OPENNPAR: A System for Developing, Programming, and Designing Non-Photorealistic Animation and Rendering

Nick Halper Tobias Isenberg Felix Ritter
Bert Freudenberg Oscar Meruvia Stefan Schlechtweg Thomas Strothotte

Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg
{nick|isenberg|fritter|bert|oscar|stefans|tstr}@isg.cs.uni-magdeburg.de

Abstract

The notable amount and variation of current techniques in non-photorealistic rendering (NPR) indicates a level of maturity whereby the categorization of algorithms has become possible. We present a conceptual model for NPR, on which we base a modular system, OPENNPAR, *which integrates NPR algorithms into distinct classes. Components in* OPENNPAR *are modularized and consequently reintegrated for various rendering purposes, allowing many kinds of NPR algorithms to be reproduced, including the integration of 2D and 3D methods. Additionally, the system provides support for a range of users (developers, programmers, designers) according to their respective levels of abstraction, thus being available in multiple contexts. Ultimately,* OPENNPAR *holds great potential as a tool in the development, augmentation, and creation of NPR effects.*

1. Introduction

The nature of non-photorealistic rendering (NPR) in its simplest definition, is a form of visual communication. As communication is virtually endless in its possibilities, NPR attempts to succinctly define options within this scope. Particular rendering styles are capable of conveying context-specific information in an application-dependent environment. Thus, there is a need for an effective rendering system which provides support for multivariate applications.

Despite the plethora of non-photorealistic effects available [2, 9], there exists a rather limited number of primitives actually employed to generate these effects [1]. There also remains a similarly limited number of general techniques for the application of these primitives. The ingenuity of the algorithms underlying the aforementioned effects lies not in the mere application of these primitives, but rather, in their *combination*. Thus, a system could be designed wherein all modular components are freely combined and interchanged. Moreover, photorealistic rendering is often used to complement NPR. Therefore, this system could also include photorealistic capabilities.

To achieve the necessary modularity for the proposed system, NPR techniques must first be categorized according to their various properties. Specific classes of algorithms can then operate on the same sets of data—consequently sidestepping unnecessary data conversions between software projects. In addition, NPR algorithms can be individually broken down into a set of smaller algorithms, wherein an 'elementary set' of algorithms are eventually defined. Logically, keeping modules small and simple increases the range and flexibility when generating more complex algorithms. Finally, functionality is little without application. An effective means of presenting available options in the system to a variety of users will allow content creation at a level that satisfies individual requirements. Involved herein are those who actually create the modules, those who plug the different modules together to create a specific effect, and finally those who use effects to produce images.

2. OPENNPAR

Our main contribution is an attempt to unify many NPR techniques into a single framework, OPENNPAR, that is accessible by users at different levels of abstraction. This section first categorizes base classes for algorithms to establish the conceptual framework for OPENNPAR and then describes OPENNPAR's architecture and user classes.

2.1. Classes of Algorithms

Classification of NPR techniques [1] aid both terminology and discussion, yet there is still no clear direction for a unification of algorithms into a single system. A recent effort specialized for stroke stylization constructed a

pipeline of operations allowing flexible combinations of effects [3]. Unfortunately, an ordered pipeline of operations within a larger framework for NPR would potentially sacrifice the system's generality. Therefore, our basic philosophy is to allow free flow of operations between sets of primitive states. By classifying primitives according to our conceptual model in Figure 1, building blocks (individual processes) can be arbitrarily combined (a sequence of processes) for different rendering pipelines (the collective sequence of processes).

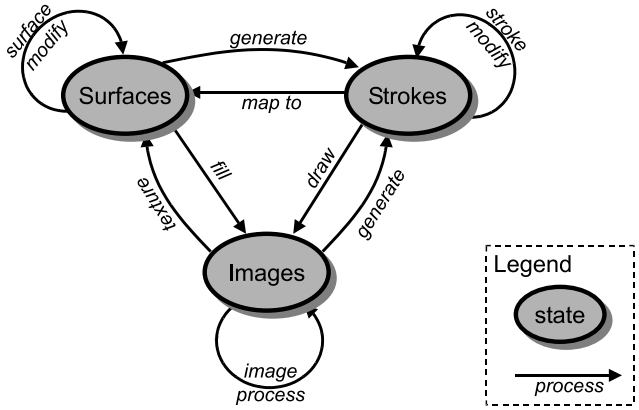


Figure 1: Our conceptual model for algorithms in NPR.

