An Experimental Investigation into Deriving Procedural Creation of Abstract Images From Photographs

Innovations Project Report

Will Alexander - d1111966

1. Initial concept and idea

Fundamentally, my project idea was to develop a program that would create dynamic, complex and engaging abstract images, which exhibited effective relationships between forms, light and colour. Early on I had the idea of constructing theses images not from scratch, but perhaps using photographs as 'seeds' to the program, hence naturally providing a range of variation, randomness and structure for the program to use and build upon. Why do this procedurally? – to take advantage of the nature of procedural programming; allowing a lot of repetition and variation to create complex results. Implementing what is essentially a filter would allow, in theory, an unlimited number of exciting compositions to be created. Some of the initial art works that inspired me were those of Matta, Kandinsky and Turner. Having produced a lot of figurative, realism-based work during my studies so far, I realised that it would be a fresh challenge to try something completely abstract, and it would be ambitious for me as a goal.

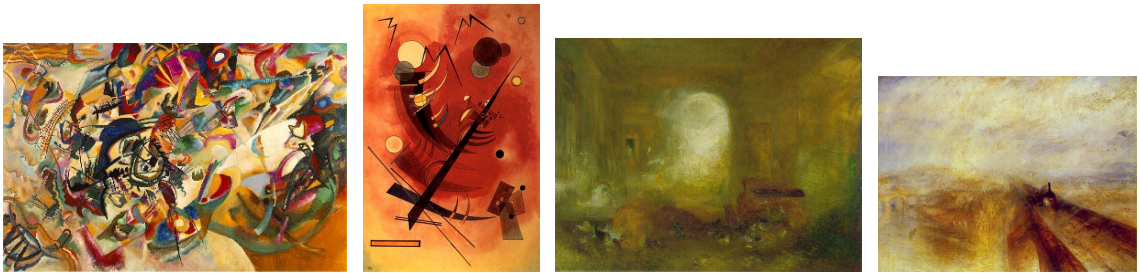
2. Development of Idea

To actually begin, I needed to ground the project down to some tangible goals. I was considering both the creative side – visual research and continuation of the idea, some practical experimentation to develop my thought processes, and the technical side – working out what I might need to research and explore, computer vision and image processing for example. Visually, I was looking mostly at the work of Roberto Matta:

Some examples of the paintings of Roberto Matta, which were the initial inspiration behind my idea

I was immediately drawn to the works of Matta upon seeing *Black Virtue* (left), because of their soft, flowing continuous look. Many of Matta's works can be seen at [9]. They are abstract images, and are unified into one, an anamorphous world of colour and light. The viewer infers a *space* in the paintings, they evoke a sense of volume, form and texture; the

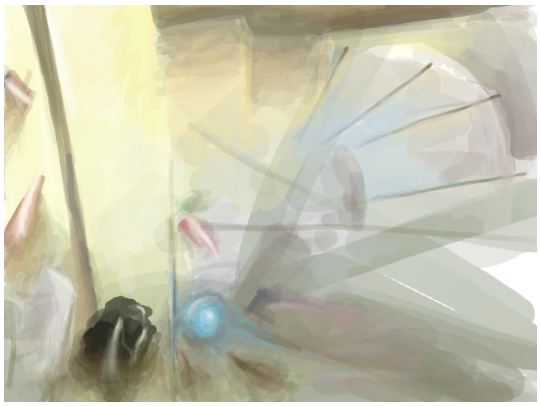
compositions are dynamic relationships, of which every feature is a part. The range of shapes and impressions of forms is really interesting – levels of translucency change around the space – open soft areas blend into solid looking surfaces that imply body parts and other forms. I also like the complex colour relationships – not just linear monotone mixes, but as in nature, a real diffusion of different hues and tones across the space. I backed up interest and observation in this area by looking at works of Kandinsky and Turner:

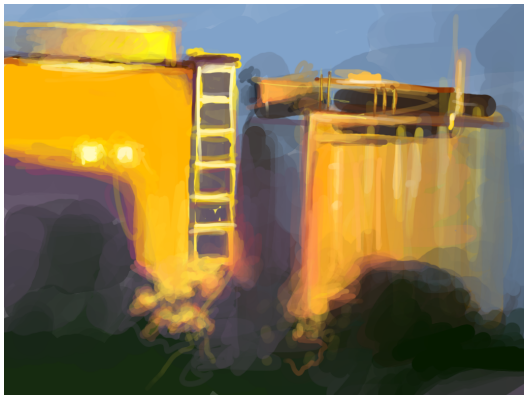


Some further inspiration and reference came from the work of artists Kandinsky and Turner

Much of Kandinsky's later abstract work has a really strong dynamic feel to it; every element of the composition moves the eyes along a fundamental direction of energy. Turner's works also inspired the direction of my project; the way Turner starts with reality, but completely romanticises it by submerging everything in a continuous, expressive space full of dynamic light and colour variation is akin to the process I wanted my works to go through. The resulting pieces by Turner are impressionistic – light and its impact are the subject of the paintings – objects and shapes fade and disperse into one another, producing a fantastic range and depth of colours and suggestions of form between.

After all this inspiration and research, I began to home in on an albeit vague aim: a self-contained program which would extract certain pre-defined types of information from an image: regions of brightness and colour, strong direction vectors in the image etc., and use these to create an abstract composition involving many of the visual ideas above. To try and observe my own thought processes tangibly, I tried manually creating some of my own compositions, based on my intended style, and found this very useful for uncovering some of my ideas; how do you break down an abstract composition into a series of logical steps?





Some examples of my simple manual experiments

In my own paintings, everything was based on the 'input' image. I tried to instinctively, without thinking too hard, pick out a handful of obvious principal regions in the image for each, and extrapolate, spread them out in a seemingly natural direction. I tried to back up these regions by 'fading them out' through different colours, to provide a diverse range of colours and tones. Where there were obvious dynamic lines, I manipulated other elements towards, or in the direction of, such lines. These large areas are supported by more 'scraggly', lively strokes driven by the rate of falloff in colour, and subtle features in the image. These trials helped me establish a few direct ideas:

- Start with a very simple, subtle base of just a few representative colours
- Take of the order of five to ten of the largest/brightest regions, and dramatically build them up on top of this
- Allow these and other smaller regions to be manipulated by existing dynamic lines in the image
- Although obviously not physically representative, I want the resulting output compositions to be faithful to the input in terms of tone, colour and 'activity'; subtle, subdued photographs will produce similar feeling images, while more diverse and dramatic photographs will also produce equivalent compositions.

