Nick Halper, Stefan Schlechtweg: "Creating Non-Photorealistic Images the Designer's Way" In: Proceedings of NPAR 2002, International Symposium on Non Photorealistic Animation and Rendering, Annecy, Frankreich, Juni 2002, S. 97-104, ACM Press, New York.

Creating Non-Photorealistic Images the Designer's Way

Nick Halper Stefan Schlechtweg Thomas Strothotte

Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg
Universitätsplatz 2, D-39106 Magdeburg, Germany
{nick|stefans|tstr}@isg.cs.uni-magdeburg.de

Abstract

We present a novel way for quickly and easily designing non-photorealistic images based on elementary operations which are linked together to create a variety of visual effects. Rather than mimicking a visual effect that an artist has already produced, we instead mimick the process undergone for the artist to produce that image. Compared to traditional approaches, this opens the possibility to have the images created by users with no programming skills. We describe a modular system that makes these elementary operatios available to the user. A specially designed user interface allows for an easy and intuitive combination of these operations to create an image. Visual feedback is provided to the user at any time and for any stage in the process.

Keywords: Non-Photorealistic Rendering, User Interfaces, Modular System, Sketch Interface, Creative Process, Design

1 Introduction

Non-photorealistic rendering (NPR) has become an important research area in computer graphics. One interesting fact in this area is that many of the techniques developed over the years in computer graphics research can be used here to create specific effects. This covers image processing filters as well as silhouette computation or special rendering algorithms. Indeed, many NPR images are created by starting from a 3D model and combining many different algorithms to yield exactly the image the user wants. Due to the number of algorithms available, there are possibly many combinations that can be made to produce an image. The approach which has to be taken to create a non-photorealistic rendition differs from photorealistic image generation – there is simply no rendering equation to be solved in order to derive a pixel's color.

In particular, non-photorealistic rendering was born out of the desire to create visual effects that convey some visual goal. The key word in the previous sentence is "create", because in order to achieve a visual goal, someone needs to think about what visual effects would successfully enhance the images for the goal in mind, but often the path to the final product or "look" is not immediately known. Instead, a more design-oriented, iterative procedure is what is called for. As a consequence, NPR systems should enable such an approach by (a) providing basic operations for the necessary modules and (b) providing a user interface which supports an iterative, creative combination of these operations.

Most NPR work to date has either "simulated" existing artistic styles (e.g. [2], [6], [11], etc.), or provided the user with an interaction method in order to control parameters for a certain rendering style (e.g. [7]), but rarely come up with "new" styles that have never been seen before. There has been work done by SNIBBE [10] that actually uses the interaction with the computer media as an art form. For a more comprehensive overview of NPR techniques see [13]. However, no such system is available whereby the user can create new styles and apply them to a rendering pipeline that was not previously intended by the system.

If the user can actually control the rendering pipeline and create new effects, then systems can employ designers, rather than programmers, to come up with new effects. What designers can produce is limited only to their imagination and their range of technical abilities. If a rendering system can provide a number of algorithms accessible to the designer, then the designer's range of abilities will be extended. However, the tools provided by the system must be sure not to constrain or influence the designers' natural creative process to which they can apply the full extent of their imagination. Presenting a system that requires the designer to adapt, distract, or place attention outside of the actual task will hinder the creative process to which the designer can come up with ideas. Therefore, rather than adapting the designer to the system, we adapt the system to the designer.

As a result, rather than mimicking a visual effect that an artist has already produced, we instead mimick the process undergone for the artist to produce that image. This in turn requires the development of a user interface, and an underlying non-photorealistic rendering system that is capable of supporting the interface functionality. We treat both aspects in the following sections. We start with rather general thoughts on the design process from which we derive goals and ideas for an NPR system. Such a system is the described in Section 3. In Section 4, we show a few examples, before we conclude the paper and give some directions for future research

2 Mimicking the Designer's Approach to Creating an Image

How a designer produces an idea varies greatly from individual to individual. Therefore, in this section, we provide a very general and simplistic overview of steps involved to generate new ideas and visual effects. From these steps, we are then able to layout the general rules for a user interface and interaction tasks.

2.1 The Designer's Approach

Some of the key issues in the design process, that will influence the user interface and the architecture of a NPR system, are the following:

- Designers usually start with a blank sheet of paper.
- Designers start sketching some simple ideas.
- Designers possibly reuse previous ideas and combine them into their new design.
- Designers play with ideas.

