Gradient Domain Rendering

Category: Research

(1) Original drawing (2) Normal Map / 2D vector field (3) Shading parameter image $c(u, v)$

(4) Control Image 0: $DI_0(u, v)$ (5) Control Image 1: $DI_1(u, v)$ (6) Rendering

Figure 1: *An example of the applications of 2D gradient domain rendering: re-interpretation of an artwork. (1) An artist's original drawing; (2) a 2D vector field (similar to normal map) created from the original drawing; (3) a shading parameter image, $c(u, v)$ is a gray-scale image that provides the combined effect of our shading, shadow and ambient occlusion computations; (4) and (5) Two control images provided by the artist; (6) The rendered image created by interpolating the two images $DI_0(u, v)$ and $DI_0(u, v)$ using $c(u, v)$ as $(1 - c(u, v))DI_0(u, v) + c(u, v)DI_1(u, v)$. The final image is more volumetric looking than the original drawing with subtle effects of shadow and shading even though there is no true 3D shape.*

Abstract

In this paper, we present an approach to allow artists to create 3D-looking stylized depictions with complete visual control. The core of our approach is to compute rendering effects directly in gradient domain. This approach allows us to use 2D vector fields that may not necessarily correspond to any 3D shape, but that can still help to compute a 3D appearance. To use our approach, an artist provides two images and a corresponding 2D vector field. Final images are created by interpolating between the two images using shading information derived from the 2D vector field.

We show that 2D vector fields is sufficient to obtain qualitatively convincing visual effects. Rendering methods we have developed include diffuse shading, ambient occlusion, soft & hard shadows; and environment reflection using rectangular images. Although these methods do not directly correspond to underlying physical phenomena, they can provide results that are visually similar to 3D realistic rendering. One of the major advantages of our approach is the ability to treat images as "mock-3D" shapes that can be visually acceptable even when images do not correspond to real 3D shapes.

Another advantage of our approach is that the artists can interactively control the illumination and rendering processes to intuitively obtain desired visual results. Since 2D vector fields, themselves, are images, so they can be represented using any convenient 2D raster or vector image format like normal maps. The fact that the 2D vector field is itself an image makes it a very painter-friendly representation, subject to creation and manipulation by both algorithmic and direct approaches.

1 Introduction and Motivation

Normal maps became instantly popular as soon as they are introduced in 1998 [Cohen et al. 1998]. Although,they are mainly used as texture maps to include details to polygonal

meshes, they can directly be used as shape representations. In fact, Johnston developed Lumo to model normal maps as mock-3D shapes by diffusing 2D normals in a line drawing [Johnston 2002]. Despite the potential power of normal maps as shape representations, only a few groups investigated the potential use of normal maps as mock-3D shapes [Okabe et al. ; Bezerra et al. 2005; Shao et al. 2012]. Existing methods focused on modeling these normal maps. To render these mock-3D shapes, only basic 3D rendering methods are employed and the rendering effects such as shadow or occlusion has never been included. This shortcoming is not really unexpected. For standard 3D rendering, shadow or occlusion computation requires a 3D shape description such as a height field. However, it is not always possible to reconstruct a height field from the normal maps since a given vector field may not necessarily be gradient of any height field.

To unleash the true power of this representation for 2D artists, there is a need for a new rendering framework that is designed to work especially in gradient domain. In this paper, we present such a gradient domain rendering framework. Moreover, our framework is completely 2D: we ignore z component of normal maps, instead, we use 2D lights and rectangular environment maps. Most importantly, we obtain realistic looking local shadows and ambient occlusion directly from 2D vector fields.

This rendering framework based on 2D vector fields is still in sync with existing normal map representation. For instance, Lumo modeling requires to interpolate only x, y components of the normal vectors, z component is ignored and computed to maintain unit length [Johnston 2002]. In fact, z component in any normal map does not add any additional information. It is therefore possible to view any normal map as a 2D vector field. However the opposite is not always correct: the length of a 2D vector can be longer than one even if each component is smaller than one.

We see this flexibility as a user-interface advantage for painters who may want to directly paint gradient domains without considering any constraint. Since our rendering framework still works with any 2D vector, possible mistakes in vector length do not affect our results. In other words, using a relatively flexible 2D vector fields provide efficiency and ability for artists to directly create and manipulate artworks and allow intuitive artistic control over visual results.

One advantage of using 2D vector fields and 2D lights is that the shading parameters we have computed for every pixel guarantee to provide a whole gamut of numbers between 0 and 1. Using this shading parameter c as interpolation parameter for every pixel, we interpolate two artist-provided control images, DI_0 and DI_1, to obtain final image. There is really no requirement for creating DI_0 and DI_1. For any given pixel (u, v) the color of DI_0 is the color the artist wants to see if there is no light reaches to that pixel, i.e. $c(u, v) = 0$. Similarly, for any given pixel (u, v) the color of DI_1 is the color the artist wants to see if this point is completely illuminated by the light, i.e. $c(u, v) = 1$. In other words, this process guarantees to obtain colors exactly like the artist wanted. An example is shown in Figure 2.

This approach can be used in a wide variety of 2D applications including digital painting, artistic filtering, reinterpretation of paintings and illustrations (See Figure 1). Artists can create artificial, but still believable, versions of the original images as well as original art work that can be dynamically manipulated with complete control over final results or they can reinterpret existing artworks such as the one shown in Figure 3.