An important part of the philosophy of this project was that the original image should dictate *everything*. This didn't just mean the colour range and contrast as I explained above, but that any decision made within the program should in some way be based upon the image. For example if there were effects being built up in layers, then the number of layers could vary to make differing results between images; this would mean somehow making a decision based on the image. Parameters to effects could be affected by the image: how large to make individual region shapes, or how pronounced dynamic effects are.

3. Getting Practically Started

At this point, I now knew some of the algorithms and techniques I was going to need to implement the project. Many of the areas such as image segmentation and image processing I new fairly little about, so I began researching the subjects in general by doing some reading, and looked into some papers on specific methods, for example image segmentation or detecting colour coherence for painterly rendering. I tried to not think too much about specific applications of the techniques I was reading about, but to just learn as much as possible to widen my 'toolset'. As expected, most literature on computer vision techniques and related areas has a very specific aim – realistic and totally correct recognition of information, such that it could be reconstructed. However for my purposes, all I needed was some interesting shapes and variation, total accuracy was not essential, so some of the more involved techniques that would involve getting distracted from my true aims, could be avoided. I reasoned that inaccuracy could almost be a bonus in my case; I thought it would be especially interesting to use several different techniques for the same purpose, e.g. region recognition, and then combining the results together. Having multiple different approximations to the same shape could, in the context of the kind of variation I was trying to achieve, be quite effective.

I knew by now that I wanted to be able to identify areas and regions in the image as 2-dimensional shapes. This area of research is called **Image Segmentation** or **Region Detection**, and is still very much an unsolved problem. After looking at some very advanced papers on this subject, for example [3], there were two problems. These were so advanced, implementing them would be a project in itself. On top of this, being papers, each had very little profound content in its own right, being built on top of years of research in some cases. Understanding the whole algorithm would have meant following numerous levels of references. I decided instead to read through some books, which actually start at the beginning. [1] was particularly useful for covering the range of concepts and possibilities within image segmentation. Basic image segmentation methods are based on one of two cores: *Splitting* or *Merging*. Splitting involves assuming that the whole 2D image is one recognisable region, then testing to see if this is really the case. If not, it is split in some way, depending on the algorithm. This produces more regions, which are also tested for coherency. The process continues recursively until all regions are assumed to be 'correct'. Merging algorithms are essentially the opposite – they assume that the smallest considerable areas (usually individual pixels) are all separate regions. Neighbouring regions are then compared and merged into a new, larger region if they satisfy some conditions depending on the algorithm. Splitting and merging are usually fairly crude, and the next level of algorithm uses both. A *Split and Merge* method involves splitting, and then considering the results for merging, before splitting again etc. But even within this, there is a range of algorithms, some outlined in [1]. The *quadtree split and merge* algorithm appealed to me most. It involves treating the whole image as a rectangular region, then, if the variance of the region is above a certain threshold, it is split into four children. This same process is then carried out recursively on all regions until the tree reaches a maximum depth. During the process, nearby regions are

considered and those deemed to be parts of the same region are merged. The algorithm was fairly skeletal in the book, and required a lot of practical development: my final region segmentation algorithm is described in **Appendix A**. Here are some of the results of the segmentation on images:

An example of my image segmentation algorithm detecting regions in an image

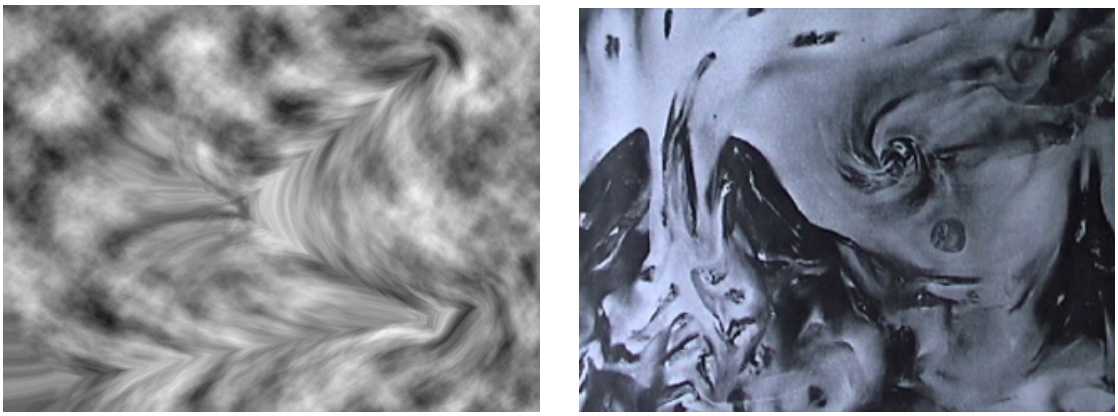
In order to pick out and include interesting details, I also wanted to implement some edge detection algorithms. Again, I knew very little, but there was a good broad discussion in [2] on the subject. Most edge detection methods, along with many other image processing methods, use the idea of a *convolution filter*. This is essentially a matrix that passes over every pixel in the image, describing how its filtered value is related to the pixels around it. In the case of edge-detection, usually each pixel only depends on its immediate neighbours. There is a range of similar filters, mostly involving subtracting neighbouring pixels' values from each other to obtain differential information (rate at which colours are changing at that point). The higher this differential, the more pronounced the change of colour in the image at that point, and the more likely it is to be on an edge. I implemented two different edge-detection filters; the more complicated examples can return other information such as the tangent and normal directions of an edge at any pixel.

At this point I feel I should address the issue of why I insisted on developing many of these algorithms myself, from scratch, when many are possibly available as usable code and APIs. My justification is that in the general context of learning, actually developing the code for the techniques is much more beneficial. On top of this, as I researched many image processing ideas, in a wide variety of fields from edge detection to cellular-automaton simulation of watercolour, I felt my intuitive understanding of, and capacity for ideas for, images and colour and shape widening dramatically; I was making notes of dozens of ideas. I would argue that this was invaluable later on in the project.

4. More Algorithmic and Visual Ideas

I found, given the intangible nature of the project, that knowing where to start was difficult, I was trying to think about everything at once. I was having so many ideas, from specific visual effects to general concepts of how the system might works as a whole. So I tried to keep track of all this by making extensive notes of all my ideas, which I could refer back to later. I reasoned that plunging in and doing research and experimentation would help shape my idea and help me think about where to go next. Having the specific challenge of implementing these routines kept me motivated, and the results, as I had hoped, began to clarify for me how I might use them and put the final working program together.