Many hybrid algorithms combine two or all three of these classes [1, 9]. Thus, by providing numerous but small and well-identified processes centered around a set of extendable basic primitives, we hope to achieve a generality of use covering a large range of NPR algorithms. The goal of OPENNPAR is to embody our conceptual model and enable users at all levels of expertise to interact with the system using knowledge requirements depicted in Figure 2.

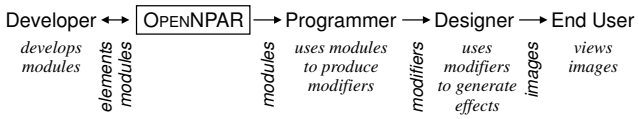


Figure 2: OPENNPAR knowledge pipeline

2.2. Developing OPENNPAR

OPENNPAR is built on the foundation provided by OPEN INVENTOR, an object-oriented graphics architecture, that employs a scene graph based approach [8]. Essentially, OPEN INVENTOR developer principles are adapted to meet our conceptual model requirements—primitives are composed from *elements* in OPEN INVENTOR, whereas processes are performed in the render action procedure in nodes during a rendering traversal. In addition, OPENNPAR restricts each node to perform one specific process only—we distinguish these nodes from normal OPEN INVENTOR operation by calling them *modules*. Thus, we execute processes in Figure 1 though the use of modules, and represent the classes of primitives with elements.

The developer adds functionality to OPENNPAR by creating numerous new modules that each compute a single, specific task. In contrast, primitive elements in the system are extended but kept to a small generic set suitable for a broad range of application. Thus, a variety of modules operate on the same primitive elements which aids the interchange of data and resulting flow of computation.

2.3. Programming with OPENNPAR

Programmers access OPENNPAR's functionality by placing modules into a rendering pipeline and taking care of module dependencies and data flow connections, either by editing an external scene description or by calling OPENNPAR's API within an application. Their primary task is to exploit OPENNPAR's existing range of effects and potentially define new algorithms by ordering modules and interchanging data in novel ways. Since NPR algorithms often require access to multiple stages in a rendering pipeline, *fields* are used to aid the propagation of data between modules within scene-graphs.

2.4. Designing with OPENNPAR

Whereas the programmer has the technical expertise to experiment with the system at a modular level, designer productivity increases when part of an entirely visual and iterative process. Therefore, we devised an interface to overcome technological knowledge requirements on designers that closely follows our conceptual model's data flow [5].

Designer's interact with OPENNPAR by applying *modifiers* that enable both (1) a *method of interaction* which leaves the designer unaware of the data being used to create a given effect, and (2) a *method of computation* which assembles a unique pipeline of graphical operations to achieve the effect. A modifier manipulates, adds, or removes modules in a scene-graph as defined by the programmer. In addition, modifiers handle all internal ordering dependencies and data conversion between modules to produce a desired effect. The end result—each modifier assembles a unique operation pipeline for the effect that is compatible with the required system data flow, yet the designer's original application order of effects is visually maintained.

3. OPENNPAR Examples

This section demonstrates the range of effects achieved with OPENNPAR, using simple examples to highlight advantages in the development and subsequent re-use of modules and modifiers by programmers and designers respectively.

3.1. Silhouettes

A number of succinct modules are defined in OpenNPAR to encapsulate elementary stroke algorithms for stylization, silhouette extraction, stroke concatenation, and artefact filtering (see [6]). Each stroke module alters the current stroke primitive state in the pipeline. Therefore, in similar fashion to the stroke stylization stages in GRABLI et al.'s system [3], the programmer selects modules and pipes them in different ways to produce different cumulative results. The result of a silhouette extraction module with subsequently applied stroke stylization modules that add a chalk texture, texture coordinate generation, stroke orientation, and thickness depth-cueing (whereby thickness is aligned according to stroke orientation) is shown in Figure 3(a).