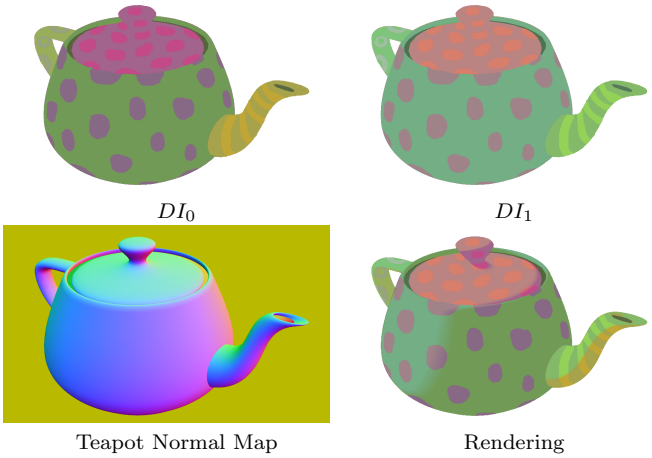


Figure 2: *This example demonstrates the results for DI_0 and DI_0 images that are freely painted by an artist. The teapot normal map is a Lumo model, used in permission.*

1.1 Summary of Contributions

The existing computer graphics approaches for creating and manipulating 2D artworks are mostly based on image processing techniques. 3D graphics techniques, on the other hand, can potentially be useful to develop efficient and simple to use methods for 2D artwork creation and manipulation. 2D vector fields provides a natural bridge between 2D and 3D computer graphics.

Our 2D vector field based framework can be broadly identified as a gradient domain approach. Solving problems in gradient domain turned out to be very useful for a wide range of 2D computer graphics applications such as tone mapping [Fattal et al. 2002], image editing [Perez et al. 2003], matting [Sun et al. 2004], image stitching [Levin et al. 2006], and even bas relief construction [Weyrich et al. 2007]. Existing approaches mainly employ the gradient domain for processing data. After the processing, they return back the original domain usually by solving an optimization problem, such as Poisson formulation [Fattal et al. 2002].

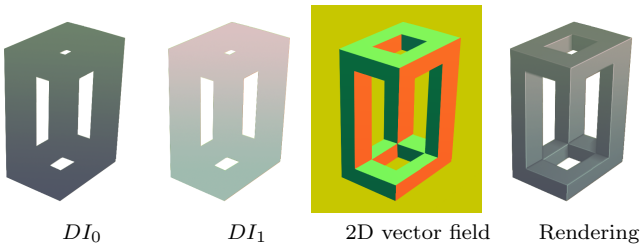


Figure 4: *Rendering an impossible object with local shadows and ambient occlusion. Particularly note the line drawing effect coming from ambient occlusion. Control images and 2D vector field are painted by an artist in a digital painting program.*

$DI_0(u,v)$ $DI_1(u,v)$ 2D vector field Rendering 1 Rendering 2

Figure 3: *A example of re-rendering of a cubist painting: A diffusely relit re-interpretation of Picasso's self-portrait from 1907. Note that although this image is intentionally flattened and do not correspond any real shape, it is still possible to illuminate it with our approach. 2D vector field and control images are all painted manually by an artist inspired by Picasso's original painting. The whole process of creating 2D vector field and control images did not take more than one hour using a digital painting program.*

In this paper, we make all our computations in gradient domain, but we never reconstruct 3D shapes. As a result our computations are naturally efficient. However, the real benefit of our approach is not computational efficiency. Since we do not have to deal with 3D reconstruction, we can represent, model, render and composite with inconsistent and impossible shapes. This property makes our approach artist friendly. Our practical contributions can be summarized as follows:

Simplified Gradient Domain Representation: We have identified that a 2D vector field is sufficient to represent mock-3D shapes. Using this minimal information, we can compute all 3D effects as if they are real 3D shapes. As a result of this minimalist approach, our 2D vector fields are backward compatible, i.e. they can be stored as images and can be considered normal maps. As a consequence of this result, any image processing method can be used to create and manipulate 2D vector fields and any image can be designated as a 2D vector field. A formal definition of a 2D vector field is made in section 2.
Parametric Shading with Control Images: We introduce a shading approach, in which the final images are computed by interpolating a set of artist-provided control images. This approach provide simple and intuitive control to artist over visual results. For practical reasons, we use only two control images, but artists can provide more control images for finer control of final results.
Shadow and Ambient Occlusion computation in Gradient Domain: We have developed image processing methods to compute ambient occlusion, soft and hard shadows directly from the 2D vector field. To compute an acceptable shading we do not require precise, correct, or consistent 2D vector fields. In addition, we have developed a generalized version of Gooch shading [Gooch et al. 1998] that provides predictable results for art direction. Basic rendering operations are formally discussed in section 3. **Rectangular Images for Mirror Reflection:** We have developed an alternative to sphere mapping to obtain mirror reflection. Sphere mapping uses circular images for reflection mapping. Photographing circular images are easy using gazing balls. However it is not intuitive to paint circular images. We, therefore, show that rectangular images can be used for mirror reflection. Painting such rectangular images is simple and intuitive for any painter.

While it is preferable for artists to directly paint 2D vector fields with complete control, it is always possible to convert real objects into 2D vector fields. The procedure for obtaining a 2D vector field is a straightforward rendering process. The x and y components of the 3D-normal vector of the visible point are simply converted to red and green colors in the image.

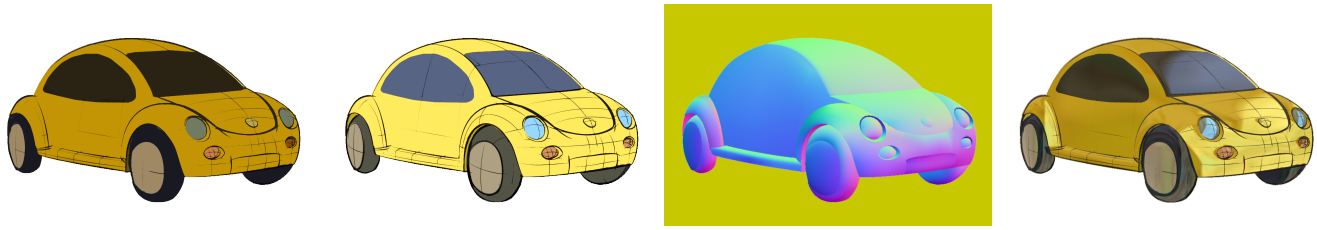
A better option, particularly for illustrators, is to model 2D vector fields directly with sketch based interface. To create the 2D vector fields, we have implemented a simple modeling system based on boundary gradients interpolation suggested by Johnston [Johnston 2002]. For interpolation, we use a simple medial-axis based [Igarashi et al. 1999] sketch based modeling tool which uses subdivision [Nasri et al. 2009]. Since modeling is not the focus of this paper, we do not include details here. Our rendering approach does not depend on our models. Normal maps created by Lumo [Johnston 2002] or CrossShade [Shao et al. 2012] can be rendered in our framework with shadow and reflection. Figure 6 shows examples of shadow and reflection effects on Lumo and CrossShade models.