The most important part here is the "playing" with ideas. A designer normally does not get the final idea at once, rather, it is a visual and creative process where the ideas evolve. Visual feedback is very important in this process: You see what you have, and you see where you can take it. Mostly designers attentions are focused on the sheet of paper in front of them that contains previous sketches comprising experimented variations on visual styles. Whenever designers are happy with particular stages of their sketching process, they may clump ideas together, or redraw sketches that evolved from others. They may even go back to a variation on an early sketch and take it in an entirely different direction. In short, at every stage in the creative process designers are constantly comparing the visual effects of styles that they have come up with to see which look best and which have potential to be taken further.

Sometimes designers even make mistakes, or come up with results that they did not intend because of the way they applied a tool, or they just expected something else to come from it. However, often results through experimentation, especially those that are unexpected or bizarre, inspire. Ask any artist, and they might tell you that some of their best ideas or works were created by "accident". As a final remark, note that anyone not involved or monitoring the creation process might have trouble understanding what was going on – most often only the designer herself can see the stages in the development clearly.

2.2 The General Interface

Here we look at which important features a system interface must comprise in order to stay close the designers creative process as described in the previous section:

- The interface starts out with a blank screen of infinite size, that can be moved freely – called the *sketchpad*.
- The interface enables the designer to introduce some original images or scenes.
- The interface allows the designer to apply existing algorithms to images or scenes.
- The interface allows the designer to play with images or scenes by adding effects or applying variations to their visual style.

An effect or variation is something that when applied, either adds to or alters the rendering style. Throughout this paper, these are also referred to as *modifiers* that a user may work with to influence the rendering output.

As a general rule, we do not want to take the users attention away from the sketchpad and break the fluidity of the interaction. Overlayed floating or pop-up windows in this respect are undesirable, because the designers attention is taken completely away from the sketchpad area into a separate window.

2.3 Interaction Tasks

In order to maintain visual feedback, rather than replacing images, we simply add the new images with their variations applied to the sketchpad. To apply a variation the user simply sketches a line starting from the image to apply the variation on to a new location, as shown in Figure 1.

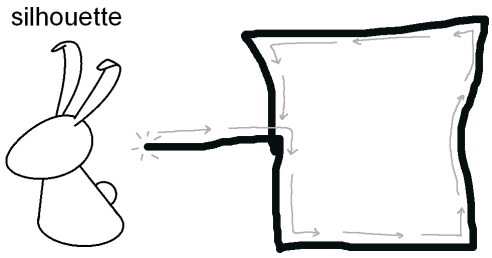


Figure 1: Sketching a new variation. The gray arrows indicate a users path of motion from the point of pressing the button to its release (note that not sketching a box will result in an image generated the same size as the previous one)

The approximate size of the box determines the size of the new image to render. If no box is sketched, then the same size as the previous image size is taken. Once the user releases the pen or mouse button, she may decide which effect or variation to apply to the image, as shown in Figure 2. Here we use pie-menus [5] for several reasons. The number of available options to the user can be large, from anywhere between one to hundreds of modifiers. Since modifiers can be grouped based on the effect they have, hierarchical pie-menus can be used. Interface widgets such as a taskbar featuring drag and drop functionality may have space problems in this respect. Also, regular pull-down menus would become quite large and also draw the user's attention to a completely different location on the screen (see [1]). Figure 3 then shows the result of these simple steps.

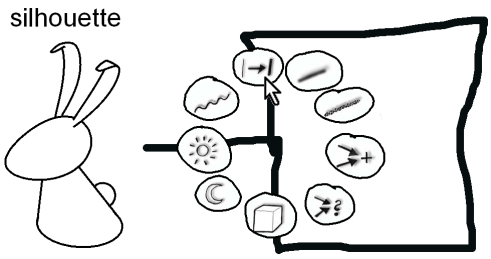


Figure 2: Selecting a variation from a pie-menu.

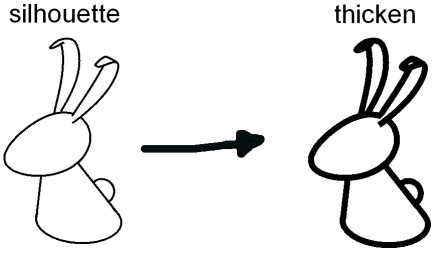


Figure 3: Final result of applying a variation

So far, we have achieved the following:

- The user's focus has remained on the area of the sketchpad comprising the images of interest at all times.
- Placement and size of the new image is intuitive.
- Both images are maintained and still accessible in the sketchpad for potentially applying further variations.
- The image on which the variation was applied is clearly traceable because of the user's original sketched line to the location of the new image.
- The name of the variation is labeled next to the image, with its effect clearly visible.

Inputs to a variation can be changed at any time, simply by sketching from a desired image to the image that has that variation. Figure 4 shows this.

