

Adaptive Stylization with Vertex Attributes

Eoin O'Connor

Submitted for
Master of Computer Science
Trinity College Dublin

Supervised by
Dr. John Dingliana

Submitted to the University of Dublin, Trinity College, May, 2018

"I, Eoin O'Connor, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signature:

Date:

0 Summary	5
1 Introduction	8
1.0.1 Line Drawing	9
1.0.1.1 Why change line thickness?	9
1.0.1.2 Why not automate the line thickness?	9
1.0.1.3 Why Inverted Hull?	9
1.0.2 Painterly Rendering	10
1.0.2.1 Why change detail in the first place?	10
1.0.2.2 Why have a human decide what's important?	13
1.0.2.3 Why not paint in image space?	14
2 State of the Art	15
2.0.1 Line Drawing	15
2.0.1.1 Line Placement	15
2.0.1.2 Line Style	18
2.0.1.3 Temporal Coherence in Line Drawing	20
2.0.2 Painterly Rendering	21
2.0.2.1 Varying Detail Automatically	22
2.0.2.2 Varying Detail with Human Input	24
2.0.2.3 Paint-Stroke Placement	26
2.0.2.4 Temporal Coherence of Paint strokes	28
2.0.3 Other Uses of Vertex attributes	28
3 Inverted Hull	29
3.1 Implementation	29
3.1.1 Painting Vertex Attributes	29
3.1.1.1 Why Maya?	29
3.1.1.2 On the Intuitiveness of Painting Vertex Attributes	30
3.1.2 Inverted Hull Rendering Algorithm	30
3.2 Results	33
3.2.0.1 Practical and aesthetic benefits	34
3.2.0.2 Different distances	36
3.2.0.3 Different viewing angles	37
4 Painterly Rendering	39
4.1 Implementation	39
4.1.1 Painting vertex attributes for level-of-detail	39
4.1.2 The Stroke-Based Rendering Algorithm	40
4.1.2.1 Rendering to the textures	40

4.1.2.2	Generating Stroke Positions with Poisson Disk Distribution	40
4.1.2.3	Rendering Strokes	43
4.2	Results	45
4.2.0.1	Aesthetics	47
4.2.0.2	Tuning the stroke parameters for full stroke coverage	48
4.2.0.3	Real time poisson disk distribution	49
5	Conclusions and Future Work	52
	References	54

0 Summary

This dissertation presents an examination of the use of vertex attributes in two non-photorealistic rendering techniques. The question I aim to answer is “How can vertex attributes be used effectively in non-photorealistic rendering?”. To answer this, I examine an existing line drawing technique that uses vertex attributes to control line-thickness, then introduce a novel painterly rendering algorithm that uses vertex attributes to mark high-detail areas.

The existing line drawing technique, referred to inside as ‘Inverted Hull’, is a standard from the days of the fixed-function pipeline and while the use of vertex attributes with this technique is not completely novel, it has not been published to this date. It is found to be primitive compared to more modern line drawing techniques, though at least some of its shortcomings can be mitigated by the use of vertex attributes.

The novel painterly rendering technique can produce images that exhibit an artistic variance in detail reminiscent of impressionism in painting or Bokeh in photography, although not quickly enough for real time interaction and not temporally coherent. The vertex attributes are shown to be a viable way to mark areas of high detail in way that is view independent.

Acknowledgements

Thanks to

Dr. John Dingliana for supervising my project,
Junya C Motomura, whose talk at GDC gave me the idea,
Ross, 'prime motivator',
Huw, who turned up on time.

Abstract: This paper examines two non-photorealistic rendering techniques for 3D models that make use of vertex attributes to control style. The first is an existing technique used for drawing outlines around objects, where vertex attributes govern the thickness of the outline. The second is a novel painterly rendering technique that uses vertex attributes to mark important areas of a 3D model so that they can be rendered in greater detail, while abstracting away the details in other areas.

1 Introduction

Non-photorealistic computer graphics are “characterized by randomness, ambiguity, or arbitrariness rather than completeness and adherence to the portrayed object’s properties” (Strothotte, 2002). Many research papers in this area aim for more automated computation of the non-photorealistic elements of an image - the randomness, the ambiguity, etc. I aim for more control over the non-photorealistic elements.

I believe that adding information to the 3D object itself is more efficient than adding information to a 2D image because the input is reusable and view-independent.

Adding an artist’s input to a 2D frame of animation could yield the same quality of result, but it must be added to every other frame. Storing this input in 3D means the information can be mapped to infinitely many 2D frames automatically.

This paper examines two non-photorealistic rendering techniques that make use of vertex attributes to influence the final image. First, I will discuss an old technique used to draw lines around the contours of a 3D model, where line thickness is governed by vertex attributes. Then I introduce a novel painterly rendering technique for 3D models that allows the level of detail to be controlled by the vertex attributes of the model.

1.0.1 Line Drawing

1.0.1.1 *Why change line thickness?*

Varying line thickness is an important expressive tool in line drawing. It can be used to convey weight, light/shadow (Goodwin, 2007) and exaggerate the shape of the subject being drawn. As stated in (Goodwin, 2007):

“Stroke width and style are key features of art, comics, and technical illustration. Line style adds life and clarity to drawings [McCloud 2006], and provides cues to 3D shape [Guptill 1997; Hodges 2003].”

1.0.1.2 *Why not automate the line thickness?*

Automating line thickness has been achieved with good results, such as by (Goodwin, 2007). However, I propose that similar results can be obtained using a much simpler algorithm with the aid of vertex attributes. Furthermore, I propose there is some added value in directly influencing the rendering of particular part of the model.

1.0.1.3 *Why Inverted Hull?*

For a detailed description of the Inverted Hull technique, see section **3.1**. I chose the inverted hull technique as a proof of concept because of its simplicity, and because it provides a very straightforward opportunity to use vertex attributes in the rendering process (that being the extrusion of vertices in the first pass). Usually, using inverted hull would mean having no control over stylization. With the help of vertex attributes, the thickness of the outline can be changed around certain areas of the model.

I believe that the input from vertex painting could be generalised to other line techniques as well.

1.0.2 Painterly Rendering

1.0.2.1 *Why change detail in the first place?*

Santella et al. addressed this issue using Johannes Vermeer's "Girl Reading a Letter at an Open Window" (1657) (figure 1) as an example.

"The realistic look belies the effort of the artist, who has organized the image to highlight meaningful information. For example, the artist creates high contrast in distinguishing the figure from the background and delineating her face, hair, dress and hands. In comparison, unimportant objects in the scene—the chair, the panes of glass, the tabletop, the back wall—lack the same sharp detail and fine contrasts" (Santella, 2002).



Figure 1 Johannes Vermeer's "Girl Reading a Letter at an Open Window" (1657)

Understating the minor features of an image serves to emphasise the important features. This is an important technique in painting that distinguishes it from a photograph. Having said that, even in photography, the abstraction of detail is an established technique known as Bokeh. By intentionally leaving elements outside the depth of field, photographers can abstract away extraneous detail and isolate the subject.



Figure 2 Santella's eye-tracking algorithm decides what is important by recording what test subjects look at - then abstracting away everything else (Santella, 2002).



Figure 3 Josefina with Bokeh, from Wikimedia Commons, <https://www.flickr.com/photos/paseodelsur/51805888/>

1.0.2.2 *Why have a human decide what's important?*

Machines are not always able to determine an artist's intention. Let's look at the cinematic use of depth of field in this shot from *The Matrix*.



Figure 4 *The Matrix*, Wachowski Brothers, (1999)

A mobile phone ringing is not a dramatic event in normal circumstances, but in this scene it is *made* dramatic by blurring everything else (and by the extreme upward angle of course). Even when sharing the screen with a human face, this phone gets all the attention. You don't need to look far for more examples of this technique in cinema. Whether it is in the foreground or background, it is difficult to guess what an artist might want to communicate when composing shots like this. For that reason, I have chosen to let a human decide what is important for this technique.

1.0.2.3 *Why not paint in image space?*

Hertzmann used a 'weight image' to govern the level of detail in his stroke based rendering, but this was intended for 2D still images only (Hertzmann, 2001). This paper presents a technique that can generate a similar type of weight image automatically by rendering it from 3D model data. This is a step towards enabling 3D animation with authored areas of high and low detail. Temporal coherence of strokes is out the scope of this paper, but I believe this technique could be made compatible with (Vanderhaeghe, 2007) in the future.

2 State of the Art

In this chapter I will describe how others have approached the important problems regarding line drawing and painterly rendering. The two areas share common issues, such as stroke placement and temporal coherency of strokes, although their methods tend to differ. Line drawings are rendered with much fewer strokes, so the major concern is how to efficiently depict an object, normally with as few strokes as possible placed on only the most important features. The main concern with painterly rendering is style, and while stroke placement and stroke efficiency don't hurt, they are not held to the same constraints. With my technique, I would like to bring the idea of *leaving out what's unimportant* from line drawing into the sphere of painterly rendering.

2.0.1 Line Drawing

There are many research papers addressing line drawing. Usually, these papers address one of three problems: line placement, line style, and temporal coherency of strokes (in stroke-based line drawings).

2.0.1.1 Line Placement

The placement of lines is obviously very important for artistic rendering. The current state of the art techniques for line placement tend to be stroke-based algorithms; that is to say they output a set of line segments which could be interpreted as pen-strokes.

DeCarlo et al. provide a solution to the problem of line placement in *Suggestive Contours for Conveying Shape* (DeCarlo, 2003). They define contours to be areas on the 3d model where the surface normal is orthogonal to the viewing direction. Building on this, they define *suggestive contours* to be areas that *would* be a contour from a nearby view. The paper actually provides two different algorithms to generate strokes: one in object space and one in image space. Their technique produces aesthetically pleasing images, but they admit that “Objects without concavities have no suggestive contours”, citing a rounded cube as an example. When rendered with suggestive contours, only the outer silhouette of the cube is drawn.