1.2 Motivation got Gradient Domain Representations

We are inspired by the recent results that suggest the gradient domain representations can be natural for humans. In particular, Cole et al. [2009] showed that people can correctly estimate normal vectors from line drawings and, Shado et al. [2012] developed CrossShade, a sketch based interface to design complicated shapes as normal maps. The normal maps generated by CrossShade can be depicted as if they are 3D shapes. Based on these results, we observe that (1) it is easy for humans to estimate the gradient domain, and (2) it should be easy to create gradient domain representations instead of full 3D representations. This suggests that gradient domain representations could be easily manipulated by artists and play an important role for designing expressive and artistic images. We expect that in the near future, we will start to see more examples of gradient domain

DI_0 DI_1 Normal Map Rendering Detail

Figure 5: *Shadow and reflection on a Lumo cat normal map model. Note in the detail image (1) the reflection of a hand-drawn rectangular environment map on eyes and nose of the cat (2) local shadows around the convex shapes such as mouth and cheeks, (3) line drawing effect obtained directly from ambient occlusion. Control images are painted by an artist starting from one of the rendered images in the paper. We only changed the backgrounds of original images from white to yellow differentiate outside regions (i.e. blue component is zero). The model is used in permission.*



DI_0 DI_1 Normal Map Rendering

Figure 6: *Shadow and subtle reflection on CrossShade WV normal map model. DI_0 and DI_1 images are painted by an artist starting from the rendered images in the paper. We only changed the backgrounds of original images from white to yellow differentiate outside regions (i.e. blue component is zero). The model is used in permission.*

approaches.

Another motivation for gradient domain representations comes mainly from inconsistencies that frequently exist in images of shapes. Turning such images of objects into 3D-realistic shapes is not always possible. The gradient domain representations provide a flexible, efficient and practical solution to this reconstruction problem. Images of objects, particularly in paintings and illustrations, introduce significant challenges for shape reconstruction. Even when images in paintings look 3D-realistic, there can still be inconsistencies. Most artists do not even try to draw truly 3D-realistic images of shapes. It is, therefore, extremely hard to create a 3D-realistic shape and identify correct transformations (including perspective) such that the boundaries of transformed 3D-realistic shapes exactly match the boundaries of painted shapes. This registration problem has been an important issue in visual perception since it is visually easier to detect discrepancies in boundary regions.

Gradient domain mock-3D shapes provides solution to this registration problem bypassing the whole process of estimating 3D transformations and shapes of objects. When we directly design corresponding shapes in 2D we effectively eliminate registration problems. In 2D it is easy to exactly match the boundaries of the images and models. This approach directly leads in re-interpretations of paintings. The 2D vector fields created by these re-interpretations are imprecise everywhere except at boundaries. Accuracy at boundaries is sufficient to avoid registration errors. To obtain accurate boundaries in 2D, many such methods exist [Li et al. 2004].

The simplicity of the gradient domain representation can also be useful in a wide variety of applications. 3D reconstruction of physical objects from a single photograph is one such application. Although these objects are real and have a well-defined shape in 3D, we still do not know actual transformations that turn them into images. In addition, the physical objects can have a wide variety of "shape imperfection", which cannot easily be captured with 3D-models. In other words, to replace any photographic image of a physical object with a virtual object, registration always presents a challenge. The problem can be even more difficult if the physical objects are transparent and reflective. In those cases, it can be impossible to estimate exact 3D shape from a single photograph [Zongker et al. 1999]. A remarkable example in this direction is Single-view relighting with normal map painting [Okabe et al.].

Another potential application of this approach is character design. When artists design unusual and exaggerated characters, 3D-realism is less important than how interesting the character will look from a set of given viewpoints. This design approach allows for the drawing of characters that look appealing in 2D, but are impossible to be converted into 3D-models without view-dependent models [Rademacher 1999]. In such cases, it again makes sense to bypass the 3D-modeling process such as Lumo [Johnston 2002; Bezerra et al. 2005]. Figure 1 is another example of 2D character that is rendered with our approach by interpolating two images.

An additional application of this approach is rendering im-

possible shapes (See Figure 4). This is particularly useful since it is not possible to reconstruct corresponding height fields for most impossible objects. For some of the impossible shapes corresponding 3D shapes can be constructed [Elber 2011], but they cannot simply be obtained by automatic reconstruction. Purposely non-realistic images such as cubist paintings, multi-perspective images, expressionist and abstract paintings can also be turned into such mock-3D models that can be dynamically manipulated/re-interpreted/re-rendered (See Figure 3).

2 Definition of Gradient Domain

2D vector fields are essentially orthographic projections of shapes in z, i.e. $(0,0,1)$ direction. Let a square $M = [0,1] \times [0,1]$ denote a 2D vector field and let $(u,v) \in M$ denote two coordinates of the field. Let $(x(u,v), y(u,v))$ denote the 2D vector field with $x : M \to [-1,1]$,, $y : M \to [-1,1]$, and $d : M \to [0,1]$.
We have identified that it is sufficient to provide two coordinates of the normal vector, $(x(u,v), y(u,v))$, to obtain realistic looking diffuse shading, ambient occlusion, local shadows, shadows, mirror and glossy reflection with artistic control. Artists can completely ignore the third dimension during the creation of this vector field. It is even possible to allow inconsistencies and imperfections.