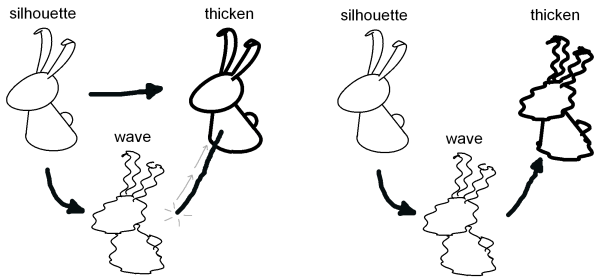


Figure 4: Changing an input by sketching the new connection as indicated by the gray arrows (left). The old connection is automatically removed (right)

Note that any changed image will propagate its changes automatically to the entire subsequent connected paths of images that apply variations on it. In addition, connections can be easily re-connected back to the original simply by re-doing the initial sketch for that connection. Some existing systems provide a history window (such as Adobe Systems Photoshop), whereby you can also see the results in the history pipeline, and you can undo a current state to a previous one. In countless design books (e.g. [9]) they highlight the importance of providing a big friendly undo button. However, in our interface, there is no explicit "undo" option – it is inherent in the process since we have visual access to the entire history and for the created images in a *non-linear* fashion. This is an important feature, since often the undo button is used when "mistakes" have been made, and it breaks the continuous process if used because it reverts "back" to a previous state. In the process of a design, users do not want to go back a step, they want to see themselves taking ideas forward. In our interface, mistakes are not considered "undone", but "redone" according to what looked best (e.g. sketching to replace a new input gives a result, and to undo this action, we just sketch the previous connection back in).

Sometimes variations may be applied as a result of a combination of more than one image or scene (i.e. a modifier takes more than one input). In this case, when the user is asked to select an effect, and chooses one which requires multiple inputs, is then asked in a similar fashion which of these inputs he wants to connect to. Any modifier that has not been given its required number of inputs places a default image at the location where the resulting image should be, so that it is clear that another input is required for the variation to be applied.

Once a user has finished parts of a design or decided that a certain group of modifiers performs an operation which can be regarded as a modifier in its own right, the user can group these and thus create a new modifier. This is done by simply circling the affected modifiers. The system will then create a new modifier that encapsulates the selected group and computes the number of necessary inputs and outputs. The new modifier can then be used in the same manner as any other previously defined or given modifier. The user can then go to a different area of the sketchpad (or start a new sheet) and simply bring that modifier back in. Most importantly, it can be assigned a name (labels may also be given for each input) and saved for later reuse in a different project.

The interface that we have shown here shares similarities with data-flow interaction techniques as used in several professional packages (such as the image compositing tools in Discreet's Combustion, or Silicon Grail's RAYZ). However, we have both tightened and simplified our interface with our task goal in mind - we wish to create new rendering algorithms. This differs from, for instance, purely image compositing tasks that can be performed most often by reusing the same combinations of tools with different parameters. We keep the workspace with the hand-drawn sketch connections, rather than computing an algorithm to make neater connections, because we do not want to give the impression of a finalised result. This takes ideas from NPR concepts in general (e.g. the presentation of sketched architectural drawings, [8]). The connections created by our system are also abstracted to the user (see Section 3.3) so that the visual display of the data-flow to the user is not necessarily the same as that undertaken by the system.

There is a natural process to design, so we want to see the workspace as natural and not given over to the system to organise it. In this sense, each workspace can be visually identified as belonging to a user (i.e. a user will have his own characteristic sketching abilities, and it will be clear to someone looking at the workspace who's workspace it is). Users that are naturally "neater" in their process, will have neater looking workspaces. Frantic users will have frantic looking workspaces. This also gives users the added impression that it is their work, and not that the system has merely taken their input and programmed a set of combinations.

Astonishingly, this is enough for the user to start playing and creating all kinds of combinations of effects. The interface provides the tools to the user quickly and comfortably. Now we need to provide the underlying system that makes this possible.

3 Providing the Underlying System for the Interface

We wish the user to have free experimentation with the system and come up with new effects and modifiers. As mentioned earlier in Section 2.2, a modifier alters the rendering style so that it produces a certain effect or variation. In this section, we will look at the possibilities that we have to influence the rendering style.

3.1 Requirements for Modifiers

A modifier influences the rendering style by adding, removing, or altering some set of operations that form part of the rendering pipeline. In order for a modifier to do so, it has to fulfill the following requirements:

- Modifiers must have (one or more) inputs and one output.
- Connections from the output of one modifier to the input of any other modifier must be possible.
- Whenever an input to a modifier is changed, its output needs to be newly generated. This output may be connected as an input to another modifier, thus propagating the re-computation to all subsequent connected modifiers.

