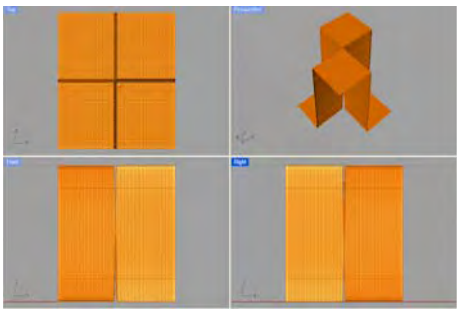


Fig. 14: Result black and white square four pieces imported to Rhino

In the next step the two colours black and white are extended to four colours when two grey scales were also used. The two new colours were grey 50% and grey 75%.

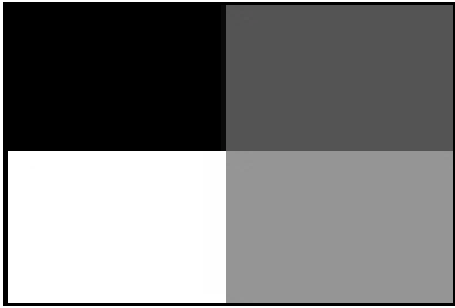


Fig. 15: Four grey gradient square four pieces, to test how the Heightfield function reacts to different grey gradients

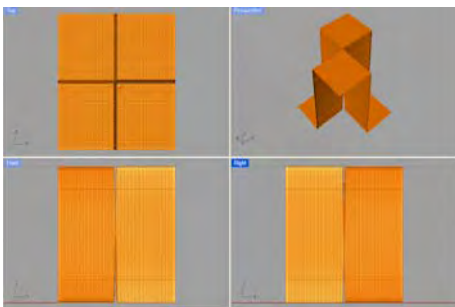


Fig. 16: Four grey gradients square imported to Rhino creating four different squares with a certain height level

The tests proved that different colours will create different heights. If one could see each of the four grey tones in the pictured square before as one pixel and each pixel produced by one LED stick one could create a surface with four different heights.

[3.2.3]
**Foundation studies
 with Photoshop**

Adobe programs are commonly used for working with images. When having a movie clip of a certain length this movie can be altered to single images. If the movie has for example 25 frames per second, each second in the movie will result in 25 images, and the translation of the movie into single images could be done in Adobe After Effects or similar video editing programs. Each image could be then altered in Photoshop from the original image to an image suitable for my research purpose.

What steps have to be done in Photoshop?

Firstly, the image tests with Heightfield from Image have shown that it is possible to create different heights from black and white images; therefore a colour image should be altered into a greyscale image.



Fig. 17: Test image step 01 in Photoshop to achieve base for Heightfield function



Fig. 18: Test image step 02

After altering the image to a greyscale image, the resolution has to be altered in regards to the size of the spatial dynamic media system. If for example the screen would be 6m*4m one would have 60 * 40 pixels, due to having 10 * 10 pixel per square metre. The test picture should therefore be altered to a resolution of 100 pixels in width and 66 pixels in height.



Fig. 19: Test image step 03

The image then would still have an undefined number of colours, but each colour would create in the Heightfield from Image function a position in the Z – plane,

therefore a specific number of colours have to be defined. When saving the image as 'save for web' one can define how many colours one will use. The range here is from 2, 4, 8, 16, 32, 64, 128, 256 possible colours. For Test image step 04 a range of 8 different colours will be used. The image would therefore have a depth of 8 light points on the LED stick.



Fig. 20: Test image step 04 where the image only contains 8 different grey gradients

When repeating these four steps for each single image a movie could be created which could be later translated to be displayed on a spatial dynamic media system screen with the dimensions of 100 pixel by 66 pixel and a depth of 8 pixel or, in metres, 10m * 6.6m * 0.8m. These dimensions have been chosen only for demonstration purposes and could be altered for all other dimensions as well depending on the existing dimensions of the spatial dynamic media system.

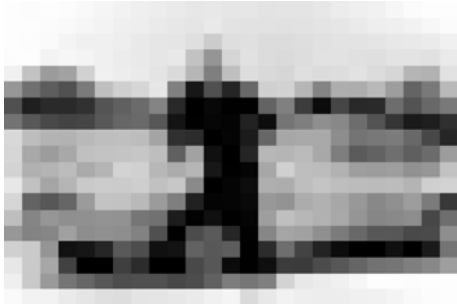
[3.2.4] Scripts to alter a surface to a light point grid

Based on the previous tests, a script should test what kind of result could be achieved when importing an image into rhino with the Heightfield from Image function. The resulting surface should then be divided by a point grid system; depending on their Z-value location each point on the surface should be stored in a separate layer. For a better visualisation each point should be changed into a sphere with different layer colours.

The image below has been altered from its original resolution and colour to an 8 bit image with a resolution of 20 pixels height and 26 pixels width.



Fig. 21: Test image for script



*Fig. 22: Test image with a resolution of 20*26*

When imported to Rhino with the Heightfield from Image function the following surface is the result when having a setup as explained in Subchapter [3.2.1.] with a number of sample points 200 x 200 and a height of 80 mm.

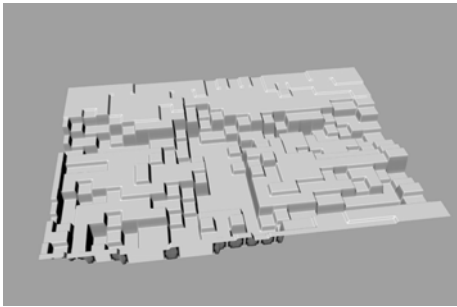


Fig. 23: surface generated out of image with the Heightfield function

The following script has been then applied to arrange points on a surface:

```
' *****
' ArrayPointsOnSurface
' Script for Rhino 3.0
' Sample Script of Rhino 3.0 altered by M .Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au
```

```
' DESCRIPTION: Creates an array of points on a surface.
' STATUS: working
' TO DO:

' *****
```

Option Explicit

```
Sub ArrayPointsOnSurface()
Dim strObject, nRows, nColumns
Dim U, V, i, j, arrParam(1), arrPoint
' Get the surface object

strObject = Rhino.GetObject("Select surface", 8)
If IsNull(strObject) Then Exit Sub
' Get the number of rows

nRows = Rhino.GetInteger("Number of rows", 2, 2)
If IsNull(nRows) Then Exit Sub
nRows = nRows - 1
' Get the number of columns

nColumns = Rhino.GetInteger("Number of columns", 2, 2)
If IsNull(nColumns) Then Exit Sub
nColumns = nColumns - 1
' Get the domain of the surface

U = Rhino.SurfaceDomain(strObject, 0)
V = Rhino.SurfaceDomain(strObject, 1)
If Not IsArray(U) or Not IsArray(V) Then Exit Sub
' Add the points

For i = 0 to nRows
arrParam(0) = U(0) + (((U(1) - U(0)) / nRows) * i)
For j = 0 to nColumns
arrParam(1) = V(0) + (((V(1) - V(0)) / nColumns) * j)
arrPoint = Rhino.EvaluateSurface(strObject, arrParam)
If IsArray(arrPoint) Then Rhino.AddPoint arrPoint
Next
Next
End Sub

' *****
```

The image below illustrates how the surface will look with points arranged on a surface.

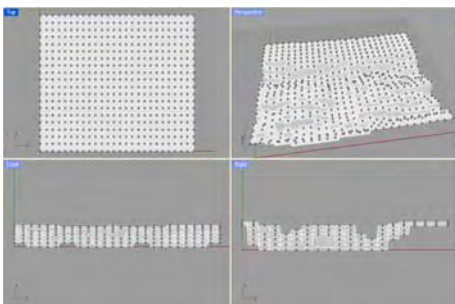


Fig. 24: surface generated out of image with points on surface when using the script

The next step/script takes a collection of points of different heights and assigns them to different layers depending on a specified range and number of divisions. It is designed to work with a surface from Heightfield from Image and the 'array points on surface' sample script.

```
' *****
' ASSIGN LAYER BY HEIGHT SCRIPT

' Script for Rhino 3.0
' Written by M .Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: This script takes a collection of points of
different heights
' and assigns them to different layers depending on a specified
' range and number of divisions.
' It is designed to work with a surface heightfield from an image
' and the 'array points on surface' sample script.

' STATUS: working
' TO DO: Automatically calculate the z value range of points.
' *****
```

Option Explicit

Sub assignlayerbyheight()

```
' define variables
Dim arrPoints, strPoint
Dim dblMinZ, dblMaxZ
Dim dblDivisions, i
Dim dblTotalRange, dblDivisionRange
Dim arrPtCoords, dblPtZ

' select points
arrPoints = Rhino.GetObjects("Select points to assign to layers",
1)

' define range of heights
dblMinZ = Rhino.GetReal("Define minimum z value", 0)
If IsNull(dblMinZ) Then Exit Sub

dblMaxZ = Rhino.GetReal("Define maximum z value", 10)
If IsNull(dblMaxZ) Then Exit Sub

' define number of divisions
dblDivisions = Rhino.GetInteger("define number of divisions", 8, 2)
dblDivisions = dblDivisions - 1

' calculate range for each layer
dblTotalRange = dblMaxZ - dblMinZ
dblDivisionRange = dblTotalRange / dblDivisions

' add new layers
For i = 0 To dblDivisions
    Rhino.AddLayer "Layer " & (i+1)
Next

i = 0
```

```

'loop through all points
For Each strPoint In arrPoints

  'get z value of point
  arrPtCoords = Rhino.PointCoordinates(strPoint)
  dblPtZ = arrPtCoords(2)

  'assign layer based on z value & calculate range dynamically
  For i = 0 To dblDivisions

    If dblPtZ > (dblMinZ + (dblDivisionRange*i)) And dblPtZ <
      (dblMinZ + (dblDivisionRange*(i+1))) Then
      Rhino.ObjectLayer strPoint, "Layer " & (i+1)
    End If

  Next
Next
End Sub
AssignLayerByHeight

' *****

```

The image below illustrates the outcome of the script.

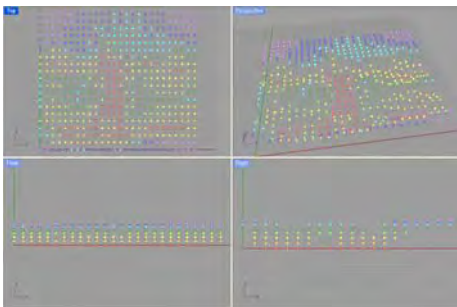


Fig. 25: surface generated out of image with points on surface and heights in different colours

For a better illustration and to generate a better render each point should have a sphere for clearer illustration. This should also be done with a script.

```

' *****
' AddSpheresToPoints

' Script for Rhino 3.0
' Written by M. Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: Script to add a sphere to a series of points
' and add it on the same layer as the point.

' STATUS: working
' TO DO:

' *****

```

Option Explicit

Sub AddSpheresToPoints()

'Define Variables

Dim arrPoints, dblRadius, strPoint
Dim strPointLayer, arrPointCoordinates

'Get user inputs

arrPoints = Rhino.GetObjects("Select points to assign to layers",
1)

dblRadius = Rhino.GetReal("Define radius of spheres", 1)

Rhino.EnableRedraw vbFalse

'Loop through all points

For Each strPoint In arrPoints

'get layer of point

strPointLayer = Rhino.ObjectLayer(strPoint)

'set current layer to same as point

Rhino.CurrentLayer strPointLayer

'get coordinates of point

arrPointCoordinates = Rhino.PointCoordinates(strPoint)

'add sphere

Rhino.AddSphere arrPointCoordinates, dblRadius

Next

Rhino.EnableRedraw vbTrue

End Sub

AddSpheresToPoints

'*****

The image below illustrates the outcome of the script.

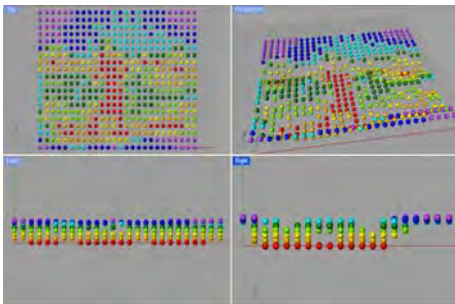


Fig. 26: surface generated out of image with points on surface and heights in different colours with points for better illustration

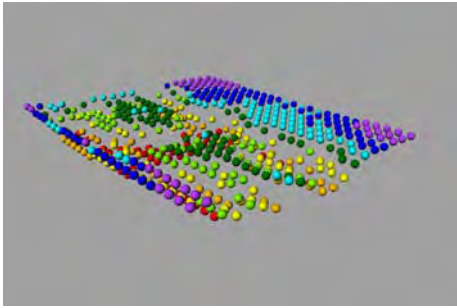


Fig. 27: Surface generated out of image view from the side

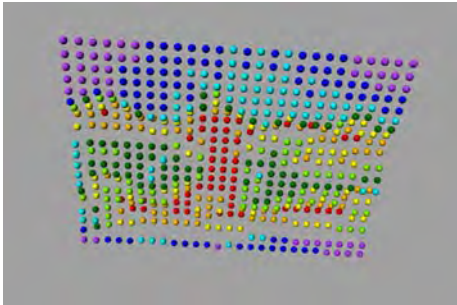


Fig. 28: Surface view generated out of image with the use of the previous designed script

Before continuing with scripting a second look should be taken at the Heightfield function to explore possible problems.

[3.2.5]
Foundation studies
with Heightfield
function II

Further tests with the Heightfield function should explore further if this function could be used for translating an image into a surface. With the results of the previous foundation study with Photoshop in mind an image containing 8 different colours should then be translated into a surface.

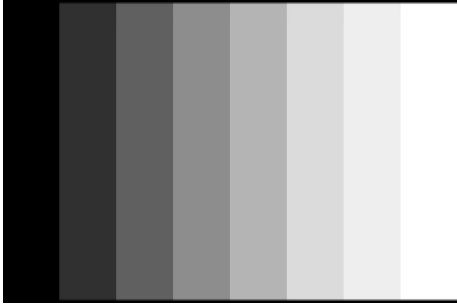


Fig. 29: 8 colours grey gradient for further tests with Heightfield function

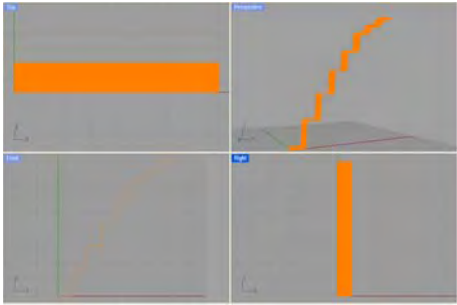


Fig. 30: 8 colours grey gradient imported to Rhino with Heightfield function

These 8 colours should then be mixed up to create different steps.

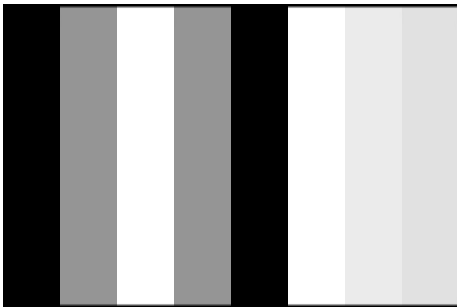


Fig. 31: 8 colours grey gradient mix for further tests with Heightfield function

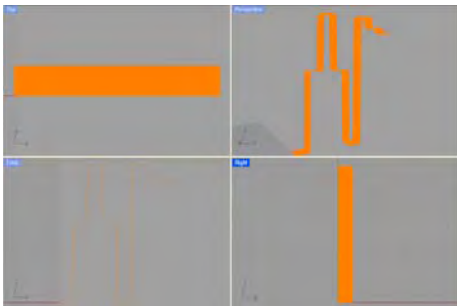


Fig. 32: 8 colours grey gradient mix imported to Rhino with Heightfield function

What has been noticed is the irregular distance between the gradients; even they had frequent gradient steps between them. It seems that the height between different colours varies depending on if they have a higher or lower level of black in them. The next image illustrates this with measured heights from one height to another height.

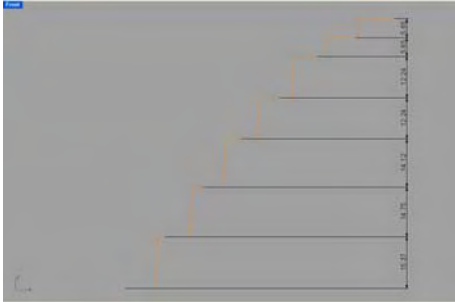


Fig. 33: Different heights when using 8 colours grey gradient, which proves that the Heightfield function do not create an exact position depending on the colour value

[3.2.6] Conclusion

Even if the Heightfield option would in principle do what is necessary to translate an image into a surface, it seems not ideal to achieve a clear image. This is due to the Heightfield function not being as precise as other options, which should be explored in the next chapter.

[3.3] Tests with translating pixel information into an Excel data sheet

[3.3.1] Introduction

When doing the foundation studies explained in Subchapter [3.2.3.] another option was discovered in how the colour information of an image could be translated into a surface. When recalling the Heightfield from Image function, one notices that the main role of this function is giving a colour a certain position in the Z – axis. This has been explained in Subchapter [3.2.2.] when analysing different colours and their position in the Z – axis. When using Photoshop the program offers an Info tool which tells the location of the pixel as well as the colour information in RGB and CMYK.