2D vector fields are a natural conceptual follow-up of normal maps [Cohen et al. 1998]. If we consider x, y of normal maps as a 2D vector field, any normal map can directly be used as a 2D vector field. If the 3D-normal vector is a unit vector, a 2D-vector field, (x, y), is sufficient to extract the third dimension information since the value of z can always be computed as a $z(u,v) = \sqrt{1 - x^2 - y^2}$ (z can never be negative). This assumption is correct only if normal vectors are computed correctly as unit vectors from a given 3D shape. Fortunately, even when the 2D vector field is imprecise it is still possible assume that z component is implicitly provided. The main issue is that in 2D vector fields it is possible to have $x^2 + y^2 > 1$ since the 2D vector field data can be imprecisely created by users. This is not really a problem since we can always assume that the 2D vector field is uniformly scaled by a parameter $w \in [0,1]$. For w values smaller than $1/\sqrt{2}$, the unit 3D normal vector always exist $n(u,v) = (wx, wy, \sqrt{1 - w^2x^2 - w^2y^2})$.

We want to point out that such a scaling in gradient domain corresponds to the flattening of the 3D shape and it is a visual equivalent of flattening sculptures into bas reliefs [Weyrich et al. 2007]. The rendering results, therefore, are not expected to create visual problems and it is possible to obtain visually acceptable results from imperfectly defined 2D vector fields using low w values. For high quality 2D vector fields, it is always possible to use $w = 1$. For others a value smaller than $w = 0.7$ always works. In practice, our default value for w is $w = 0.5$.

The advantage of 2D vector fields is that they are not required to correspond to any given shape. Even when there is no imperfection in the data, there may still not exist any height field whose gradient can produce the given 2D normal vector field. To obtain a height field, that can minimize errors, we have to solve Poisson Equation [Fattal et al. 2002]. This expensive computation is unnecessary for 2D vector fields. It is still possible to estimate shadows in a given light direction without explicitly obtaining a height field by computing a line integral convolution [Cabral and Leedom 1993].

2.1 Gradient Domain Painting

One significant advantage of using normal maps is that we can readily convert 2D vector fields into Low Dynamic Range (LDR) images and save them using any common image format which can easily passed to GPU. We assume that a LDR image on M is denoted by $c(u,v) = (r(u,v), g(u,v), b(u,v))$ where $c : M \to [0,1]^3$. The conversion from (x,y) to (r,g) is given as $(r = 0.5(x+1), g = 0.5(y+1))$.

Our figures show examples of 2D vector fields that are encoded as images. Note that in these images blue values are not zero although we do not really need blue component. In our case, we use blue as a thickness parameter; i.e. $b = 0$ means object does not exist. The yellow backgrounds in our 2D vector field images are simply non-object regions. Inside of the object, blue value can be anything other than zero. As can be seen in the figures, the 2D vector field images are more colorful than normal maps since we do not really compute the blue color in tandem with red and green.

The main advantage of thinking of 2D vector fields as images is that artists can create 2D vector fields directly using a 2D painting software. Most importantly, the painters do not need to think that they are working on a gradient domain to paint these images. They imagine an object lit by 2-point lighting illuminated from left with a directional (parallel) red light and from above by a directional green light. Ignoring shadows, they can paint the image based on how much red and green light they want to see in every pixel. For instance, a pixel color red=0.75 and green=0.3 means, the artist wants 75% of the light from the left and 30% of the light from the top can illuminate that particular pixel.

These red-green paintings, although imperfect, provide good estimations of 2D gradient domains. Since red and green light vectors $(1,0)$ and $(0,1)$ are linearly independent from each other, any 2D light can be given a linear combination of the two as $(x_L, y_L) = x_L(1,0) + y_L(0,1)$. Therefore, to compute illumination coming from an arbitrary parallel light, we simply compute the contribution from two linearly independent components.

Providing direct control to artists is essential for the creation of unusual images since the artists can deliberately introduce imperfections that lead to expressive and artistic effects. The same imperfections also provided crosshatching in cartoon shading under diffuse illumination as shown in Figure 7. It is also easy to paint these 2D vector fields as images. Creation of none of painted 2D vector fields shown in this paper took more than one hour using a digital painting program by an experienced painter.

2.2 Parametric Shading with Control Images

One of the contribution of this paper is the introduction of the concept of control images to obtain predictable visual results. We assume that final diffuse image is simply computed as a weighted average of control images using a Bezier curve as

$$I(u,v) = \sum_{i=0}^{N} DI_i(u,v) B_i(c(u,v)) \tag{1}$$

2D vector field Normal Cartoon

Figure 7: *The effect of the cartoon shading. In this case, control images are simply black and white. Note that in this case crosshatching effect comes directly from hand-drawn vector field. We expect direct involvement of painters into the process of creating normal maps can result in innovative solutions.*

where $B_i(c(u,v))$ denotes Bezier basis functions, $DI_i(u,v)$ denotes Bezier control images, $0 \leq c(u,v) \leq 1$ is a shading parameter that is computed for one key light. One advantage of this approach is that the color of the key light and ambient illumination is embedded in the control images and directly controlled by the painter. Because of convex hull property of Bezier curves, the equation guarantees that the result is also an image and it stays in the convex hull of control images. The advantage of Bezier basis functions over others that provide convex hull property such as B-spline is that Bezier formulation guarantees that the curve interpolates the first and last control images. In other words, if our shading computation can guarantee to obtain 0 and 1, the painter defined colors are always obtained in final image. As a conclusion, the obvious advantage of this formulation is that it can help artists to plan exactly what kind of results they expect to see.