The actual rendering is done by what we call *modules*. These modules are placed in a rendering pipeline and thus can be regarded as "basic operations" in such an NPR pipeline. A modifier changes the rendering style by selecting which modules form part of the actual rendering pipeline that is used to create an image. The modules themselves are independent, but the results of the overall system output are dependent on the modules. This means that we may add a module to any stage of the rendering pipeline that may influence the subsequent rendering output. As a result, the rendering pipeline is not fixed. This is by far the largest difference to photorealistic rendering approaches, where the rendering pipeline is tightly defined (see [3]).

We choose to base our system on object-oriented architectures, such as [12] and [14]. This enables us to use a scenegraph-based approach. Our system, OpenNPAR, extends OpenInventor to incorporate as many NPR techniques as we can. In addition, we provide image modules that can potentially perform any image operation or apply any image processing filter.

The user interface as introduced earlier, requires that we show the result of applying a modifier and thus provide direct visual feedback to the user. Hence, whenever a modifier is introduced to the sketchpad, we need to store its output as an image (this is done simply by rendering to and reading from an off-screen buffer) that is constantly linked and updated to the presentation of the modifier in the sketchpad.

Our system goal is to provide a set of elementary modifiers that perform very simple alterations to the rendering output. They should be as simple as possible since this allows for a larger range and variety of visual styles. These elementary modifiers can then be linked together or grouped by the user at run-time to produce more complex rendering styles. This does not require any programming skills. We shall now look at a few of these elementary modifiers.

3.2 Example Modifiers

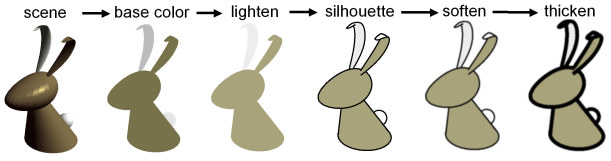
Here, we give a few examples of modifiers that use various modules. For the interface any number of modifiers may be provided. For instance, strokes are an NPR technique that may be added to the scene or image, such as when drawing silhouettes [4, Chapter 7], but they can also be extended to fur and art-based rendering [6]. For the purpose of this paper we select only a few simple modifiers that are used throughout as examples:

- *Scene modifiers:*
  - *Light Intensity Filter:* Parses an input scene and replaces all materials with a white, non-specular material. On its own, the result would be that the rendered scene is white in areas with high illumination, and dark in areas with little or no illumination.
  - *Base Color:* Changes the lighting model in the rendering pipeline to base color illumination. On its own, this would effectively render colors based only on the materials' diffuse properties.
- *Stroke Modifiers:*
  - *Silhouette Generation:* Adds a module that computes a silhouette based on our viewpoint. The silhouette module constructs a set of strokes that define the silhouette.
  - *Stroke Thicken:* Adds a module that influences stroke thickness.
  - *Stroke Waviness:* Adds a module that applies a wave function over the stroke paths.
- *Image Modifiers:*
  - *Lighten:* Takes an image as an input, and lightens all the pixels in the image for the output.
  - *Dither:* Takes as input an image that defines a dither-matrix and a source image, and applies a dithering algorithm to produce an output image that comprises only black and white pixels.
  - *Selector:* Takes as input three images. One of the inputs is computed as an alpha-image, that selects which of the pixels in the other two images are filtered through to create the output image. If the alpha-image pixel is above a certain threshold, the first image pixel passes through, otherwise the second image pixel passes through.

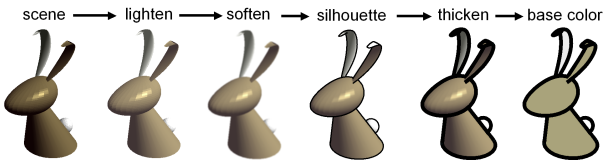
3.3 Piping Modifiers Together

The system must make sure that any connections between modifiers are successfully made, and in particular, that no connections are deemed impossible or not allowed. In order to get the system to allow the user this total freedom to connect anything to anything, we abstract from the various types. Now the user does no longer need to be concerned about, for example, connecting the output of an image modifier – which is usually an image – to the input of a scene modifier – which is usually a 3D scene. All the user needs to know, is that a modifier alters the output in some way so that it can produce some sort of variation on the rendering style.

Here we will see two sequences of modifiers that are piped together. Note, that the results displayed are those that are intuitive or most likely to be expected by a user.