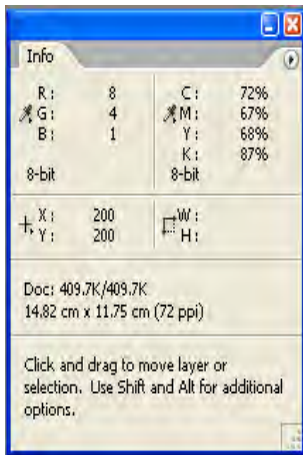


Fig. 34: Info tool Photoshop

This kind of information could be used when finding a way to export the colour information of each pixel to an Excel data sheet. The datasheet would then feed a script which would translate the colour information into a Z – value. The size of the image would define the size of the surface in the X and Y plane and the colour information in the Z plane.

[3.3.2]
ImageJ program

The ImageJ program allows its user to save an image file such as *.jpeg or *.tif into an *.txt file. The txt file could be then opened in Excel to create an Excel data sheet.

The following steps illustrate what the program does:



Fig. 35: Test image 5 x 5 pixel

When opening the image in ImageJ, ImageJ shows the pixel size, colour information and size of image in kilobit.

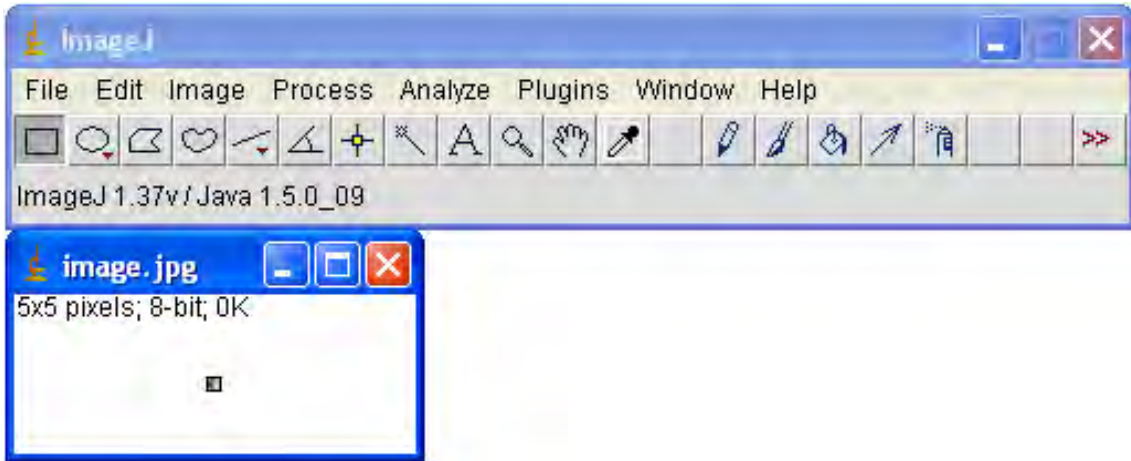


Fig. 36: ImageJ program

It is then possible to change the image to a 'Text image' which will produce the before mentioned *.txt file. This file will be then opened in Excel.

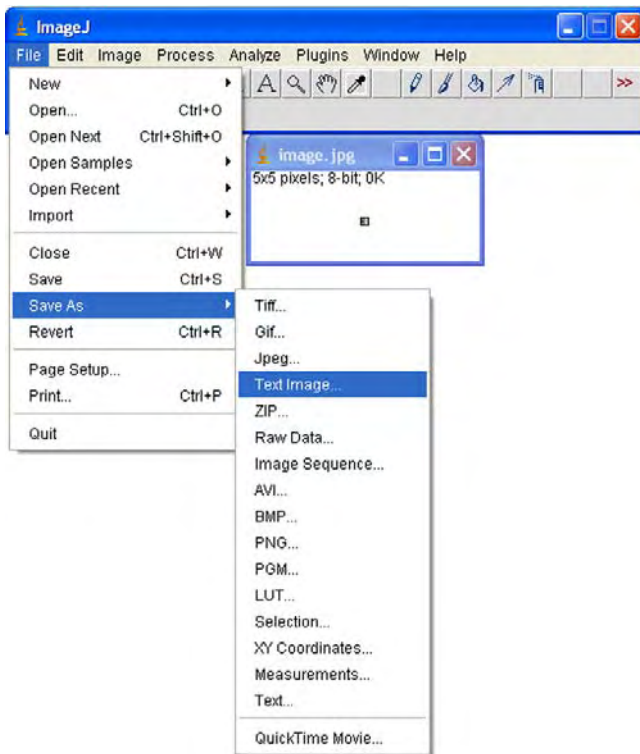


Fig. 37: ImageJ program roll out of 'Save as' command which allows saving file as txt file

The following array of data is visible in Excel:

138	125	139	201	203
114	118	143	205	187
58	82	154	215	191
111	114	147	237	226
143	126	131	219	208

Each of these numbers represents colour information in greyscale. Greyscale has a pixel value from 0 to 255, 0 meaning the darkest and 255 meaning pure white.

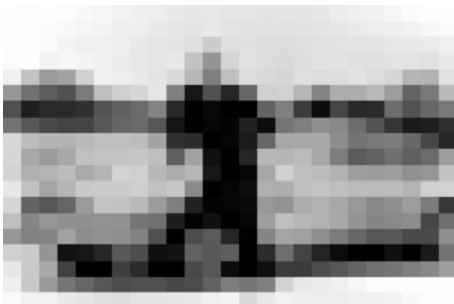
[3.4]
Script based on
previous tests

[3.4.1]
Working on script

The following script makes it possible to convert the information stored in the Excel sheet into a surface. To illustrate the process a test images previous used in Subchapter [3.2.4] should be reused.



Fig. 38: Test image for script



*Fig. 39: Test image with a resolution of 19*25*

When saved as a text image in ImageJ and exported to Excel the following data array has been the result:

```

7 7 7 7 7 7 4 4 4 4 4 4 1 4 1 4 1 4 4 4 4 7 7 7 7
7 7 7 7 4 4 4 4 4 1 1 4 1 1 1 1 1 4 1 4 4 4 4 7 7
7 7 7 7 4 4 4 4 4 4 1 1 1 1 1 1 1 1 4 1 1 4 4 4 4
7 7 4 7 4 4 4 4 1 4 0 2 1 1 1 1 1 1 1 1 1 1 1 4
4 0 2 2 1 4 4 4 1 1 0 5 0 1 1 1 1 1 1 1 1 1 0 0 1
2 5 5 3 5 0 1 1 1 5 6 3 3 5 0 1 0 5 0 0 0 2 5 5 0
3 3 6 3 3 3 5 5 3 6 6 6 6 5 5 3 6 3 6 3 5 3 5 5
5 3 3 3 3 5 5 3 3 6 6 6 3 6 5 5 2 2 5 3 3 6 6 3
1 0 0 0 0 0 0 0 2 6 3 6 6 6 5 2 0 2 2 5 5 5 3 3
1 2 2 2 0 0 0 1 0 5 2 6 6 3 0 0 0 2 5 3 5 5 3 5 2
1 0 0 0 0 0 1 0 0 0 0 6 6 6 2 1 0 0 0 2 2 2 2 2 0
1 0 2 0 0 0 1 0 0 0 2 6 6 6 0 2 0 0 0 0 0 0 0 0 1
1 5 3 5 2 0 0 0 2 5 6 6 6 6 2 2 0 0 0 2 5 2 2 5 3
0 5 2 2 2 2 5 5 3 6 6 6 6 2 0 0 2 2 2 2 5 2 3 5
2 5 2 2 5 2 2 2 3 6 3 3 3 6 2 2 2 2 5 5 3 3 3 2
1 0 2 6 3 6 3 3 6 6 5 3 5 6 3 5 3 3 6 6 6 6 6 5
1 1 2 3 6 6 3 3 6 6 3 3 6 6 3 3 3 3 5 5 5 5 2 2
4 0 2 5 5 3 3 3 3 3 5 5 5 5 1 1 4 4 7 7 7 7 7 7
7 7 4 4 4 1 1 1 4 4 4 4 7 1 4 4 4 4 4 4 4 7 7 7

```

The script below would then be applied to alter this data into a surface generated by points:

```

' *****
' CREATES A MATRIX OF POINTS BASED ON AN IMAGE TEST FILE

' Script for Rhino 3.0
' Written by M. Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: creates a matrix of points based on an image txt file
' STATUS: working
' TO DO:

' *****

Sub PointsFromImage()

' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

' gets information from excel

Dim currentPoint
Dim MyPointCount As Double
Dim AllpointCoords
Dim MyXRow
Dim MyYRow

```

```

MyYRow = Rhino.GetReal ("How big is picture in X direction?")
MyXRow = Rhino.GetReal ("How big is picture in Y direction?")

newPointCount = MyXRow * MyYRow
ReDim All pointCoords(newPointCount)

For i = 1 To MyXRow
  For j = 1 To MyYRow
    Z = Cells(i, j).Value
    currentPointCoord = Array(i, j, Z)
    currentPoint = Rhino.AddPoint(currentPointCoord)
  Next
Next

End Sub

```

'*****'

The image below illustrates the outcome of the script. One can recognize that each point has exactly the same distance to another point in all three axes. This is a main difference to the tests conducted with the earlier scripts, which were based on the Heightfield from Image function. The Heightfield function had differences in the distance from one point to the other in the Z – axis. This is much better controlled when altering an image with the ImageJ program.

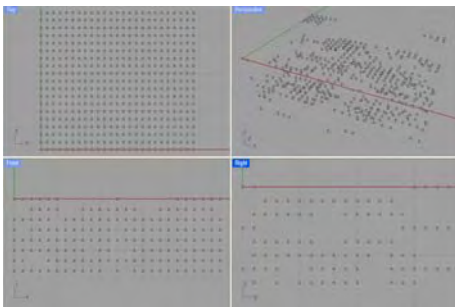


Fig. 40: surface generated out of feeding information provided by ImageJ into script

The next step is applying the script used before in Subchapter [3.2.4] 'ASSIGN LAYER BY HEIGHT SCRIPT' this script will then pick all points and select them regarding the position in the Z – axis into different layers.

When applying the script and analysing the result, a few alterations and improvements of the script were necessary.

The image was upside down and also the surface height not displayed correctly with having white as the brightest colour close to the viewer and black as the darkest colour at the end. The first problem of the upside down image could be solved when altering the line:

currentPointCoord = Array(i , j , Z)

in the script above to:

currentPointCoord = Array(i , -j , Z)

to allow the image to be built up in rhino not close to the location 0,0,0 in the coordination system and then moving towards + X und +Y direction, but in a same way one would read a page, starting at the top left corner.

The analysing the images created in Photoshop one important difference was noticed. When saving the image in Photoshop as a 'save for web' image the colours and the colour index have been in an order so that the colour index of the brightest colour (white) had the colour index 7 and the darkest (nearly black) the colour index 0. This was not the case when the image was saved and analysed again in Photoshop (see images P_3.4.1.B and P_3.4.1.C). The colour index was then mixed up. An effect of depth created through colour could not be achieved with a not-ordered colour index. This mix-up was corrected within the script when altering the colour index, which drives the Z – value, into a corrected new Z – value.

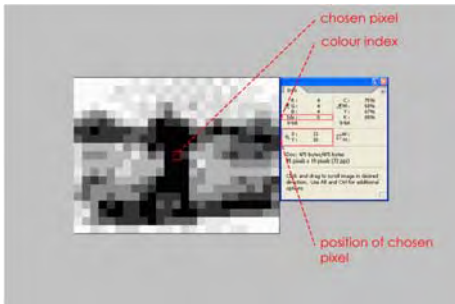


Fig. 41: Analysis of image with a black image
RGB value 4, 4, 4 having index 0

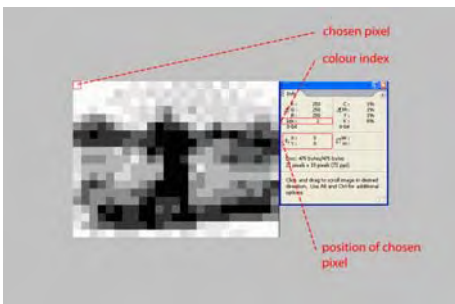


Fig. 42: Analysis of image with a black image
RGB value 250, 250, 250 having index 2

For this image following the alteration of the Z – value has been included in the script.

```
'New Z value to create a depth in regards to greyscale
  If z = 6 Then MyNewZ = 0
  If z = 3 Then MyNewZ = 1
  If z = 5 Then MyNewZ = 2
  If z = 2 Then MyNewZ = 3
  If z = 0 Then MyNewZ = 4
  If z = 1 Then MyNewZ = 5
  If z = 4 Then MyNewZ = 6
  If z = 7 Then MyNewZ = 7
```

This alteration has to be done for each picture series, or at least has to be controlled before applying an image to guarantee a gradient from black to white.

[3.4.2]
Final script

Resulting on the above mentioned testings and alterations to work towards a final script, the following script is able to take a movie and translate movement happening within it into a movement of a surface. As an example, to test the script and demonstrate how the result could look, a movie showing a time lapse of clouds moving across a blue sky has been chosen.

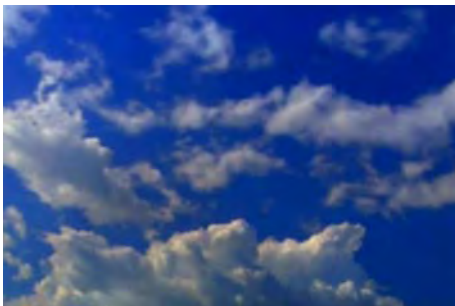


Fig. 43: Image taken out of movie clip, which functions as a sample image for testing script

This image has then been altered into a greyscale 8 bit image and then into a text image as described in Subchapter [3.4.1].

```
' *****
' CREATES A MATRIX OF POINTS BASED ON AN IMAGE TEST FILE

' Script for Rhino 3.0
' Written by M .Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au
```

```

'DESCRIPTION: creates a matrix of points based on an image txt file
'STATUS: working
'TO DO:

' *****

Sub PointsFromImage()

' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

' Variables
  Dim currentPoint
  Dim MyPointCount As Double
  Dim AllpointCoords
  Dim MyRowX
  Dim MyRowY
  Dim arrAllPoints
  Dim MyNewZ
  Dim arrOldObjects
  Dim MySheetCount
  Dim Loop1

  MySheetCount = 1

' define image size in pixel + size of sphere
  MyRowY = Rhino.GetReal("How big is picture in Y direction?")
  MyRowX = Rhino.GetReal("How big is picture in X direction?")
  MyRadius = Rhino.GetReal("What is the radius of spheres?")

' redim code
  newPointCount = MyRowY * MyRowX
  ReDim AllpointCoords(newPointCount)

' Counts sheets in excel
  For k = 1 To MySheetCount

' EnableRedraw - switch of screen to start / finish script
  Rhino.EnableRedraw False

' Removes all objects.
  arrOldObjects = Rhino.AllObjects()
  If IsArray(arrOldObjects) Then
  Rhino.DeleteObjects (arrOldObjects)
  End If

' Loop for creating an image of one sheet out of excel
  For i = 1 To MyRowX 'i gets X Value
  For j = 1 To MyRowY 'j gets Y Value
  ' z = Cells(j, i).Value 'gets Z Value
z = Worksheets(k).Cells(j, i).Value

' New Z value to create a depth in regards to greyscale

  If z = 255 Then MyNewZ = 0
  If z = 254 Then MyNewZ = 0
  If z = 253 Then MyNewZ = 0
  If z = 252 Then MyNewZ = 0
  If z = 251 Then MyNewZ = 0

```

If z = 249 Then MyNewZ = 0
If z = 248 Then MyNewZ = 0
If z = 247 Then MyNewZ = 0
If z = 246 Then MyNewZ = 0
If z = 245 Then MyNewZ = 0

If z = 230 Then MyNewZ = 1
If z = 229 Then MyNewZ = 1
If z = 228 Then MyNewZ = 1
If z = 227 Then MyNewZ = 1
If z = 226 Then MyNewZ = 1
If z = 225 Then MyNewZ = 1
If z = 224 Then MyNewZ = 1
If z = 223 Then MyNewZ = 1
If z = 222 Then MyNewZ = 1
If z = 221 Then MyNewZ = 1

If z = 202 Then MyNewZ = 2
If z = 201 Then MyNewZ = 2
If z = 200 Then MyNewZ = 2
If z = 199 Then MyNewZ = 2
If z = 198 Then MyNewZ = 2
If z = 197 Then MyNewZ = 2
If z = 196 Then MyNewZ = 2
If z = 195 Then MyNewZ = 2
If z = 194 Then MyNewZ = 2
If z = 193 Then MyNewZ = 2

If z = 175 Then MyNewZ = 3
If z = 174 Then MyNewZ = 3
If z = 173 Then MyNewZ = 3
If z = 172 Then MyNewZ = 3
If z = 171 Then MyNewZ = 3
If z = 170 Then MyNewZ = 3
If z = 169 Then MyNewZ = 3
If z = 168 Then MyNewZ = 3
If z = 167 Then MyNewZ = 3
If z = 166 Then MyNewZ = 3

If z = 145 Then MyNewZ = 4
If z = 144 Then MyNewZ = 4
If z = 143 Then MyNewZ = 4
If z = 142 Then MyNewZ = 4
If z = 141 Then MyNewZ = 4
If z = 140 Then MyNewZ = 4
If z = 139 Then MyNewZ = 4
If z = 138 Then MyNewZ = 4
If z = 137 Then MyNewZ = 4
If z = 136 Then MyNewZ = 4