If there are more than one light source, we simply use tensor product Bezier functions. For such cases, the artist needs to provide $K(N+1)$ images for K lights and N degree Bezier functions. In practice, it is hard to create such a large number of images consistently especially when both K and N is large. In practice, we noticed that only one key light and $N = 1$ is sufficient for basic control of visual results. Therefore, without loss of generality, all our examples in this paper uses only two control images and only one key light. In this case, the general Bezier curve equation simplifies into

$$I(u,v) = DI_0(u,v)(1 - c(u,v)) + DI_1(u,v)c(u,v) \qquad (2)$$

where $DI_0(u,v)$ and $DI_1(u,v)$ denote two images that are provided by the artist. The first image defines the color when the key light does not illuminate any given point (u,v). The second image $DI_1(u,v)$ defines the color when the key light fully illuminates any given point. In all our examples DI_0 and DI_1 are always painted by an artist. It is also noted that that this formulation Gooch shading formulation [Gooch et al. 1998]. The advantage of Bezier formulation can be demonstrated by comparing this equation with classical rendering formulation that is given in a ray formula as

$$I(u,v) = DI_0(u,v) + VI(u,v)c(u,v) \qquad (3)$$

Note that we did not change $DI_0(u,v)$, since mathematically speaking it corresponds to the ambient term of classical shading equation. However, the term correspond to diffuse term of classical equation is a vector and will be computed as $VI(u,v) = DI_1(u,v) - DI_0(u,v)$ to obtain the same result. Note that $VI(u,v)$ may not necessarily be an image since for some (u,v) values $DI_1 - DI_0$ can be negative. In practice this can happen very frequently for artist defined control images (See control images in Figure 2). In other words, if we do not use a formulation that does not guarantee convex hull property, such as ray equation, we significantly restrict the creativity of the artist.

In this formulation, it is also possible to turn diffuse shading into cartoon shading by re-mapping c values. We provide controls to create sharp change from one control image into another, which can also results in cartoon shading (See Figure 7).

In conclusion, to create dynamic 2D artworks, artists have to create at least three images: one 2D vector field, two control images. In most figures we included artist created control images to provide an idea about the process.

3 Gradient Domain Diffuse Shading

In this section, we show how 2D vector fields can be used to compute diffuse shading, ambient occlusion and soft and hard shadows using simple image operations. We provide formulations for single parallel or point lights, but these formulations can be extended to multiple and area lights.

Let $c_d(u,v), c_a(u,v), c_s(u,v) \in [0,1]$ denote shading parameters that are calculated for every pixel using our diffuse shading, ambient occlusion and shadow computations respectively. In other words, these three parameters are B&W images that describe particular effect for all pixels. For instance, for a given (u,v), $c_a(u,v) = 0$ means that that particular point is completely occluded and $c_a(u,v) = 1$ means that (u,v) is not occluded at all. These three parameters can be used in Bezier formulation separately to obtain final diffuse image. On the other hand, to have a consistent visual results, we think that it is better to combine these parameters into one overall shading parameter $c(u,v) \in [0,1]$ if we use only ne key light. To obtain an overall shading parameter $c(u,v) \in [0,1]$ we need an operator that guarantees $c(u,v)$ is always between $max(c_d(u,v), c_a(u,v), c_s(u,v))$ and $min(c_d(u,v), c_a(u,v), c_s(u,v))$. Examples of such operators are multiplication $c(u,v) = c_d c_a c_s$ and mean $c(u,v) = w_d c_d + w_a c_a + w_s c_s$ where $w_d + w_a + w_s = 1$ and $w_d, w_a, w_s \geq 0$. In this paper, we use multiplication operator. The $c(u,v)$ value, then, is used to create the final diffuse rendered image using Equation 2.

3.1 Shading Computation

As discussed in section 2.1 by a linear combination of two lights we obtain the equation $rx_L + gy_L = 0.5(x+1)x_L + 0.5(y+1)y_L = 0.5(x.x_L + y.y_L) + 0.5(x_L + y_L)$. In this equation, the first term $0.5(x.x_L + y.y_L)$ corresponds to the dot product and the second term $0.5(x_L + y_L)$ is a constant – a fraction of light intensity computed in L_1 norm, which is not exactly 0.5 but it is close to 0.5 since the light vector is a unit vector. We, therefore, observed that the second term can be further simplified to provide a better user control as

$$c_d = 0.5(x.x_L + y.y_L) + 0.5 \qquad (4)$$

One advantage of this particular equation; $x = 0$ and $y = 0$ always correspond to $c = 0.5$. Therefore, regardless of the light direction, the value of c_d in the position where $x = 0$ and $y = 0$ does not change. We use a scaling term that can be considered as light intensity. This scaling term (combined with truncation operator, which keeps the shading parameter values between zero and one) also helps to obtain cartoon shading.

For parallel light, the light vector (x_L, y_L) is the unit vector and it is the same for every point. Therefore, it is really straightforward to implement parallel lights. For point lights, the 2D cursor position u_c, v_c will be the position of the point light and the light direction is computed for every point as $(x_L(u,v), y_L(u,v)) = normalize(u - u_c, v - v_c)$, where $normalize()$ is a function that normalizes a given vector. One problem with such point light is that the light distribution around $u = u_c$ and $v = v_c$ is singular. Such a singularity does not look visually appealing. Fortunately it is possible to make this singularity acceptable by increasing the intensity of the light around $u = u_c$ and $v = v_c$ with a linear fall-off term. Note that since our computations are in 2D, the linear, instead of quadric, fall-off makes more sense mathematically.