In the above example, the model's colors are changed first to base colors only, then lightened, and then a silhouette is placed over the image that is subsequentely softened, and then thickened. Note that the thickened silhouette remains softened. In the underlying system, the rendering pipeline first replaces the scene's light models to use base color lighting, which is then rendered to an image that is then lightened. Afterwards, a silhouette is computed from the scene, and rendered on top of the lightened image. This result is rendered back to an image, so that this image can then be softened. In the next stage, the silhouette is thickened, but must be rendered on top of the *lightened* result, so that it is then softened to give the resulting soft thick silhouette shown.



In the second example above, the scene is lightened, then softened. A silhouette is then drawn around the scene, that is subsequently thickened. Finally, the image changes its scene colors to base colors. Note that the base colors are also lightened and softened, whereas the silhouette remains sharp. In the underlying system, the scene is rendered to an image, that is then lightened, and then softened. The silhouette is computed from the original scene, and rendered on top of the lightened softened image. The silhouette is thickened, but this silhouette must again be rendered on top of the lightened softened image. Finally, the base color modifier is added to the sequence. This modifier works on the original scene to change the lighting model to base color lighting, ther result of which is then lightened and softened before adding the silhouette.

From these examples, we observe that:

- Image Modifiers inputs are always connected to image outputs. If the input is not an image, it is rendered to an offscreen-buffer, read into an image and connected to the input.
- Scene Modifiers always operate on the last scene that served as an output (i.e. image modifiers $\leftrightarrow$ scene modifiers are commutative). For instance, if we have a sequence of several image modifiers, we must trace back to the available scenes which originally served as an input to those modifiers.
- Stroke Modifiers also operate on the last scene that served as an output, but a new stroke modifier is connected directly after the last stroke modifier in the series of connections if there is one present, otherwise it is added at the end.

This allows the user to connect the outputs of any available modifiers directly to the inputs of any other modifiers, leaving the type conversion and actual pipeline order of the affected modules to the system.

In some cases, we may have multiple-input modifiers that can directly configured to take a variety on input types. For instance, a combiner modifier that takes two inputs and merges them into one output, may overload the inputs in a similar fashion to object-oriented overloaded functions (several functions having the same name, but the called function depends on the number and type of parameters given). This allows us to have functions that merge two scenes together, two images together, or a scene with an image, without having to undergo a conversion process.

However, we must be careful to maintain consistency in our results so as not to confuse the user. For instance, consider a modifier that darkens an input source. If we have two methods, one that uses modules in the scene, and one that uses image modules, the results may differ. Darkening a scene may be done either by dimming the light source in the scene, or by darkening the material's diffuse color in the scene. Darkening an image might darken each pixel comprising the image. Note that the results between the scene and the image modifier in this case are different – the darkened image converges to black over the entire image, whereas the scene modifier only converges to black where there is geometry present (background areas where no polygons are rendered will be unaffected).

4 Examples

Here we provide a sequence of screenshots to give an overview of the interface and some feel for the creative process that the user undergoes during the interaction. Looking at the screenshots may be a little overwhelming at first, but actually being part of the process as the ideas build up feels very natural to the user.

Our examples use a rather limited set of modifiers, but even so, a large range of visual effects are created, despite the limited number of modifiers given to the user.

In Figure 5, the user has imported a 3D model that is rendered using PHONG shading and displayed in the upper left corner of the sketchpad. From this, the user has found a way to create a cartoon effect by using a selector that takes the light intensity of the model to choose whether or not to use a lightened base color image, or a darkened base color image. At the top, we also see that the user has created a dithered image by importing a dither matrix that is fed into the dither modifier and thus applied to the scene image. The user has then extended the cartoon image and dithered it using the same dither matrix.