If z = 120 Then MyNewZ = 5
If z = 119 Then MyNewZ = 5
If z = 118 Then MyNewZ = 5
If z = 117 Then MyNewZ = 5
If z = 116 Then MyNewZ = 5
If z = 115 Then MyNewZ = 5
If z = 114 Then MyNewZ = 5
If z = 113 Then MyNewZ = 5
If z = 112 Then MyNewZ = 5
If z = 111 Then MyNewZ = 5

If z = 90 Then MyNewZ = 6
If z = 89 Then MyNewZ = 6
If z = 88 Then MyNewZ = 6
If z = 87 Then MyNewZ = 6
If z = 86 Then MyNewZ = 6
If z = 85 Then MyNewZ = 6

```

If z = 84 Then MyNewZ = 6
If z = 83 Then MyNewZ = 6
If z = 82 Then MyNewZ = 6
If z = 81 Then MyNewZ = 6

If z = 60 Then MyNewZ = 7
If z = 59 Then MyNewZ = 7
If z = 58 Then MyNewZ = 7
If z = 57 Then MyNewZ = 7
If z = 56 Then MyNewZ = 7
If z = 55 Then MyNewZ = 7
If z = 54 Then MyNewZ = 7
If z = 53 Then MyNewZ = 7
If z = 52 Then MyNewZ = 7
If z = 51 Then MyNewZ = 7

If z = 30 Then MyNewZ = 8
If z = 29 Then MyNewZ = 8
If z = 28 Then MyNewZ = 8
If z = 27 Then MyNewZ = 8
If z = 26 Then MyNewZ = 8
If z = 25 Then MyNewZ = 8
If z = 24 Then MyNewZ = 8
If z = 23 Then MyNewZ = 8
If z = 22 Then MyNewZ = 8
If z = 21 Then MyNewZ = 8

If z = 10 Then MyNewZ = 9
If z = 9 Then MyNewZ = 9
If z = 8 Then MyNewZ = 9
If z = 7 Then MyNewZ = 9
If z = 6 Then MyNewZ = 9
If z = 5 Then MyNewZ = 9
If z = 4 Then MyNewZ = 9
If z = 3 Then MyNewZ = 9
If z = 2 Then MyNewZ = 9
If z = 1 Then MyNewZ = 9
If z = 0 Then MyNewZ = 9

' creates point in Rhino
currentPointCoord = Array(i, -j, MyNewZ)
' creates sphere in Rhino
currentSphere = Rhino.addsphere(currentPointCoord, MyRadius)

' call colour function
MyChangeColour currentSphere, currentPointCoord(2)

Next 'next for (i)
Next 'next for (j)

' EnableRedraw - switch of screen to start / finish script
Rhino.EnableRedraw True

' call screengrab function
screengrab (k)

Next 'next for (k)

' Tells that script is finished
MsgBox ("Script Finish")

End Sub

```

The script also has a function which will capture an image at the end of each loop to illustrate the result and the changes of the surface while running the script.

```
' *****
Function screengrab(i)
' Function for Rhino 3.0
' Written by M .Hank Haeusler 23.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: Incremental Screen Capture to file location
' STATUS: working
' TO DO:

' *****
' start of Rhino Connection
Dim RhinoApp As Object
Set RhinoApp = CreateObject("Rhino3. Application")
Dim Rhino As Object
Set Rhino = RhinoApp.GetScriptObject()
RhinoApp.Visible = True
'end of Rhino Connection
' *****
' function screengrab
Dim sPath

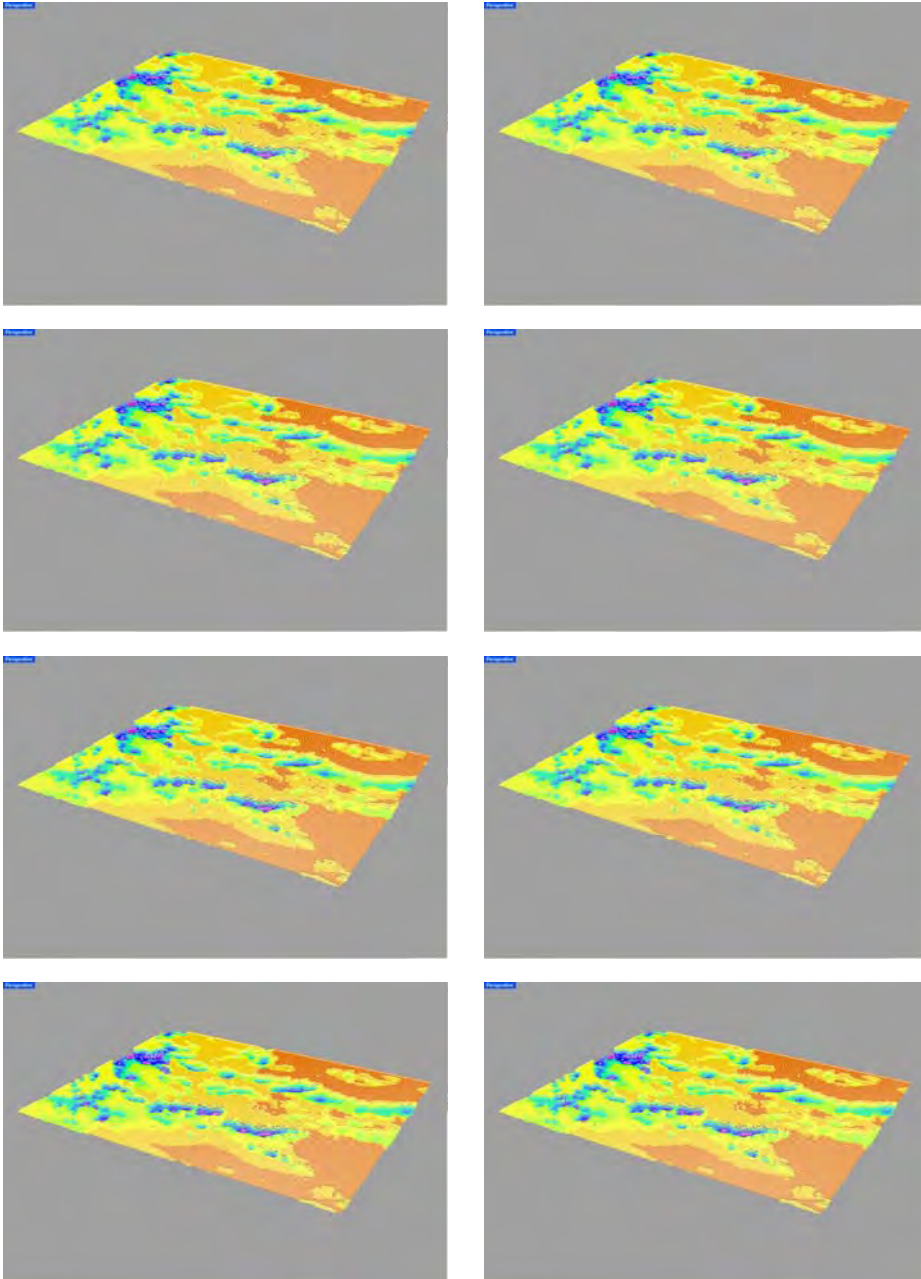
' Specifies Path location and file type
sPath = "S:\Students\Hank\01_RESEARCH\Surface
scripting\02_PICS\screengrab of multilayered surface\" & i &
".jpeg"

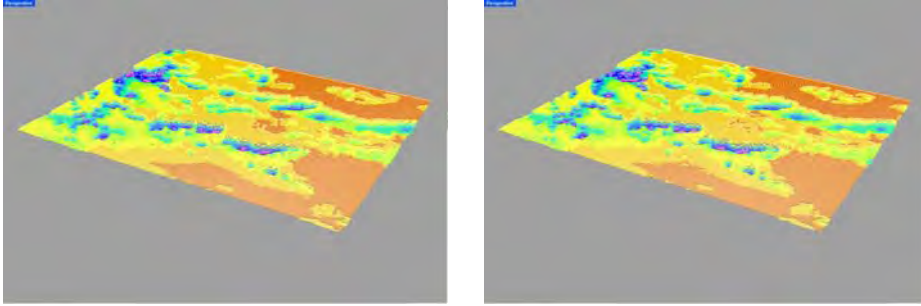
Rhino.Command "-_ScreenCaptureToFile " & Chr(34) & sPath & Chr(34)
& " Enter"
' *****
End Function
```

When altering the images generated from the cloud movie into txt files and then importing them into Excel, Excel limited the amount of columns one could have to 255. Therefore the image had a pixel resolution of 255 * 200. Furthermore Photoshop had difficulties when saving the images as a *.gif file and altered some of the colours. As a result of this the images were not saved with the 'save for web' command but with the 'Posterize' command, which allows the user to define how many colours one would like to have in an image. I have limited the number to 10 colours, which has been as the excel data sheet later proved, more than exact 10 colours, it produces a range of 10 very colours with having each single colour with some same variations. This have been considered in the script and altered to a total of 10 colours which feed into the script.

[3.4.3]
Images

The following 10 images have been selected to represent how the result could look. Due to there being nearly 600 single images not all of them could be displayed here.





Picture Series 1: Sequence of images first 10 of 589 the information of the cloud movie which were fed into the script

[3.5] **Conclusion**

It has been possible to provide future artists with a tool that would allow them to explore the potential of a Spatial Dynamic Media System when coming from a film or image based background. The code also allows the user to set certain parameters such as the number of colours or how these colours are then translated.

[4.]
P4 /

Test Series III:
Creating a surface based
on a 3D form

[4.1]
Introduction

This test addresses a solution for an application similar to that in the designed media content in Project 3 where human faces were scanned in order to analyse their emotions. Here a 3D surface, that being the face, has been scanned and displayed via a 3D light point matrix. This will be achieved by using the following script, where any surface could be analysed and separated in a grid of point. Each point should then have its own colour depending on its Z position. Surfaces to be scanned are not always in a rectangular form, as a standard Spatial Dynamic Media System Hardware configuration would be, and assuming that the screen on which any kind of animation or movement is to be displayed is in a rectangular form the surface must be trimmed and altered to fit. Therefore the user has the option of defining the size of the screen before translating the scan into a 3D light point matrix.

[4.2]
Set up

[4.2.1]
Test surfaces

Once again, the surfaces used for the test are examples only and have no significance in of themselves. More than one example should be used to prove that the script functions with a variety of surfaces. Here two different test surfaces are used:

- a simple 2D plane, further called Surface 1
- a surface generated through the Heightfield from image function, further called Surface 2

[4.2.2]
Images of test surfaces

The following two images are used for the tests:

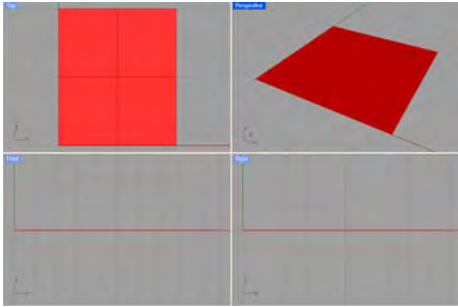


Fig. 44: 2D plane surface as test surface for script

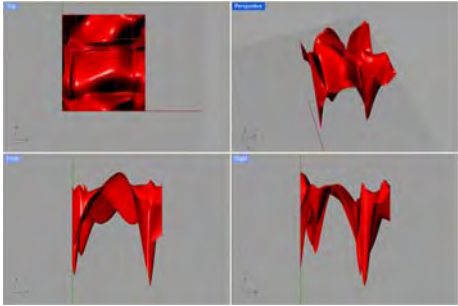


Fig. 45: Surface generated of a heightfield from image as a second test surface for script

[4.3]
Script based on previous set up

[4.3.1]
Testing script

The following script is able to divide a surface into a resolution of light points required by the user.

```
' *****
' CREATING A SURFACE GENERATE OF LIGHT POINTS OUT OF A 3D FORM
' Script for Rhino 3.0
' Written by M .Hank Haeusler 07.02.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: selecting surface and alter surface into a 3D pint
grid with different colours for different heights
' STATUS:
' TO DO:

' *****
```

```
Sub SurfaceOut3DForm()
```

```

' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

' SELECT SURFACE*****

' variables SELECT SURFACE
Dim strObject
Dim arrDomainU
Dim arrDomainV
Dim arrParam(1)
Dim arrpoint
Dim MyFirstPt
Dim MyResolution
Dim AllPts()
Dim arrPointGridOnSrf
Dim MyBasePoint

' *****
' get resolution
  dbl res = Rhino.GetInteger("Resolution?")

' Get the surface object
  strObject = Rhino.GetObject("Select surface", 8)
  If IsNull(strObject) Then Exit Sub

' Get the domain of the surface
  arrDomainU = Rhino.SurfaceDomain(strObject, 0)
  arrDomainV = Rhino.SurfaceDomain(strObject, 1)

' Turn off redraw function
  Rhino.EnableRedraw vbFalse

' Add the points
  If Rhino.IsSurface(strObject) Then
    arrParam(0) = arrDomainU(0)
    arrParam(1) = arrDomainV(0)
    arrpoint = Rhino.EvaluateSurface(strObject, arrParam)
  End If

' Array points with equal distance
  counter = 0

  For i = 0 To arrDomainU(1) Step dbl res
    arrParam(0) = i

    For j = 0 To arrDomainV(1) Step dbl res
      ReDim Preserve AllPts(counter)

      MyFirstPt = Array(arrpoint(0) + i, arrpoint(1) + j, 0)
      AllPts(counter) = MyFirstPt
      counter = counter + 1

    Next
  Next

  ReDim arrPointGridOnSrf(UBound(AllPts))
  counter = 0
  For Each pt In AllPts
    tempPtOnSrf = Rhino.SurfaceClosestPoint(strObject, pt)

```

```

arrPoint2 = Rhino.EvaluateSurface(strObject, tempPtOnSrf)
Rhino.AddPoint arrPoint2
arrPointGridOnsrf(counter) = arrPoint2
counter = counter + 1

```

```

Next
End Sub

```

When testing the script on Surface 1, the script generated the following result with the resolution set to five, which is equal to a distance of 5 cm from LED stick to LED stick.

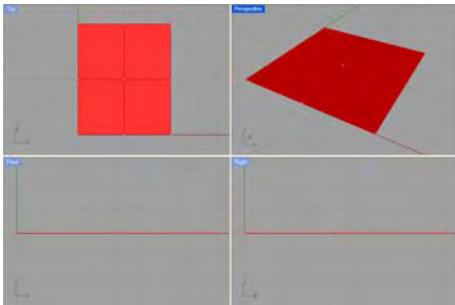


Fig. 46: Surface 1 divided in points by the script

Applying the same script on Surface 2 with a resolution of 5 generated the following result:

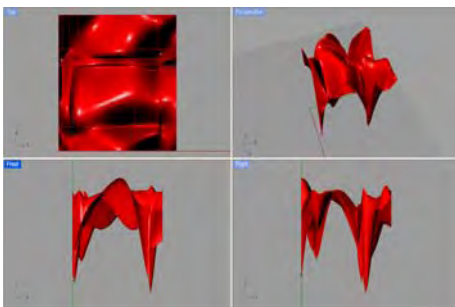


Fig. 47: Surface 2 divided in points by script

Here the script did not create an array of points and the points were not spread evenly over the surface. This uneven spread is due to Rhino script getting the first point at the fringe of the surface arraying them and then creating surface closest point on the surface based on the U and V value of the surface. This surface closest point is not projected parallel to the surface, therefore the distortion.