3.2 Ambient Occlusion

Ambient occlusion can simply be considered a function of mean curvature [Griffin et al. 2011]. Since 2D vector fields do not necessarily correspond to any real shape, a mean curvature cannot be defined. Fortunately, it is still possible to estimate the curvature around any point by just looking at the 2D vector field. For a given δ_u and δ_v, let $\delta SM_{i,j}(u,v)$ denote the vector that is given as the difference between the two 2D vectors of given point (u,v) and its neighborhood point $(u + i\delta_u, v + j\delta_v)$ as

$$\delta SM_{i,j}(u,v) = \begin{pmatrix} x(u + i\delta_u, v + j\delta_v) - x(u,v) \\ y(u + i\delta_u, v + j\delta_v) - y(u,v) \end{pmatrix} \quad (5)$$

We can determine if this vector is directed to or from the given point (u,v) by simply computing the dot product as

$$A_{i,j}(u,v) = \delta SM_{i,j}(u,v) \bullet (i\delta_u, j\delta_v). \quad (6)$$

Positive $A_{i,j}$ means the vector is directed to (u,v) and negative $A_{i,j}$ means that it is directed from (u,v). We observe that if all $A_{i,j}(u,v)$'s are positive, the point is a maximum. If they are all negative, the point is a minimum. If some are positive and some are negative, the point correspond to a saddle point. Therefore it is easy to differentiate between these points by simply computing the following summation which roughly corresponds to mean curvature:

$$c_a(u,v) = 0.5 \sum_{i=-N}^{N} \sum_{j=-N}^{N} \frac{w_{i,j} A_{i,j}(u,v)}{\sum_{i=-N}^{N} \sum_{j=-N}^{N} w_{i,j}} + 0.5 \quad (7)$$

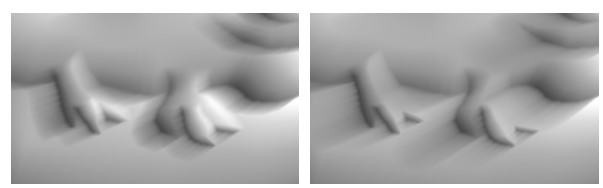
The weight terms, $w_{i,j}$'s, are Gaussian coefficients that are given as $w_{i,j} = e^{-(i^2+j^2)/\gamma^2}$. These coefficients are used to make sure that distant neighbors do not impact the result more than close neighbors and the summation in the denominator is just a normalization term to make sure that $c_a(u,v)$ will not be larger or smaller than the maximum and minimum values of $A_{i,j}$. Despite the normalization term, the result can still be larger than 1, or smaller than 0 since $A_{i,j}$'s can be smaller than -1 and larger than 1. We therefore apply the re-mapping operation given in Equation **??** to guarantee that the results fall in the range between 0 and 1.

$\delta_u \approx 1.5$ $\delta_u \approx 2.5$

Figure 8: *The effect of $\delta_u = \delta_v$ parameter in ambient occlusion computation for $N = 2$. In this case, $DI_0(u,v)$ is a black image and $DI_1(u,v)$ is a white image and background is chosen white. The value of δ_c is set to 1 to display the full range of computed ambient values.*

The quality of ambient occlusion depends on the size of the filter $2N + 1$ and the terms δ_u and δ_v. Figure 8 shows the effect of δ_u in the visual quality of ambient occlusion. As it can be seen in the figure, small δ_u values creates flat images. On the other hand, large δ_u values introduce noise for the same number of samples.

Ambient occlusion is the most expensive rendering operation since for every point we need to compute the effects of the $(2N + 1)^2$ neighboring points. Fortunately, for reasonable quality results ambient occlusion can be computed in real-time. For very high quality results $c_a(u,v)$, can be computed only once and re-used during interactive sessions since it does not depend on light position.



$\delta_s \approx 0.5$ $\delta_s \approx 1.5$

Figure 9: *The effect of δ_s parameter in shadow computation for $N = 10$ on the feet of the creature in Figure 1. In this case, $DI_0(u,v)$ is a black image and $DI_1(u,v)$ is a white image and the background is chosen white. The value of $\delta_c = 1$ to see the full range of ambient values.*

3.3 Shadow Computation

One of our most unexpected results is that realistic looking soft shadows can be computed by line integral convolution over a 2D vector field. We again assume that the light is a unit vector given as (x_L, y_L). We compute a set of line integral convolutions in different lengths along the ray that starts at (u,v) in the direction of $(-x_L, -y_L)$ to identify shadows for a given point (u,v).. Among these line integral convolution values, the highest value provides the tallest obstacle along the direction. The higher the number, the more

likely this point would be in shadow. The line integral convolution for a given length $n\delta$ is given as:

$$S_n(u, v, \delta_s) = \sum_{i=0}^{n} a(u, v, i\delta_s) \tag{8}$$

where $a(u, v, n\delta_s) = (x(u - i\delta_s x_L, v - i\delta_s y_L).x_L + y(-i\delta_s x_L, v - i\delta y_L).y_L)$. Note that $S_n(u, v, \delta_s)$ can be computed recursively as

$$S_n(u, v, \delta_s) = S_{n-1}(u, v, \delta_s) + a(u, v, n\delta_s) \tag{9}$$

We want to clarify that the term $(x.x_L + y.y_L)$ does not really come from our diffuse shading computation. Let the landscape be a piecewise linear surface that consists of planar regions, in every step we traverse only one planar region that is represented by a unit normal vector whose (x, y) coordinates are given by 2D vector field. Under this assumption, if we compute the height difference along the ray $(u - t\delta_s x_L, v - t\delta_s y_L)$, for a unit step, we compute height difference from the beginning of the step to the end of the step as

$$h = \frac{(x.x_L + y.y_L)}{\sqrt{1 - x^2 + y^2}}. \tag{10}$$

In the equation of h, the denominator term $\sqrt{1 - x^2 + y^2}$ creates singularities around shape boundaries, i.e. silhouette edges. These singularities disturb computation since they can dominate line integral convolutions. Since the nominator term $x.x_L + y.y_L$ is already large in boundary regions, we remove the denominator term and end up with a completely 2D algorithm. Based on $S_n(u, v, \delta_s)$'s, final c_s is computed as

$$c_s = 0.5 \;\; max\{S_1, S_2, \ldots, S_{N-1}, S_N\} - 0.5 \tag{11}$$

We can again obtain soft and hard shadows from this formula by using the re-mapping formula given in Equation **??**. If the light is a parallel light, vectors (x_L, y_L) are the same for all points in 2D space, we can also use the same N and δ_s for every point. On the other hand, if the light is point light, some minor adjustments are necessary. For instance, for points that are closer to 2D light, we do not expect to see long shadows. Therefore, for larger distances to the point light, we use higher values for N and δ_s.