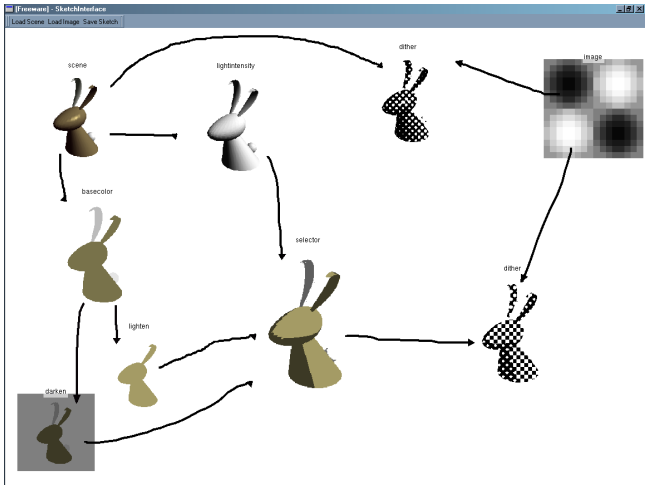


Figure 5: First stages in the development of effects

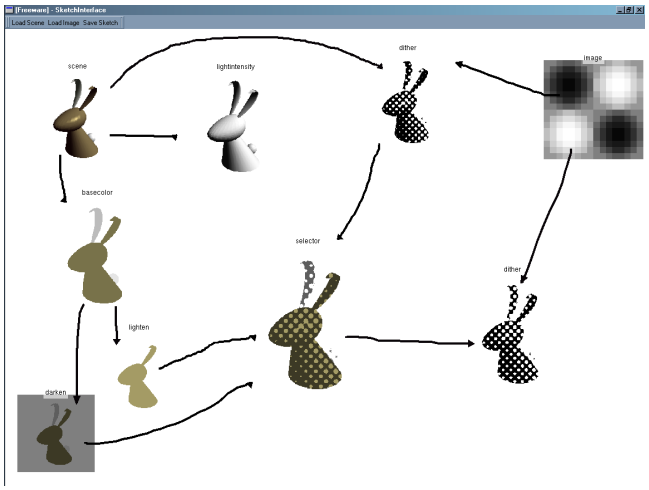


Figure 6: Changing an input and propagating effects

In Figure 6 the user changes the result simply by replacing one input to the selector modifier. This modifier now takes as input the dithered image instead of the light intensity image. The result is that the graphical representation of the selector modifier is immediatly updated, and also the dither modifier placed at the bottom right of the sketchpad that it connects into. The original connection from the light intensity modifier to the selector is removed.

The user has continued with the same sketchpad and played with the interface to give a variety of results as shown in Figure 7. The modifiers from the first two examples can still be seen in the upper left section of the sketchpad. Note that in all these examples, all results are produced by only using one imported image (the dither matrix), one imported scene, and nine elementary modifiers. Expanding the modifier library would widen the range of technical abilities to which the user has access, and thus greatly expand the range of effects that may be produced.

5 Conclusion

We have provided a modular rendering system that fits the needs for non-photorealistic image generation by providing different modules which themselves perform basic operations on images or 3D models and which can be linked together in order to achieve certain visual effects. Based on this system, we have provided a user interface that enables and encourages the user to "play with ideas" in order to combine modifiers that affect the rendering pipeline without actually having to write any code. Each application of an idea generates a new image, such that each image in the process of the evolved idea can be seen. Thus, at every point in time, visual feedback is given and maintained. Grouping mechanisms allow to create new modifiers, that evolve from a set of ideas. Also, the user may take any image in his creative process, and start a new direction of ideas. The undo feature in our interface is both implicit and non-linear. Mistakes are not "undone", but effects "redone", which maintains the continuity of the creative process.

Also of interest, is that the style of the user in coming up with ideas is recognizable. The way the sketchpad is organized is dependent on the way the user likes to draw out new variations. It does not matter if things look unorganized afterwards – this is analogous to a designer sketching on paper – the only person who understands what is going on is the designer herself. Even the screenshots that the reader sees in this paper will appear quite confused, except to the original author of the sketch.

We have left as many possibilities open as possible so that the user is not limited by the system. The user is allowed to link modifiers together in arbitary ways, regardless of how inappropriate the connections may seem. Results may not quite be to the user's liking, it might not even be what was expected, but we decided not to give the power to the programmer to constrain what the user may or may not combine. Instead, we leave the users to make their own mistakes, with the hope of providing inspiration throughout the course of the creative process.

6 Future Work

Many questions and directions for future work arise from the work shown in this paper. First, we want to provide more and more powerful modifiers. These should not only allow a variation to be applied to a certain state but also actually provide functionality such as connecting two entire seperate sequences of modifiers. Such a combiner modifier could, for instance, connect a cartoon-shader and a pencil-sketch shader as shown in Figure 8. This might also require an intelligent selection of parts of an image or of a scene so that the effect of these modifiers will be constrained to these parts.

