Another Look at Graftels

Cindy M. Grimm

Washington Univ. in St. Louis[*]

Abstract

We present a hybrid object-image algorithm for the placement of graftels in a scene. Our algorithm addresses the problem of frame-to-frame coherency while maintaining image-space constraints. Graftels are attached to objects in the scene. Each graftel has a behavior function that determines its size and orientation based on the current view direction and image-space size. The placement of the graftels and the behavior functions are optimized to meet 2D image design constraints for a given set of views and image sizes. These constraints may be generated automatically using information such as the surface normal and object color, or constructed interactively.

CR Categories: I.3.m [Computing Methodologies]: Computer Graphics—Misc.

Keywords: graftel

1 Introduction

Texturing with graftels [Kowalski et al. 1999] is a technique for adding visual complexity to a scene. Graftel textures differ from traditional techniques, such as texture mapping and per-pixel shaders, in that the placement, scale, and orientation of individual graftels is determined by 2D design constraints. For example, the orientation of the graftel might depend on the surface normal projected into the image plane or on the local curvature of an object. The number of graftels in an area might depend on the averaged local shade values near that point in the image.

Graftels are very closely related to stroke-based systems [Kalnins et al. 2002; Meier 1996; Hertzmann and Perlin 2000; Daniels 1999]. One characteristic that distinguishes graftels from strokes is that graftels are discrete entities drawn from a small set of simple shapes, while strokes can differ greatly and are often composited together in the final image. Both techniques, however, present similar challenges when animated. In this paper we will use the term graftel to mean a 2D drawing element, which may be textured or alpha blended into the image.

The appeal of graftels is their ability to sparsely indicate texture without overwhelming the visual system. A few, well-placed texture elements can indicate the general direction and shape of the local texture, leaving the viewer to “fill in” the remaining texture. Principles of 2D texturing are fairly well-understood in the art community, although translating these informal rules and guidelines to specific algorithms can be challenging [Winkenbach and Salesin 1994]. How to apply these rules to animating 3D scenes is less well-understood, since there is a fundamental tension between the 2D design rules and frame-to-frame coherency. A graftel rendering system must provide solutions for two main problems.

First, as the scene changes, the graftels must move, scale, and re-orient themselves to the new image. Too many graftels appearing (or disappearing) between frames produces “popping” artifacts. Similarly, graftels that radically change their orientation or scale can produce visually disturbing artifacts. The system must provide a mechanism for controlling the introduction and movement of the graftels.

The second problem is that the specification of graftel behavior in 3D animation [Pastor and Strothotte 2002; Markosian et al. 2000; Kaplan et al. 2000] can be quite complex. There are a large number of variables in the 3D scene that indirectly influence how graftels are placed in an image: camera angle, image size, rendered shade values, and projected surface normals, to name a few. The system must provide a method for specifying 2D design rules which take into account these 3D data.

We present a system for the placement of graftels that addresses the problem of frame-to-frame coherence while maintaining desired 2D design constraints. The graftels are attached to objects in the scene in order to provide coherence. Each graftel has a *behavior function* that determines the size and angle of the graftel based on the current camera and image size. The placement of the graftels on the object, and the parameters of the behavior functions, are optimized to meet desired 2D design constraints. We explore two methods for specifying graftel behavior functions and 2D design constraints. In the first approach, the user specifies constraints for a (typically) small set of view points. These constraints may depend on values calculated from 3D data. The system then finds a set of graftels and corresponding behavior functions that meet those constraints. In the second approach, the user interactively positions, scales, and orients the graftels from arbitrary view points.

There are several advantages to our approach over an entirely image-based or object-based one. First, the behavior functions plus the object-based placement of the graftels enforce frame-to-frame coherency. Second, there is no need for costly reads from the image buffer during rendering to enforce 2D design constraints. In fact, depending on the form of the behavior function, it can be implemented directly as a hardware pixel shade operation. Third, unlike a purely object-based approach, the behavior functions are optimized to meet 2D image constraints, such as density and orientation. Fourth, we present a very free-form behavior function (and corresponding user-interface) which makes it possible to create very complex behavior functions.

The paper is organized as follows. We begin with a review of related work (Section 2). Section 3 outlines the approach we take. This consists of defining graftels (Section 4), the 2D design constraints (Section 5), and behavior functions (Section 7. The heart of our approach is an optimization routine, which consists of an algorithm for initial placement of the graftels (Section 6.1) and a behavior optimization algorithm (Section 6.2). The interactive system is discussed in Section 8. We conclude with the discussion of several examples (Section 9).