[4.3.2]
Final script

This error is corrected in the following script.

```
' *****
' CREATING A SURFACE GENERATE OF LIGHTPOINTS OUT OF A 3D FORM
' Script for Rhino 3.0
' Written by M .Hank Haeusler 07.02.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: selecting surface and alter surface into a 3D pint
grid with different colours for different heights
' STATUS:
' TO DO:
' *****

Sub SurfaceOut3DForm()

' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

' SELECT SURFACE*****
' variables SELECT SURFACE
Dim strObject
Dim arrDomainU
Dim arrDomainV
Dim arrParam(1)
Dim arrpoint

' *****
' get resolution
  dbl res = Rhino.GetInteger("Resolution?")

' Get the surface object
  strObject = Rhino.GetObject("Select surface", 8)
  If IsNull(strObject) Then Exit Sub

' Get the domain of the surface
  arrDomainU = Rhino.SurfaceDomain(strObject, 0)
  arrDomainV = Rhino.SurfaceDomain(strObject, 1)

' Turn off redraw function
  Rhino.EnableRedraw False

' Get min u/v domains
  If Rhino.IsSurface(strObject) Then
  arrParam(0) = arrDomainU(0)
  arrParam(1) = arrDomainV(0)

' converts u/v parameter into X,Y,Z parameter
  arrpoint = Rhino.EvaluateSurface(strObject, arrParam)
  End If
```

```

' create point matrix
MyMatrix = PointGrid(arrDomainU, arrDomainV, arrpoint, dblres,
strObject)

' Turn on redraw function
Rhino.EnableRedraw True

End Sub

' *****

Function PointGrid(arrDomainU, arrDomainV, arrpoint, dblres,
surface)

' *****
' start of rhino connection
Dim RhinoApp As Object
Set RhinoApp = CreateObject("Rhino3. Application")
Dim Rhino As Object
Set Rhino = RhinoApp.GetScriptObject()
Rhino.Visible = True
' end of rhino connection
' *****

Dim allPoints()
Dim arrPoints()

newcounter = 0

For i = 0 To arrDomainV(1) Step dblres

counter = 0
For j = 0 To arrDomainU(1) Step dblres

' add points in u direction
coordPoint = Array(arrpoint(0) + j, arrpoint(1) + i,
arrpoint(2))

arrStart = Array(coordPoint(0), coordPoint(1), -1000)
arrEnd = Array(coordPoint(0), coordPoint(1), 1000)

sampleLine = Rhino.AddLine(arrStart, arrEnd)
stringy = ("intersect SelID " + surface + " SelID " +
sampleLine + " enter")
tempIntersect = Rhino.Command(stringy)

If tempIntersect = True Then
ReDim Preserve arrPoints(counter)

strPoint = Rhino.LastObject
tempPoint = Rhino.PointCoordinates(strPoint)

arrPoints(counter) = tempPoint

counter = counter + 1
End If

tempCurve = Rhino.ObjectsByType(4)
Rhino.DeleteObjects (tempCurve)
Next

ReDim Preserve allPoints(newcounter)
allPoints(newcounter) = arrPoints
newcounter = newcounter + 1
Next

PointGrid = allPoints

```

End Function

The following images show Surface 2 with a resolution set to 10:

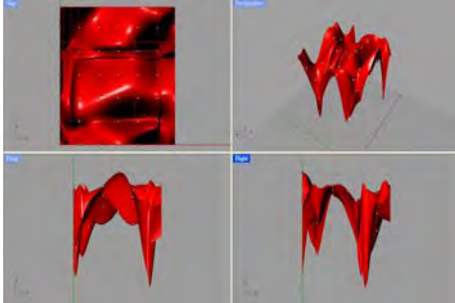


Fig. 48: Surface 2 divided in points with improved script

[5.]
P4 /
Test Series IV:
Creating a multilayered
surface

[5.1]
Introduction

The codes written earlier for sensor input, image, video or 3D form should now be mixed with each other to create a multilayered surface. Here two or more surface movements should be displayed at the same time. The surfaces should be stored in separate files and not in the Excel sheet as has been the case earlier. With having the data stored in one location and the running code in another location, one has the option to upload data by simply altering the name of the file location which inhabits the stored information. Both of the used input surface movements are for representation purposes only and are rather simple movements. Nevertheless they can demonstrate how such a surface could look. Furthermore the crossing point of both surfaces should be illustrated in a different colour and the two original surfaces in a second and third colour.

[5.2]
Tests towards
writing a code

[5.2.1]
Creating two
different animations

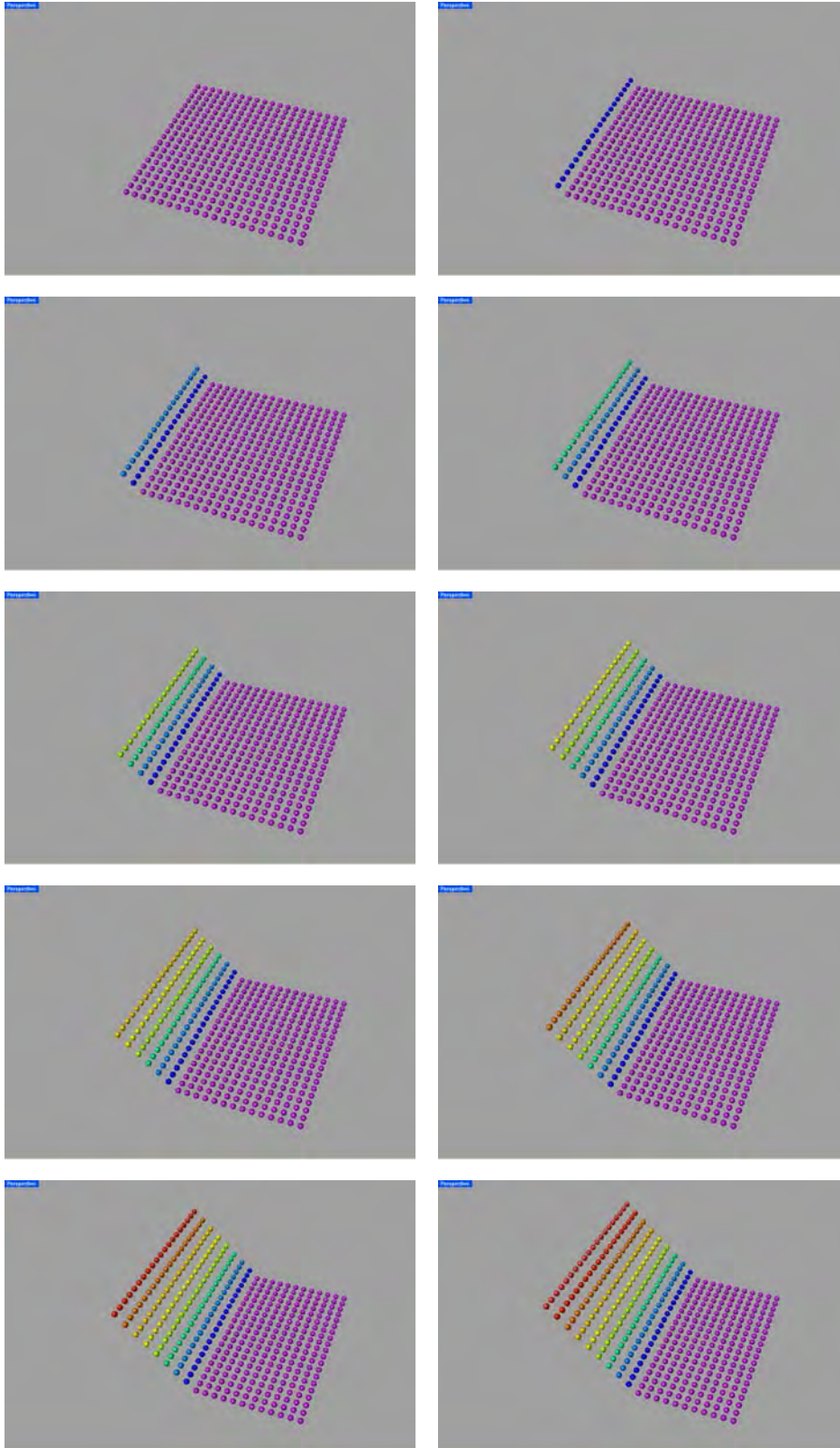
Wave form

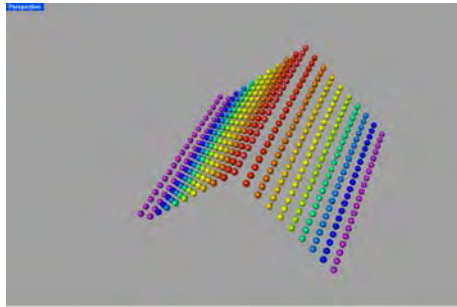
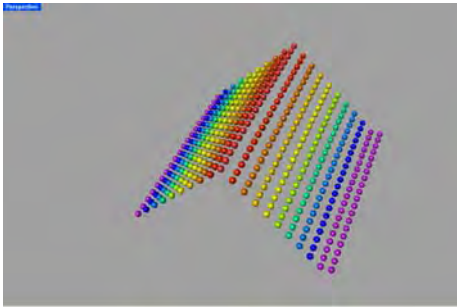
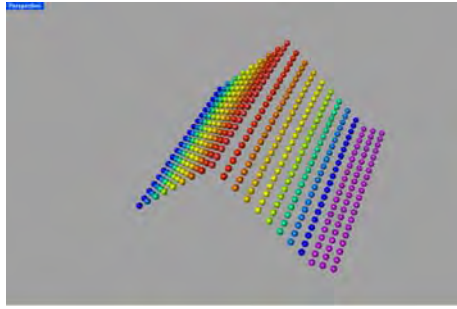
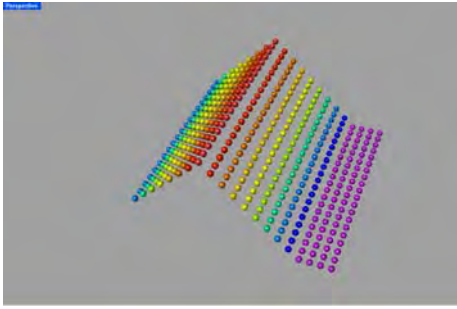
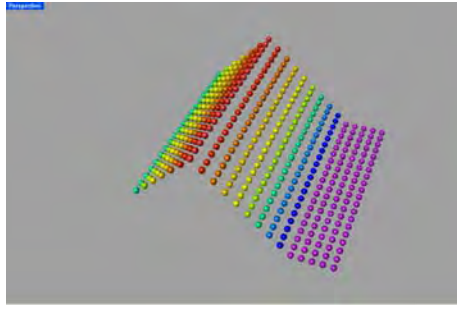
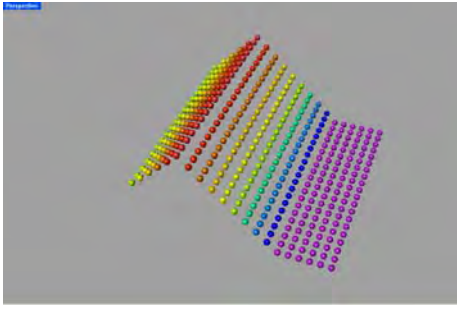
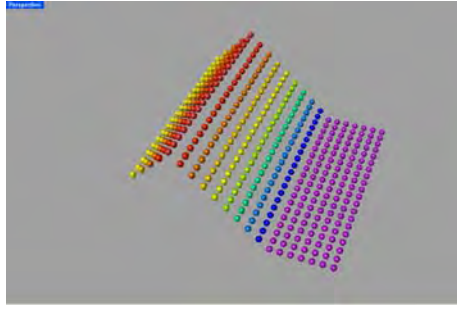
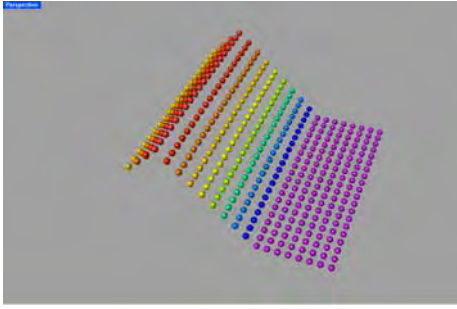
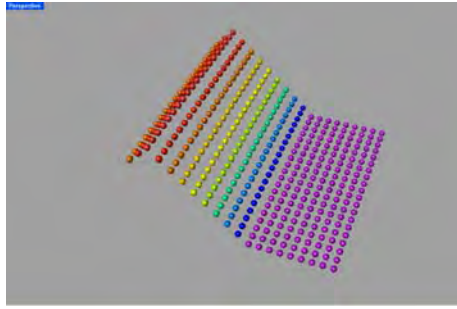
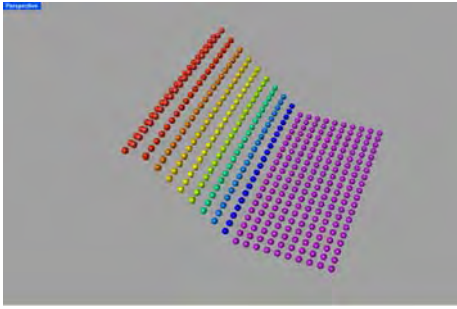
To illustrate the code two simple movements should be generated. These movements are expressed in Excel due to this being the program used so far mainly as an input source for the codes. The information for this surface movement is stored in a remote data sheet and will run using a code stored in a separate sheet.

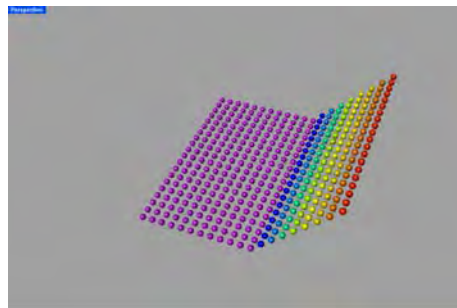
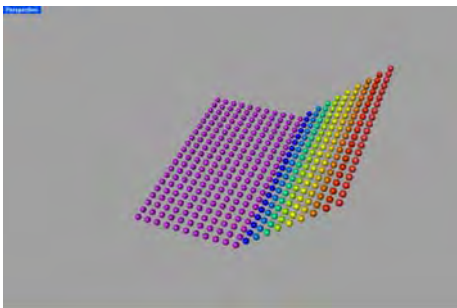
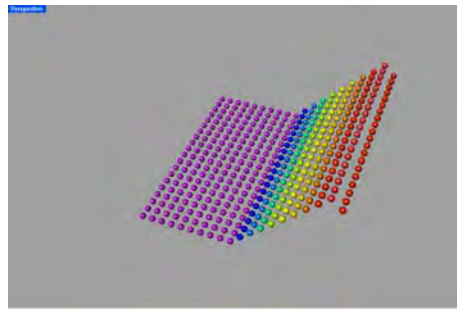
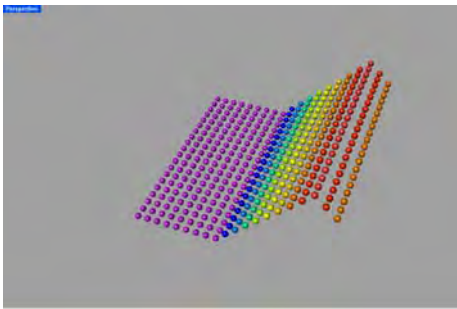
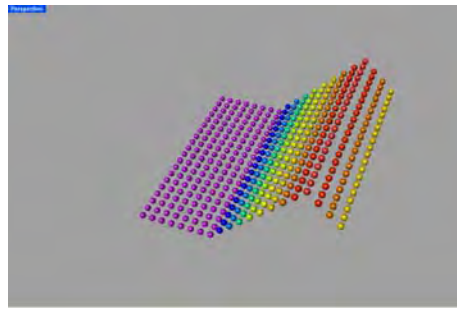
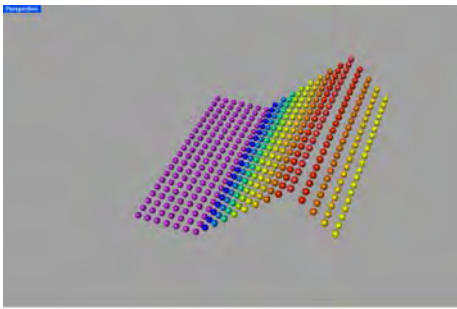
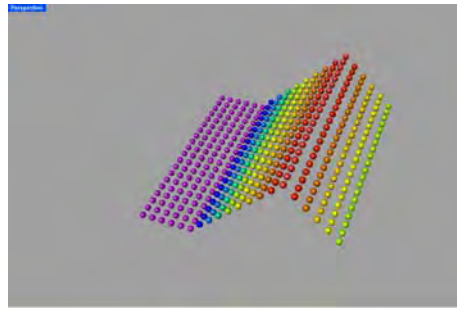
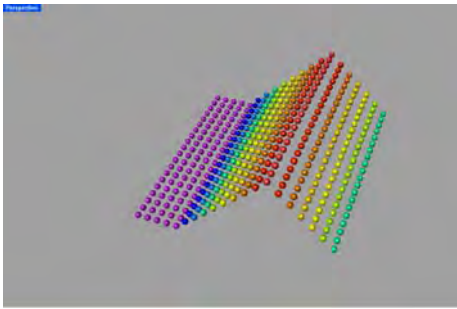
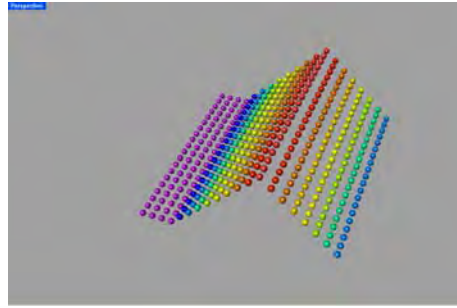
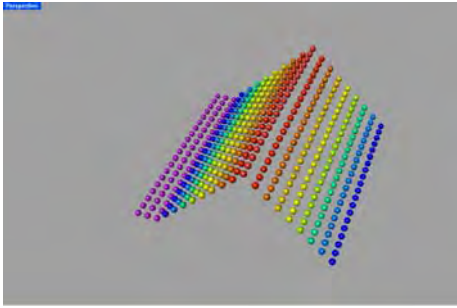
Both images are 20 * 20 pixels in size and have either a depth of 7 or 10 pixels. The Excel datasheet therefore contains 38 spreadsheets with 20 columns and 20 rows, with the numbers arranged in a way that it either creates a wave form or a drop form and changes from value 0 to 7 (10) to create a surface movement.