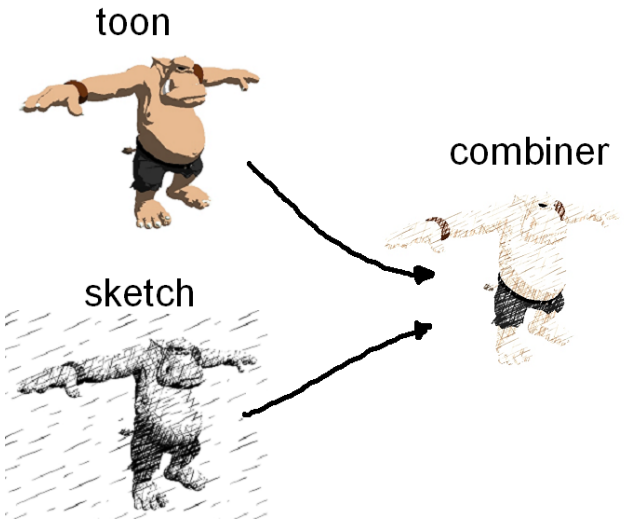


Figure 8: an intelligent combiner modifier

Another powerful feature would be to be able to directly interact and sketch "into" each presented image at any given stage in the sketchpad. This would let us vary parameters, or simply add to the images with additional editing tools, and see the effects propagate to the subsequent connected modules automatically. In effect, by sketching into a modifier, we are editing the modifier and may even save its changes as a new modifier. This leads to the general question of how does the user specify parameters for a modifier. Taking into account the design goals stated earlier, the use of sliders, dialog boxes, or menus would not be the method of choice. The level of abstraction when using such tools is different from the user's task so that she would be distracted. A direct manipulation by, for instance, sketching the desired result of a parameter change, would greatly enhance the interface.

The grouping functionality that is implemented so far works well for "regular" cases. It can, however, only be applied if the modifiers to be grouped are spatially close together on the sketchpad and if the user is able to circle the modifiers in such a way that the circled region contains *only* the modifiers to be affected. This might not necessarily always be the case, so a re-arrangement of the modifiers might be needed before the grouping which would destroy the sketchpad layout evolved from the design process. It is not clear so far how the interaction should be designed for such an operation.

A further development of an NPR system that is designed to primarily support the user and the creative process in designing images is only possible with the participation of the target group. So a comprehensive user study will not only clarify some of the issues raised earlier but also lead to new ideas which themselves will move the development forward.

Finally, it is also possible to use the interface to present available resources. Many NPR methods, especially simulation approaches, are rather time and resource consuming. On the other hand, many methods can also be implemented using modern graphics hardware (for example, register combiners). Due to a limited number of such hardware resources, we may extend the interface to provide some sort of resource managment, such as suggesting better paths through modifiers, or highlight paths that use too many resources.