[*]email:cmg@wustl.edu

2 Related work

There are two basic approaches to graftel or stroke placement and movement. The first is to attach the graftels to a 3D object [Meier 1996; Kaplan et al. 2000; Pastor and Strothotte 2002], the second is to place the graftels in image space [Markosian et al. 2000; Kalnins et al. 2003] and, as much as is possible, re-use the previous frame's strokes in the next frame.

There are also two approaches to determining where and how to draw the graftels. The first approach projects the 3D scene data to 2D, then uses just the 2D information [Kowalski et al. 1999; Winkenbach and Salesin 1994]. The second approach begins with points already attached to the 3D object, and determines, based on how the 3D graftel projects to 2D, how to draw it. Meier [Meier 1996] uses projected screen size and orientation to determine the direction of the strokes. Kaplan [Kaplan et al. 2000] extend this to a function based on the viewing direction, lighting, surface normal, and camera view point distance. Pastor and Strothotte [Pastor and Strothotte 2002] use a fairly complicated method based on subdividing the surface to place stipples on the object. Their method has the advantage that the user can zoom in arbitrarily close, and the surface will continue subdividing as needed.

There are also hybrid 2D-3D systems that have incorporated 3D viewing parameters into essentially 2D drawing systems. Cohen [Cohen et al. 2000] allows the user to draw 3D billboards which then rotate to continuously face the viewer. Disney [Daniels 1999] has developed a system that lets the artist paint a 2D scene onto a 3D model. Each individual brush stroke is attached to the corresponding 3D geometry. As the view point is changed, the strokes are re-painted in their new locations, with an adjustment made for changing screen-size and orientation. Fiore [Fiore et al. 2003] uses a combination of 3D geometry and user-defined 2D strokes to render trees in a cartoon style. WYSIWYG NPR [Kalnins et al. 2002] combines many traditional non-photorealistic rendering techniques with the ability to paint strokes which attach themselves to objects. Kowalski [Kowalski et al. 2001] uses a set of composition rules to define how objects are rendered and combined based on their expected screen-space projection.

Tone or art maps [Praun et al. 2001; Webb et al. 2002; Klein et al. 2000] are used for applying a certain class of stroke textures to surfaces. They automatically cope with changing tone values and changing resolutions in a smooth, continuous way by blending in (or fading out) additional strokes where needed. This is implemented as a series of mip-maps, with strokes in the lowest level of the mip-map appearing in all higher levels. This approach works very well for textures that are self-similar and can be procedurally generated, such as hatching or stippling. Since the textures are applied as texture maps to the object, the strokes' directions can only depend on the orientation of the object, not on image-space constraints. Also, it is not clear how to extend tone maps to more arbitrary textures.

3 Automated approach overview

The following sections describe the first of two basic approaches for placing graftels and defining behavior functions. The second approach is interactive and is discussed in Section 8. In the first approach, the user first defines 2D design constraints, such as the desired density, size and angle (Section 5). These constraints may be calculated from 3D data such as the surface normal, shade values, *etc.* They next choose a set of view angles and image sizes over which to optimize. These views may be evenly distributed in view space, or follow a pre-determined camera path. The user also specifies the form of the behavior function and its free parameters (Section 7). Note that we must have a behavior function for each graftel, but that more than one graftel can share a behavior function.

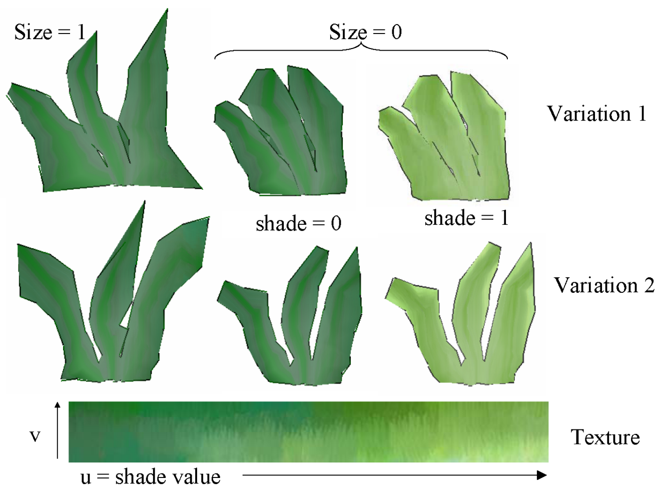

Figure 1: The grass graftel. Shown are two possible mesh variations, each with a "big" and a "small" version. The small versions are shown shaded "dark" and "light", using the texture map at the bottom.

Once the user has specified the necessary information, the system searches for a set of graftels and behavior functions that meet the desired 2D design constraints. This optimization routine alternates between placing graftels and optimizing the behavior function parameters. Graftel locations are chosen based on the desired local density and graftel size in one of the chosen viewpoints. Additionally, we try to place the new graftel next to an existing graftel at the desired spacing. Note that as graftels are placed in one image, they will also appear in any other view that contains that part of the scene.