A key development that I made was a system to help bring dynamism to my compositions. As described above, an exciting and coherent dynamic dense of *flow*, present in Matta, Kandinsky and Turner's work, was something I wanted to bring to the images. I came up with the idea of extracting a small number of principal straight lines from the image, and then using these to drive a 'vector field', which could be queried at any point in the 2D image. This could then by used to create a sense of motion in many different ways. I decided a good way to create such an effect could be to treat the lines as magnets, propagating parallel forces into the space around them. The longer the line, the stronger the forces it causes. The overall 'force' at any given point could be computed by summing the forces of the lines in the image, using their distance as a falloff. For a detailed explanation of the algorithm, see **Appendix B**. An example of how the magnetic field looked on its own is shown here on a plane of noise for clarity:



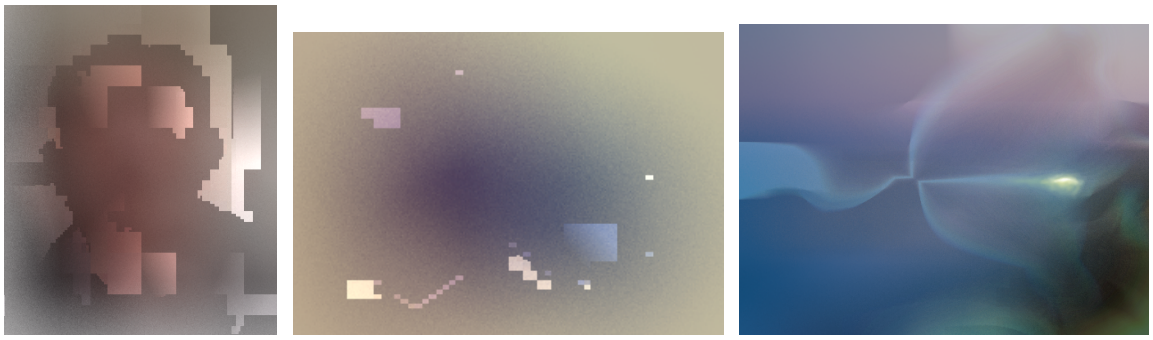
The effect of a series of magnetic lines on some noise (left), with an example of a similar effect in Matta's work (right)

Producing this magnetic effect made me think a little about the philosophy of the program as a whole – here was a mechanism that could be used several times in conjunction with different 'elements' to produce a really varied and dynamic result.

5. Consolidating the Tools to Create Visual Results

As I began to end up with a set of usable tools, I also developed a framework for dealing with images, to make the whole process easier. I wanted to start using the tools to actually create something, and I began a series of early tests, using my code for its purpose. While individually these were reasonably successful, I was still having trouble seeing the big picture. At several points throughout development, I re-evaluated my method, by taking a step back and trying to think about how my high-level solution would work. Each time, I started from scratch in terms of how I would control the functions and the image, and each time the result was better organised, since I was now better informed as to what did and did not work, and what tools I had to actually work with.

With the image segmentation, edge detection and dynamic field, I had a small set of tools, and sources of information, which I could begin to use creatively. One of the key elements of Matta's paintings that I liked was the contrast between open-feeling spaces and solidity. I thought that to achieve this I could use my region-detection routine, and isolate the particularly large regions, then build these up from a soft background by continually blurring and adding them together, or fading them out in certain directions, or manipulating them with the magnetic field, forming a transition from the background into a very outstanding and overt shape. This result would, I thought, capture some of the most important elements of the works I was trying to emulate. After a few experiments, I had some reasonably promising results:

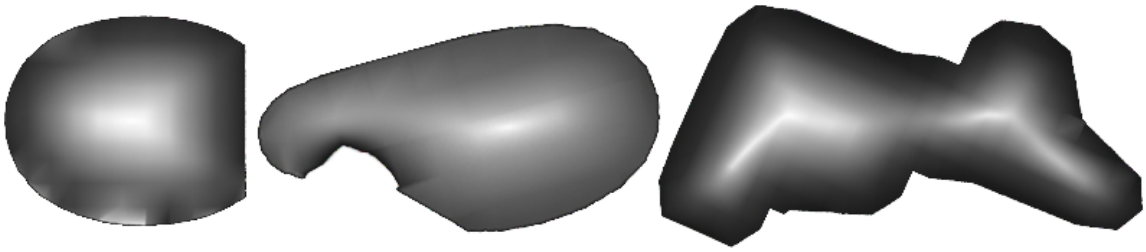


Some early tests of various combinations of the tools I had so far

Even with the fairly limited toolset and specific visual goal, it was nice to see that running the program on a range of images produced some quite unpredictable results.

Some of the 'solids' in Matta's work are less subtle and extremely overt, one can make out actual 3-dimensional forms. My solution to generating such an effect was to take the region detection tool that I already had at my disposal, and extrapolate some of the resulting regions into 3D shapes. These could then be rendered, and combined into the composition. Researching such an algorithm could have been a difficult task; converting a quadtree-structured shape into a smooth piece of 3-dimensional geometry is

quite a specialised requirement! I felt confident enough in my own problem-solving ability to design a method myself. Firstly, I needed a method for converting a region, which comes out of the quadtree as an incoherent group of rectangles, into a single outline. Then from this, an algorithm for extrapolating into a smooth, blobby 3-dimensional form such as those seen in Matta's work. After several stages of attempts, I ended up with a successful method, the results of which can be seen below:



For a full description of the algorithm, see **Appendix C**.

6. Establishing an Overall Working Method

At numerous points throughout the project, I had given thought to how the program would work as a whole. Having several interesting visual results was good, but how to put everything together to produce a final self-contained program that would work consistently and variedly with a range inputs, and produce results that looked exactly as I intended, was a challenging idea.