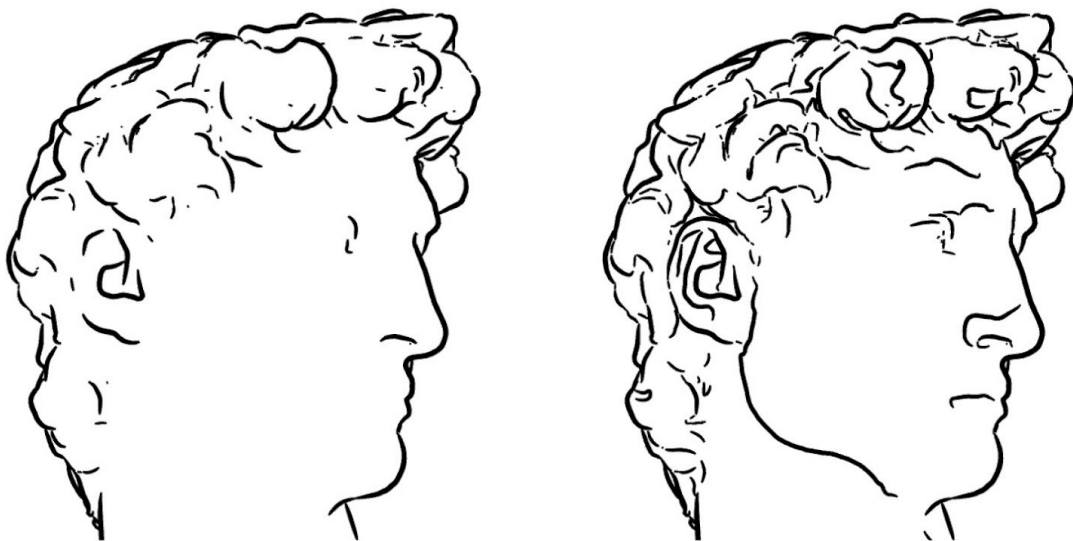


Figure 5 Left: contours only. Right: contours and suggestive contours (DeCarlo, 2003)

Judd et al. propose a different solution in which they address some of the issues with suggestive contours (Judd, 2007). This technique uses the *principal curvature* of the model as well as *view-dependant curvature*. The principal curvatures at a point are the maximum and minimum of the normal curvature at that point. View-dependant

curvature is a measure of how the surface normal changes from a particular perspective. The titular *apparent ridges* are found where the direction of the *maximum* view-dependant curvature of a point is the same as the direction of the principal curvature at that point (minimum, or maximum). Strokes are placed at apparent ridges. This technique improves on suggestive contours in that objects do not need concavities for their features to be drawn. Apparent ridges is better able to depict the rounded cube that posed a problem for suggestive contours.

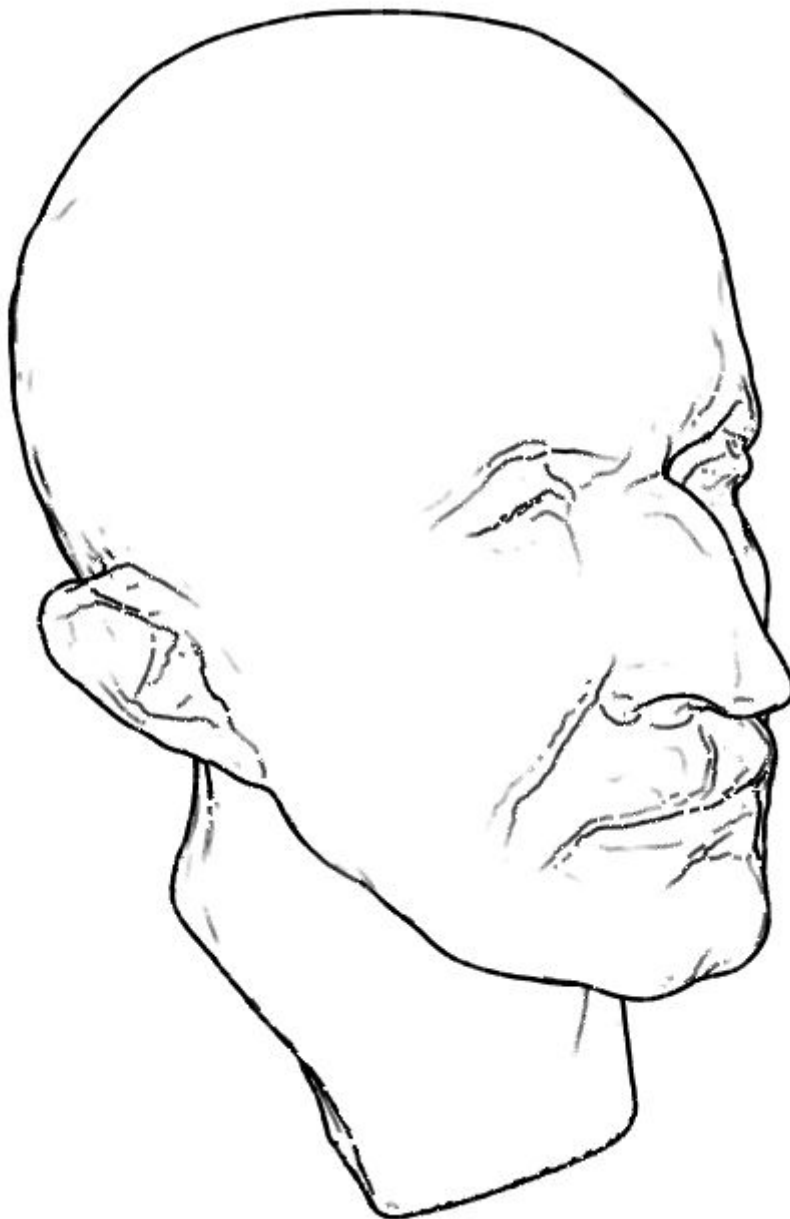


Figure 6 Apparent Ridges (Judd, 2007)

Cole et al investigated how human artists used lines to depict a 3d render as a line drawing (Cole, 2008). Among other things, they found that 57% of all lines drawn appeared at interior and exterior occluding contours. The inverted hull algorithm may not be as sophisticated as the aforementioned stroke-based algorithms with respect to line placement, but it is capable of drawing lines around these contours.

2.0.1.2 Line Style

The strokes produced by stroke based algorithms can be rendered in a number of different ways, including using a texture map (of an ink brush stroke for example). This can be tapered or widened based on the position within the stroke. However, the current state of the art algorithms do not use vertex attributes to govern line style.

The problem of variable line thickness has been tackled by Goodwin et al in the paper *Isophote Distance: A Shading Approach to Artistic Stroke Thickness*, where line thickness at each point on a stroke was governed by the distance from that point to an isophote along the radial plane (Goodwin, 2007). This technique produces lines whose thickness varies with the curvature of the model, the light direction, and the viewing distance. It does not allow vertex attributes to affect the thickness of lines around particular areas.

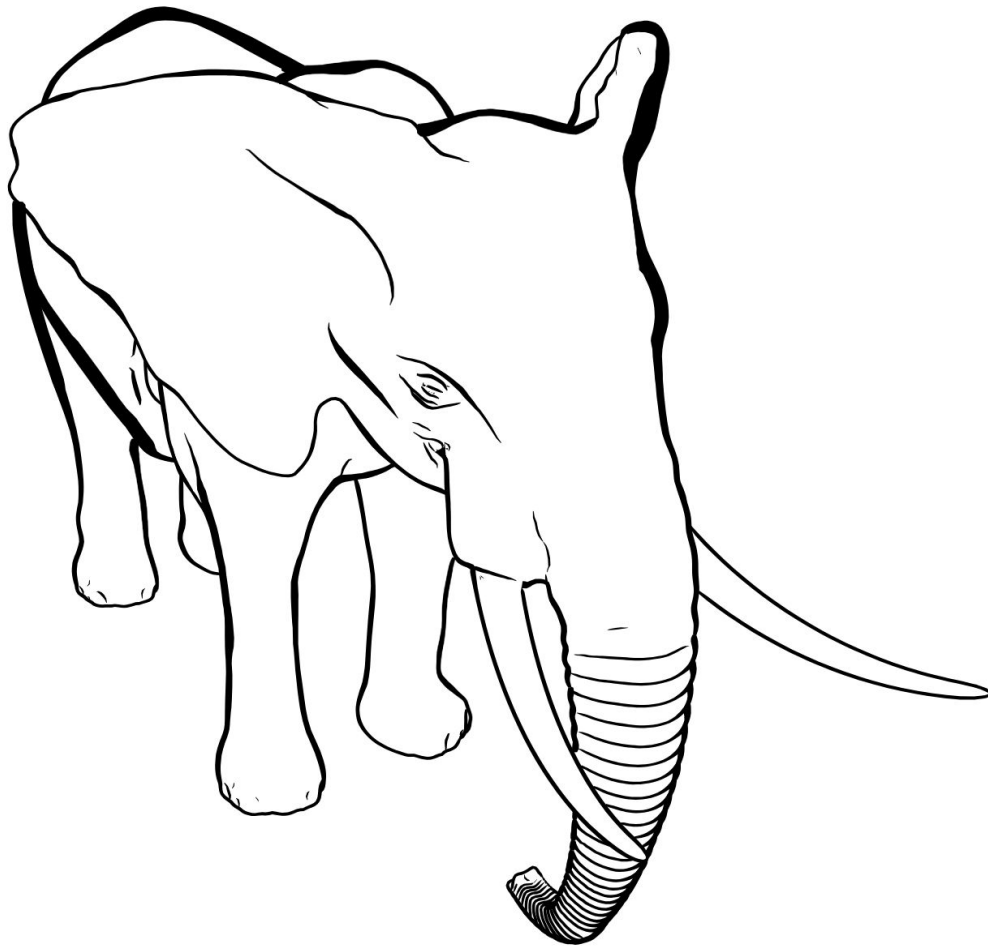


Figure 7 An example of an Isophote Distance render. (Goodwin, 2007)

Grabli et al. introduced a highly generalised approach to line style inspired by programmable shaders in their paper: *Programmable Style for NPR Line Drawing* (Grabli, 2004). This paper describes a multiple stage procedure for creating line drawings much like the traditional 3D graphics pipeline. In the first stage, they create a 'view map' which describes the potential lines that might appear in the drawing and their topological arrangement from a 3D model. In the second stage, they apply 'style modules' to the view map. Style modules might select and chain or split potential lines, or change how they are rendered for the final image. Style modules are given all the information of the view map to work with, so they can be quite sophisticated - it is possible for style modules to operate only on particular types of lines. This paper

mentions 'object importance' among the possible data in the 3D scene. This idea is quite similar to what I propose in the next section on painterly rendering in that some areas of the render are considered more important than others. However, Grabli et al. do not examine the use of vertex attributes as a measure of importance.

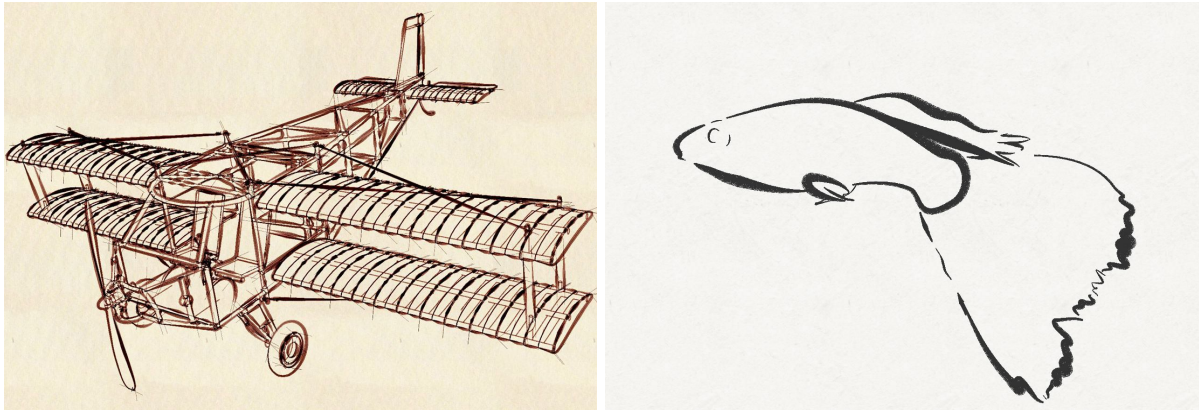


Figure 8 Left: A sketchy style drawing with 'construction lines' generated by the program and drawn in a different style. Right: A Japanese style drawing with two style modules simulating different brushes. (Grabli, 2004)

Other papers dealing with line style include (Kaplan, 2000), who attaches geograftals to a 3D surface to represent strokes and (Kalnins, 2002) who introduces a system to design stylization, like hatching patterns and line styles, where users can draw strokes directly onto 3D models.