When a graftel is first created, its behavior function is set to a default value. The second part of the optimization procedure adjusts a graftel's behavior function to better meet the 2D design constraints over the entire set of specified views.

4 Graftel definition

Each graftel has a position p, surface normal $\vec{n}$, screen space angle α, rotation matrix r, 3D size s_3, screen size s_2, texture-mapped mesh, minimum pixel size m, and maximum pixel size M. The position, screen space angle, and screen size are used by the behavior function to calculate the graftel's drawing position. The graftel's position and surface normal are found by intersecting the object with a ray through the pixel.

The mesh for the graftel is centered at $(0, 0.5)$ and is 1 unit in the y direction. The base of the graftel is at $(0, 0)$, allowing the graftel to grow "up" and to rotate around its base point.

The texture values for the graftel are calculated, in part, from the shaded value of the graftel [Kulla et al. 2003]. The v texture coordinate for each vertex is fixed, but the u texture coordinate is set to the shade value for the graftel, plus 0.05 of the vertex's x position. The shade value is calculated from the lights, the position, and the surface normal using the standard lighting equation. See Figure 1.

For each graftel type we allow the user to edit one or more shapes for the graftel, with each shape having a big and a small version. The 2D size parameter calculated by the behavior function (Eq. 9) is used to interpolate between the two graftel sizes.

To render the graftel from an arbitrary camera we "undo" the rotation in the camera (screen-space alignment). Screen-space size and orientation are accomplished by scaling and rotating the graftel in its coordinate system before applying the camera transformation.

Let $P_c = PS_zS_{xy}RT$ be a standard camera defined by an eye point E_c, a `look` vector $\vec{l}$, and an `up` vector $\vec{u}$. Then the matrix V_g applied to the graftel is:

$$s_p = \frac{s_2}{||P_c p - P_c(p+\vec{u})||} \quad (1)$$

$$\vec{v} = R(p - E_c) \quad (2)$$

$$V_g = PS_zS_{xy}T(\vec{v})R(z,\alpha)S(s_p,s_p,1) \quad (3)$$

where $T(\vec{v})$ is translation by $\vec{v}$, $R(z,\alpha)$ is a rotation around the z axis, and $S(s_p)$ is a scale matrix that compensates for perspective scaling.

The rotation matrix r is used by the graftel's behavior function to align the local graftel's coordinate system with that of the behavior function. This matrix is applied to the camera's `look` vector before passing it to the behavior function.
This equation determines the projected size of a sphere placed at the graftel's location and scaled by the 3D size of the graftel. We normalize by the maximum size of the graftel.

5 Image optimization

The 2D design constraints are stored as data at each pixel. How this data is constructed depends on the desired constraint. Here we describe the format of the required image-space data in general; specific equations for constructing the data will be provided in Section 9. Example images are shown in Figure 2. For each pixel we store the following:

- **Depth channel:** This is the standard depth buffer created by rendering the object.
- **Spacing channel:** The value I_{sp} specifies the desired spacing between graftels. This is used during the placement algorithm.
- **Overlap channel:** The value I_o specifies the maximum amount of overlap allowed at the pixel. This is used during the optimization algorithm to determine the best size of the graftel.
- **Angle channel:** The value I_α specifies the desired screen space angle.
- **Size channel:** The value I_s specifies the desired maximum size of a graftel at that pixel.
- **Coverage channel:** This channel indicates how much overlap there is of graftels at that location. Let o_g be the overlap value assigned to one graftel, and I_c be the pixel value. Then I_c/o_g is the number of graftels overlapping that pixel.
- **Placement channel:** This channel indicates where graftels should be placed in order to maintain the desired spacing. The higher the value I_p, the better the pixel choice. I_p/o_g is approximately how many graftels would be adjacent to a graftel placed at that pixel.

5.1 Coverage and placement channels

The coverage and placement channels are created by rendering the graftels at different sizes (see Figure 2) and summing each graftel contribution.

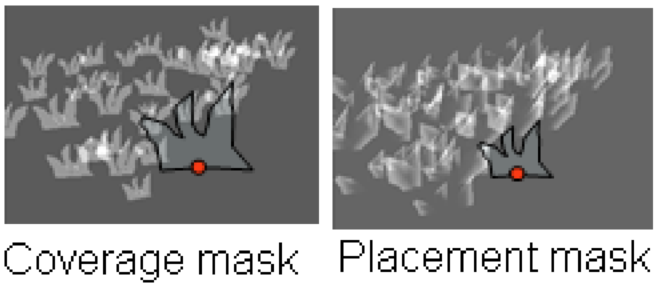

Figure 3: Left: Placing the coverage mask on the coverage channel to determine if a graftel at the indicated location would overlap an existing graftel. Right: A similar calculation to determine how good the placement would be.