Since basic algorithms are individually encapsulated in modules, a different effect is easily achieved by replacing the silhouette extraction module with a surface skeletonization module-rather than setting stroke primitives to form a silhouette, strokes are generated based on the model's skeletal structure. Rendering of skeletal strokes is more efficient due to its view-independency in contrast to silhouette extraction that requires re-computation with every change in viewer position. Observing that skeletal strokes are effective when the object is at a distance (see Figure 3(b)), both the silhouette extraction and skeletonization module are grouped under a level-of-detail module that selects which of the two to process depending on the object's projected size. Thus, when the object is visibly small, skeletal strokes are used to depict the object's form instead of its actual silhouette.

3.2. Interactive Illustration

Since OPENNPAR's functionality is layered on top of Open Inventor, we already have substantial support for user interaction. For instance, a 3D Painter application (Figure 4(a)) is written in less than 1000 lines of code that enables the user to 'paint' onto the surface in various styles in similar fashion to [7]. Each point picked on the surface maps its surface coordinates, normal, and color (material or texture) to a newly generated stroke point. The stroke stylization modules from our previous example subsequently determine the quality, texture, and style of the stroke.

Interaction can also be extended specifically for OPENNPAR. Figure 4(b) shows an interactive illustration that allows users to pick shadows while combining photorealism and NPR to provide relevant abstraction in guiding user focus. Silhouettes and stroke stylization modules are applied to shadow data to illustrate the interaction context, whereas skeletonization modules help determine anchor text placement.

3.3. Real-Time Shading

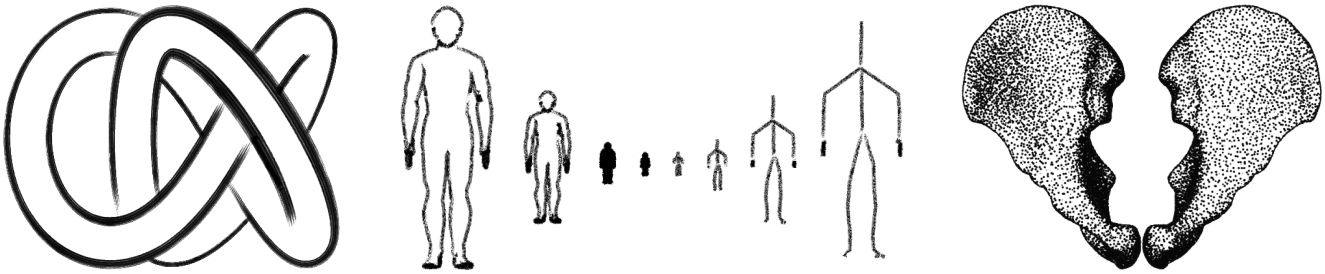
Rendering modules access primitive states to determine output. Default conditions are assumed for primitive state components that remain undefined, otherwise values introduced by previous modules are used. This gives modules freedom to manipulate primitive elements to alter a subsequent effect. To achieve real-time shading, developers introduce modules that alter programmable features of graphics hardware. The programmer defines the code to be compiled at run-time, and the hardware configuration subsequently alters output from standard rendering modules. Figure 4(c) shows examples for cool-to-warm shading, colored hatching, cel shading, and black-and-white stroke textures all implemented with vertex programmable modules and texture-combiner modules. Hardware programmable features in OPENNPAR can also render output in real-time from data generated from external applications. Stippling points and conditions are generated off-line and output to an OPENNPAR compatible scene graph defining modules for real-time stipple and silhouette rendering (Figure 3(c)).

3.4. Animation

OPENNPAR is capable of reading VRML files extended to include animation modules. To produce the variety of results in Figure 5, the programmer first edits the exported VRML scene description from a professional 3D animation package to include an image module that reads frame-buffer contents after each animation frame rendering. A variety of image processing modules are then appended to the scene graph description and their output connected to a video module parameterized to insert images at specific time intervals to a video file. The scene description is imported into an OPENNPAR viewer application that subsequently writes the animation to disk in a specified video format.

3.5. Designer Interaction

The advantages of OPENNPAR become apparent with its designer's interface: modifiers are easily implemented by programmers, and designers subsequently explore diverse effects to reach a communicative intent using even a limited set of modifiers. For example, Figure 6 demonstrates a designer's work on two initial images using only 10 modifiers to come up with a 'sponge-painting' effect within a



(a) Visible silhouette rendered in real-time with oil paint texture and depth cueing.