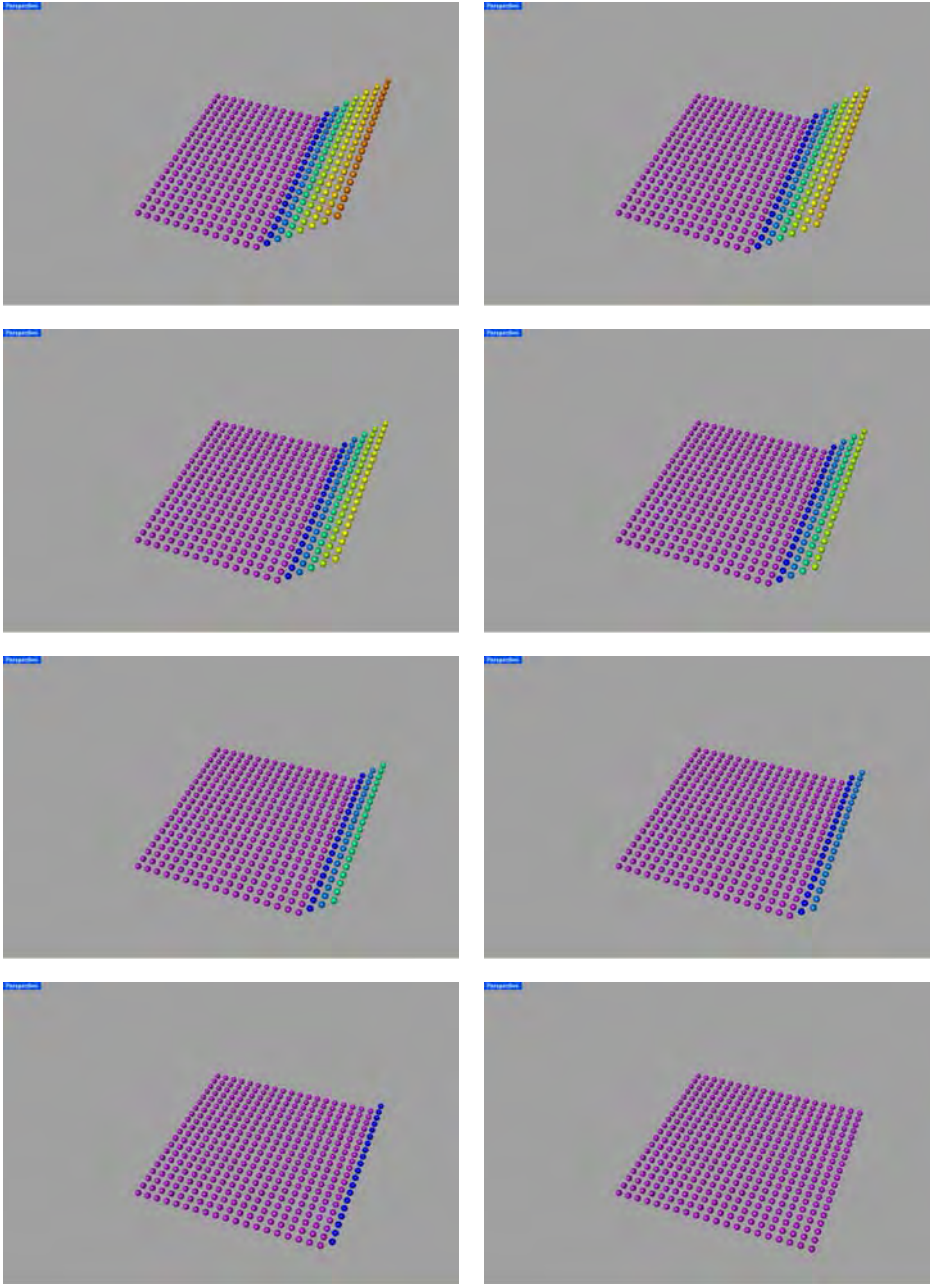
The code is similar to those created earlier, for example when creating a surface out of an image. Here the surface is still in different colours; in the final demonstration it will be in one colour only.

The following images (*Picture Series 1*) illustrate the movement of the surface with a wave form:





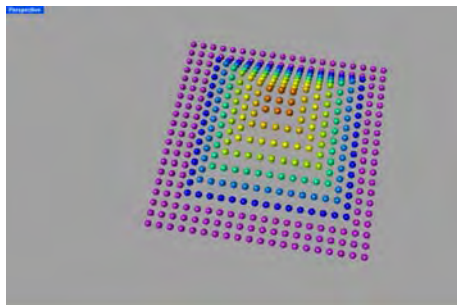
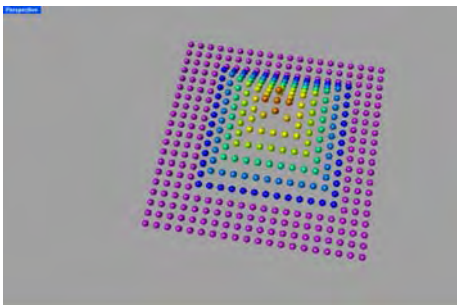
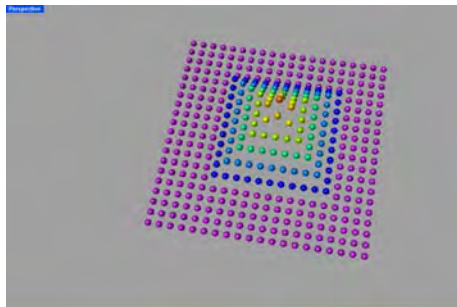
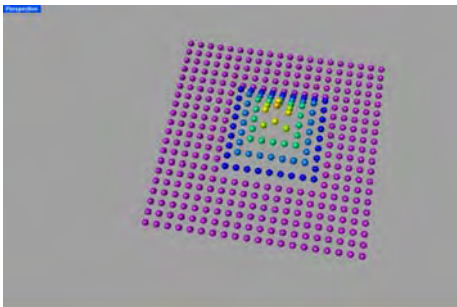
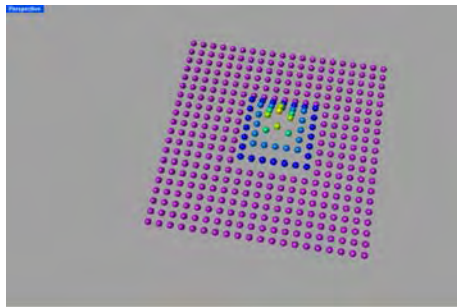
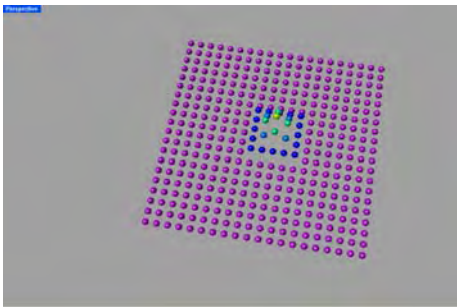
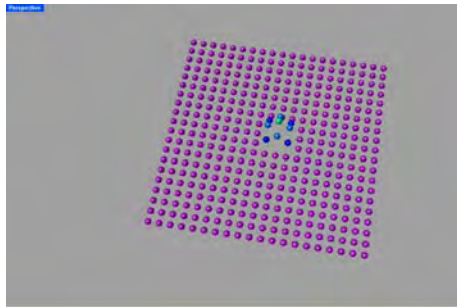
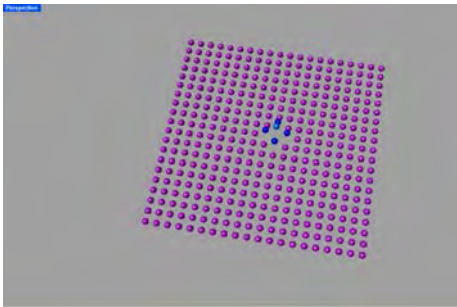
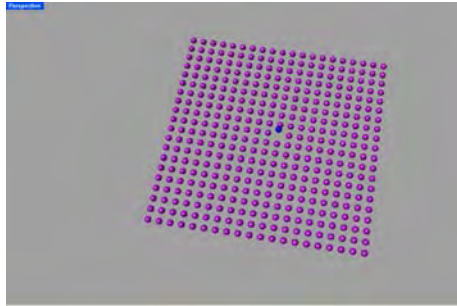
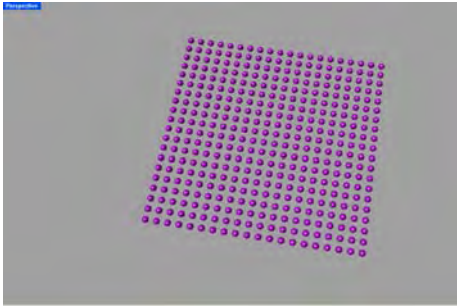


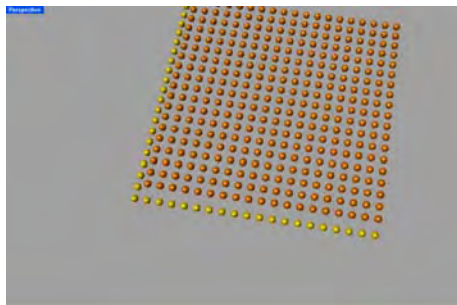
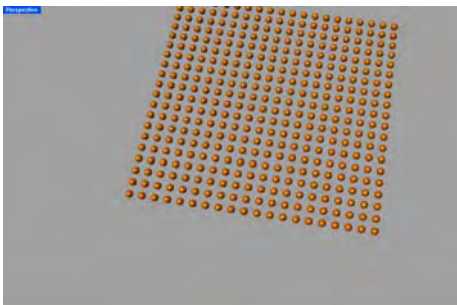
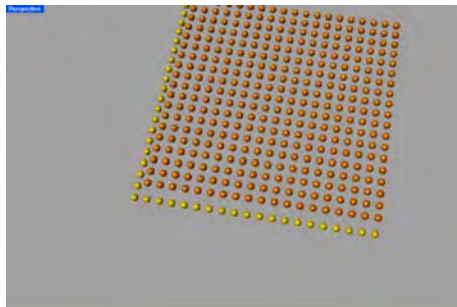
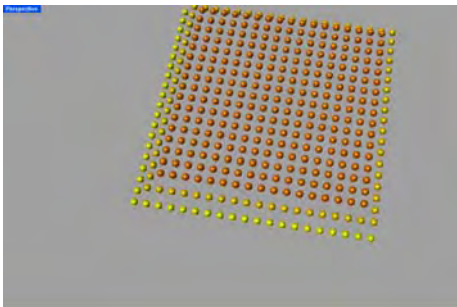
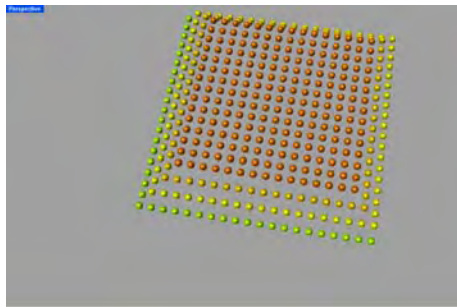
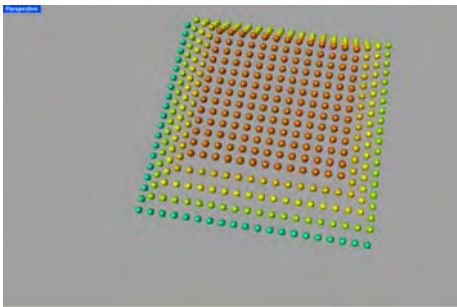
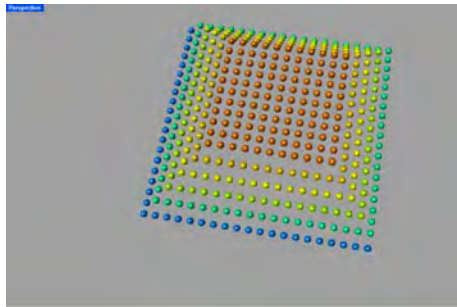
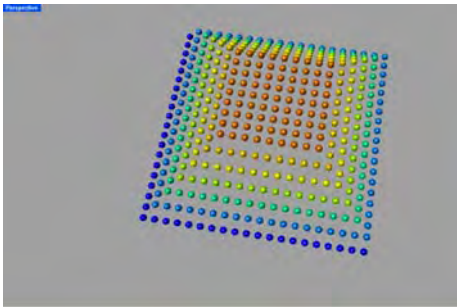
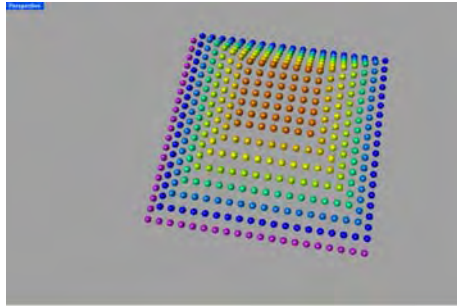
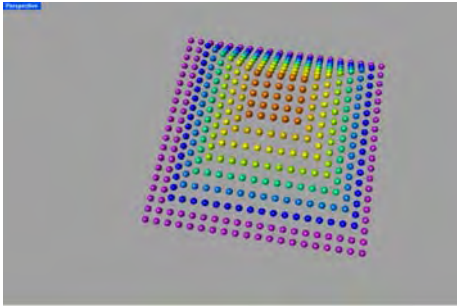


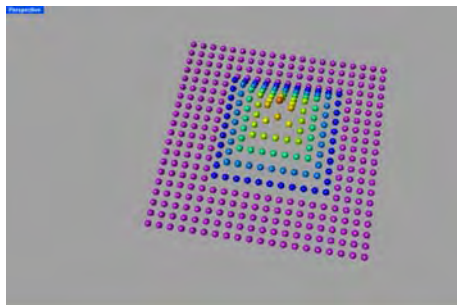
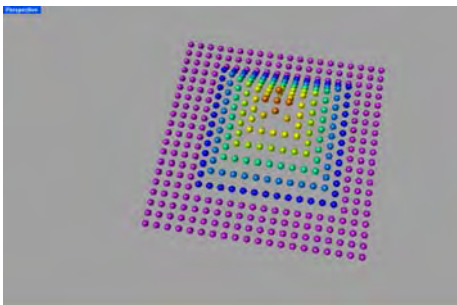
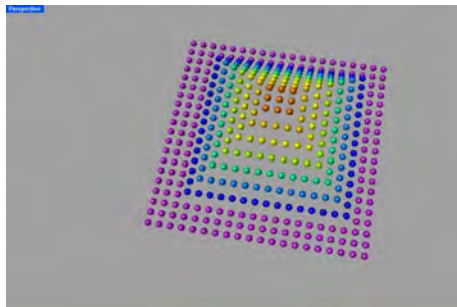
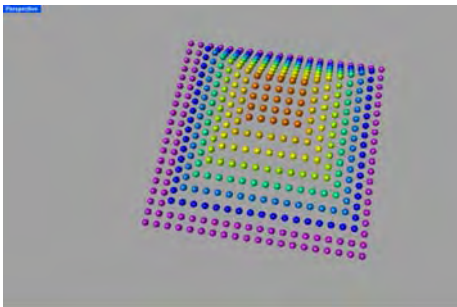
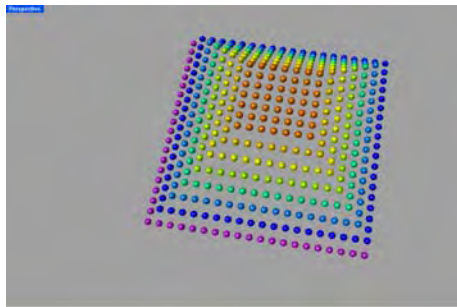
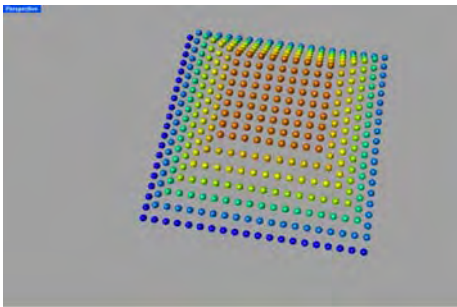
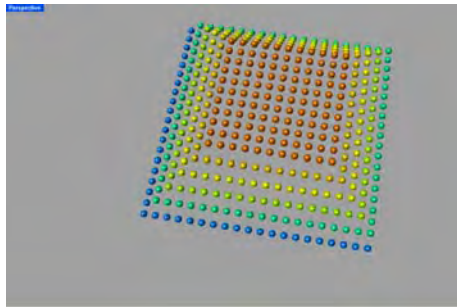
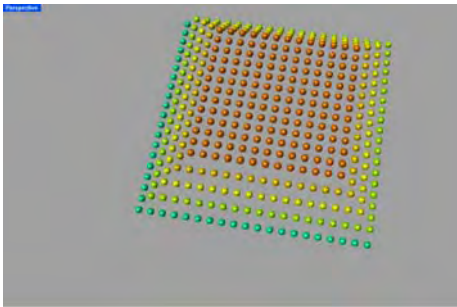
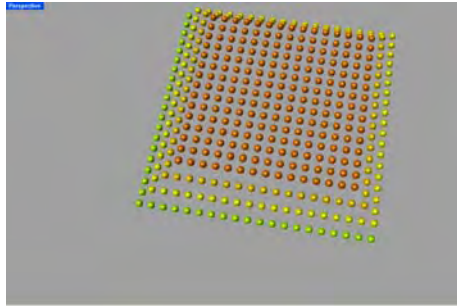
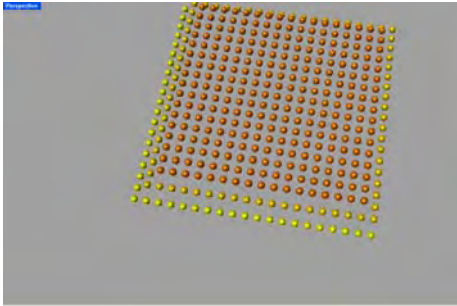
(Picture Series 1: Screenshots of surface movement when running script to generate surface one)