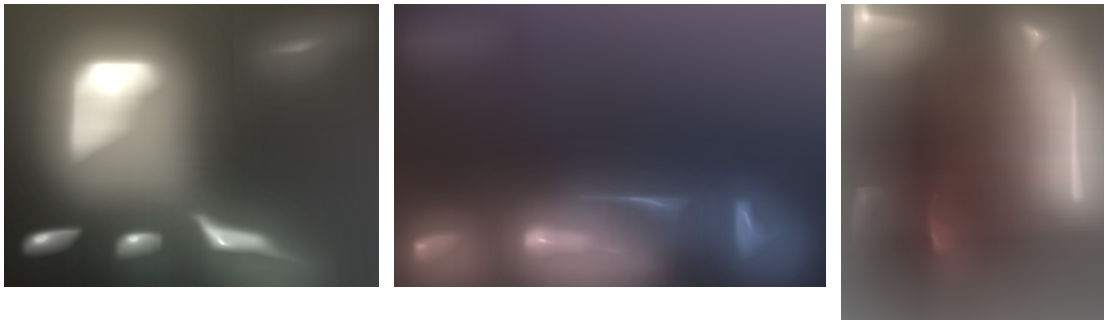
One such idea was slightly removed from the course of the project so far: I considered setting up every 'effect' as a module or node, which would depend on as many or as few 'attributes' of the image or previously generated effects as it likes. These nodes could then be linked together at random, to create all sorts of unpredictable and more importantly *complex*, inter-dependent results. This was in a way linked to genetic programming, and I considered going down that route, allowing a user to choose from a variety of different dependencies and node combinations. However I decided not to pursue this possibility for a number of reasons. After researching genetic programming, it seems a fairly complex field, which would require too much time to introduce at this late stage in the project. Another reason for keeping a distance was the question of whether I actually wanted the results to be *that* unpredictable, or for the 'user' to have to intervene and make choices; the original intended outcome of the project was to create images closely resembling the works I had looked at. So as a result of this consideration, I decided to adopt a more straightforward approach.

Another possibility that I considered was implementing several modular 'stages' to the program. Each stage would evaluate 'attributes' of the image, such as a 'dynamism'

value, and a ‘constrast’ value. It could then use these attributes to make decisions about what to carry out. Other modules could follow, taking more information out of the composition so far, to make more decisions etc. However, by the time many of my features were working, their success made me think more clearly and simply, so I decided to try instead to build the whole program up like a painting: get the fundamental shapes and composition working first, then build up with complex and dynamic detail, all using the now extensive toolset I had. I would be able to keep adding features and effects to the system, augmenting it as much as I could in the available time. This also allowed me to be a little creative; some of my earlier results had shown me that once one begins coding to process images, there are many possibilities and results that can be followed up, that had not been planned. I wanted to not necessarily be completely tied down to the artworks I was inspired by, but for there to be room for my own input as I developed the effects.

7. Developing the Final Program

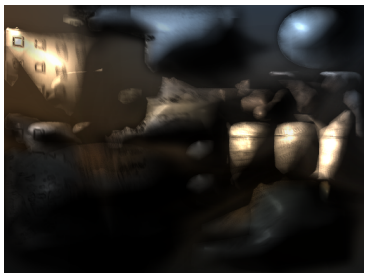
As described above I decided to press ahead, building up my program in layers, setting intermediate goals after which I could evaluate and decide where to go next. I decided that the starting point should be to take a very blurred version of the original image, removing the brightest colours, as the starting layer. Then identify the handful of largest regions, about 5, and gradually and softly build these up on top of the background in decreasingly blurred layers, so moving from an empty space to an impression of greater thickness and shape. The blurred layers would take the region’s average colour combined with a randomly selected colour from within the region, providing already a nice range of colour variation across the composition. Because each built up region takes its colour from its area of the original image, the brighter regions are more pronounced, the duller ones more subtle, so the result varied, and faithful to the original image. At the ‘peak’ of each of these built up regions, I render the region as geometry, to give that impression of surface. The bright specular highlight on the geometry is the key element that ‘sells’ the shapes as a subtle suggestion of surface. This would result in quite an interesting composition, on top of which everything else could be built. As I developed, I always tested the algorithm out on *several* input photographs to ensure, as far as possible, that the method was a general one that would work and produce the desired result in many situations. I found that the results were usually consistent across several examples, but interestingly, there were always some that gave more unpredictable results. Here are some of my initial tests into this first layer:



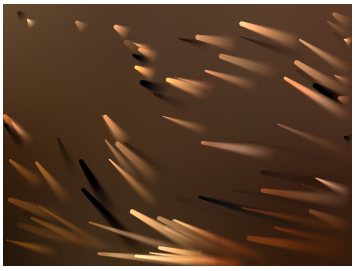
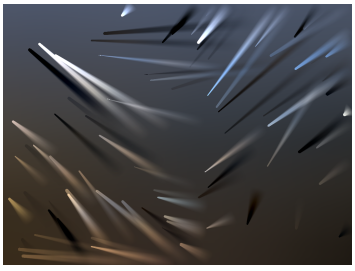
To further enhance the impact of the regions, I also replicated one of my more successful earlier tests, namely using the magnetic field effect on each layer to give the result a painterly, swirling, flowing feeling. In keeping with the philosophy that every element and decision in the composition should be derived from the input image somehow, I generated the magnetic field's driving lines by creating a line from each region to the space between two random others. This also had the advantage that the field would appear to be related to the region shapes, bringing coherency to the effect. As an interesting aside, when allowed to dominate, this effect created some nice results in its own right. When applied in a more subtle way, the results of this layer brought a lot to the compositions:

At this stage, I felt that the compositions needed more detail, something to break up the total smoothness present. I started to think about the beginning of the end, trying to move towards a final result. Looking at the artists' work again, they seemed so much more complex than my results. I realised that I needed to be a little less conservative; so far, I had very tight control over what the program was creating, but the complexity and excitement of some of Matta's work could not be thought out so methodically at every stage. I realised that the various layers and effects needed to merge together and be part of a whole, and not obviously discernable; the resulting images would not work if they were all clearly a formulaic combination of three algorithms. I decided that I should create a more general effect for the next stage, based perhaps on several of the visual elements I wanted to convey, and also very broad so that it could be 'added on' multiple times to produce a varied cumulative effect. It made a lot of sense to get the most out of the complex routines I hade developed so far – the image segmentation and geometry construction among other things had taken a long time, and so far were underused. For example, I found that some of the smaller regions produced were interesting shapes that

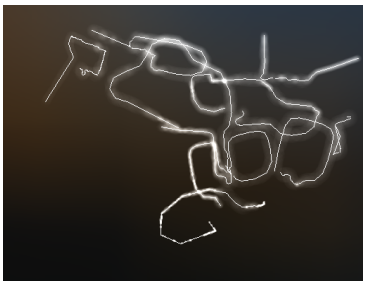
could really add to the composition. I realised as I started combining these with other existing algorithms: magnetic fields, lines etc, that I had barely scratched the surface of what could be done by combining these tools. The result was a ‘general’ layer which created geometry from a large number of regions in the usual way, and allowed these regions to be added on, or to be ‘multiplied’ by the current composition. Regions are also manipulated by magnetic fields and are blurred. Intentionally, there are several parameters to this routine, the idea being that the controlling function will call the apply the effect several times, varying the parameters each time:

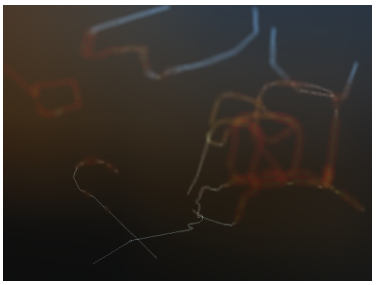



A number of more specific effects were developed as well. I wanted to include some direct straight lines in the composition, because of their strong directional graphic effect. Lines were generated by picking a random point in a magnetic field as a start point, then following the force vector at that point, to create an end point. This resulted in quite an effective dynamic visualisation of the field. In order to avoid the lines appearing like a separate entity, they ‘taper off’ toward their rear end, to blend seamlessly with the rest of the composition:

I felt that more scraggly and random ‘stringy’ lines were also necessary. To generate such artefacts, I found another good way to get more out of the methods that I already had in place. The intermediate result from the geometry creation algorithm, is an outline with nice shape to it. This outline can then be rotated and scaled to slightly disassociate it from the regions geometry. Varying the thickness of these strings and burring them ‘strings’ depending on the brightness of the image below, gave a range of results from bold lines to flickering suggestions.





This latter stage of the project was in some ways the most enjoyable part; I had developed all of the complex tools and algorithms I needed, and I was now able to use them to create all sorts of dynamic relationships. I realised that all the research and hard work early on was worth it, to give me so many workable options and information later on, although inevitably I had to write some new routines at certain points. I did also find this final stage quite slow, capturing some subtle effects in code and in a general sense was hard, but a great challenge.

8. Succinct Description of Final Production

I decided to use the C programming language for development, since I was most comfortable with it, and I used the Image Magick Library for dealing with importing and exporting images. I developed a simple set of data structures and functions for using images, which I stored as unsigned bytes. The fundamental structure of the program is as follows:

- Import input image
- Compute a set of information about the image – for example edge detection, raster and geometric results from image segmentation. Return this through a data structure
- First a soft blurred background is applied.
- Smear function called with a handful of regions, to add a dynamic swirling effect to the background
- Region build up function called with similar regions, to give a transition from soft empty background to hard suggestions of solidity
- Several detail-producing functions called, the parameters of which are derived from the input image itself, to produce varied results
- Extra effects such as straight and scraggly lines are added
- Export the image

Demonstration of the layered progression of images

9. Conclusion and Evaluation

In conclusion, I would say the project has been mostly a success. It has challenged me even more than I expected, in the areas I wanted to be challenged, and I am quite satisfied with many of the results. The nature of the project was certainly different to most programming that I have done before; the lack of singly defined logical or even visual goals was a sharp shock for me, and I've enjoyed and learned from the very experimental process which resulted. The focus and process of what I was doing did begin to evolve as the project neared completion, I realised that trying to have one single 'filter' at the end was pointless, given the sheer number of varied effects that I was trying to capture. My work became very much an experimental process of attempting differently styled compositions, brought about by concentration on different specific effects. I also let the project go in its own direction to an extent – letting incorporating ideas from results that were unplanned and unexpected. Given this, I have been pleased with the results; I have successfully managed some of the specific features that I was trying to create, for example the blending from soft colours to a subtle suggestion of volume and form. Many of the compositions that I was creating toward the end really captured, I thought, my fundamental goal of a feeling of empty space, solidity and *three dimensions*. Compositions are also fairly faithful to their input images, where colour and dynamism are concerned.

However, in all honesty I would say that the one distinct failure, if any, was one of my concerns throughout: tying all of these aspects together effectively such that they are all apparent and have their desired effect, but also produce a successful composition overall. On top of this, I didn't manage much substantial work into varying the significance and 'parameters' of the various effects/layers based upon the input image – many of my results are similar in style and scale of effects, not *getting the most* of the effects.

I feel in retrospect that I have only begun to scratch the surface of what would be possible with this idea; not just in terms of different ways of going about my aims or different effects entirely, but even in that the ways I am combining the algorithms I have already devised could have gone much further. In terms of how I might approach the project were I to attempt it again, I would try to carry out more extensive research into the area of abstract image creation, better feeding my imagination and awareness of existing ways of thinking. An obvious question of difference of method were I to change the way I worked would be the way I developed everything from scratch, rather than drawing on existing resources. Some complex routines, such as region detection, could have been done 'for free', but I am pleased that I developed those as I did, because the aim of this project was to challenge myself in an area that I knew little about, and researching and implementing those algorithms myself allowed me to do that.

Future work could, I thought, involve creating moving images, perhaps through processing a whole sequence of images. But quite an exciting idea would be to use the effects that I have created, and generate sequences from those. For example allowing the magnetic fields to actually move colours and shapes over time, or taking advantage of the three-dimensional shapes by moving that camera viewpoint around and through them, creating captivating *moving* compositions just from one image.

[1] JAIN, R., KASTURI, R. AND SCHUNCK, G., 1993. *Machine Vision.* Singapore: McGraw-Hill.

[2] UMBAUGH, S.E., 1998. *Computer Vision and Image Processing*. London: Prentice-Hall, Inc.

[3] FELZENSZWALB, P.F., HUTTENLOCHER, D.P, 2004. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision,* 59 (2).

[4] HERTSMANN, A., 1998. Painterly Rendering with Curved Brush Strokes of Multiple Size. *In:* Proceedings of ACM SIGGRAPH 1997, Annual Conference Series, ACM, pages 453-460.

[5] McMILLAN, L., *Line-Drawing Algorithms* [online]. Dept. Computer Science, University of North Carolina.
Available from: http://www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html

[6] VINCE, J., 2001. *Essential Mathematics for Computer Graphics Fast*. London: Springer-Verlag Ltd.

[7] COMNINOS, P., 2006. *Mathematical and Computer Programming Techniques for Computer Graphics*. London: Springer-Verlag Ltd.