2.0.1.3 Temporal Coherence in Line Drawing

Another area of line drawing research is concerned with keeping strokes temporally coherent. The configuration of strokes in a line drawing can be dramatically different between two frames. Papers aiming for temporal coherence seek to keep strokes consistent between frames. This involves moving and reshaping strokes, introducing new strokes where necessary and culling strokes when they become occluded or too densely packed.

(DeCarlo, 2004) handles temporal coherence for their own suggestive contours from (DeCarlo, 2003). (Kalnins, 2004) addresses the problem of temporally coherent line style by building upon their previous work where users can draw strokes directly onto 3D models by ensuring that the strokes stay consistent between animation frames as well as running at interactive rates. (Lou, 2015) focuses on line drawing and introduces a new technique to map strokes onto 3D models to help maintain temporal coherency even with bumpy models - where normal techniques would fail. (Bénard, 2012) addresses the temporal coherency problem of line strokes and also eliminates cluttered lines based on a screen space density value.

2.0.2 Painterly Rendering

Painterly rendering is a vast field. Hegde et al. (Hegde, 2013) and Hertzmann (Hertzmann, 2003) have done surveys of different techniques. Hegde approached the problem of painterly rendering as a whole, whereas Hertzmann focused on stroke-based rendering techniques alone.

Meier introduced a technique to render 3D objects with particles (or strokes) for animation. (Meier, 1996) In this case, the particles were distributed across the model's geometry rather than screen space. This provided temporal coherency. Some of the particle's attributes are sampled from a 'reference image' which was rendered image that the paint strokes are intended to depict. The technique

introduced in this paper also samples a reference image to obtain stroke attributes.

Meier's technique does not vary detail across the image.



Figure 9 Meier's painterly rendering of a haystack (Meier, 1996)

2.0.2.1 *Varying Detail Automatically*

Many painterly rendering algorithms attempt to automatically determine what parts of the image to render in high detail. This obviously has the benefit of requiring less input from a human to produce good images.

(Collomosse, 2002) uses image salience to prevent details from being lost in the rendering process. The technique reduces the size of strokes in salient regions, as well as ordering them such that they won't be hidden behind the larger, less salient strokes. The technique doesn't alter the density of strokes and, being completely automatic, doesn't allow for authored low detail or high detail regions.

Hertzmann introduced a technique to bring a render from low detail with large brush strokes up to high detail with small brush strokes (Hertzmann, 2003) wherein the

source image is blurred to a degree corresponding to the brush size. Then progressively smaller strokes are layered on top where the source image differs from the render, referencing a less blurred source image for each brush. Many 2D techniques use a layering approach to detail including (Haeberli, 1990) and (Hays, 2004).



Figure 10 Hertzmann's brush strokes of different sizes. Left is the source image, centre is the brush strokes, and right shows the same strokes with texture mapping. (Hertzmann, 2003)

(Hays, 2004) goes a step further and enables temporal coherency for stroke based rendering from video data. Their technique produces very high quality renderings built up from layers of varying detail, shown in figure 11. What goes into each layer is determined by increasing frequencies of gradient. In theory, these layers could be filtered through a weight mask such as the vertex colour image described in this paper. Unfortunately, implementing this was infeasible within the time frame.

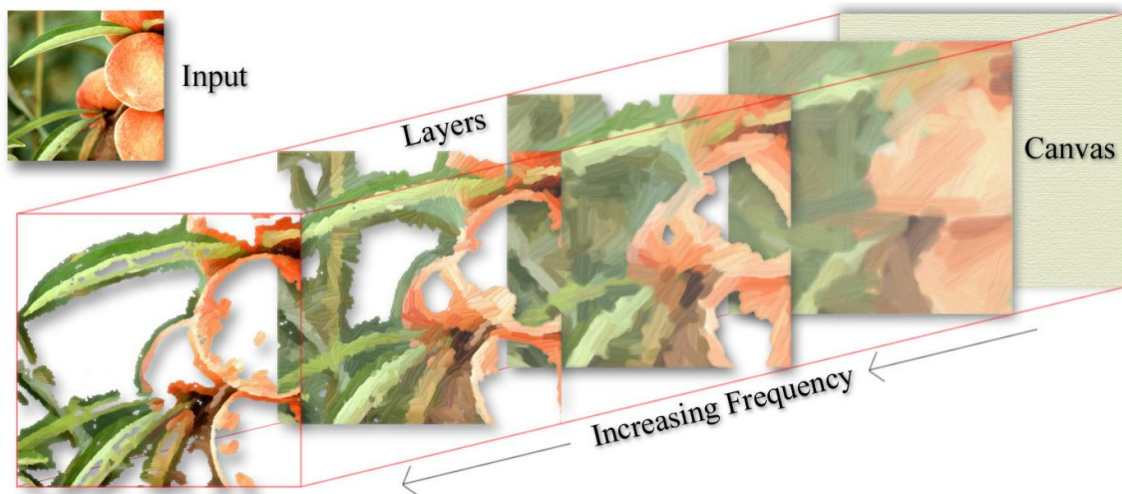


Figure 11 Hays' multiple layers of detail. (Hays, 2004)

2.0.2.2 *Varying Detail with Human Input*

Bridging the gap between automatic and human-authored detail is (Santella, 2002).

Using an eye tracker, Santella et al recorded the parts of an image test subjects looked at. With this data, they were able to raise and lower the detail in areas based on where their subjects looked. This technique is different to the one presented in this paper because it only applies to still images, and isn't exactly 'authored' (anyone who has experience with an eye tracker will understand how little control we have over our own eyes).

Varying detail by painting detail masks has been done before for 2D images.

(Hertzmann, 2001) uses an interactively painted weight image to mark the level of detail desired for each area in the image. Hausner, although his technique was intended for mosaic blocks rather than paint strokes, demonstrates the use of user defined regions to changes level of detail (Hausner, 2001). In doing so, the the mosaic blocks are made smaller and more dense like the paint strokes in the

technique describes in this paper. What these two papers lack is the capability to attach the desired detail information to a 3D model which can be moved around.



Figure 12 Paint By Relaxation: Weight image and the corresponding painterly render. (Hertzmann, 2001)

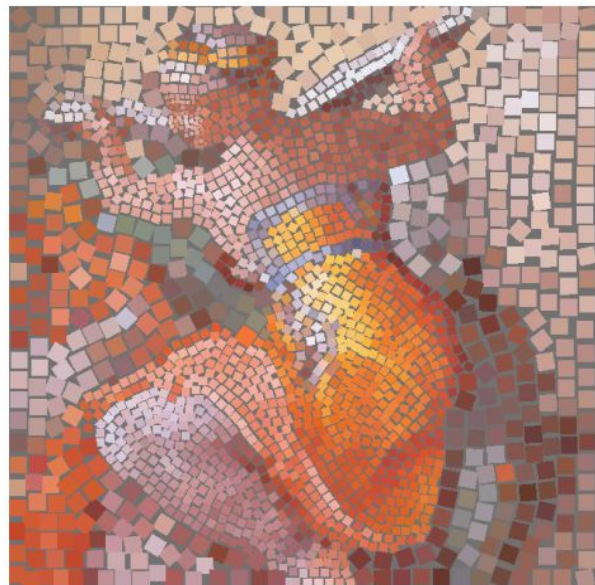
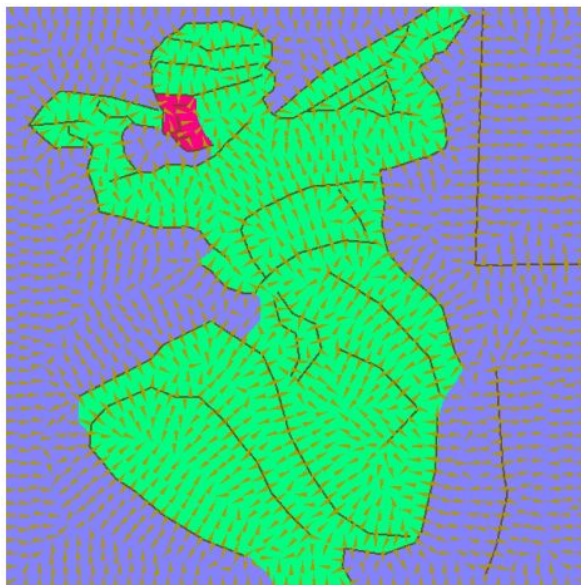


Figure 13 Simulating Decorative Mosaics: The user-defined detail areas and the mosaic image. (Hausner, 2001)

The issue of detail in 3D been addressed in (Winkenbach, 1994), where a user marks areas of a 3D model to be rendered in detail in a pen and ink style, while other areas are left flat (this is known as indication). This is very much a texture oriented approach. The detail marks are placed interactively onto the texture where the

surface is to be 'indicated'. The model is then rendered and texture mapped rather than abstracted and depicted with particles like paint stroke algorithms.



Figure 14 Computer-Generated Pen-and-Ink Illustration: User-defined marks, with indication, and without indication. (Winkenbach, 1994)

2.0.2.3 *Paint-Stroke Placement*

Placing paint strokes usually requires extensive trial-and-error iteration in order to guarantee full coverage of the canvas as well as to achieve an accurate rendering of the source (Hertzmann, 2001). Usually this involves making a random change to one stroke and evaluating some energy function over the new image - if the energy decreases, the change is accepted - and once the energy function converges, the job is finished. There are other ways to achieve acceptable stroke placements which are quicker and simpler, though less exact. The fastest way would be a uniform random distribution, but as you can see in figure 15, this results in large gaps and clusters everywhere - due to the fact that it's random. The next best option is Poisson disk distribution. Samples are placed randomly, but no two samples can be closer than some minimum distance.