To create the coverage image we first render the object, shifted slightly in the `look` direction. We then clear the color buffer, and turn off writing to the depth buffer. This insures that occlusion of graftels by the object happens correctly. Next, we render each graftel at intensity o_g, summing the color contributions of each graftel. The graftel has a size g_s, calculated by the behavior function (Section 7). The graftel also has a desired spacing provided by the spacing channel. We render the graftel scaled by s_r so that it is the desired size plus half of the spacing on either side:

$$s_r = \frac{I_{sp}/2 + g_s}{g_s} \quad (4)$$

We currently set o_g to 0.2, which lets us distinguish between 0 to 5 graftels overlapping at one point, while still being able to visualize the coverage channel.

The placement channel is created in a similar manner, except that the intensity of the graftel is o_g at the boundary, and fades to $0.1o_g$ in the interior. The graftel is translated down by $1/2$ and scaled by $2s_r$. This places a halo around the graftel at the desired spacing. Pixels with the highest placement values are those that are the desired distance away from one or more graftels.

6 Image optimization

The goal of the optimization algorithm is to determine where to place the graftels, how to orient them, and how to set their behavior functions. Both optimization algorithms (placement and behavior function) operate on the image-space data defined in the previous section. We alternate between one pass of the placement algorithm, and one pass of the behavior optimization algorithm. The optimization terminates when there are no more places to add graftels.

6.1 Placement algorithm

The goal of the placement algorithm is to add a new graftel near an existing graftel, separated by the desired spacing. If no suitable graftel is found, we simply pick an image point at random that meets certain criteria.

We first create a mask of the graftel. We place this mask (scaled and rotated appropriately) at the pixel under consideration and add up the contributions of the channels (see Figure 3). If the mask does not overlap an existing graftel, but is adjacent to one, then we add that pixel to the list of pixels under consideration. We visit every pixel in every image, keeping a list of all candidate pixels. For each candidate we store how many graftels the pixel is adjacent to. We sort this list, and begin adding candidate graftels. After a

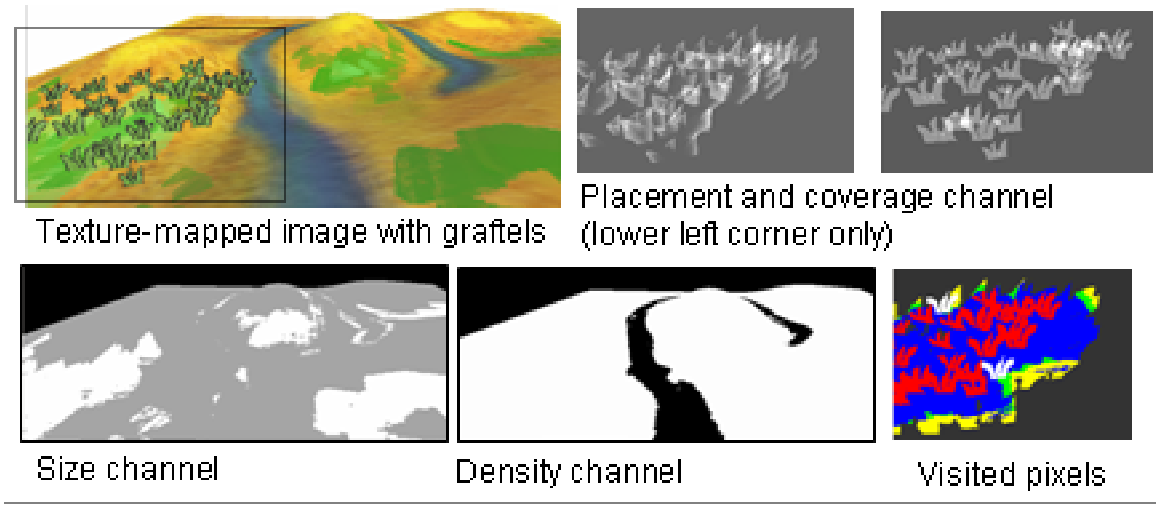
Texture-mapped image with graftels
Placement and coverage channel (lower left corner only)
Size channel
Density channel
Visited pixels

Figure 2: Image-space constraint data for the grass and ground. The lower right image shows the pixels visited in the placement algorithm. Red pixels were automatically discarded because they lie inside existing graftels (including graftels added from other viewpoints during one pass of the placement algorithm). Blue pixels were discarded because the coverage mask would overlap an existing graftel. Grey pixels were too far away from an existing graftel to be considered. Green and yellow pixels are the candidate pixels. White pixels show the two graftels that were added from this viewpoint. (The placement, coverage, and visited images only show a blow-up of the lower-left corner.)

graftel is added, we find its projection in all images and update the corresponding coverage channels appropriately. (The depth buffer is used to determine if the graftel is occluded). Candidate pixels still in the queue are checked against the updated coverage channel before being added.