Figure 9 shows the effect of δ_s over the visual quality of shadows. As it can be seen in the image, small δ_s values creates short shadows. Increasing δ_s makes shadows longer. However, once the algorithm detects the highest point in the given direction, shadows stop growing – as expected.

4 Specular Reflection

To combine specular and diffuse effects, we use transparency coming from the two images DI_0 and DI_1. Let $EI(u, v)$ denote "environment image' and let $\alpha_I(u, v)$ denote opacity of combined diffuse image I. Note that normal maps do not have a transparency, they only provide information about how to render the final image. On the other hand both DI_0 and DI_1 can have transparent regions. The term α_I of I results in combined transparencies of the two images DI_0 and DI_1 during rendering. This term is used to make only certain parts of the image reflective (see Figures 10 and 11). We also have a user-defined global parameter, α_G, such as those in GIMP or Photoshop, that can make the entire layer more reflective. Then $\alpha_C(u, v) = \alpha_G \alpha_I(u, v)$ denotes combined transparency of the layer $I(u, v)$. Using combined transparency, α_C, we compute the composited image as follow:

$$CI(u, v) = \alpha_C I + (1 - \alpha_C) EI(R).$$

where $R(x, y) = (u, v)$ represents reflection mapping.

For reflection mapping our goal is to provide a simple and intuitive-to-use method for 2D artists. Our method is closely related to sphere mapping, which is one of the most widely used reflection methods and it is used in normal map representations such as Lumo to obtain rendering effects [Johnston 2002; Bezerra et al. 2005]. In sphere mapping a far away spherical environment is stored as an image that depicts what a gazing ball (i.e. mirrored sphere) would reflect if it were placed into the environment, using an orthographic projection. Although sphere mapping is one of the simplest methods for obtaining reflections, it can still be complicated for some 2D artists who want to work only with rectangular images. Moreover, in our case, an additional problem comes from the fact that $x^2 + y^2$ can be larger than 1. Therefore, we need a similar method that can use rectangular environments. Fortunately, there exists a gazing ball shape that can reflect the whole environment into a square image. This particular gazing ball shape can be given by an implicit surface $max(x^2, y^2) + z^2 = 1$ [Akleman and Chen 1999]. We do not provide the details here, but it can be shown that this shape, when placed in an environment, reflects the whole environment in a square using orthographic projection along the z direction. We can, then, create reflection simply by using the following operation:

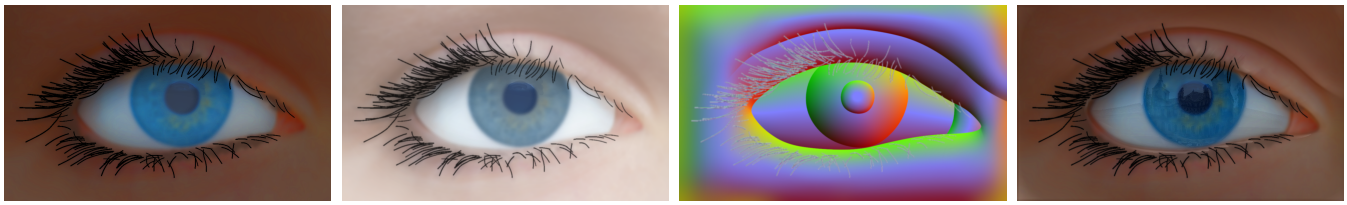
$$(u, v) = R(x, y) = (0.5x + 0.5, 0.5y + 0.5) \tag{12}$$

The only caveat in this approach is that every point on the boundary of a square image reflects the same point in the environment sphere. Therefore, every point on the boundary of the square image has to be the same. In practice, any seamlessly tile-able wallpaper image can be used as an environment map. In our experience, any image works as an environment map, probably due to humans' high tolerance for discontinuities in mirror images.

For realistic reflections, we move the center of the environment image in tandem with the light position. This creates visually acceptable specular reflections. Therefore, we do not think an additional specular highlight is necessary. Glossy reflection is simply obtained using smoothed versions of environment maps provided by mipmap. Glossy reflections combined with other compositing operations such as multiplication can be used to obtain other effects such as environment illumination. These artist painted rectangular environment maps allows to create mirror reflections in re-interpretation of existing painting as shown in Figure 11

5 Discussion and Conclusion

2D vector fields are particularly useful for 2D artists whose primary focus is painting and illustration. These artists have a good understanding of how shading works, but they may not want to follow the conventional projection for obtaining 2D works. 2D vector fields allows them to create paintings that can be illuminated exactly as they want. In essence, a 2D vector field provide the rules to combine three control images, namely DI_0 and DI_1 with a rectangular environment image that is used for reflection. The combination operator



$DI_0(u,v)$ $DI_1(u,v)$ 2D vector field Rendering

Figure 10: *Reflections on the eyeball is obtained with α_I term by using slight transparency on eyeball region. Since the overall α is 1, the rest of image is diffuse. The 2D vector field, and a line drawing are obtained by a sketch based interface. Control images are, then, painted by an artist using the line drawing as a guide.*

$DI_0(u,v)$ $DI_1(u,v)$ 2D vector field Rendering 1 Rendering 2

Figure 11: *Re-interpretation of Warhol's Campbell Soup painting with art directed reflections. In his original painting, Warhol painted mirror reflected black areas on top of the can and a very subtle and almost invisible diffuse reflection on body of the can. Using a black and white striped image as an environment map and by painting slightly varying control images, an artist was able to move both subtle diffuse reflection and mirror reflected black in tandem, which can help to better appreciate the idea behind this painting. Creation of control images and 2D vector field did not take more than one hour.*

is, in fact, a bilinear equation, in which environment image appears twice. It is, therefore, reflection dominates the equation when α is close to 1. Artists can further exploit this process to obtain abstract images.