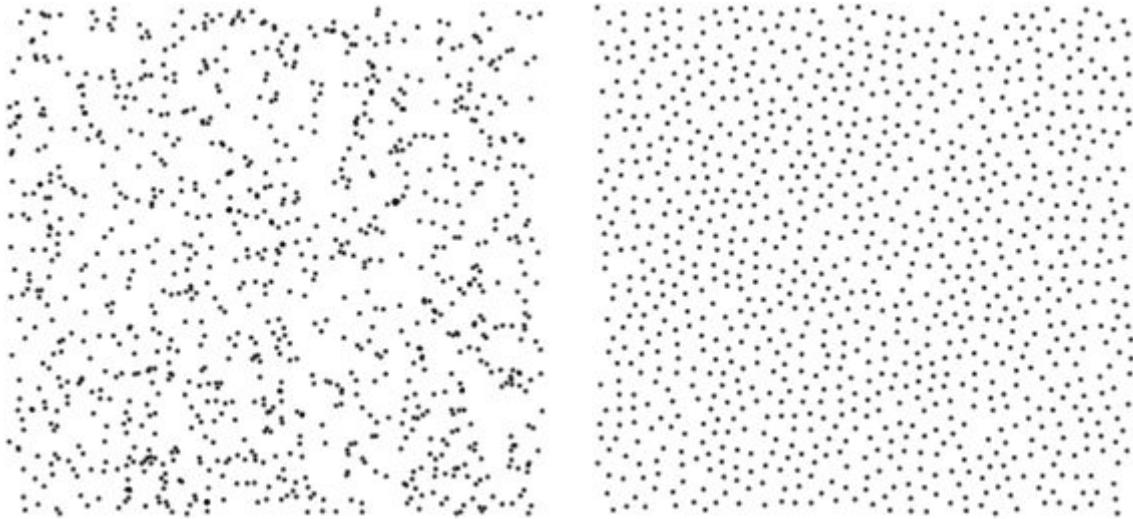


Figure 15 Uniform random vs Poisson disk distribution

Different methods for generating Poisson disk distributions have been examined thoroughly in (Lagae, 2008) with respect to quality mostly - though the quality of the distribution is not a vital concern for technique introduced in this paper. Achieving a maximal distribution, where the samples are so densely packed that no more can be added without violating the conditions, is also not a concern for this paper.

An added advantage of Poisson disk distribution is that the minimum distance can vary per location, ensuring sparsity and therefore low detail, making it an ideal fit for the technique in this paper. Varying sample density was described in (McCool, 1992), using a function rather than a texture lookup as in this paper. A fast technique for generating samples has been described in (Dunbar, 2006), but I don't believe it is fast enough to run this technique in real time, so I use an adapted version of *Fast Poisson Disk Sampling in Arbitrary Dimensions* (Bridson, 2007) because it is simpler and still quite fast.

2.0.2.4 Temporal Coherence of Paint strokes

While sadly out of scope for this paper, temporal coherence for paint strokes is a very important issue for the future. The main papers that deal with this problem are (Vanderhaeghe, 2007), where screen space paint strokes are projected back onto 3D models before they are rotated; (Meier, 1996) where paint strokes are attached to the 3D model as it moves around the scene; and (Hays, 2004) where the temporal coherence of strokes is maintained by tracking the movement of pixels in image space.

2.0.3 Other Uses of Vertex attributes

Anyone who has rigged a 3D model for animation will be no stranger to painting vertex attributes as influence weights for joints on a skeleton. (Bando, 2002) used vertex attributes as direction vectors for modeling how human skin wrinkles, shown in figure 16.

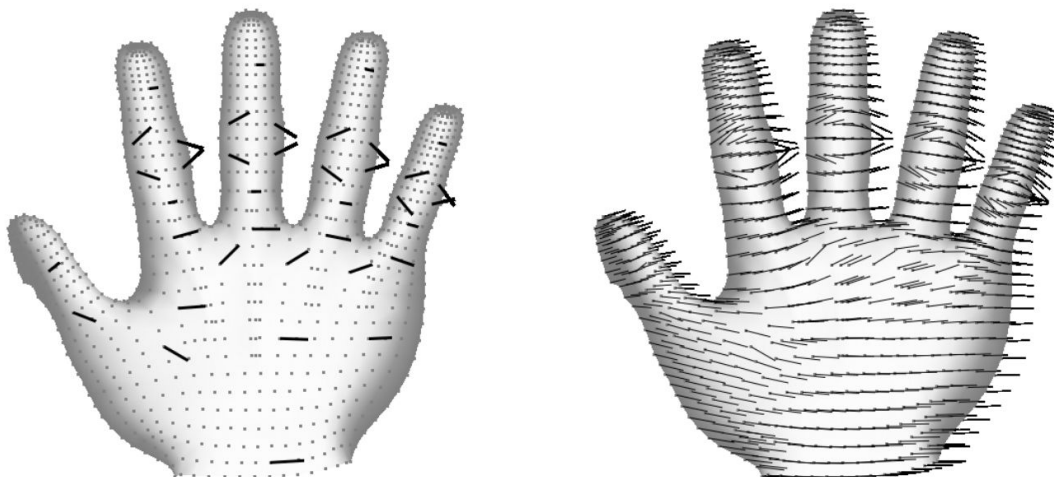


Figure 16 Left: a few direction vectors are defined at key points. Right: A direction field formed by interpolation (Bando, 2002).

3 Inverted Hull

This chapter describes a technique to draw lines around the contours of a 3D model where line thickness varies with vertex attributes.

3.1 Implementation

For the implementation of this technique, vertex colours were painted on to each 3D model using Autodesk Maya. The models were exported with their colour attributes and then loaded into the rendering program. In the rendering program, each of the model's vertices is extruded outwards proportionally to its colour value. The back-faces of this extruded model are rendered black, to represent the outlines. Finally, the front-faces of the model are rendered, without extruding the vertices, to colour inside the lines.

3.1.1 Painting Vertex Attributes

3.1.1.1 *Why Maya?*

Autodesk Maya provides a tool to paint vertex colours directly onto a 3D model using a 3D brush. I believe many artists will be familiar with tools like this, since similar tools are used when skinning meshes. For this reason, and because of time constraints, I chose to use this tool as opposed to creating my own. As a result of this decision, the vertex attributes cannot be changed during rendering. I chose to use Maya over other programs simply out of convenience. Any tool that supports the authoring of vertex attributes would be suitable.

3.1.1.2 *On the Intuitiveness of Painting Vertex Attributes*

For the Inverted Hull technique, I make use of the red channel only, although in theory it would be possible to use the full RGBA supported by Maya. Maya's brush tool allows for making changes to a single colour channel, but does not allow for viewing the colour from a single channel. This would not be suitable for independent parameters

The process of painting areas that should be more boldly outlined was unusual. Colours are painted onto a vertex when it is facing the viewer (or painter), but the rendering program only uses these colour values when the vertex is facing orthogonally to viewer. For this reason, some knowledge of how the rendering algorithm works is necessary to apply a good paint layer.

Furthermore, due to the long response time between painting a vertex in Maya and seeing how it affects the render, it is impractical to be meticulous with details. For the demonstrations, I used a broad brush to block out areas (like a face or a hand) and then used a smoothing tool to smooth over the edges. If the tool was interactive, it might be more practical to apply vertex colours with more precision (like on the side of a nose or finger, for example).

3.1.2 Inverted Hull Rendering Algorithm

This rendering algorithm is an old technique that works in two passes.

During the first pass, each vertex is extruded by a small amount in the direction of its normal. This produces a slightly larger, and slightly deformed model. The back-faces of this model are rendered black, or whatever colour is desired for the outline.

During the second pass, the front-faces of the model are rendered without any vertex extrusion, using any desired shader.

The vertex attributes are used in the first pass. As mentioned earlier, only the red channel is used. For each vertex, a 'thickness' value is interpolated (based on the value of the red channel at that vertex) between a pair of minimum and maximum thickness values shared across all vertices.

Here is an example vertex shader (in GLSL) for the first pass of the algorithm:

```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 5) in vec3 aColour;

uniform mat4 projectionview;
uniform mat4 model;

uniform float thicknessMin;
uniform float thicknessMax;

void main()
{
    float thickness = thicknessMin * (1 - aColour.r) + thicknessMax *
aColour.r;
    vec3 extendedPos = aPos + aNormal * thickness;
    gl_Position = projectionview * model * vec4(extendedPos, 1.0);
}
```

The inverted hull algorithm is a hybrid algorithm . These algorithms are “characterized by operations in object space that are followed by rendering the

modified polygons in a more or less ordinary way using the zbuffer.” (Isenberg, 2003).

3.2 Results

As expected, the lines are thicker around areas with higher red values, as shown in figure 17.



Figure 17 Left: variable line thickness rendered over with white. Right: illustrating the red channel of the vertex colours.

In reality, the vertex colours on the models are RGB values ranging from black (0,0,0) to red (1,0,0). In order to illustrate them clearly alongside the black outlines, I colour them with $RGB(1, 1-r, 1-r)$, where r is the red channel of the vertex colour. The result is that black areas appear white, while red areas still appear red.

The second pass of the inverted hull algorithm can really be performed by any shader program. Figure 18 shows the use of a quantised shader.

It is also possible to render only the outer silhouette of the model by disabling depth testing before rendering the back-faces in the first pass, as shown in figure 18.

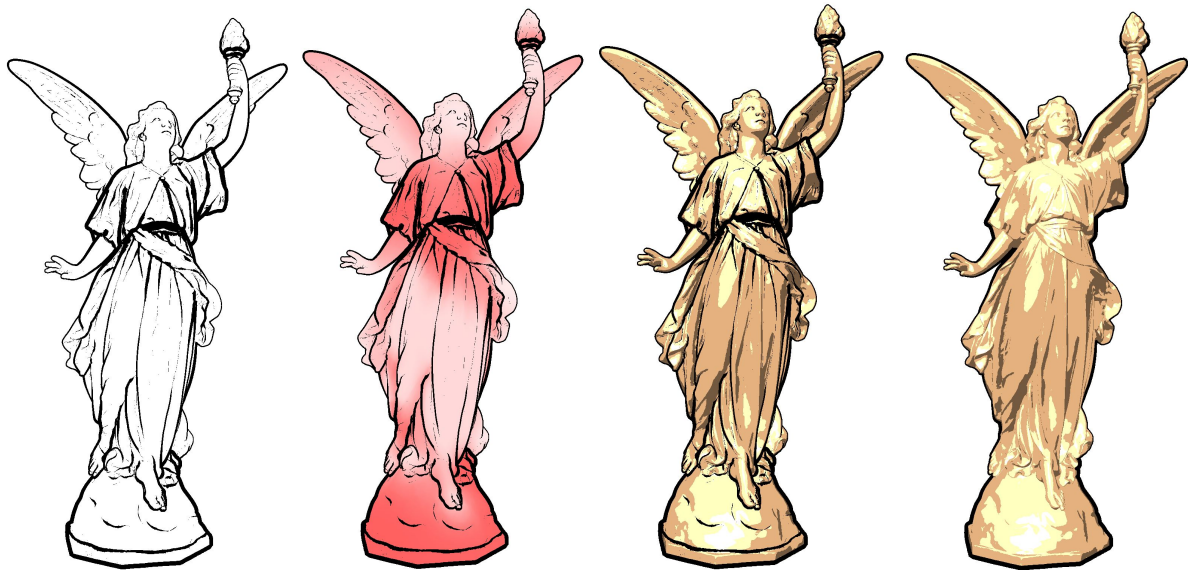


Figure 18 Far left: variable line thickness. Centre left: illustration of vertex colours. Centre right: quantised shading (toon shading) over inverted hull. Far right: depth test disabled for inverted hull.