[8] HUXTABLE, J. *Java Image Processing* [online]. Available from: http://www.jhlabs.com/ip/blurring.html

[9] ROCK, T., 1997, *Matta: Art Gallery* [online]. Available from: http://www.matta-art.com

Appendix A: A detailed Quadtree Split and Merge Image Segmentation Algorithm

The object of this algorithm is to split the image up into a set of **regions**. Region*s* are defined as a group of **cells**. Cells are rectangles, of the same ratio as the image itself. These cells belong to a **quadtree** on the image, and are defined as a structure in the program. A cell has pointers to its children and parent, and its boundaries are represented by 4 integers:

```
typedef struct quadNode
{
	int left,right,top,bot;
	color colorAv;
	int leaf;
	int further;

	struct region *segmentRegion;
	struct quadNode *next[4];
	struct quadNode *prev;
}quadNode;
```

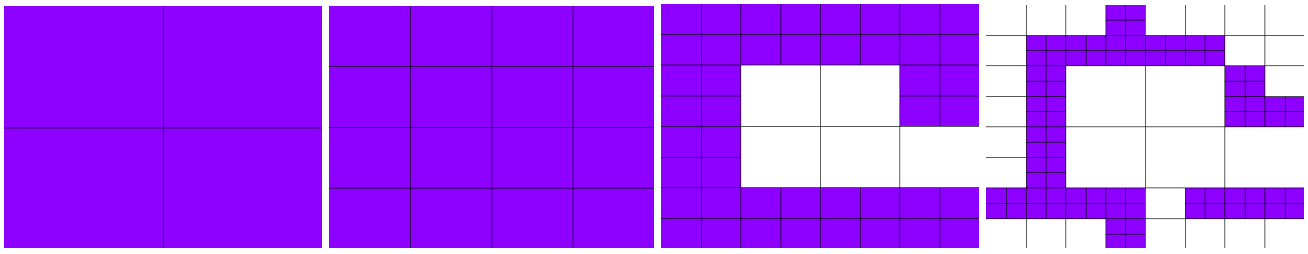
The regions of the image are stored in a doubly-linked list, so that as regions are merged and split, the list can be easily edited. Each region contains a linked-list of all its constituent cells, as well as the 'area', the number of pixels covered by the region:

```
typedef struct region
{
	struct quadPointer *cellRoot;
	int area;

	struct region *prev;
	struct region *next;
} region;
```

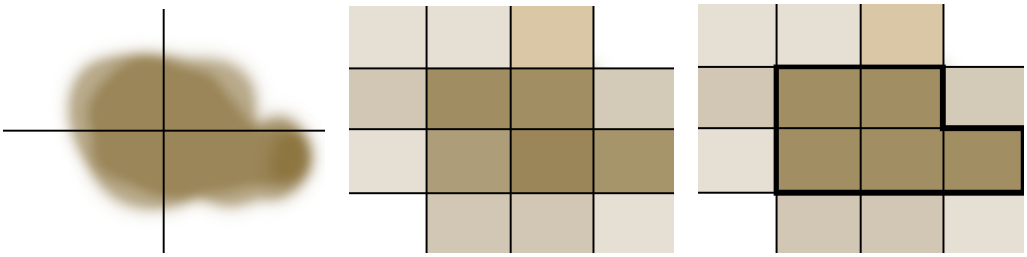
The largest cell possible is the perimeter of the whole image. Depending upon the level of depth the quadtree is allowed, the smallest potential cell is the size of one pixel.

1) Assume that the whole image is one single region defined by one cell, and that it will be broken down at least once



At each level of depth, cells with more than acceptable variance are broken down into four children

2) At the current level of depth, loop over all cells that have a high enough variance to split into four children. For each cell:
    a. Split into four. Make the original cell node point to its four children, and each child point to the original cell node as its parent.
    b. Create a brand new region with this new cell as its only member
    c. Compute, from the original image, the average colour of this cell
    d. Compute the **variance** of the cell by taking a set number of samples. If this variance is below a certain threshold, this cell contains just one region, and will never be split again. If not, this cell needs to be split and go **further**. Mark it as such.

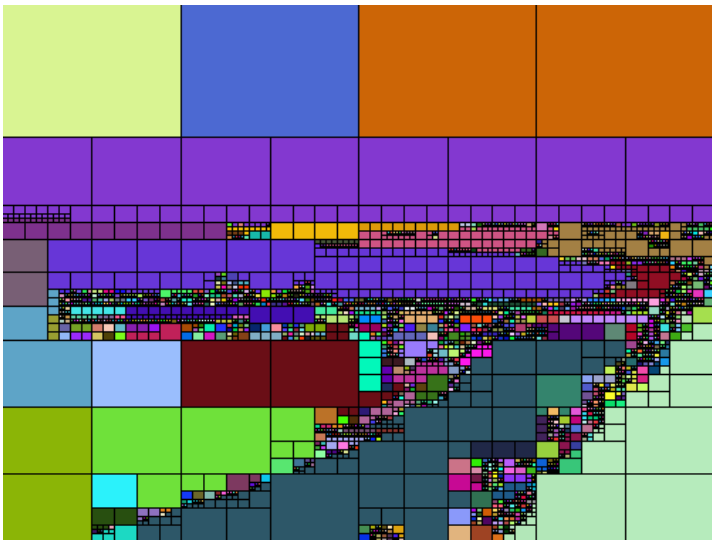


Some cells may be broken down due to high variance, after which some of their children join together again to form the coherent region

3) Take the list of these new depth level cells, and loop over all of those that do *not* need to go **further**. For each:
    a. *Internal Merging:* Compare the cell to other sibling cells that also need to go no further. If the distance between their average colours is below a certain threshold, then merge the two regions of which the cells are a part.
    b. *External Merging:* Use the parent and child links in the quadtree structure to navigate, and, based upon where the cell shares an outer edge with its parent, test each adjacent cell.
    c. If the average colour of the adjacent cell and this are below the threshold, then merge the two regions of which the cells are a part.

4) Repeat Step (2) for as long as the current level of depth does not exceed the set number.

The structure of the quadtree is now lost, and we have a linked list of regions covering every pixel on the image. Each region contains a list of its constituent cells.