We use two masks, each at a different scale. The coverage mask is scaled so that it is the expected size of the graftel, plus the expected spacing. The coverage mask scale value is:

$$\frac{I_{sp}(x,y)/2 + I_s(x,y)}{I_s(x,y)} \tag{5}$$

The value mask is used to calculate the size, density, overlap, angle, and placement values; it is scaled to $I_s(x,y)$. For example, to calculate how "good" a pixel is, we place the mask at that pixel, and sum the values of the placement channel where the mask is non-zero, and divide by the number of non-zero mask pixels.

Candidate pixels are those that meet the following criteria:

- The coverage mask does not overlap any non-zero values in the coverage channel.
- The value mask does not overlap any zero values in the density channel (zero indicates that there should be no pixels here).
- The value mask overlaps at least one non-zero value in the placement channel.

In practice, we only need to look at pixels that are near places where the placement channel is non-zero and the coverage channel is zero. This is typically an expanding ring in each image. If we find no pixels that meet all three of the above criteria, then we pick a random pixel that meets all but the last criteria.

6.2 Behavior function optimization

We optimize each behavior function independent of the others; theoretically, we could achieve better results by optimizing all of the functions simultaneously. Unfortunately, this would be prohibitively expensive. Instead, we randomly pick a small number of graftels (typically 10-20), optimize them using the same coverage channels, then repeat.

We first find all of the views in which a graftel is visible. For each of these views we find the ideal size and angle for the graftel from the coverage and angle images. These `<view, size, angle>` triplets are then sent to the behavior function; the actual optimization depends on the type of behavior function (Section 7).

The ideal angle value is fairly straightforward to calculate: we simply place the value mask from the previous section on top of the graftel's location and sum the overlapped angle values.

The ideal size is the biggest possible graftel size such that the coverage value is less than the allowed overlap $I_o(x,y)$. We do a brute-force search over the sizes, from zero up, stopping when the coverage mask overlap exceeds the allowed threshold or we reach the biggest allowed size.

Figure 4 shows the inter-relation between the desired spacing and the allowed overlap. The spacing channel controls where graftels are added, but the overlap channel controls their sizes across the images.

7 Behavior functions

The purpose of the behavior function is to map information about the current camera and the graftel's position and orientation to a size and orientation on the screen. Rather than use all of the camera's parameters (which would be prohibitive) we extract two pieces of information from the camera. The first is the direction of the `look` vector of the camera relative to the local coordinate system of the graftel. The second is the projected size of the graftel, or how far the graftel is from the camera.

Each graftel has a rotation matrix r that defines its orientation; this rotation matrix can either be based on the local surface frame or on the viewing coordinate system in which the graftel was created. We multiply the `look` vector of the camera by this rotation matrix to give the actual view vector that is passed to the behavior function.

Each graftel has a 3D size, which is the size of a sphere placed around the graftel's position. This is calculated at creation time by finding the sphere size that, when projected, has radius g_s on the screen, where g_s is the size of the graftel. More specifically,