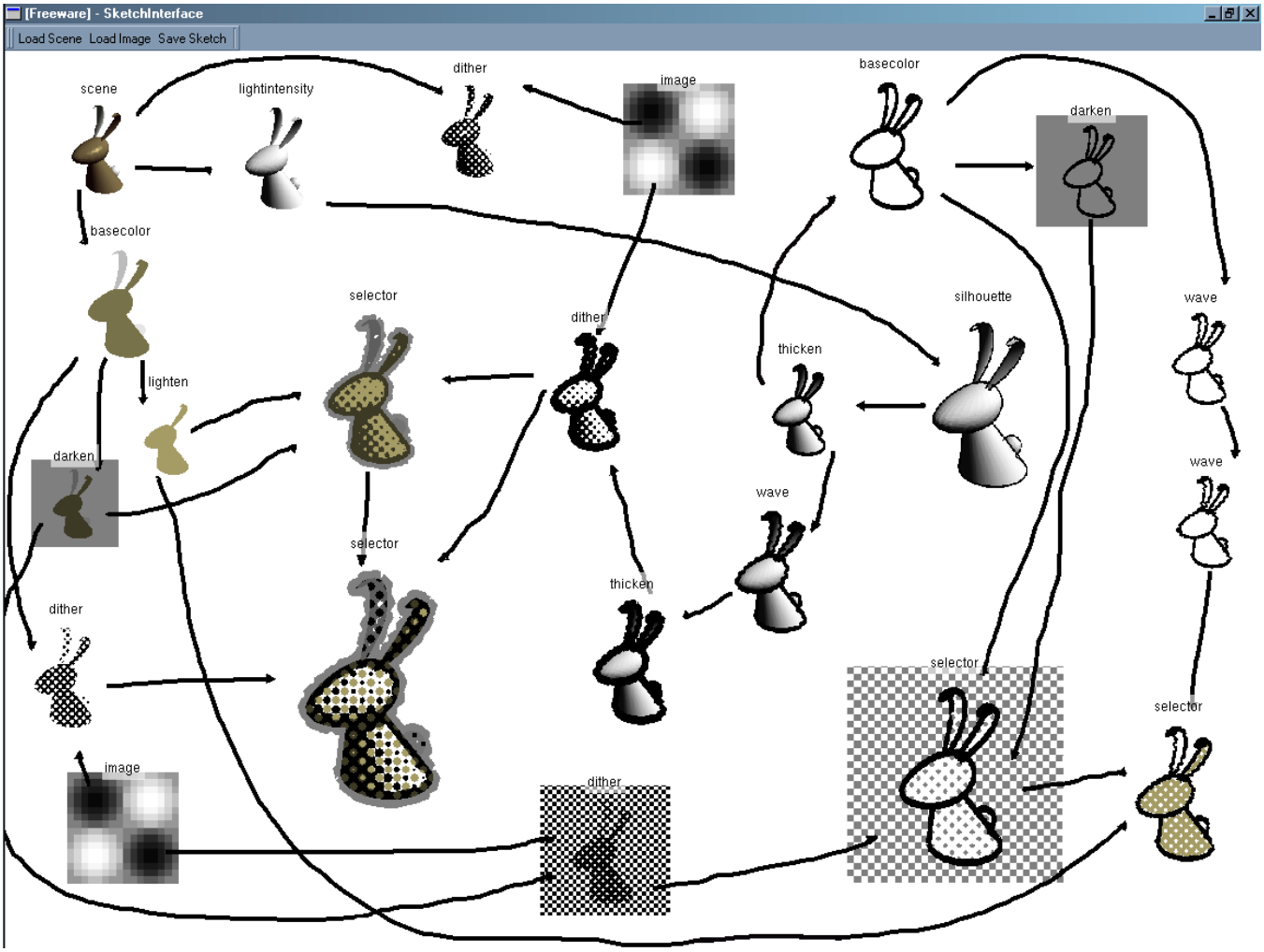


Figure 7: Zoomed out view of the sketchpad after the user has played with it for some time

7 Acknowledgements

The authors would like to thank Nico FLOHR and Jana HINTZE for discussions, designs, and help with the paper. A special thank you goes to Tobias ISENBERG for his support in helping to provide the underlying system of the project at such a critical stage early in its development. This work was in part financed by a doctoral scholarship from the Province of Saxony-Anhalt.

References

[1] Jack Callahan, Don Hopkins, Mark Weiser, and Ben Shneiderman. A Comparative Comparison of Pie vs. Linear Menus. In *Human Factors in Computing Systems, Proceedings of CHI 1988*, pages 95–100, New York, 1988. ACM Press.

[2] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-Generated Watercolor. In Turner Whitted, editor, *Proceedings of SIGGRAPH'97 (Los Angeles, August 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 421–430, New York, 1997. ACM SIGGRAPH.

[3] James D. Foley, Andries van Dam, Steve K. Feiner, and John F. Hughes. *Computer Graphics. Principle and Practice*. Addison Wesley Publishing Company, Reading, 2nd edition, 1990.

[4] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. AK Peters Ltd., San Francisco, 2001.

[5] Don Hopkins. The Design and Implementation of Pie Menus. *Dr. Dobb's Journal*, 1991.

[6] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John F. Hughes. Art-Based Rendering of Fur, Grass, and Trees. In *Proceedings of SIGGRAPH'99 (Los Angeles, August 1999)*, Computer Graphics Proceedings, Annual Conference Series, pages 433–438, New York, 1999. ACM SIGGRAPH.

[7] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive Pen-and-Ink Illustration. In Andrew Glassner, editor, *Proceedings of SIGGRAPH'94 (Orlando, July 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 101–108, New York, 1994. ACM SIGGRAPH.

[8] Jutta Schumann, Thomas Strothotte, Andreas Raab, and Stefan Laser. Assessing the Effect of Non-photorealistic Rendered Images in CAD. In *Proceedings of CHI'96 Conference on Human Factors in Computing Systems (Vancouver, Canda, April 1996)*, pages 35–42, New York, 1996. ACM SIGCHI.

[9] Ben Shneiderman. *Designing the User Interface*. Addison Wesley Publishing Company, Reading, 1992.

[10] Scott Sona Snibbe and Golan Levin. Interactive Dynamic Abstraction. In *Proceedings of NPAR 2000, Symposium on Non-Photorealistic Animation and Rendering (Annecy, France, June 2000)*, pages 21–29, New York, 2000. ACM.

[11] Mario Costa Sousa and John W. Buchanan. Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models. In Pere Brunet and Roberto Scopigno, editors, *Proceedings of EuroGraphics'99 (Milano, Italy, September1999)*, pages 195–207, Oxford, 1999. NCC Blackwell Ltd.

[12] Paul Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. In Edwin Catmull, editor, *Proceedings of SIGGRAPH'99 (Chicago, July 1992)*, Computer Graphics Proceedings, Annual Conference Series, pages 341–349, New York, 1992. ACM SIGGRAPH.

[13] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann Publishers, San Francisco, to appear March 2002.

[14] Josie Wernecke. *The Inventor Mentor: Modeling: Programming Object-Oriented 3D Graphics with Open Inventor*. Addison Wesley, Reading, 1994.