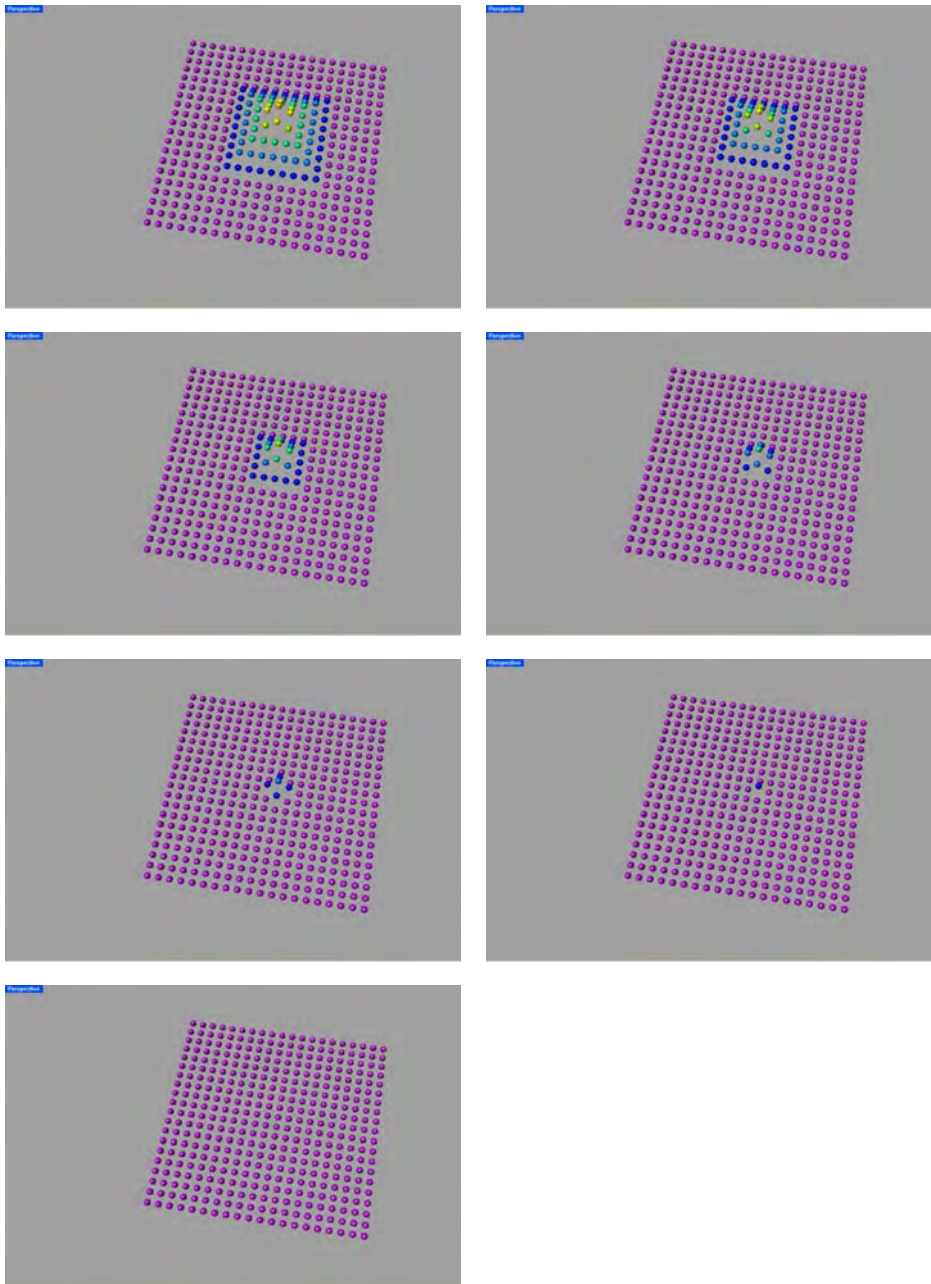
Drop form

The following images (*Picture Series 2*) illustrate the movement of the surface with a wave form:









(Picture Series 2: Screenshots of surface movement when running script to generate surface Two)

[5.3] Final script

Following is the code for creating a multilayered surface based on the two surfaces introduced in the previous Chapter. As mentioned in the introduction of this chapter both sets of data for creating the surface are stored in a remote file, and the location of

the file could be changed within Window Explorer when creating a multilayered surface based on other data.

```
' *****
' CREATING A MULTILAYERED SURFACE OUT OF A REMOTE EXCEL SHEET

' Script for Rhino 3.0
' Written by M .Hank Haeusler 04.02.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: Looping through x amount of sheets in a remote excel
' worksheet and create a multilayered surface
' STATUS:
' TO DO:

' *****
Sub MultilayeredSurface()
' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

' variables
Dim SurfaceFolderOne
Dim SurfaceFolderTwo
Dim MyXRowSurfaceOneAndTwo
Dim MyYRowSurfaceOneAndTwo
Dim arrOldObjects
Dim MyMultiSurfaceColor

Dim MyWorksheetCountSurfaceOne
Dim obj XLOne, obj WBOne, obj SHOne

Dim MyWorksheetCountSurfaceTwo
Dim obj XLTwo, obj WBTwo, obj SHTwo

' define image size in pixel + size of sphere
  MyYRowSurfaceOneAndTwo = Rhino.GetReal("How big is picture in Y
  di recti on?")

  MyXRowSurfaceOneAndTwo = Rhino.GetReal("How big is picture in X
  di recti on?")

  MyRadius = Rhino.GetReal("What is the radius of spheres?")

' SURFACE_ONE_EXCELCONNECTION*****
*****

' defines number of worksheets for SurfaceOne
  MyWorksheetCountSurfaceOne = 38
  For m = 1 To MyWorksheetCountSurfaceOne

' start of connection to Excel for SurfaceOne
  Set obj XLOne = CreateObject("Excel . Appl i cati on")
  Set obj WBOne = obj XLOne.Workbooks.Open
  ("S:\Students\Hank\01_RESEARCH\Surfacescripting\08_SCRIP TS\MS_Wave_0
  70205.xls")
```

```

Set obj SHOne = obj WBOne.Sheets(m)
' *****
' SURFACE_TWO_EXCELCONNECTION*****
*****

' defines number of worksheets for SurfaceTwo
MyWorkSheetCountSurfaceTwo = 3
  ' For n = 1 To MyWorkSheetCountSurfaceTwo

' start of connection to Excel for SurfaceTwo
Set obj XLTwo = CreateObject("Excel.Application")
Set obj WBTwo = obj XLTwo.Workbooks.Open
("S:\Students\Hank\01_RESEARCH\Surfacescripting\08_SCRIPPTS\MS_drop_0
70205.xls")
Set obj SHTwo = obj WBTwo.Sheets(m)

' *****

' Removes all objects.
arrObj = Rhino.AllObjects()
If IsArray(arrObj) Then
Rhino.DeleteObjects (arrObj)
End If

' makes a wireframe viewport
Rhino.Command ("_WireframeViewport")

' EnableRedraw - switch of screen to start / finish script
Rhino.EnableRedraw False

' SURFACE_ONE_CREATE_POINTS*****
*****

' Loop1 to create SurfaceOne out of txt file
For i = 1 To MyXRowSurfaceOneAndTwo
  For j = 1 To MyYRowSurfaceOneAndTwo
    ZSurfaceOne = obj SHOne.Cells(j, i).Value()

' creates sphere surface in Rhino for SurfaceOne
currentPointCoord = Array(i, j, ZSurfaceOne)
currentSphereOne = Rhino.addsphere(currentPointCoord, MyRadius)

' ColorForSurfaceOne
Rhino.ObjectColor currentSphereOne, RGB(255, 0, 0) ' red

' SURFACE_TWO_CREATE_POINTS*****
*****

ZsurfaceTwo = obj SHTwo.Cells(i, j).Value()

' creates sphere surface in Rhino for SurfaceTwo
currentPointCoord = Array(i, j, ZsurfaceTwo)
currentSphereTwo = Rhino.addsphere(currentPointCoord, MyRadius)

' ColorForSurfaceTwo
Rhino.ObjectColor currentSphereTwo, RGB(255, 0, 255) ' purple

If ZSurfaceOne = ZsurfaceTwo Then

' Function Multilayered surface colour
Overlapping currentSphereOne, ZSurfaceOne, currentSphereTwo,
ZsurfaceTwo

End If
Next          'next for (i)

```

```

        Next          'next for (j)
' *****
' makes a shaded viewport
  Rhi no. Command ("ShadedViewport")
' EnableRedraw - switch of screen to start / finish script
  Rhi no. EnableRedraw True
' call screengrab function
  screengrab (m) 'value out of MySpreadSheetCount
' Next for getting data of remote Excel data sheet for SurfaceOne
  Next          'next for (m)
' Quite the connection with the remote excel data sheet for
SurfaceOne
  obj XLOne. Quit
' Quite the connection with the remote excel data sheet for
SurfaceTwo
  obj XLTwo. Quit
' Tells that script is finished
  Rhi no. MessageBox "Script Finished"
End Sub

' *****
Function myColor(strObject, RGB)
' Function for Rhino 3.0
' Written by M .Hank Haeusler 13.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: create the colour for surface 2
' STATUS: WORK IN PROGRESS
' TO DO:
' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

Rhino.ObjectColor strObject, RGB

End Function

' *****
Function screengrab(m)
' Function for Rhino 3.0
' Written by M .Hank Haeusler 23.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: Incremental Screen Capture to file location
' STATUS: working
' TO DO:
' *****
' start of Rhino Connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object

```

```

Set Rhino = RhinoApp.GetScriptObject()
RhinoApp.Visible = True
'end of Rhino Connection
' *****
' function screengrab
Dim sPath

' Specifies Path location and file type
sPath = "S:\Students\Hank\01_RESEARCH\Surface
scripting\02_PICS\screengrab of multilayered surface\" & m & ".jpeg"

Rhino.Command "-_ScreenCaptureToFile " & Chr(34) & sPath & Chr(34) &
" Enter"
' *****
End Function

' *****
Function Overlapping(currentSphereOne, ZSurfaceOne,
currentSphereTwo, ZsurfaceTwo)
' Function for Rhino 3.0
' Written by M .Hank Haeusler 23.01.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: Incremental Screen Capture to file location
' STATUS: working
' TO DO:
' *****
' start of rhino connection
Dim RhinoApp As Object
Set RhinoApp = CreateObject("Rhino3. Application")
Dim Rhino As Object
Set Rhino = RhinoApp.GetScriptObject()
RhinoApp.Visible = True
'end of rhino connection
' *****

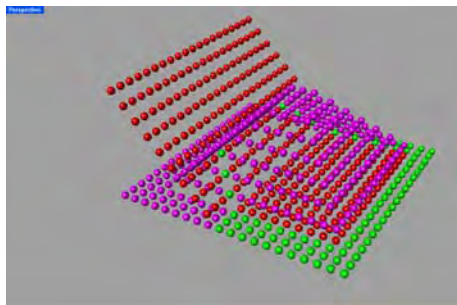
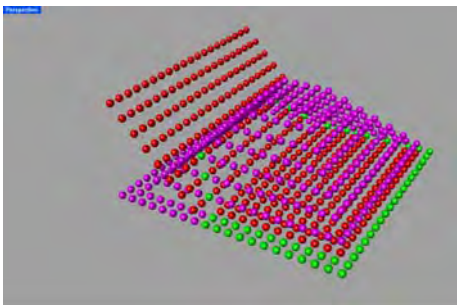
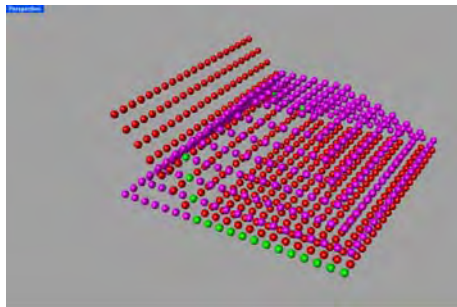
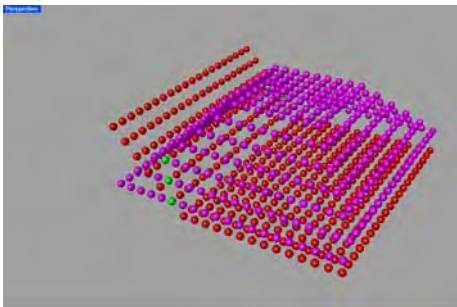
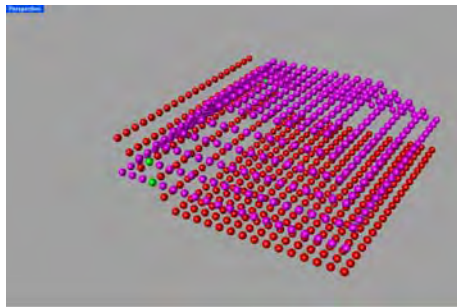
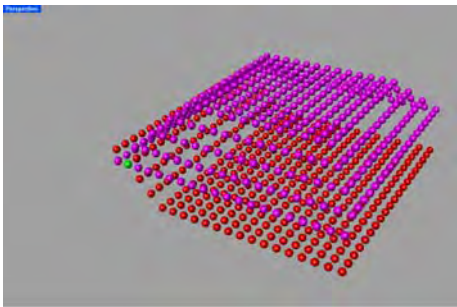
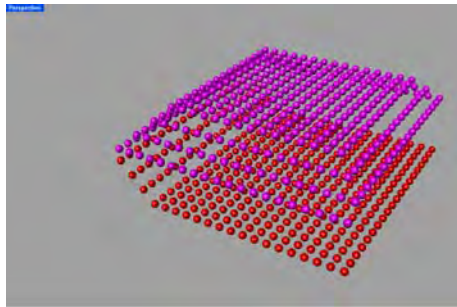
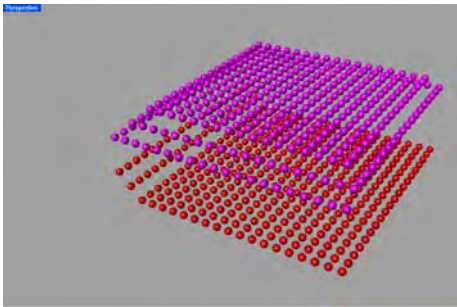
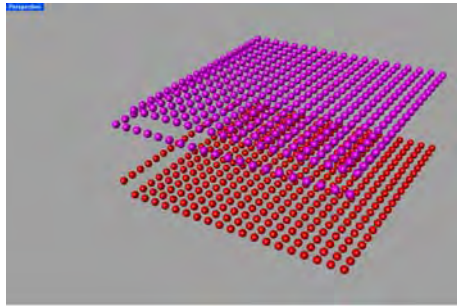
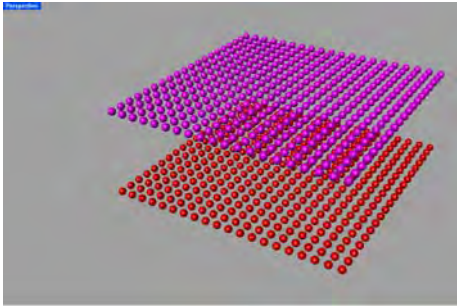
' rule for overlapping points out of surface 1 and surface 2
Rhino.ObjectColor currentSphereOne, RGB(0, 255, 0) 'green
Rhino.ObjectColor currentSphereTwo, RGB(0, 255, 0) 'green

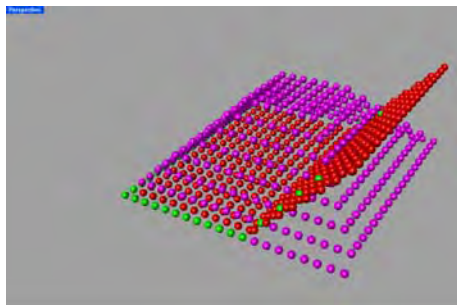
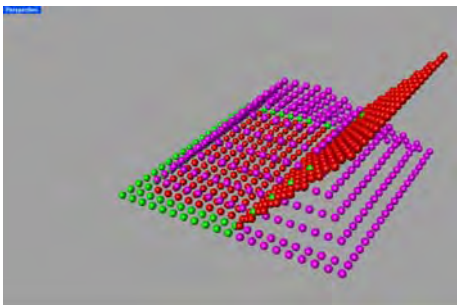
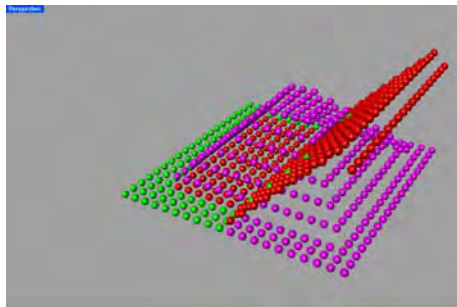
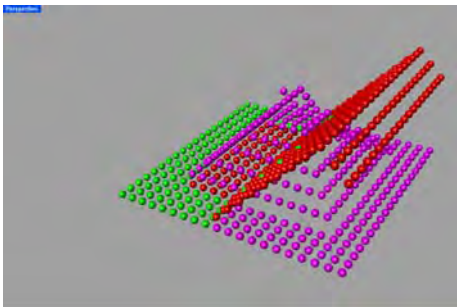
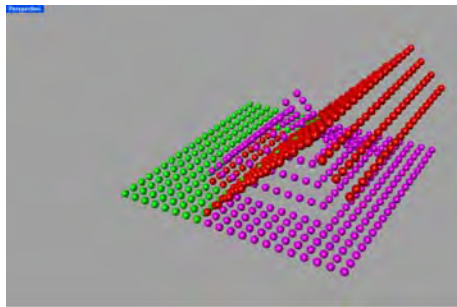
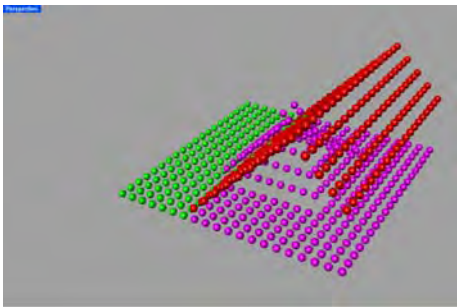
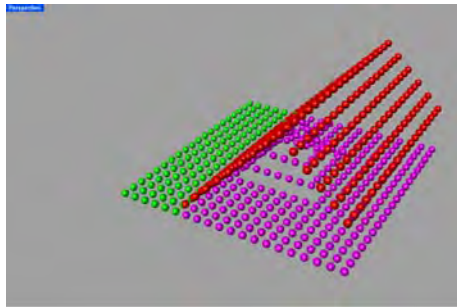
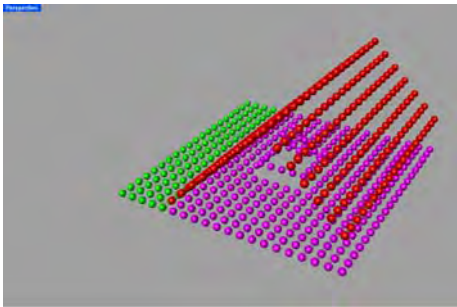
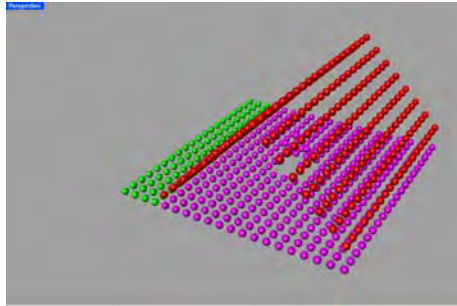
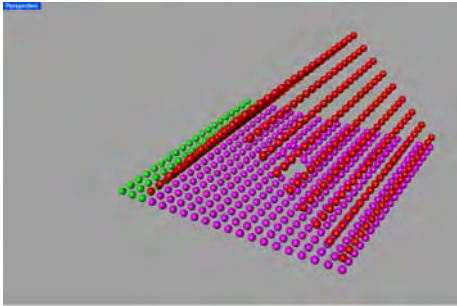
End Function

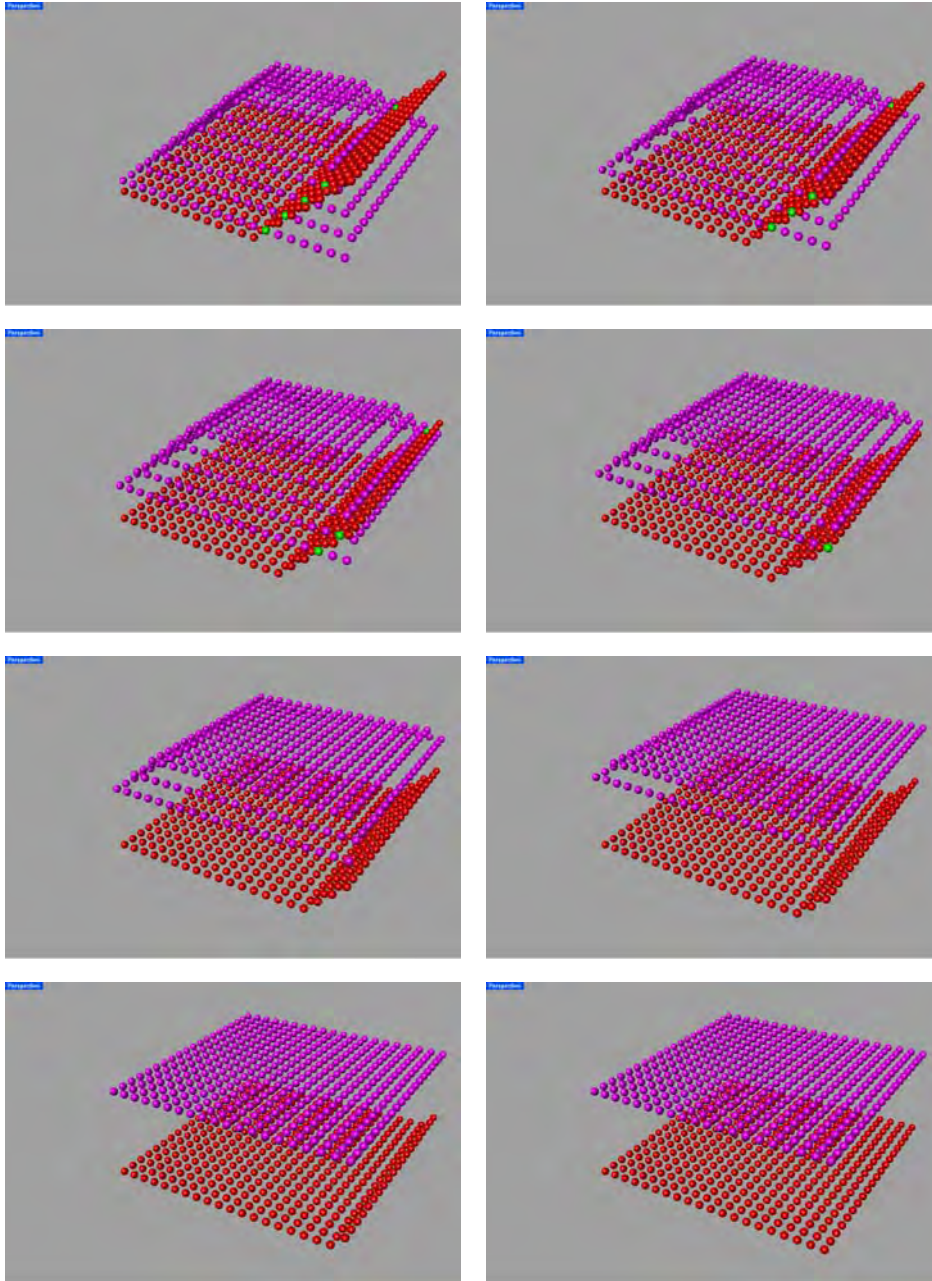
```