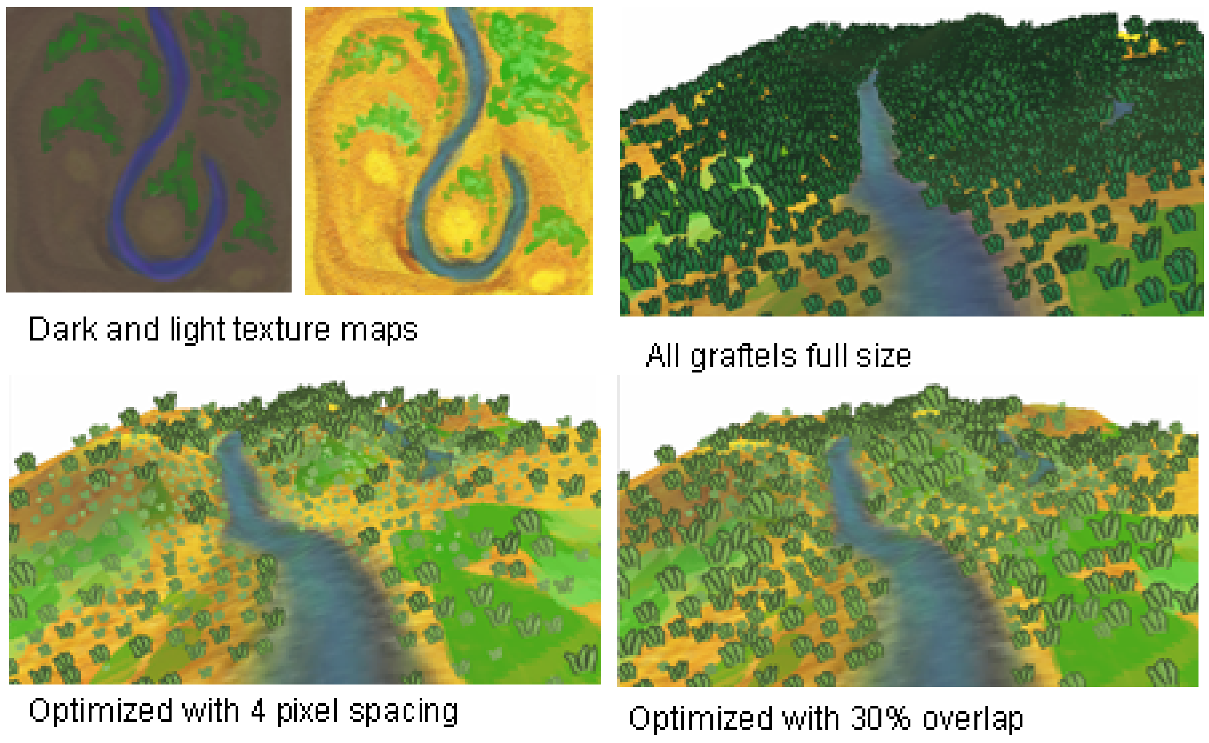
Dark and light texture maps
All graftels full size
Optimized with 4 pixel spacing
Optimized with 30% overlap

Figure 4: Left: The grass with a desired spacing of four, and an allowed overlap of zero. Right: The same grass with a desired spacing of four, but an allowed overlap of approximately 30

$$
\begin{aligned}
ratio &= \frac{g_s}{(W+H)/2} && (6)\\
lenUp &= 2||E_c - p||\tan(\theta/2) && (7)\\
s_3 &= ratio * lenUp && (8)
\end{aligned}
$$

where E_c is the eye point of the camera, θ the zoom angle, W, H the size of the window, p the graftel's point.

The screen space of the graftel is then found by mapping p and $p + \vec{u}ps_3$ to the image and taking the difference (in pixels) of their length. If $V(p)$ is the function that maps a point to the screen:

$$s_2 = \frac{||V(p) - V(p+\vec{u}p)||}{g_s} \qquad (9)$$

7.1 Look and Size behavior function

This function depends on the screen-size s_2 of the graftel and the relative orientation of the viewing direction $\vec{l}$ with respect to the surface normal $\vec{n}$. We use a standard 2D $\rightarrow$ 3D spline patch to represent the function:

$$S(s_2, \vec{l}\cdot\vec{n}) \quad \rightarrow \quad (s_x, s_y, \alpha) \qquad (10)$$

We use this behavior function for the grass and the Looks-A-Lot character (Figure 6). The default values are constant $(1,1,0)$. To fit the patch we use a least-squares approach [Fowler and Bartels 1991]. The domain of the patch is set to be slightly bigger than the maximum and minimum s_2 and $\vec{l}\cdot\vec{n}$ values for the user-defined view set. For the examples in this paper we used a degree 1 patch with 6 control points on a side. We also include a smoothing parameter in the least-squares fit which controls how much the derivative is allowed to change.

7.2 Arbitrary behavior

This behavior function is very general, but more space-intensive than the previous one. We define a 3D spline function on the sphere [Grimm 2002], with the first two parameters corresponding to the view direction in spherical coordinates (θ, ϕ), and the third parameter to the screen-space size. The total number of degrees of freedom is $4\times4\times4\times6$ for a C^1 spline with four control points in each dimension.

We use this behavior function when doing user interaction (the dinosaur, see Figure 5). The user chooses a view and a screen size by positioning the camera, then re-scales the graftel. The appropriate degrees of freedom in the spline function are then changed to meet the new constraint; this is identical to least-squares direct manipulation of spline curves [Fowler and Bartels 1991; Grimm 2002]. Essentially, we find the minimal movement of the spline control points such that the new output of the spline function matches the user change at that point.

Note that we can make this a pure view-based function by using a 2D spline function (dropping the third, size-based parameter).

8 User interface

The user interface is very straightforward. The user can "comb" the graftels by drawing directions in the image. Graftels that lie underneath the cursor are combed in the indicated direction.