As mentioned earlier to create our 2D vector fields we used a simple sketch based modeling tool, but sketching is not the only option to model 2D vector fields. 2D polygons or splines can be used to model 2D vector fields. Basically, the gradient vectors to the curves provide constraints to compute entire 2D vector field. The entire field can be computed with Poisson equation resulting a method similar to the diffusion curves [Orzan et al. 2008] or the values can be interpolated resulting a method similar to mean value coordinates [Floater 2003; Farbman et al. 2009]. These shapes can also be animated in 2D. With 2D animation, it is easy to provide topology changes that can provide unrealistic motion exactly as artists may look for. For instance, one shape can divide into two, or a hole in the middle of a shape can appear. With the development of such modeling and animation tools, we expect this approach has a potential to be popular also among animators who wants to create small-budget expressive animations with complete visual control.

For modeling and animating with 2D vector fields it is also important to have layered 2D vector fields that can be manipulated individually. Having layers also provide "a qualitative depth information" which can improve shadow computation. With this qualitative depth information, it can be possible to identify the parts that are behind and not supposed to cast shadow.

References

Akleman, E., and Chen, J. 1999. Generalized distance functions. In *Shape Modeling and Applications, 1999. Proceedings. Shape Modeling International '99. International Conference on*, 72 –79.

Bezerra, H., Feijo, B., and Velho, L. 2005. An image-based shading pipeline for 2d animation. In *Computer Graphics and Image Processing, 2005. SIBGRAPI 2005. 18th Brazilian Symposium on*, 307–314.

Cabral, B., and Leedom, L. C. 1993. Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, 263–270.

Cohen, J., Olano, M., and Manocha, D. 1998. Appearance-preserving simplification. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, 115–122.

Cole, F., Sanik, K., DeCarlo, D., Finkelstein, A., Funkhouser, T., Rusinkiewicz, S., and Singh, M.

2009. How well do line drawings depict shape? In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, 28:1–28:9.

Edwards, B. 1979. *Drawing on the Right Side of the Brain.* Tarcher/Putnam.

Elber, G. 2011. Smi 2011: Full paper: Modeling (seemingly) impossible models. *Comput. Graph. 35*, 3 (June), 632–638.

Farbman, Z., Hoffer, G., Lipman, Y., Cohen-Or, D., and Lischinski, D. 2009. Coordinates for instant image cloning. *ACM Trans. Graph. 28*, 3, 67:1–67:9.

Fattal, R., Lischinski, D., and Werman, M. 2002. Gradient domain high dynamic range compression. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, 249–256.

Floater, M. 2003. Mean value coordinates. *Computer Aided Geometric Design 20*, 1, 19–27.

http://www.opengl.org/documentation/glsl/.

Gooch, A., Gooch, B., Shirley, P., and Cohen, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, 447–452.

Griffin, W., Wang, Y., Berrios, D., and Olano, M. 2011. Gpu curvature estimation on deformable meshes. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, 159–166.

Igarashi, T., Matsuoka, S., and Tanaka, H. 1999. A sketching interface for 3d freeform design. In *Proceedings of ACM SIGGRAPH'99*, 409–416.

Johnston, S. F. 2002. Lumo: illumination for cel animation. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, 45–52.

Levin, A., Zomet, A., Peleg, S., and Weiss, Y. 2006. Seamless image stitching in the gradient domain. In *Proceedings of ECCV, Springer-Verlag, vol. IV*, 377–389.

Li, Y., Sun, J., Tang, C.-K., and Shum, H.-Y. 2004. Lazy snapping. *ACM Trans. Graph. 23*, 3, 303–308.

Nasri, A., Karam, W. B., and Samavati, F. 2009. Sketch-based subdivision models. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, 53–60.

Okabe, M., Zeng, G., Matsushita, Y., Igarashi, T., Quan, L., and Shum, H.-Y. Single-view relighting with normal map painting.

Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., and Salesin, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. 27*, 3, 92:1–92:8.

Perez, P., Gangnet, M., and Blake, A. 2003. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, 313–318.

Petrović, L., Fujito, B., Williams, L., and Finkelstein, A. 2000. Shadows for cel animation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, 511–516.

Rademacher, P. 1999. View-dependent geometry. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, 439–446.

Shao, C., Bousseau, A., Sheffer, A., and Singh, K. 2012. Crossshade: shading concept sketches using cross-section curves. *ACM Trans. Graph. 31*, 4, 45:1–45:11.

Sun, J., Jia, J., Tang, C.-K., and Shum, H.-Y. 2004. Poisson matting. *ACM Trans. Graph. 23*, 3, 315–321.

Weyrich, T., Deng, J., Barnes, C., Rusinkiewicz, S., and Finkelstein, A. 2007. Digital bas-relief from 3d scenes. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07.

Wu, T.-P., Tang, C.-K., Brown, M. S., and Shum, H.-Y. 2007. Shapepalettes: interactive normal transfer via sketching. *ACM Trans. Graph. 26*, 3, 44.1–44.6.

Zongker, D. E., Werner, D. M., Curless, B., and Salesin, D. H. 1999. Environment matting and compositing. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, 205–214.