[5.4] Images

The following images show a multilayered surface generated from the two surfaces introduced in the previous Subchapter [5.2.1]. This time they have been unicolour with the drop form in purple and the wave form in red. When both surfaces overlap, the colour changes to green to symbolize where the connection of surfaces occurs.







(Picture –Series 3: Screenshots of surface movement when running script to generate surface one)

[5.5] Conclusion

When discussing a multilayered surface in Chapter 3 [3.3] the idea and notion of a multilayered surface stayed on a theoretical level, but here a multilayered surface can be presented when applying two surfaces. These two surfaces with their determinate movements give an idea of what could be done with a multilayered surface. The

potential of this has not been fully explored, but it was not the intention to deliver a state of the art multilayered surface installation in this exercise. The main focus has been on providing an artist or designer with a tool to create multilayered surfaces and for them to explore new methods of how to design multilayered surfaces.

[6.]
P4 /
Test Series V:
Plug-in for decay
function

[6.1]
Introduction

The script will function as a plug-in to create a decay function. Furthermore it will function as a prove if it is possible to create a decay function as it has been discussed in Chapter 3 in Subchapter [3.3.3.]. The decay function will be simulated with the size of a sphere which will symbolise the light and its colour.

[6.2]
Final Script

Following is the code for creating a decay function .

```
' *****
' CREATES A MATRIX OF POINTS WITH A DECAY FUNCTION

' Script for Rhino 3.0
' Written by M .Hank Haeusler 05.04.2007
' Matthias.Haeusler@ems.rmit.edu.au

' DESCRIPTION: creates a matrix of points and give them a decay
                function
' STATUS:
' TO DO:

' *****

Sub PointsFromImage()

' *****
' start of rhino connection
  Dim RhinoApp As Object
  Set RhinoApp = CreateObject("Rhino3. Application")
  Dim Rhino As Object
  Set Rhino = RhinoApp.GetScriptObject()
  RhinoApp.Visible = True
' end of rhino connection
' *****

' Variables
  Dim currentPoint
  Dim MyPointCount As Double
  Dim AllpointCoords
  Dim MyRowX
  Dim MyRowY
  Dim arrAllPoints
  Dim MyNewZ
  Dim arrOldObjects
  Dim MySheetCount
  Dim Loop1
```

```

MySheetCount = 38

' define image size in pixel + size of sphere
MyRowY = Rhino.GetReal("How big is picture in Y direction?")
MyRowX = Rhino.GetReal("How big is picture in X direction?")
dbl Radius = Rhino.GetReal("What is the radius of spheres?")

' redim code
newPointCount = MyRowY * MyRowX
ReDim AllpointCoords(newPointCount)

' Counts sheets in excel
For k = 1 To MySheetCount

' EnableRedraw - switch of screen to start / finish script
Rhino.EnableRedraw False

' Removes all objects.
arrOldObjects = Rhino.AllObjects()
If IsArray(arrOldObjects) Then
Rhino.DeleteObjects (arrOldObjects)
End If

' Loop for creating an image of one sheet out of excel
For i = 1 To MyRowX          'i gets X Value
  For j = 1 To MyRowY      'j gets Y Value
z = Worksheets(k).Cells(j, i).Value

' New Z value to create a depth in regards to greyscale
If z = 0 Then MyNewZ = 0
If z = 1 Then MyNewZ = 1
If z = 2 Then MyNewZ = 2
If z = 3 Then MyNewZ = 3
If z = 4 Then MyNewZ = 4
If z = 5 Then MyNewZ = 5
If z = 6 Then MyNewZ = 6
If z = 7 Then MyNewZ = 7
If z = 8 Then MyNewZ = 8
If z = 9 Then MyNewZ = 9

' creates point in Rhino
currentPointCoord = Array(i, -j, MyNewZ)

MyRadius = ChangingRadius(MyNewZ, dbl Radius)

' creates sphere in Rhino
currentSphere = Rhino.addsphere(currentPointCoord, MyRadius)

' call colour function
MyChangeColor currentSphere, currentPointCoord(2)

      Next      'next for (i)
Next      'next for (j)

' EnableRedraw - switch of screen to start / finish script
Rhino.EnableRedraw True

' call screengrab function
screengrab (k)

Next      'next for (k)

' Tells that script is finished
MsgBox ("Script Finish")

End Sub

```

```

' *****
Function MyChangeColour(strObject, ColourIndex)
' Function for Rhino 3.0
' Written by M .Hank Haeusler 05.04.2007
' Matthias.Haeusler@ems.rmit.edu.au

'DESCRIPTION: Changes the color of the layer
'STATUS: working
'TO DO: add parameters required in the () out of sub main to run

' *****
' start of rhino connection
Dim RhinoApp As Object
Set RhinoApp = CreateObject("Rhino3. Application")
Dim Rhino As Object
Set Rhino = RhinoApp.GetScriptObject()
RhinoApp.Visible = True
' end of rhino connection
' *****

' Color definition for 10 different rows
Dim mycolour00, mycolour01, mycolour02, mycolour03,
mycolour04, mycolour05
Dim mycolour06, mycolour07, mycolour08, mycolour09, mycolour10

' enhance with making it dependent on variable
mycolour00 = RGB(0, 0, 255)
mycolour01 = RGB(25, 25, 255)
mycolour02 = RGB(50, 50, 255)
mycolour03 = RGB(75, 75, 255)
mycolour04 = RGB(100, 100, 255)
mycolour05 = RGB(125, 125, 255)
mycolour06 = RGB(150, 150, 255)
mycolour07 = RGB(175, 175, 255)
mycolour08 = RGB(200, 200, 255)
mycolour09 = RGB(225, 225, 255)
mycolour10 = RGB(255, 255, 255)

' gets Z value from coordinate of cell and give it colour

If ColourIndex = 0 Then
Rhino.ObjectColour strObject, mycolour00
Exit Function

Elseif ColourIndex = 1 Then
Rhino.ObjectColour strObject, mycolour01
Exit Function

Elseif ColourIndex = 2 Then
Rhino.ObjectColour strObject, mycolour02
Exit Function

Elseif ColourIndex = 3 Then
Rhino.ObjectColour strObject, mycolour03
Exit Function

Elseif ColourIndex = 4 Then
Rhino.ObjectColour strObject, mycolour04
Exit Function

Elseif ColourIndex = 5 Then
Rhino.ObjectColour strObject, mycolour05
Exit Function

Elseif ColourIndex = 6 Then
Rhino.ObjectColour strObject, mycolour06

```

Exit Function

```
Elseif ColourIndex = 7 Then  
Rhino.ObjectColor strObject, mycolour07  
Exit Function
```

```
Elseif ColourIndex = 8 Then  
Rhino.ObjectColor strObject, mycolour08  
Exit Function
```

```
Elseif ColourIndex = 9 Then  
Rhino.ObjectColor strObject, mycolour09  
Exit Function  
End If
```