Similarly, the user can paint the graftels with a size brush, making them either shrink or grow by some amount. If the user scales the graftel past the maximum pixel size for that graftel then we update the maximum size accordingly.

Once the user has finished painting, we fit the behavior function. We currently use the 3D arbitrary function for screen size, and the 2D arbitrary function for the angle.

To add new graftels the user just clicks on the object and draws a line in the desired direction. This sets the initial screen-space size, screen orientation, and 3D orientation. The graftel's rotation matrix r is set to the matrix that takes the camera frame to the coordinate axes.

Multiple graftels can use the same behavior function. In this case, every graftel that was edited contributes to the least-squares fitting (i.e., one constraint row for each graftel).

9 Specific models

9.1 Grass

The size and density images were constructed from the texture mapped object. Areas that were primarily green had a graftel size of 32 assigned to them. The remainder of the image had a graftel size of 24. Areas that were blue were marked as no graftels, while the remainder of the pixels were set to have a spacing of 4 pixels.

The screen-space angle for every pixel was set to zero, and we used a constant angle behavior function. We used the Look and Size behavior function for the graftel size. We used a set of 25 optimization images evenly spaced along the camera path through the scene.

Figure 4 shows some images from the grass scene. We performed two optimizations, one with an allowed overlap of zero, and one with an allowed overlap of approximately 30%, which produces much denser grass.

9.2 Dinosaur

The dinosaur (Figure 5) was constructed by hand, using the arbitrary behavior function, in approximately 10 minutes. There are four different grafel behavior functions; the neck, body, feet, and eyes. Note that, by repositioning the camera, we can use the same behavior function for both sides of the dinosaur, and re-orient the graftels along the tail.

9.3 Looks-A-Lot

The 2D constraints for the Looks-A-Lot (Figure 6) character were specified by painting the canonical views. The angle was set to be the normal projected into the image plane. We used a total of 14 images, seven small and seven big. The Look and Size behavior function was used for the graftel size. The size image was scaled from 0 to 10 pixels, the overlap image from 0 to 30%, and the spacing image from 0 to 3 pixels.

10 Conclusion

In conclusion, we have described a system that uses 2D image-space constraints to define general behavior functions for graftels. The system maintains frame-to-frame coherency while meeting 2D design constraints. The user can adjust the trade-off between coherencey and the 2D constraints by changing the smoothness of the behavior function (i.e., number of control points and continuity of the spline function). The user can also specify design functions by sketching the desired results, rather than specifying equations.

There are some limitations in the current system. For example, we do not take into account the up vector of the camera. For most applications (walk throughs, characters) this is acceptible because there is a default orientation and the viewer is not expected to look at the scene upside down. If orientation were important, the system could be extended by using a quaternion (four variables) to represent the view direction.

The behavior functions are valid outside of their set range, but they may not give the desired behavior. Therefore, the canonical view set should cover the desired camera angles for the best results.

References

COHEN, J. M., HUGHES, J. F., AND ZELEZNIK, R. C. 2000. Harold: A world made of drawings. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, 83–90.

DANIELS, E. 1999. Depp canvas in disney's tarzan. In *Siggraph sketches*, 200.

FIORE, F. D., HAEVRE, W. V., AND REETH, F. V. 2003. Rendering artistic and believable trees for cartoon animation. In *Computer Graphics International*, 144–151.

FOWLER, B., AND BARTELS, R. H. 1991. Constraint based curve manipulation. *Siggraph course notes 25* (July).

GRIMM, C. 2002. Simple manifolds for surface modeling and parameterization. *Shape Modelling International* (May).

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*.

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 21*, 3 (July), 755–762.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July), 856–861.

KAPLAN, M., GOOCH, B., AND COHEN, E. 2000. Interactive artistic rendering. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, 67–74.

KLEIN, A. W., LI, W. W., KAZHDAN, M. M., CORREA, W. T., FINKELSTEIN, A., AND FUNKHOUSER, T. A. 2000. Non-photorealistic virtual environments. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 527–534.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. F. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 433–438.

KOWALSKI, M. A., HUGHES, J. F., RUBIN, C. B., AND OHYA, J. 2001. User-guided composition effects for art-based rendering. In *2001 ACM Symposium on Interactive 3D Graphics*, 99–102.

KULLA, C., TUCEK, J., BAILEY, R., AND GRIMM, C. 2003. Using texture synthesis for non-photorealistic shading from paint samples. In *Pacific Graphics*, 477–481.

MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., HOLDEN, L. S., NORTHRUP, J. D., AND HUGHES, J. F. 2000. Art-based rendering with continuous levels of detail. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, 59–66.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 477–484.