A fully segmented image

Appendix B: An Algorithm for Creating a Dynamic Magnetic-like Field in a 2D Space

The magnetic field algorithm works by allowing a set of lines, defined by data structure 'line2D', to apply a force field in the two-dimensional space of the image.

```
typedef struct line2D
{
        point2D s,e;

        struct line2D *next;            /*for use as a linked list*/
}line2D;
```

The 'force' caused by a line is computed depending on the position from where the force is queried, and is the sum of the forces from all of the lines on that point. For each line:

1) Transform the point by transformation that moves the line to the origin, and oriented with the x-axis. The point's position relative to the line can now easily be computed.
2) If the point's x coordinate is within the bounds of the line, then the force applied by the line is parallel.
3) If the point is beyond either end of the line by more than a quarter of the line's length, the direction of the force is from the closer end of the line, to the point, or the reverse depending on the end in question.
4) If the point is in either outer 'quadrant', i.e beyond the line by less than a quarter of its length, then the direction of the force applied is a linear blend between the other two possibilities.
5) The magnitude of this force is diminished exponentially based upon the smallest distance from the line
6) Transform everything back to original space

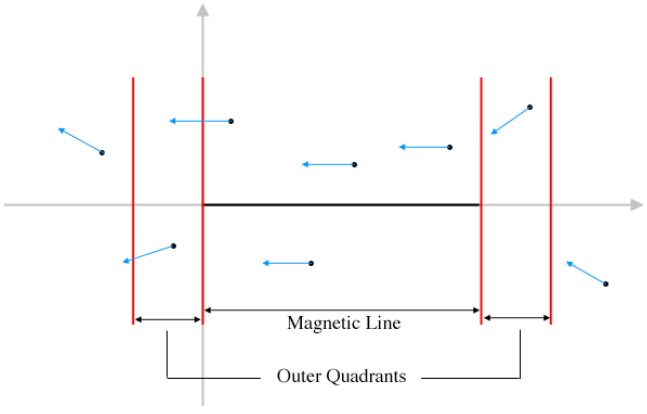


Diagram showing the how force direction depends upon position relative to the magnetic line

The sum of all of these forces gives the net magnetic force in the field at that point:

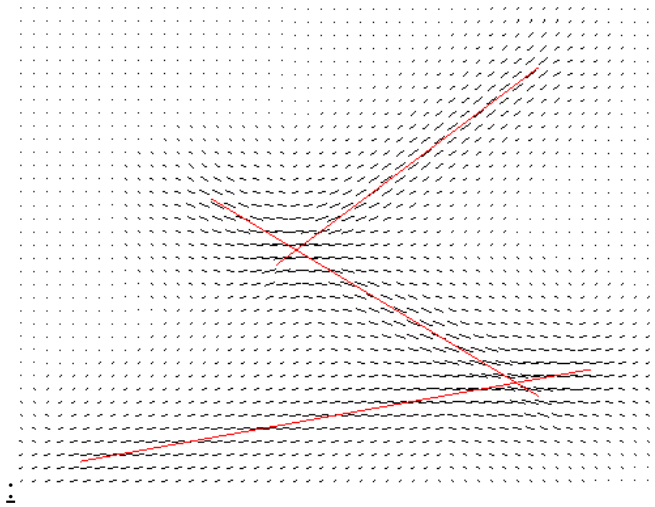
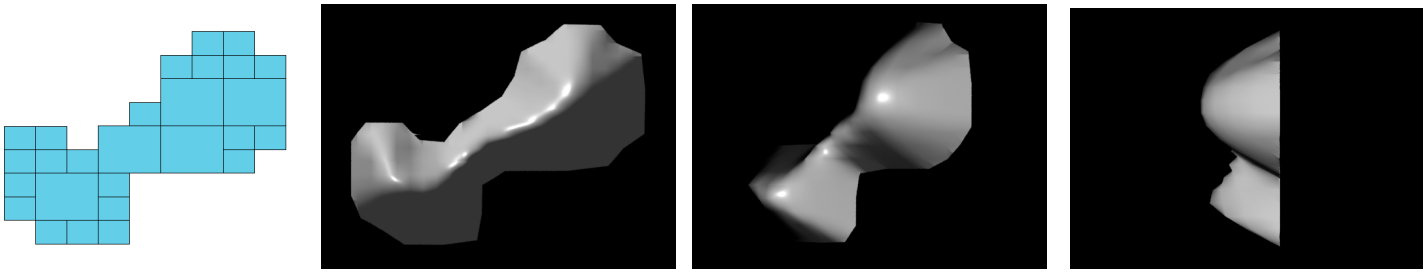


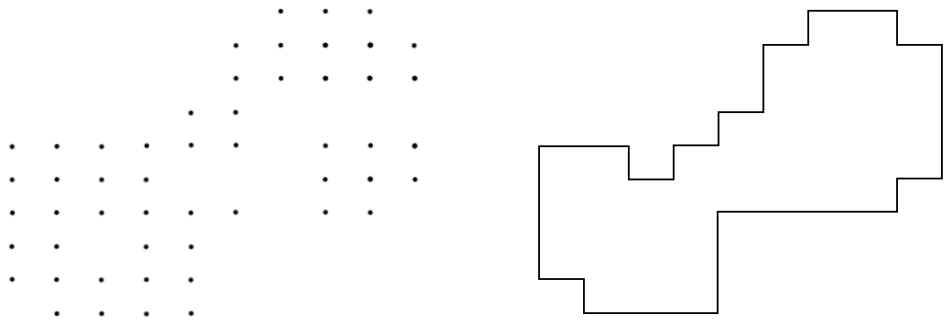
Diagram showing the cumulative effect of several ‘magnetic lines’ as a field

Appendix C: Method for converting groups of rectangles into a single coherent, rendered piece of 3-dimensional geometry

This algorithm starts with a region, defined as in the result of Appendix A, i.e. as an unordered list of rectangular cells, and converts it into a single, smooth rendered three-dimensional shape:



Firstly, the region must be turned into a polygonal outline, that is to say an ordered list of points that follow around the perimeter in a cycle:

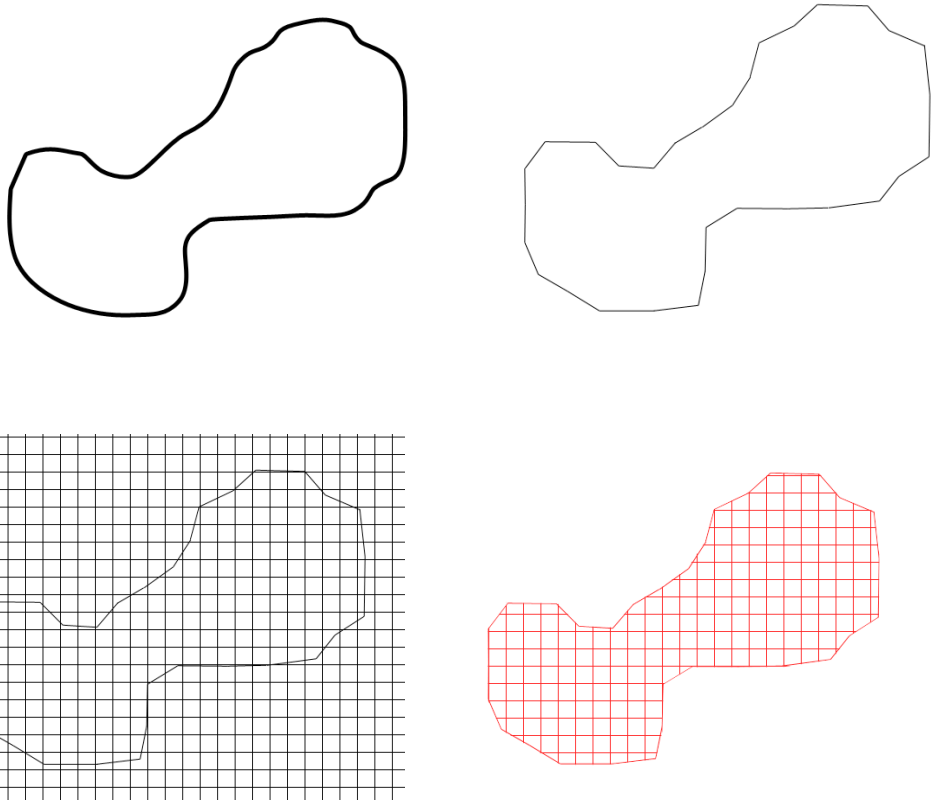


1) Redefine list of cells as sets of four corner points, rather than sets of four boundaries.
2) Cycle through the resulting cells to find one with the left-most edge. This is the 'current cell' to start with
3) Start with the top-left vertex of the cell as the current 'point'
4) Add the position of the current point to a list
5) Transform the whole set of cells such that the *current point* lies at the origin, and the direction from the current point to the current cell's next point *anti-clockwise,* is vertical.
6) Assume that this is the next point on the perimeter
7) Process through all other cells to see if any has an edge with a Y-coord less than the next on the current cell
8) If there is a closer edge, then its cell becomes the new current cell, and the

point on the edge with an X-value of zero is the new current point.

9) For as long as we don't come across the original cell and point, repeat steps 4 – 7.

The result of this is a list of points that describe the outline of the region. In order to remove that obvious rectangular-edged look that results from the quadtree method, this list of points is used to construct a simple B-spline. The curve theory in [6] describes the equations that were used. Sampling the spline at regular intervals then generates a new set of points. The result is a shape that is still fundamentally the same as the original region, but continuously smooth.



The next step is to turn this outline into a piece of geometry. This I achieve by looping over a regular grid in 2D space, and each square from the grid that intersects the new polygon, becomes a polygonal face. This gives a sensible, regular distribution of points over the region's surface. The data structure for storing the geometry is as follows:

- Both vertices and faces are stored in a linked list
- Each member or each list refers to the other list, that is to say each vertex has a list of the addresses of the faces it is part of in the face list, and each face has a list of its constituent vertices in the vertex list
- This way, *no vertex is stored twice,* and the data is stored in a convenient

way to convert to Renderman's format later

The nodes of the data structures are as follows:

The geometric information is created as follows:

1) For a given granularity, loop over all squares in the grid of the image.
2) For each, check to see if any of its four vertices are within the shape.
3) If so, then this square forms part of the final polygonal mesh.
4) Clip the square against the outline, so that any squares at the edges lose their square shape and match that of the region shape. Use the Sutherland-Hodgman clipping algorithm, as described in [7].
5) For each vertex of this final constituent face:
    a. Compute the closest distance between it and the edges of the region outline. Record this value
    b. Add to a global list of vertices, *only* if the point has not already been recorded by a different face sharing the same vertex.

6) Once all squares have been processed, Loop over all vertices and discover the maximum distance any point was from the outline. Use this maximum distance to set the Z coordinate, such that points further from the edges are extruded, and points closer to the edges remain closer to the XY plane. This results in a nice 'bobby' shape.
7) Now the final 3D coordinates of each vertex has been set, loop over all of the faces again to compute their normal vectors.
8) Finally, loop over each vertex and compute its normal as an average of the normals of all the faces of which it is a part.

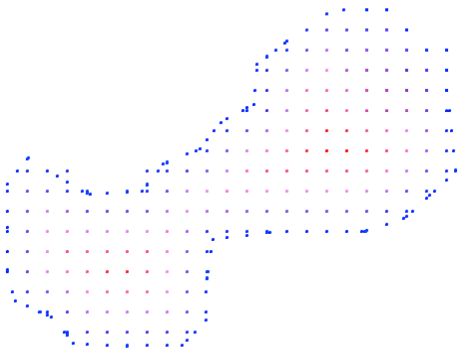
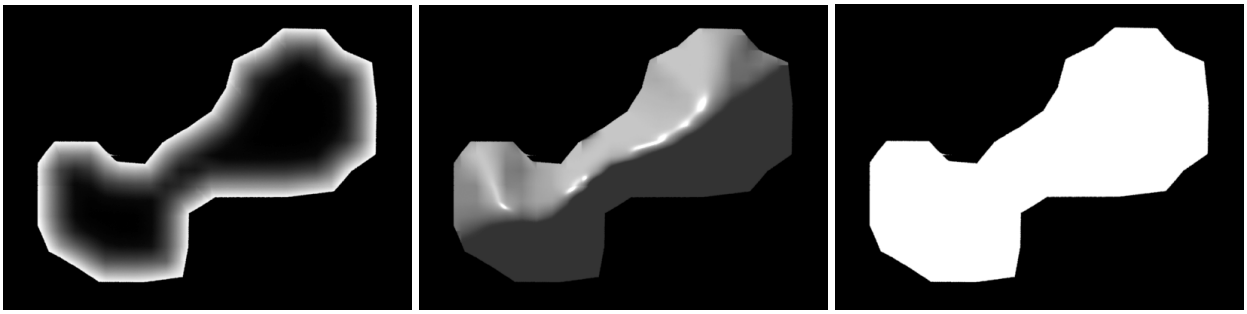


Diagram showing how vertices' distances from the shape perimeter can be computed. Blue represents points directly on the perimeter, red represents points furthest away from the perimeter

We have now obtained a sensibly structured description of a piece of geometry. The next step is to process, and write the data to the Renderman RIB file format. A simple shader is attached to the geometry, which is then rendered from an orthographic

viewpoint to produce an image of the blobby-looking version of the region. Three greyscale images come out: a depth shaded version so that the region's edges can be identified, the fully shaded version, and a flat white version displaying just the 2D shape of the region. This image can now be re-imported by my program, to be used appropriately.



Depth, shading and shape information are rendered for use by the program