End Function

```
' *****  
Function screengrab(i)  
' Function for Rhino 3.0  
' Written by M .Hank Haeusler 23.01.2007  
' Matthias.Haeusler@ems.rmit.edu.au  
  
' DESCRIPTION: Incremental Screen Capture to file location  
' STATUS: working  
' TO DO:  
  
' *****  
' start of Rhino Connection  
Dim RhinoApp As Object  
Set RhinoApp = CreateObject("Rhino3. Application")  
Dim Rhino As Object  
Set Rhino = RhinoApp.GetScriptObject()  
RhinoApp.Visible = True  
' end of Rhino Connection  
' *****  
  
' function screengrab  
Dim sPath  
  
' Specifies Path location and file type  
sPath = "S:\Students\Hank\01_RESEARCH\Surface  
scripting\02_PICS\screengrab of multilayered surface\" & i &  
".jpeg"  
  
Rhino.Command "-_ScreenCaptureToFile " & Chr(34) & sPath &  
Chr(34) & " Enter"  
' *****  
End Function
```

```
' *****  
Function ChangingRadius(MyNewZ, dbl Radius)  
' Function for Rhino 3.0  
' Written by M .Hank Haeusler 05.04.2007  
' Matthias.Haeusler@ems.rmit.edu.au  
  
' DESCRIPTION: Change of radius to simulate decay function  
' STATUS: working  
' TO DO:  
  
' *****  
' start of rhino connection  
Dim RhinoApp As Object  
Set RhinoApp = CreateObject("Rhino3. Application")  
Dim Rhino As Object  
Set Rhino = RhinoApp.GetScriptObject()
```

```

    RhinoApp.Visible = True
    'end of rhino connection
    '*****

'gets Z value from coordinate of cell and give it a size

If MyNewZ = 0 Then
    ChangingRadius = 0.1 * dblRadius
Exit Function

ElseIf MyNewZ = 1 Then
    ChangingRadius = 0.2 * dblRadius
Exit Function

ElseIf MyNewZ = 2 Then
    ChangingRadius = 0.3 * dblRadius
Exit Function

ElseIf MyNewZ = 3 Then
    ChangingRadius = 0.4 * dblRadius
Exit Function

ElseIf MyNewZ = 4 Then
    ChangingRadius = 0.5 * dblRadius
Exit Function

ElseIf MyNewZ = 5 Then
    ChangingRadius = 0.6 * dblRadius
Exit Function

ElseIf MyNewZ = 6 Then
    ChangingRadius = 0.7 * dblRadius
Exit Function

ElseIf MyNewZ = 7 Then
    ChangingRadius = 0.8 * dblRadius
Exit Function

ElseIf MyNewZ = 8 Then
    ChangingRadius = 0.9 * dblRadius
Exit Function

ElseIf MyNewZ = 9 Then
    ChangingRadius = 1 * dblRadius
Exit Function

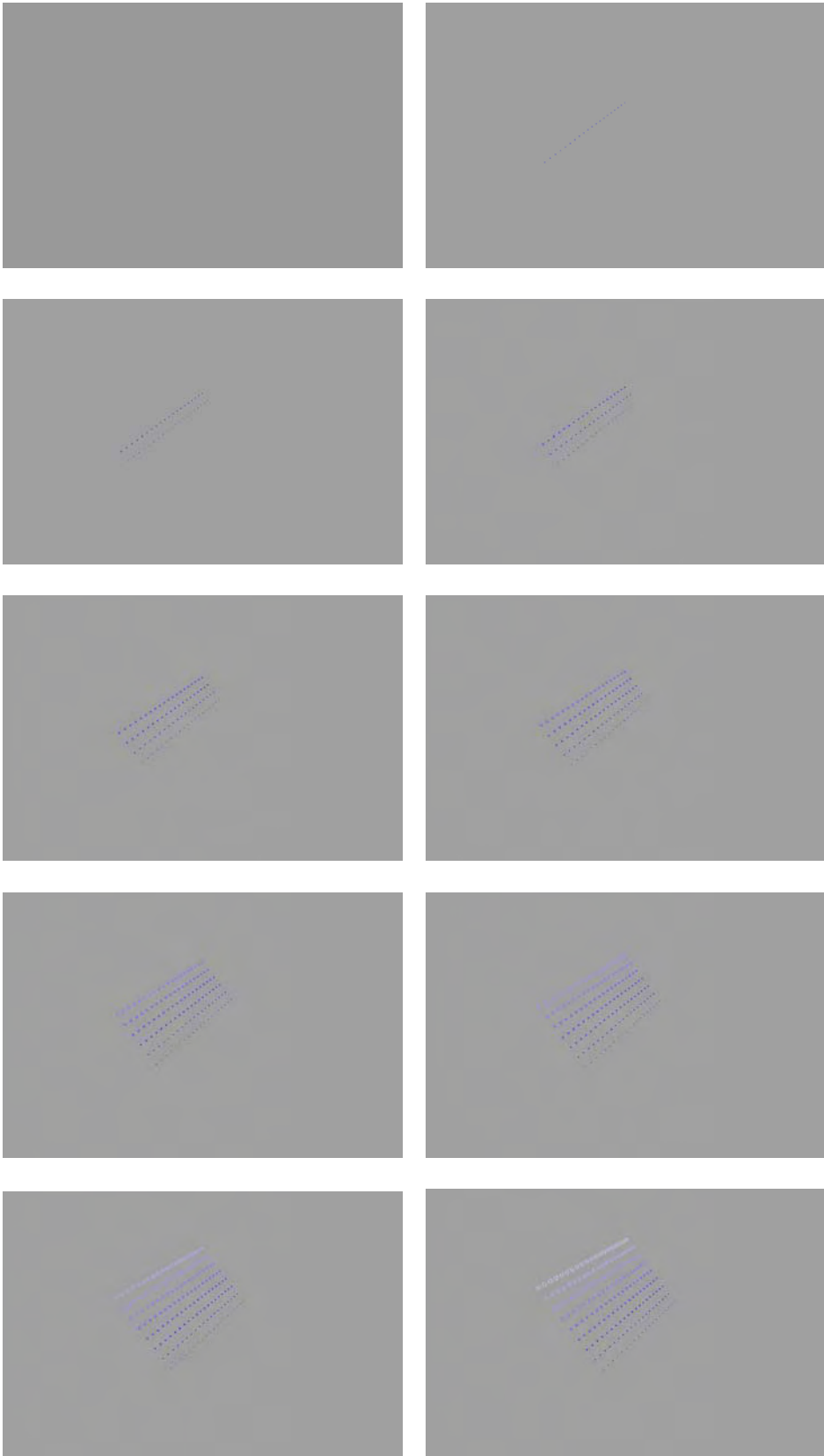
End If

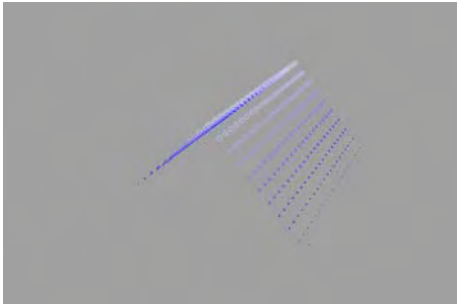
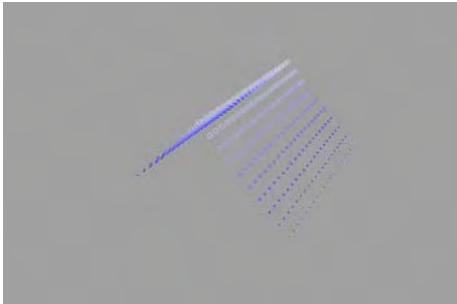
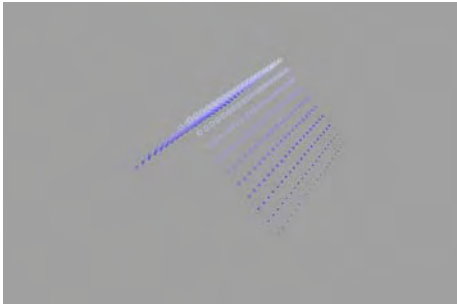
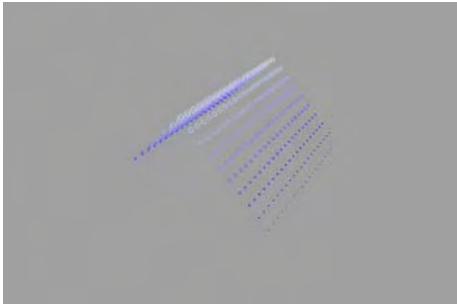
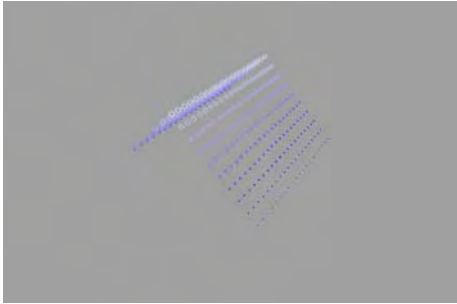
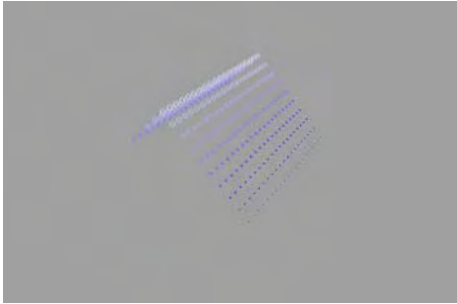
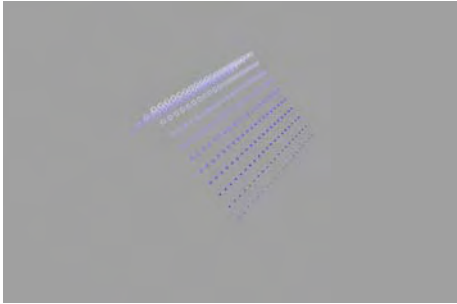
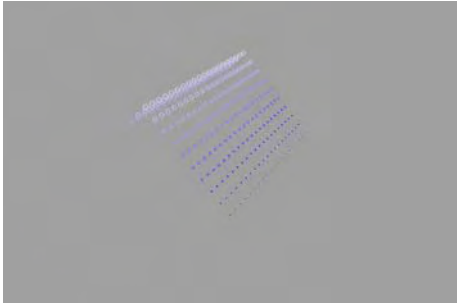
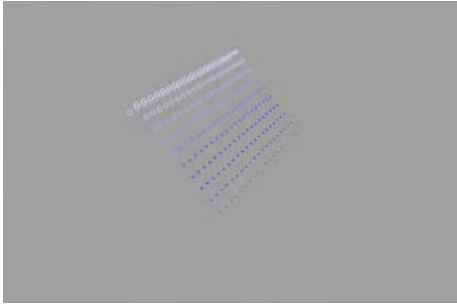
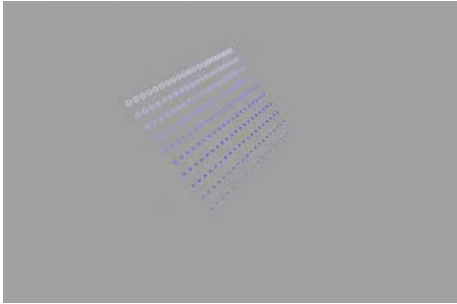
End Function

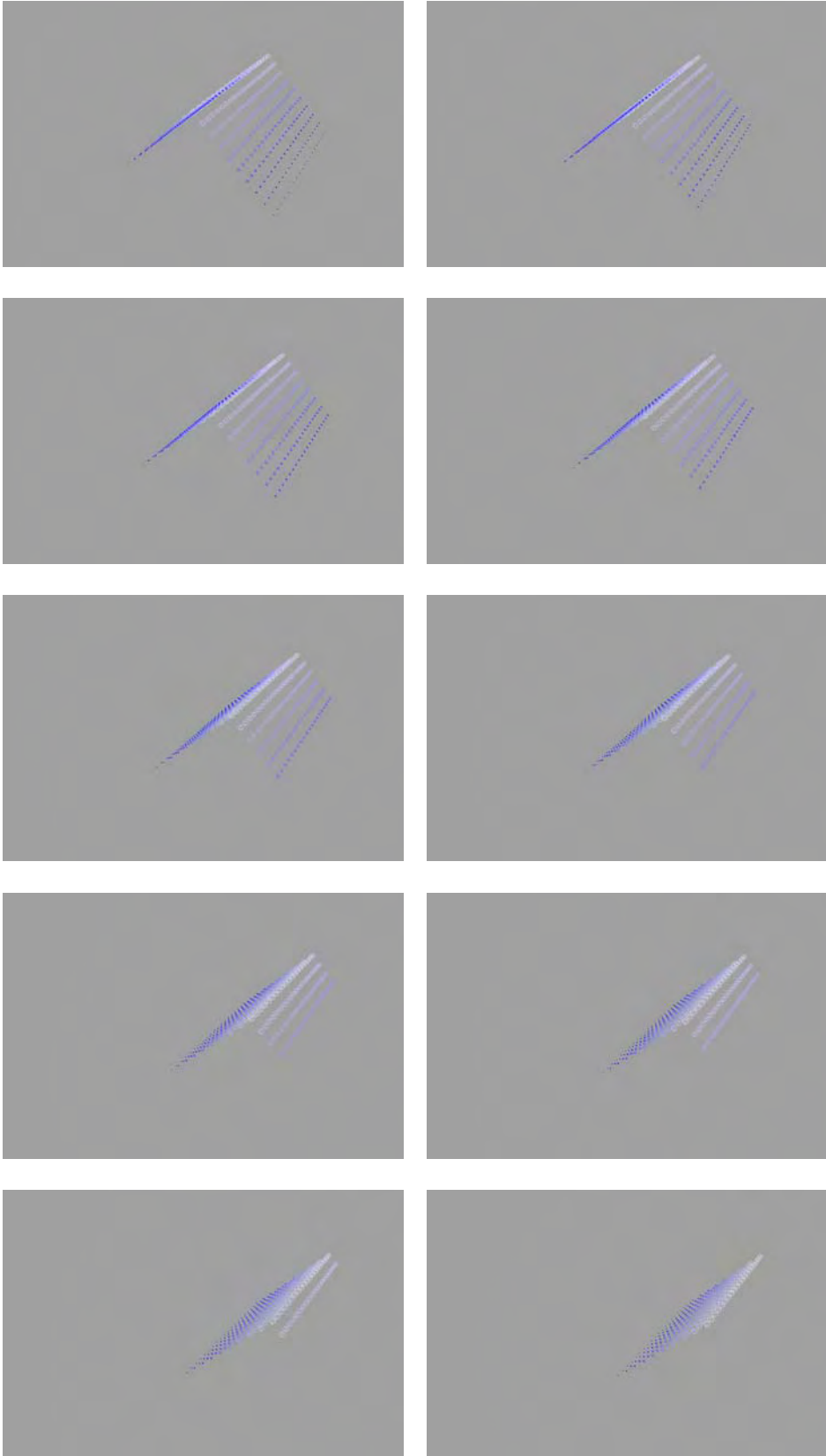
```

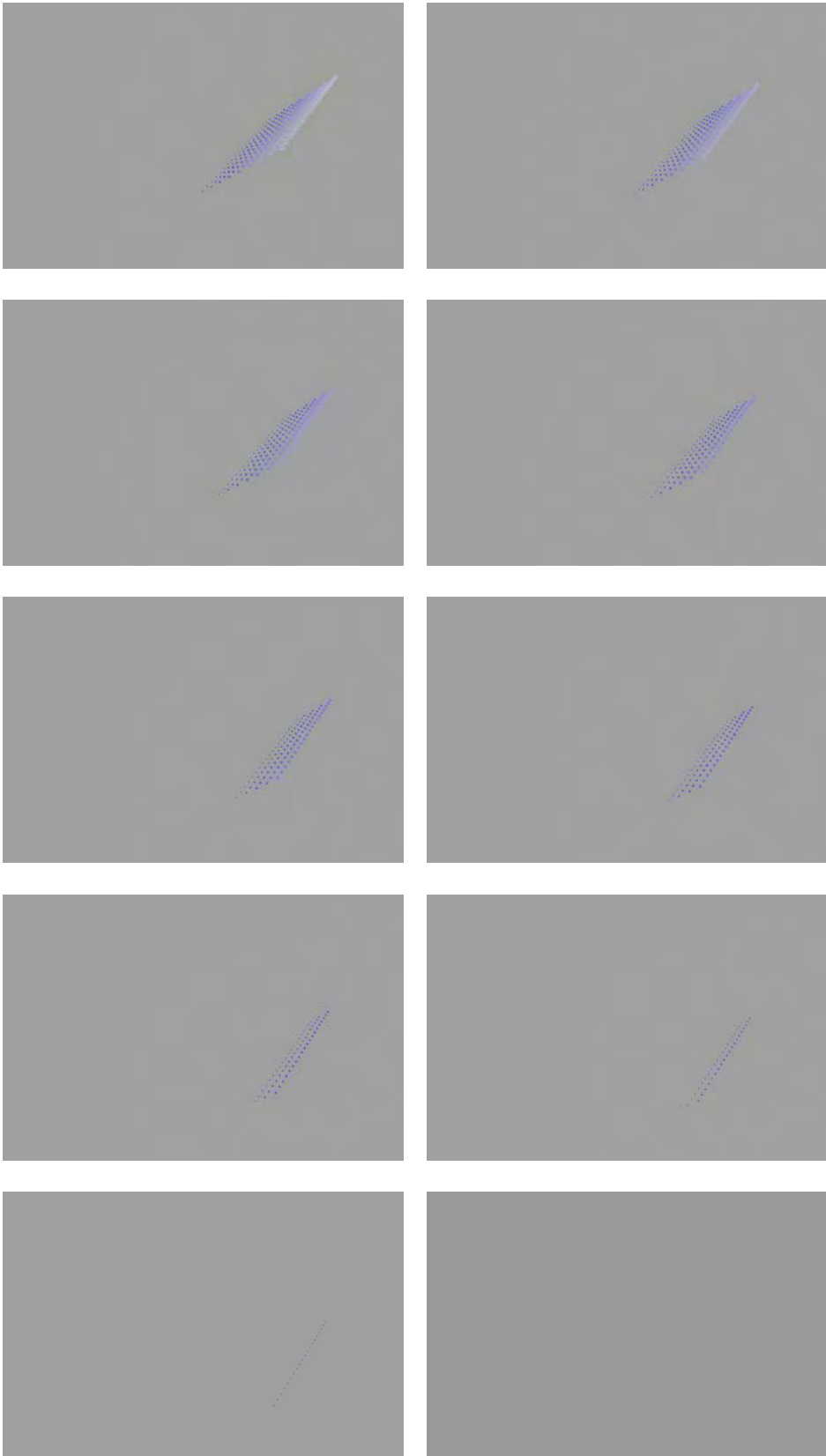
[6.3] Images

Following a series of screen shots captured while running the script to simulate a decay function by altering the size of a sphere.









(Picture Series: Screenshots of surface with a simulated decay function by changing the size of a sphere)

[7.]
P4 /

**Conclusion of Project 4:
Effect and use – working
as a curator for different
media contents**

Project 4 is the last of the four projects, providing a number of codes written in VB script in order to curate different media content designed by others. All these codes are written on an Excel platform and mainly use data fed from an Excel data sheet, with the exception of Test Series 4 where the input has been generated from a 3D scan. The use of Excel, a Windows program which enables the storing, working and analysing of data, shows the origins of potential media content. The Spatial Dynamic Media System is when using scripts able to visualize information as a spatial arrangement, viewable for the beholder and understandable in its spatiality. Excel as an input source was used when providing a script in Test Series I, which allowed the user access to all kinds of data which could be stored and saved in Excel. This includes input from various sensors such as temperature or wind among other allowing the system to react to changes in the environment, as shown by the example of water levels of reservoirs in Melbourne.

The tests have also, by the creation of a surface based on a movie clip as in Subchapter [3.] of Chapter 5 Project 4, proven that it is possible to weave together architecture and image with a result closely related to the original image but is more than barely a 3D representation of the image. Here again a code is provided for the artist to create movies and video clips by focussing on a new way of conducting a movie narrative. With a film tailor-made for the Spatial Dynamic Media System, the translation of the movement within the film will be the main focus. As my research in Test Series II has shown, when the position and colour information of a pixel is altered, the ability to create movement of a surface is enabled. It will not provide a straight translation of 2D form into a 3D form, ie. a 2D object in the movie such as a person will not be directly translated into a 3D object, but the colour information stored in the 2D object can be used to transform it into a 3D object. Thus the altering of a pixel's colour information consequently creates a movement.

The test in Test Series III originates in the proposition made in Chapter 5 Project 3, where the designed media content was based on the 3D scan of a face and the human emotion expressed via the face.

Essentially the conducted five tests have provided two general types of results: that it is possible to inject information into the system, and also to generate a moving surface in real-time. The written codes can be used as tools for ideas by artists and designers using the Spatial Dynamic Media System, as described above when discussing different tests such as the visualisation of information or the amalgam of form and image.

In addition the tests proved propositions made in Chapter 3. Here my research discussed a multilayered surface and a decay function for a surface, and the significance of these possibilities for architecture. Both options were discussed on a theoretical level in Chapter 3 Subchapter [3.3.], but the tests in this Chapter 4 Project 4 have proven that it is in fact possible to create such surfaces.

¹ <http://www.melbournewater.com.au/>, (accessed October 2006).