(b) Comparison of rendering silhouette drawings or skeleton drawings for different sizes of the object.

(c) Using OPENNPAR to render stipples and silhouettes.

Figure 3: Sample algorithms implemented using OPENNPAR.

few minutes. Notice how the designer applies a new effect directly after each previous visual result without the need to observe module dependencies.

4. Conclusion

We have presented OPENNPAR, a system for creating NPR and animation. OPENNPAR appears to be the first system of its kind that allows for a range of different user classes to both reproduce a variety of algorithms as well as create new ones. This was made possible by structuring OPENNPAR onto a conceptual framework for NPR that categorizes algorithms and primitives to support the interchange and reuse of data. Consequently, OPENNPAR offers potential for defining an effective presentation method within the wide scope of NPR.

Developers increase OPENNPAR's functionality by constructing modules and extending primitives. Programmers access functionality either by linking an application directly to OPENNPAR or through textual descriptions of modules in a rendering pipeline. Due to its modular structure, predefined effects can be reproduced or entirely new ones created by the manipulation of interchangeable modules.

A potential limitation of the system is that algorithms are constrained to formulations in the scene graph. Thus, certain NPR pipelines utilizing multiple primitives, in particular those requiring feedback loops, require atypical structuring of the scene-graph. This may invoke additional implementation overhead and loss of performance. Although we are using OPENNPAR in both advanced education and research, we have not yet conducted a broad evaluation of the usability of the tool. However, we hypothesize that many existing NPR algorithms can be created by modularizing components into OPENNPAR. The effectiveness of the system lies in the flexibility of available modules and the completeness of its elements.

In conclusion, OPENNPAR holds potential as a powerful tool for the development, augmentation, and creation of NPR. Further details about OPENNPAR can be found in [4], and additional animations and example application are available at `www.opennpar.org`.

References

[1] F. Durand. An Invitation to Discuss Computer Depiction. In *Proceedings of NPAR'2002*, pages 111–124, New York, USA, 2002. ACM Press.

[2] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. A K Peters, Ltd., Natick, 2001.

[3] S. Grabli, F. Durand, E. Turquin, and F. Sillion. A Procedural Approach to Style for NPR Line Drawing from 3D Models. Technical Report 4724, INRIA, 2003.

[4] N. Halper. *Supportive Presentation for Computer Games*. PhD thesis, University of Magdeburg, Submitted 2003.

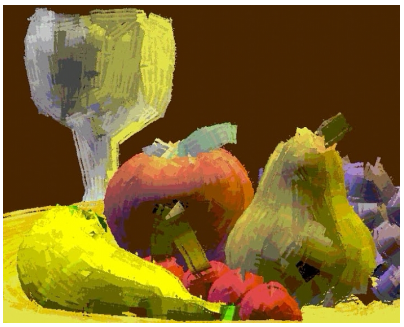
[5] N. Halper, S. Schlechtweg, and T. Strothotte. Creating Non-Photorealistic Images the Designer's Way. In *Proceedings of NPAR'2002*, pages 97–104, New York, 2002. ACM Press.

[6] T. Isenberg, N. Halper, and T. Strothotte. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum (Proceedings of Eurographics 2002)*, 21(3):249–258, Sept. 2002.

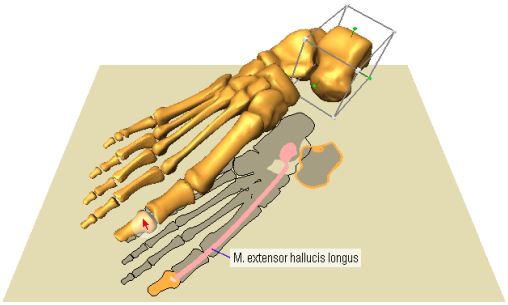
[7] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In *Proceedings of SIGGRAPH'2002*, *Computer Graphics* Proceedings, Annual Conference Series, pages 755–762, Reading, MA, July 2002. Addison Wesley.

[8] P. Strauss and R. Carey. An Object-Oriented 3D Graphics Toolkit. In *Proceedings of SIGGRAPH'99*, pages 341–349, New York, 1992. Addison Wesley.