3.2.0.1 *Practical and aesthetic benefits*

Extruding vertices along their normal effectively *deforms* the mesh. Larger extrusions will naturally deform the mesh more. Likewise, more intricately detailed features of the mesh (like eye sockets, toes, creases etc) will suffer more from deformation than broader, smoother areas (like shoulders and the outer silhouette). By painting low red values around detailed areas and higher red values around broad shapes, the technique can produce delicate lines around the details without sacrificing boldness in other areas.



Figure 18 Left: constant vertex extrusion. Right: variable vertex extrusion

As can be seen in figure 18, the variable line thickness afforded by vertex attributes can reduce deformation in sensitive areas like the face and hands, without sacrificing the bold lines on the robes. Figure 19 shows how vertex attributes can be used to add an artistic edge to the render, giving the fingers and folds of fabric a delicate quality. Figure 20 and figure 23 show how vertex attributes might help depict the weight of the lion's forelegs with respect to the snake's head.



Figure 19 Left: constant vertex extrusion. Centre: variable vertex extrusion. Right: illustration of vertex colours.



Figure 20 Left: constant vertex extrusion. Centre: variable vertex extrusion. Right: illustration of vertex colours. This model was obtained from <https://sketchfab.com/models/d5e6b6a11da646f68a5fcba661dcae99>

3.2.0.2 Different distances

Since outline thickness is calculated in object space, the inverted hull algorithm naturally produces narrower outlines for distant objects. Figure 21 shows how this property can be removed by multiplying thickness by distance. When distance is nullified, the only factor affecting line thickness is the vertex colour.

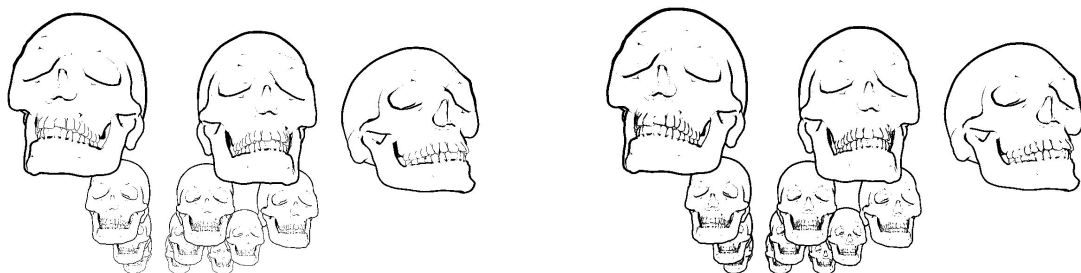


Figure 21 Left: Distance affects line thickness. Right: thickness is multiplied by distance to keep lines uniform. The skull model was obtained from <https://sketchfab.com/models/1a9db900738d44298b0bc59f68123393>

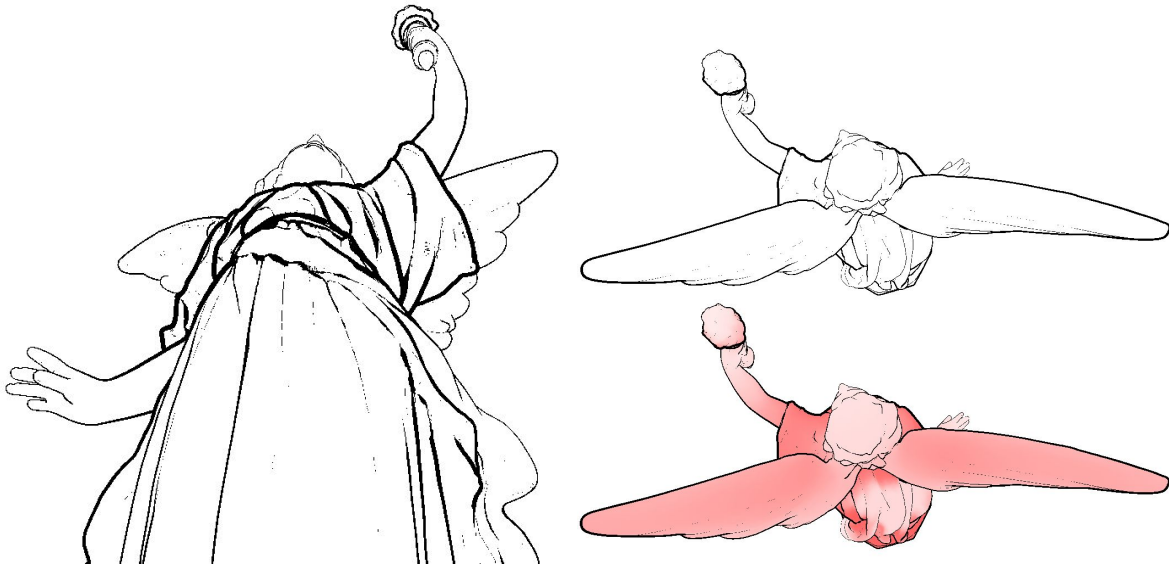


Figure 22 Left: view from below. Top right: view from above. Bottom right: illustration of vertex colours from above.

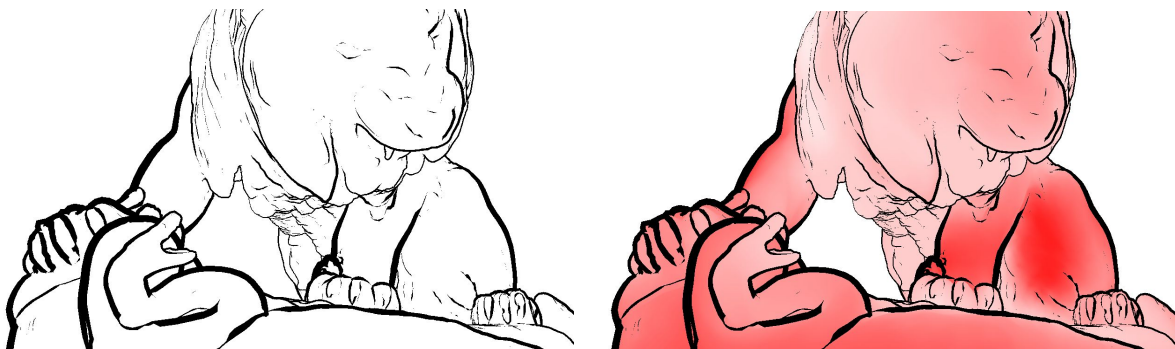


Figure 23 Left: alternate view of the lion model. Right: illustration of vertex colours.

3.2.0.3 *Different viewing angles*

Since the colour of a vertex only affects its outline when viewed (roughly) orthogonally, there may be some concern about whether certain viewing angles produce poor results. Indeed, a vertex-attribute may influence an edge when viewed from anywhere along the tangent plane. In practice, this did not prove to be much of a problem - due in part to the broad, low detail vertex colour paint job on the models. Figure 22 and figure 23 show how unusual angles have little effect on image quality. If the vertices were painted very finely (for example, red on one side of a nose or

finger and black on the other), then this issue may be worth considering. Without an interactive paint tool, such fine painting would be impractical anyway, due to the long wait for feedback.

4 Painterly Rendering

This chapter describes a technique to render a 3D model using a set of ‘strokes’ where vertex attributes determine the level of detail. The technique generates strokes each frame such that high-detail areas have more densely packed strokes.

4.1 Implementation

Before the program is run, vertex colours must be painted on to the 3D model. Each frame, a target image is rendered to a texture. The corresponding vertex colours from that image are rendered to a second texture. A set of strokes are generated with poisson disk distribution across screen space. Stroke size and density varies according to vertex colour texture.

4.1.1 Painting vertex attributes for level-of-detail

This was accomplished using the vertex paint tool in Autodesk Maya (see Painting Vertex Attributes). The process of painting vertex attributes to mark levels of detail was more intuitive than for line thickness. As mentioned in (Painting Vertex Attributes), the inverted hull painting process involved a disconnect where the vertex attributes were applied to the *surface* of the model during painting, but these attributes only affected the render when the surface faces *orthogonally* to the viewer. Painting for level-of-detail does not suffer from this disconnect. Thus, less knowledge of the algorithm is necessary to apply a good vertex paint job.

As with the inverted hull algorithm, this stroke-based rendering algorithm only uses the red channel from the vertex colours. Vertices with a high value in the red channel are considered *high detail*.

4.1.2 The Stroke-Based Rendering Algorithm

4.1.2.1 *Rendering to the textures*

Each frame, an image is rendered to a texture. We will call this the *target image*. This image can be rendered using any arbitrary rendering technique. A second texture contains the vertex colour information from the image, the *vertex colour image*. In my implementation, this texture was in full RGB format, although I only made use of the red channel.

4.1.2.2 *Generating Stroke Positions with Poisson Disk Distribution*

Stroke positions are generated with poisson disk distribution over a weighted random distribution for several reasons. The primary reason to use Poisson disk distribution is that random strokes are naturally unpredictable and do not provide consistent coverage. This is a very important trait for strokes, especially those whose size is not cleverly adapted to fill empty space. Poisson disk distribution reliably provides good coverage and it can guarantee no two strokes will be closer than a given radius; this is very important when we want low detail areas.

Another important reason is that poisson disk distribution is better suited to adapting for temporal coherence. (Vanderhaeghe, 2007) describes a method to achieve

temporal coherence of strokes using Poisson disk criteria. Unfortunately implementing this method was out of the scope of this paper.

The Poisson disk distribution method used in this paper is based on the algorithm introduced by (Bridson, 2007).

Bridson's algorithm takes as input the dimensions of the sample domain (screen size in our case), the minimum distance r between samples, and a constant k - the number of attempts each point has at generating a new sample. The contribution of this paper is the sampling of the vertex colour image to change the value of r , thereby varying the sample density across the image. So instead of one value r , this algorithm takes a *minimum distance* and *maximum distance*.

```
Begin with one random point in a list of active points.
While active points is not empty:
  Choose a random point p from active points.
  For k iterations:
    Choose a nearby point at random.
    If the nearby point is not too close to any other:
      Add it to active points.
  Remove p from active points.
```

A 2-dimensional grid is created to store samples and localise the search space. The cell size is $\frac{\text{minimum distance}}{\sqrt{2}}$ so that each cell contains at most one sample. In other words, the grid contains $\frac{\text{screen width}}{\text{cell size}} \times \frac{\text{screen height}}{\text{cell size}}$ cells. Each point in screen space maps to a cell in the grid. The items in the grid will be the locations of sample points that have been placed already.