PASTOR, O. E. M., AND STROTHOTTE, T. 2002. Frame-coherent stippling. In *Eurographics short papers*.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *SIGGRAPH 2001, Computer Graphics Proceedings*, E. Fiume, Ed., 579–584.

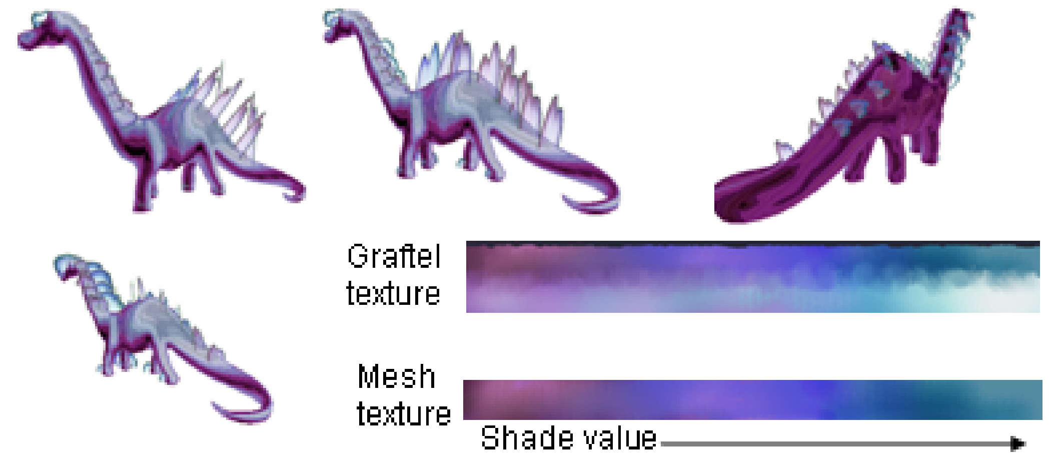

Figure 5: Graftels produced using the user-interface.

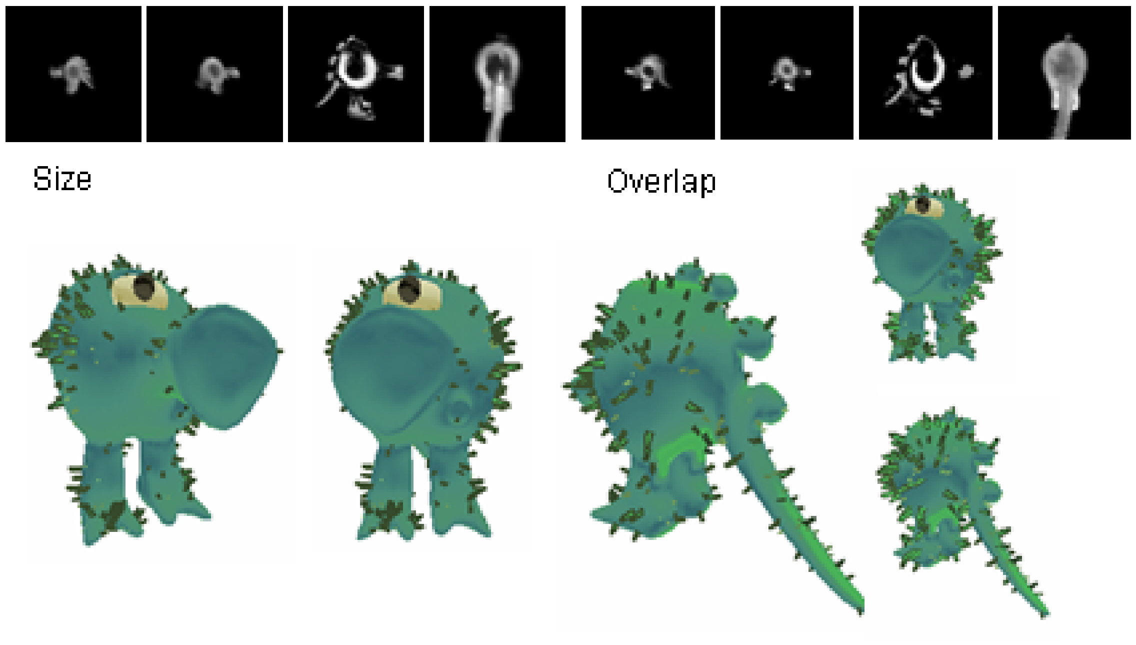

Figure 6: Top row: Images painted by the user to indicate the desired size, allowed overlap, and spacing (same as overlap images). Bottom row: Images from an animation.

WEBB, M., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2002. Fine tone control in hardware hatching. In *NPAR 2002: Second International Symposium on Non Photorealistic Rendering*, 53–58.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, 91–100.