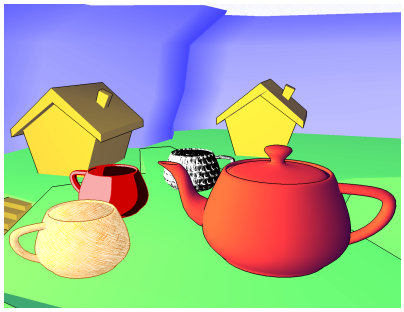
[9] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann, San Francisco, 2002.



(a) *3D Painter*: Textured strokes 'paint' over models in this still life scene



(b) *Illustrative Shadows*: Shadows convey the current interaction context



(c) *Real-time NPR*: Modular surface shaders integrated into a game

Figure 4: Applications that use OPENNPAR.

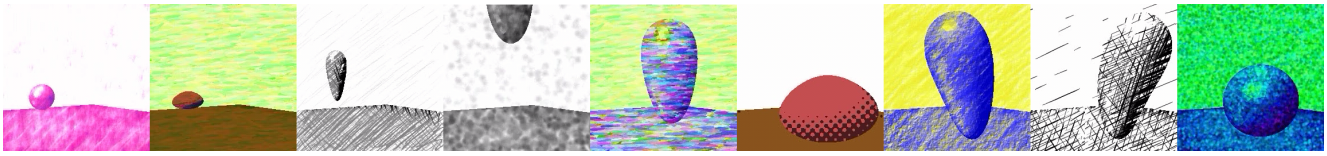


Figure 5: An animation played back with nine different image filters for artistic effect.

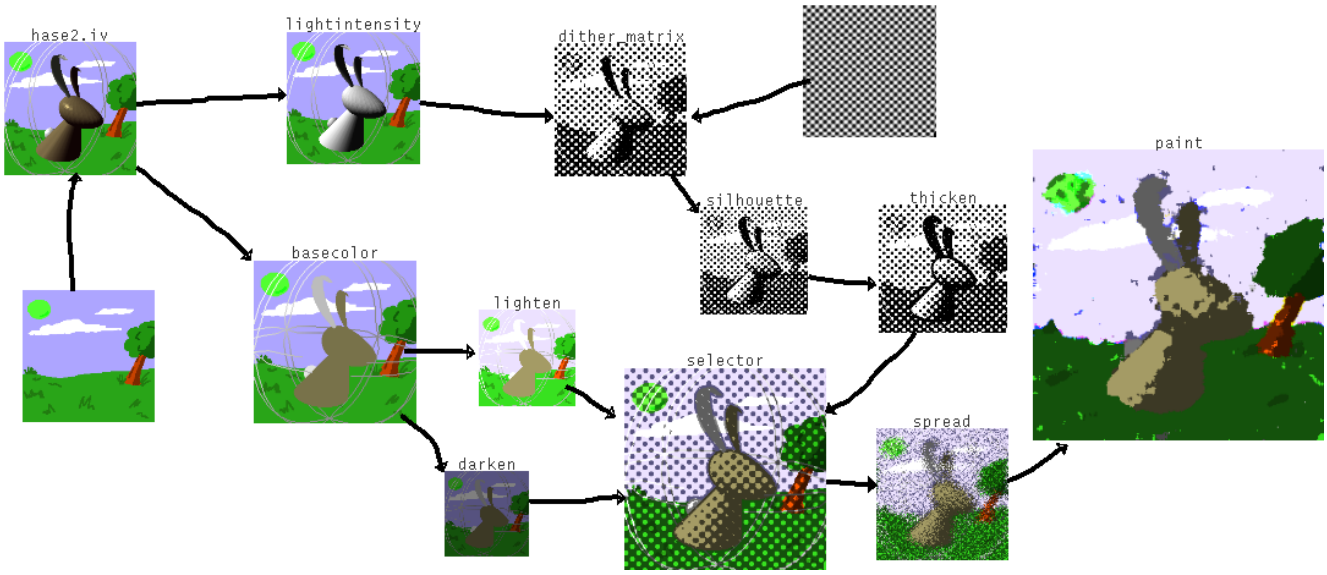


Figure 6: Producing complex effects from modifiers that manipulate modules in a rendering pipeline.