A list is created to store 'active points' - these are points who have not been given the chance to reproduce yet. A starting point is chosen at random and added to this list - and to the grid.

While the list of active points is not empty, a point p is chosen at random from the list. **The novel aspect of this paper** is in sampling the vertex colour image at this point using the screen-space coordinates of p . The red value of the vertex colour image at this sample is used to interpolate between *minimum distance* and *maximum distance*, giving us the new value r' . For k iterations, a new point is generated randomly between radius r' and $2r'$ from p . If the new point is within the bounds of the image and not too close to any other point, then it is added to the list of active points and to the grid. After k iterations pass, p is removed from the list of active points.

The grid enables the search space for nearby points to be hugely reduced. Normally, 2 squares in each direction is enough to find any sample close enough to violate the distance condition. However, when r' increases beyond *minimum distance* (recall that we used this value to calculate cell size), the amount of grid squares containing potential violators increases. When there is a great difference between *minimum distance* and *maximum distance*, it is necessary to calculate how many cells to search. The search radius in number-of-cells is given by the expression

$(r' \div \frac{\textit{minimum distance}}{\sqrt{2}}) + 1$ to ensure all possible violations are found (Dunbar, 2006).

4.1.2.3 Rendering Strokes

Strokes are rendered in a similar way to (Collomosse, 2002), with the parameters *length*, *width*, *size*, *angle*, and *depth*. Instead of taking the form of superquadratic cones like (Collomosse, 2002), I used texture mapped quads. This is because I expect to have much larger strokes in the final images than (Collomosse, 2002).

At the location for each stroke, the gradient of the target image is calculated. The magnitude of the gradient determines the *length* of the stroke. As the stroke gets longer, it must get thinner, so *width* is calculated as $1 - \text{length}$. *length* is kept between 0.5 and 1 so that strokes in 0 gradient areas are not stretched laterally. These *length* and *width* values are used to stretch the quad upon which the stroke texture is rendered.

The gradient is used again to calculate the *angle* for the stroke. The *angle* is used to rotate the stroke in alignment with the gradient line it lies on. If it is not on a gradient line, then the stroke will be shorter, and its orientation less important.

The size of the stroke is obtained by sampling the vertex colour image's red value at the stroke location, and using that to interpolate between *minimum size* and *maximum size* variables which must be defined beforehand.

The *depth* of the stroke is equal to the depth of the pixel it lies on. (Collomosse, 2002) had the most salient strokes rise to the fore. I attempted to adapt this by having the stroke depth tied to the vertex colour image value, but with poor results.

Smaller, middleground strokes could overlap large strokes depicting low-detail foreground items. This ruined the illusion of depth in the image.

4.2 Results

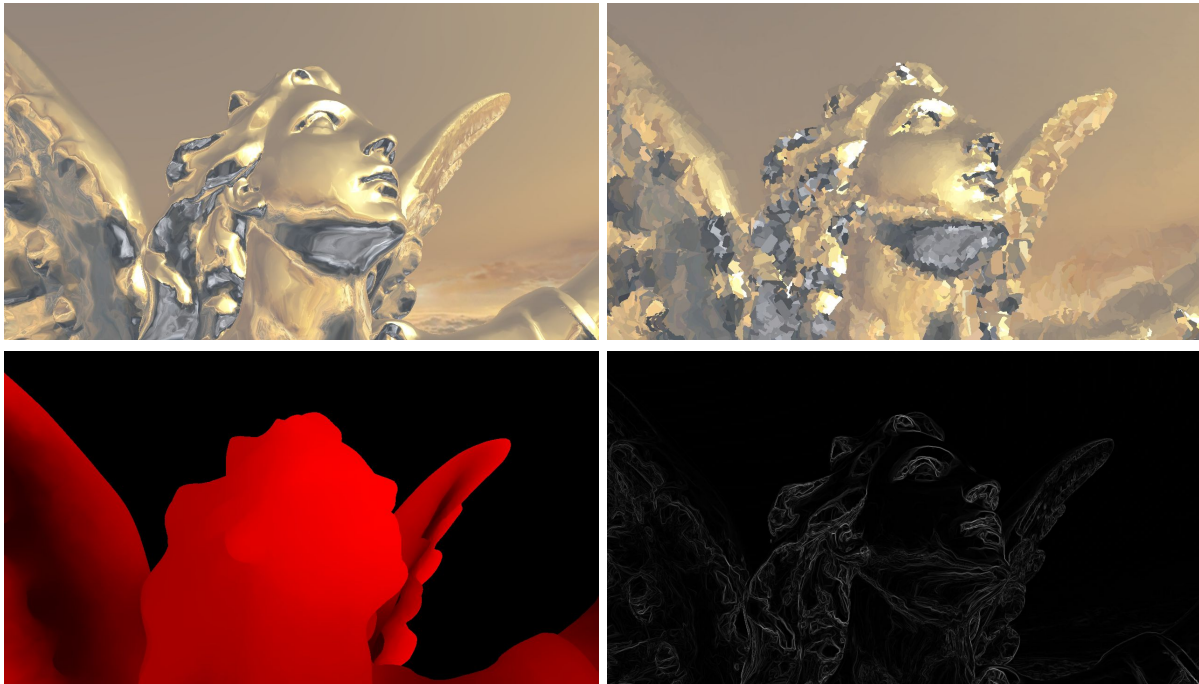


Figure 24 Clockwise from top left: (a) Target image; (b) Final stroke-based render; (c) Gradient magnitude of the target image; (d) Vertex colours;

Figure 24 shows the rendering process at different stages. The stroke size and density in (b) correlates to the colour in (c). Strokes are rotated to align with gradient lines, which is illustrated in (d). This effect can be seen more explicitly in figure 30. The gradient magnitude (d) elongates the strokes.

Different areas of the model can be emphasised by the vertex colour layout. This is shown in figure 25, where on one side, the rear fin is detailed, and on the other, the top and side fins are. Figure 29 compares uniform stroke density with varied stroke density.

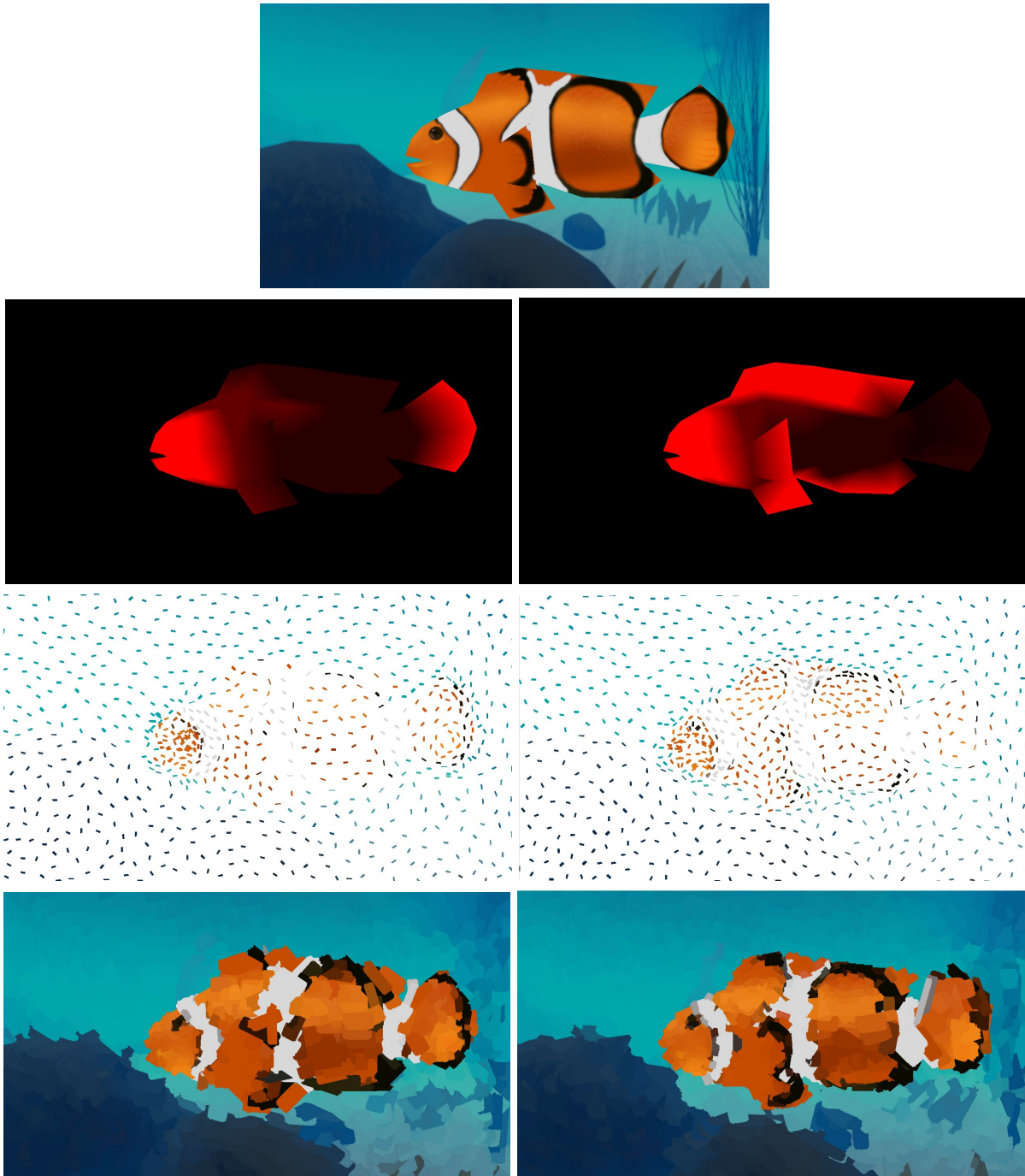


Figure 25 Two different vertex colour configurations on the same model. Source image on top; Left column shows one vertex colour configuration, stroke placement, and an example rendered image; Right column shows the same with another vertex colour configuration.

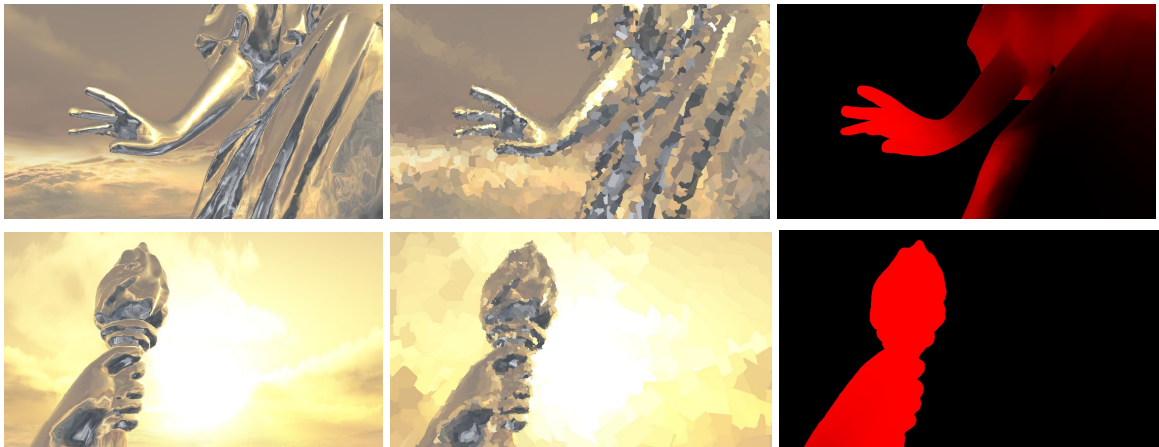


Figure 26 From left to right: Target image; Final render; Vertex colours.

4.2.0.1 Aesthetics

The images can be quite pleasing. As shown in figure 26, the high-detail areas are clearly a focal point of the images, while the sky and robes melt into the background. Figure 27 shows how high-detail areas can be in the middle-ground of the image as well. figure 28 shows how an arbitrary shader can be used to produce the target image.

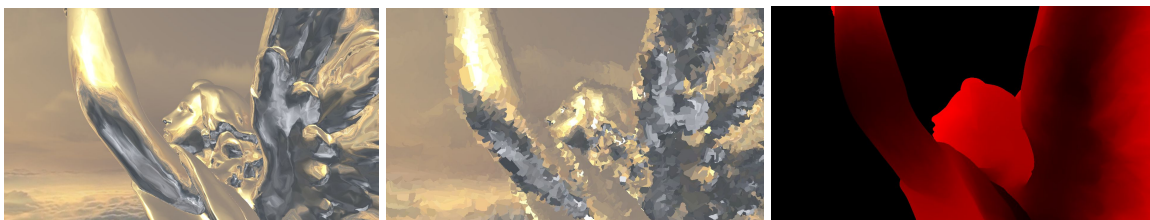


Figure 27 From left to right: Target image; Final render; Vertex colours

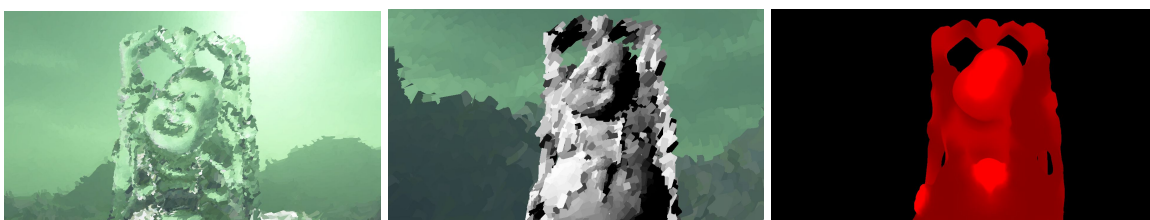


Figure 28 From left to right: Glass-like shader; Basic lambertian shader; Vertex colours.

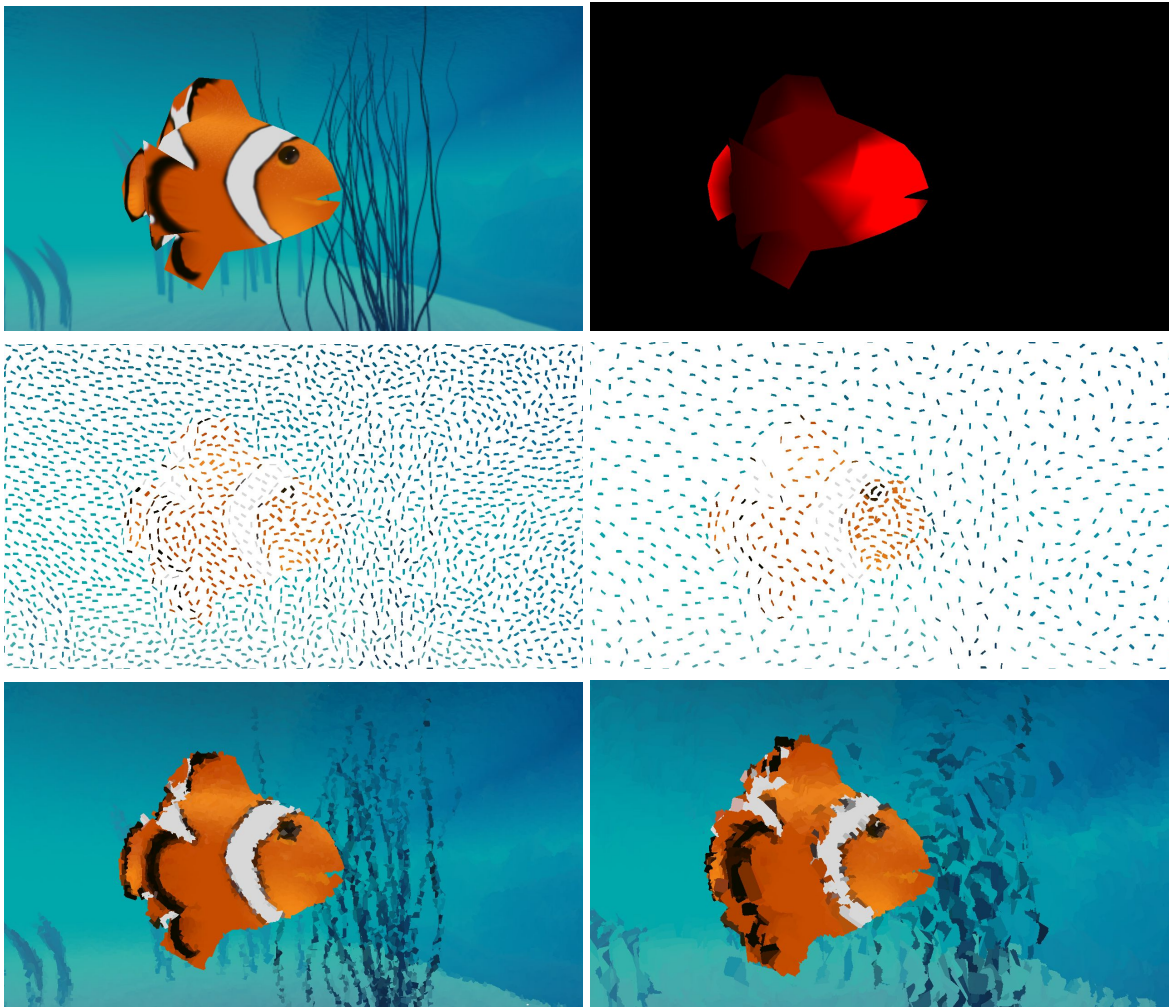


Figure 29 Showing the difference between a uniform minimum distance and the varied minimum distance offered by this technique. Top left is the source, beside it is the vertex colour image. Middle row shows uniform minimum distance vs varied minimum distance. Bottom row shows example renders with each.

4.2.0.2 *Tuning the stroke parameters for full stroke coverage*

As shown in figure 30, the stroke parameters can make or break the quality of the image. The stroke size range and stroke coverage must be configured separately for full coverage, even though they are codependent.

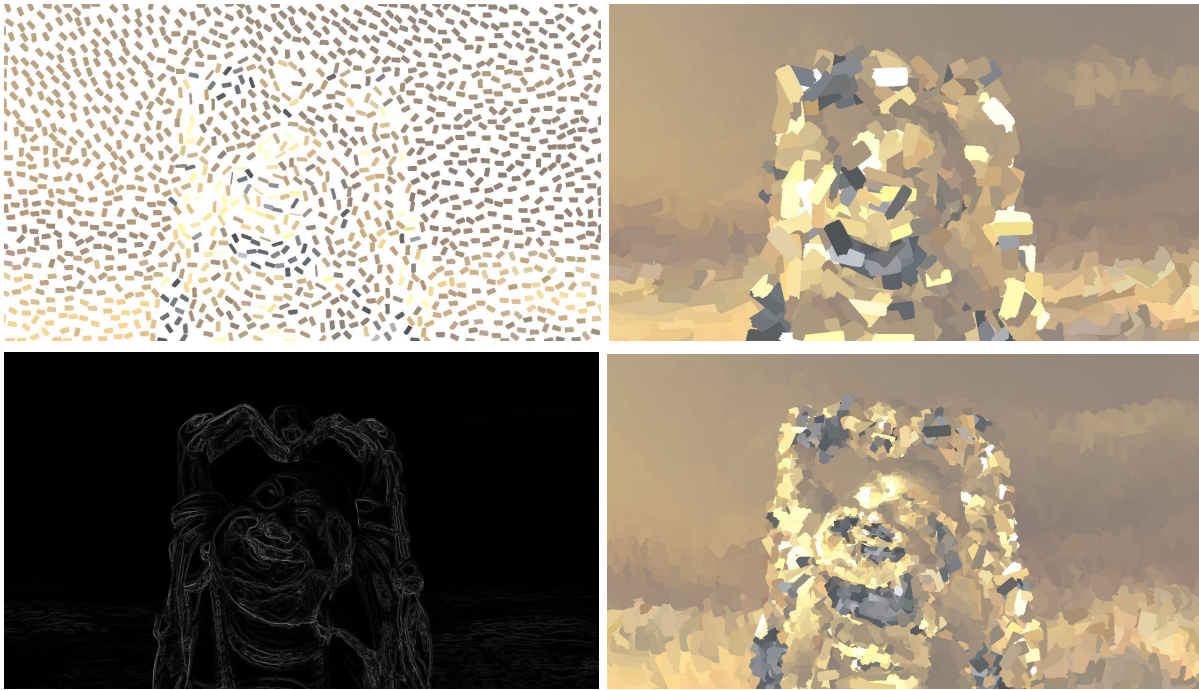
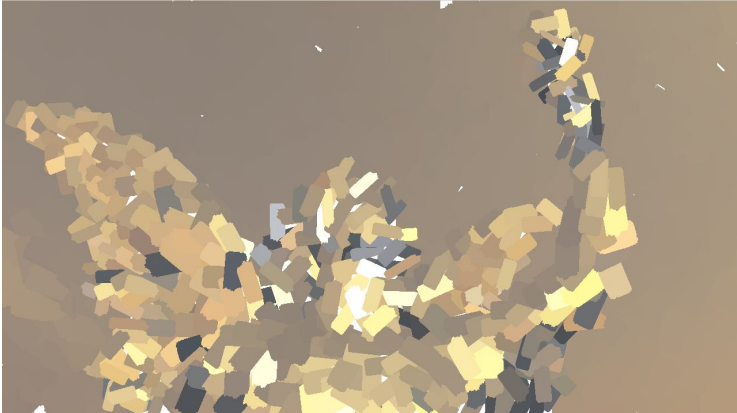




Figure 30 Clockwise from top-left: (a) Stroke size is too small with respect to stroke density; (b) Strokes are too large and sparse to convey details; (c) Stroke parameters tuned for full coverage and better detail; (d) Image gradient magnitude.

4.2.0.3 *Real time poisson disk distribution*

Generating the set of strokes is the main performance bottleneck. Figure 31 shows how frame times vary with different stroke generation parameters. Notice how the image is quite hard to decipher with just ~1000 strokes. The implemented algorithm does not limit the number of strokes, so it can be difficult to tune the parameters for performance. Furthermore, the amount of strokes generated depends largely on the amount of red in the vertex colour image. Figure 32 shows how different viewing positions can have vastly different rendering times, even when using the same stroke parameters.

Mean stroke count per frame	Mean frame time (milliseconds)	Sample image
1013	66.669	
2031	119.745	
4347	270.814	

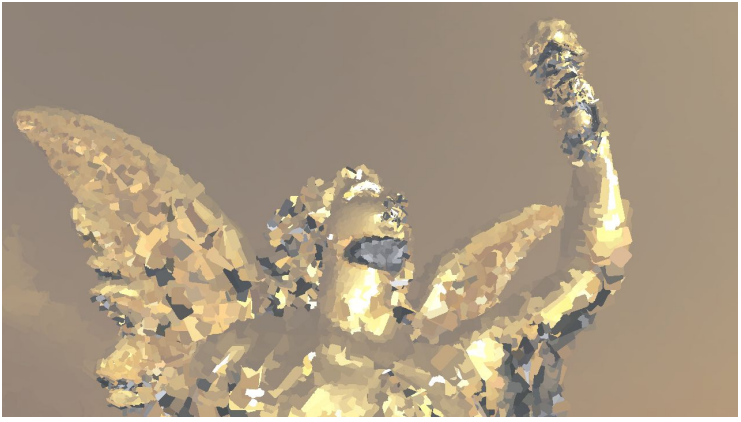
16086	974.877	
-------	---------	--

Figure 31 Same target image rendered with different stroke generation parameters. Times were recorded on Intel Core i7-7700K CPU @ 4.20 GHz and NVIDIA GeForce GTX 1080 Ti.

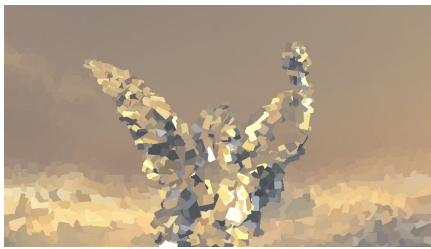

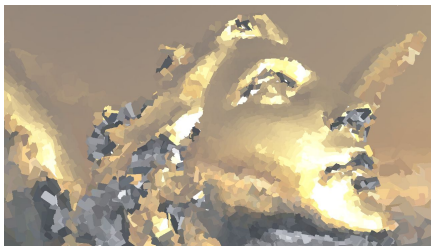

Stroke count	Time (ms)	Rendered image	Vertex colours
2453	155.149		
4082	247.81		

Figure 32 Different target images rendered with exactly the same parameters. Notice the effect of large red areas on performance. Again, times were recorded on Intel Core i7-7700K CPU @ 4.20 GHz and NVIDIA GeForce GTX 1080 Ti.

5 Conclusions and Future Work

In this paper we introduced a painterly rendering technique that uses vertex attributes to control level of detail. We also examined the use of vertex attributes in an old line drawing technique. In general, the vertex attributes were intuitive to apply using available tools. The varying of line thickness in the Inverted Hull algorithm is aesthetically pleasing and helps to overcome the deformation side-effect. The painterly rendering technique produces some aesthetically pleasing images with controlled detail. It was possible to emphasize the focal points of a 3D model by using a deliberate vertex colour configuration. Perhaps more importantly, it allows the less important elements of the image - which may be just as detailed - to be abstracted away by large, sparse strokes.

There are many ways vertex attributes can be exploited in future. First of all, an interactive paint tool wouldn't go amiss, since the delay between painting vertex colours and seeing the results is not conducive to experimentation with different layouts.

As for line drawing, the inverted hull rendering algorithm is primitive. The lines it draws are not the most efficient for rendering the shape of a 3D model alone. It is best used as a silhouette shader, with another algorithm handling the interior. An obvious step would be to use vertex attribute influenced variable stroke thickness with one of the more sophisticated algorithms like (DeCarlo, 2004) or (Judd, 2007) -

but this would be in direct competition with (Goodwin, 2007) which does a good job of varying line thickness already. Instead, I think a good way to proceed would be to take the approach of detail painting used in section 4.1.1 and use the vertex attributes to cause or prevent stroke culling. Both (DeCarlo, 2004) and (Judd, 2007) use thresholds which could be changed per vertex.

There is no limit to what could be done with painterly rendering. The first might be to implement temporal coherency as described in (Vanderhaeghe, 2007). This would greatly improve the suitability of the technique for use in animation.

Some small improvements to the technique presented could include better abstraction of low detail areas - perhaps using a gaussian blur on the source image with the vertex colour image as mask.

There are many painterly rendering techniques (both image-based and 3D) that could be adapted to use vertex attributes. Since the 3D models can be rendered to 2D images, 2D techniques would work just as well. In my opinion, the best step forward for 2D based painterly rendering would be to adapt (Hays, 2004) to use an image rendered from 3D data with accompanying vertex colour image as a detail mask, since it addresses temporal coherency.

As for making the technique realtime, it would be a dramatic pivot, but adapting an approach like (Meier, 1996) and pre-attaching strokes of varying density to a 3D model might be worthwhile.

References

Bando, Y., Kuratate, T. and Nishita, T., 2002. *A simple method for modeling wrinkles on human skin*. In Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on (pp. 166-175). IEEE.

Bridson, R., 2007, August. *Fast Poisson disk sampling in arbitrary dimensions*. In SIGGRAPH sketches (p. 22).

Bénard, P., Lu, J., Cole, F., Finkelstein, A. and Thollot, J., 2012, June. *Active strokes: coherent line stylization for animated 3D models*. In Proceedings of the symposium on non-photorealistic animation and rendering (pp. 37-46). Eurographics Association.

Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H.S., Finkelstein, A., Funkhouser, T. and Rusinkiewicz, S., 2008, August. *Where do people draw lines?*. In ACM Transactions on Graphics (TOG) (Vol. 27, No. 3, p. 88). ACM.

Collomosse, J.P. and Hall, P.M., 2002. *Painterly rendering using image salience*. In Eurographics UK Conference, 2002. Proceedings. The 20th (pp. 122-128). IEEE.

DeCarlo, D., Finkelstein, A., Rusinkiewicz, S. and Santella, A., 2003. *Suggestive contours for conveying shape*. ACM Transactions on Graphics (TOG), 22(3), pp.848-855.

DeCarlo, D., Finkelstein, A. and Rusinkiewicz, S., 2004, June. *Interactive rendering of suggestive contours with temporal coherence*. In Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering (pp. 15-145). ACM.

Dunbar, D. and Humphreys, G., 2006, July. *A spatial data structure for fast Poisson-disk sample generation*. In ACM Transactions on Graphics (TOG) (Vol. 25, No. 3, pp. 503-508). ACM.

Goodwin, T., Vollick, I. and Hertzmann, A., 2007, August. *Isophote distance: a shading approach to artistic stroke thickness*. In Proceedings of the 5th international symposium on Non-photorealistic animation and rendering (pp. 53-62). ACM.

Grabli, S., Turquin, E., Durand, F. and Sillion, F.X., 2004. *Programmable style for NPR line drawing*. In Rendering Techniques 2004 (Eurographics Symposium on Rendering). ACM Press.

Haeberli, P., 1990, September. *Paint by numbers: Abstract image representations*. In ACM SIGGRAPH computer graphics (Vol. 24, No. 4, pp. 207-214). ACM.

Hausner, A., 2001, August. *Simulating decorative mosaics*. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (pp. 573-580). ACM.

Hays, J. and Essa, I., 2004, June. *Image and video based painterly animation*. In Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering (pp. 113-120). ACM.

Hegde, S., Gatzidis, C. and Tian, F., 2013. *Painterly rendering techniques: a state-of-the-art review of current approaches*. Computer Animation and Virtual Worlds, 24(1), pp.43-64.

Hertzmann, A., 2001. *Paint by relaxation*. In Computer Graphics International 2001. Proceedings (pp. 47-54). IEEE.

Hertzmann, A., 2003, July. *A survey of stroke-based rendering*. Institute of Electrical and Electronics Engineers.

Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S. and Strothotte, T., 2003. *A developer's guide to silhouette algorithms for polygonal models*. IEEE Computer Graphics and Applications, 23(4), pp.28-37.

Judd, T., Durand, F. and Adelson, E., 2007, August. *Apparent ridges for line drawing*. In ACM Transactions on Graphics (TOG) (Vol. 26, No. 3, p. 19). ACM.

Kalnins, R.D., Markosian, L., Meier, B.J., Kowalski, M.A., Lee, J.C., Davidson, P.L., Webb, M., Hughes, J.F. and Finkelstein, A., 2002. *WYSIWYG NPR: Drawing strokes directly on 3D models*. ACM Transactions on Graphics (TOG), 21(3), pp.755-762.

Kalnins, R.D., 2004, June. *WYSIWYG NPR: Interactive Stylization for Stroke-Based Rendering of 3D Animation*. PhD Thesis, Princeton University.

Kaplan, M., Gooch, B. and Cohen, E., 2000, June. *Interactive artistic rendering*. In Proceedings of the 1st international symposium on Non-photorealistic animation and rendering (pp. 67-74). ACM.

Lagae, A. and Dutré, P., 2008, March. *A comparison of methods for generating Poisson disk distributions*. In Computer Graphics Forum (Vol. 27, No. 1, pp. 114-129). Blackwell Publishing Ltd.

Lou, L., Wang, L. and Meng, X., 2015. *Stylized strokes for coherent line drawings*. Computational Visual Media, 1(1), pp.79-89.

McCool, M. and Fiume, E., 1992, September. *Hierarchical Poisson disk sampling distributions*. In Proceedings of the conference on Graphics interface (Vol. 92, pp. 94-105).

Meier, B.J., 1996, August. *Painterly rendering for animation*. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (pp. 477-484). ACM.

Santella, A. and DeCarlo, D., 2002, June. *Abstracted painterly renderings using eye-tracking data*. In Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering (pp. 75-ff). ACM.

Strothotte, T. and Schlechtweg, S., 2002. *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann.

Vanderhaeghe, D., Barla, P., Thollot, J. and Sillion, F.X., 2007, June. *Dynamic point distribution for stroke-based rendering*. In Proceedings of the 18th Eurographics conference on Rendering Techniques (pp. 139-146). Eurographics Association.

Winkenbach, G. and Salesin, D.H., 1994, July. *Computer-generated pen-and-ink illustration*. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques (pp. 91-100). ACM.