Supportive Presentation for Computer Games

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch der Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von	MEng Nicolas Halper
geb. am	22. 03. 1978 in Paris, France
Gutachter:	Prof. Dr. Thomas Strothotte
	Prof. Dr. David Duce
	Prof. Dr. Steven Feiner

Magdeburg den 17. Oktober 2003

Abstract

Optimizing computer game play necessitates an understanding of specific game design goals and general presentation strategies, wherein game play can be both challenging and fun. Although only rarely utilized in this context, the ever-advancing field of computer graphics, and particularly non-photorealistic rendering (NPR), hold untapped potential in optimizing game play. We introduce the concept of *supportive presentation*, wherein graphical presentation planning aims to maximize player enjoyment. To this end, we provide (1) tools and empirical psychological evidence utilizing NPR-specific presentation methods; (2) tools for advanced camera control; and (3) a system for integrating key aspects of game design and development for supportive presentation. Lastly, drawing on examples of supportive presentation in potential game applications, we outline directions for future research between graphics and psychology to promote a better understanding of the influence of presentation methods.

Zusammenfassung

Um die Interaktion in Computerspielen verbessern zu können, ist ein tieferes Verständnis der spezifischen Ziele bei der Erstellung eines Spiels und der dabei angewandten allgemeinen Präsentationsstrategien notwendig. Diese ermöglichen es erst, dass das Spielen eines Computerspiels sowohl eine Herausforderung darstellt als auch dem Spieler Freude bereitet. Obwohl beide Forschungsgebiete in diesem Zusammenhang nur selten Verwendung finden, stellen das ständig weiter voranschreitende Gebiet der Computergraphik und insbesondere das Gebiet des nicht-photorealistischen Renderings (NPR) noch nicht genutztes Potential zur Optimierung des Spielgeschehens zur Verfügung. Die vorliegende Arbeit stellt das Konzept der unterstützenden Präsentation vor, welches die Maximierung des Spielspasses durch graphische Präsentationsplanung beinhaltet. In diesem Zusammenhang werden (1) Werkzeuge zur Verwendung von NPR-Methoden entwickelt und dazu empirische psychologische Untersuchungen präsentiert, (2) Methoden für eine fortgeschrittliche Kamerasteuerung entworfen und (3) ein System vorgestellt, das die Integration von Schlüsselaspekten der Spieleentwicklung für eine unterstützende Präsentation integriert. Abschliessend werden, basierend auf Beispielen für die unterstützende Präsentation in potentiellen Computerspiel-Anwendungen, Möglichkeiten aufgezeigt für die weiterführende Forschung auf dem inderdisziplinären Gebiet zwischen Computergraphik und Psychologie, um ein besseres Verständnis des Einflusses der Präsentationsmethoden zu ermöglichen.

Acknowledgements

The work and parts of this thesis were developed in collaboration with the following people: Mara Mellin, David Duke, Stefan Schlechtweg, Patrick Olivier, Ralf Helbing, Tobias Isenberg, Maic Masuch, Nico Flohr, Phil Barnard, Volker Linneweber, Christoph Herrmann, and Thomas Strothotte. Parts of this thesis are published as: [Olivier et al., 1999; Halper and Olivier, 2000; Halper et al., 2001, 2002; Isenberg et al., 2002; Halper and Masuch, 2003; Halper et al., 2003c; Duke et al., 2003; Halper et al., 2003b,a]

I am particularly grateful to Thomas Strothotte, who let me explore and be free in my work, and to Petra Specht, whose general help, support, and organizational skills are second to none.

A final and additional word of thanks to Mara Mellin, whose critical analysis and editing skills for parts of this thesis enabled me to take this work up another level. She also lent me her artistic talents to assist in the production of many wonderful images: Figures 2.9(a), 2.10(b), 2.11(b), 2.12(a), 2.13, 5.1, 5.3, 5.4, and 5.7.

Contents

1 Introduction

Fundamentally, every good game maintains certain basic elements of play that make it distinctly compelling—uniquely combining enjoyable challenges, clear goals, specific but broad strategies for success, and the opportunity to socialize with other players or simply escape from reality altogether. The understanding and effective application of a successful game design, much like the understanding of art, is an internalized process—typically relying on a 'gut' feeling [Rouse III, 2003] rather than a formal combination of the above elements with other design strategies. In order to maximize the player's enjoyment of the game, the designer must 'feel' and bring out those elements that makes it both absorbing and fun.

As computer and video games are now firmly established within the modern gaming culture, game design and development has taken on a slick, dynamic, and increasingly realistic face. Technologically, computer games have drastically advanced over the past decades as is clearly seen in the astonishing progress of real-time photorealism (PR). With the establishment of 3D graphics, gamers are playing a new kind of game—one that more closely reflects the physical capabilities and limitations of our own senses. In a world where game developers tend to invest a majority of their resources in advancing current rendering algorithms it seems counter-intuitive that these algorithms rarely contribute to actually optimizing the elements of play. Thus the question emerges: aside from achieving PR, how *else* can computer graphics be employed within game design?

In this thesis, we explore alternative presentation strategies and tools that allow game designers to more closely correlate game design intentions with visual variables in order to augment individual game experiences. Specifically, we formally introduce the concept of *supportive presentation*, wherein graphics can be employed in direct support of game design and respective elements of play:

- *Supportive Presentation:* A graphical presentation specifically tailored to augment individual user experiences by supporting the design intentions underlying the immersive virtual environment.

Formally, the task of augmenting 'individual user experiences' necessitates a deeper understanding of the user on multiple levels—specifically, a concise knowledge of the parameters and limitations within user-game interactions. Although 'rules-of-thumb' can and do contribute to designing enjoyable games, they frequently overlook certain variables or erroneously ascribe the assumed boundaries of a non-formal paradigm. Consequently, an empirical approach—one that explores the psychological parameters of the user-game interactions in terms of graphics, is clearly necessary to effectively implement supportive presentation strategies within a game. In essence, supportive presentation must go beyond creating and manipulating computer-generated images to formally probe the psychological potential therein.

1.1 Graphical Presentation Planning

Clearly, the premise of understanding user-game interactions is to enable the game design process to be better tailored to the goals of supportive presentation via the informed development of new tools and presentation strategies. Moreover, in order for supportive presentation to be effective, the communicative goals within the game must first be satisfied, such that the player successfully absorbs and responds to the *meaning* of a graphical presentation within the framework of the game objectives. A central tenet of graphical presentation planning is what can be conveyed to the viewer, or more precisely, what might or might not be understood by different viewers. Indeed, Seligmann [1993] argues that communication within computer graphics can be problematic—even though graphics can generate the exact same presentation to different viewers, they may be interpreted by individuals in different ways. Thus, we outline a strategy which reflects aspects of Seligmann's presentation pipeline, while considering how supportive presentation can be best integrated into this model.

(1) **What is the communicative goal?**: To begin, the game designer must define what the graphical presentation should *accomplish*, with the intentions of supportive presentation in mind. Depending on the game, communicative goals will be framed within a conceptual premise. At this stage, the actual styles and images are not being considered, but rather the game intentions—such as clarifying an aspect of the game, conveying object or character values, or influencing the next move of the player. For example, multi-player games will consider such things as how best to level the playing field between players of different skill levels, while exploratory games might attempt to better coordinate player performance with successive challenges in completing future goals.

(2) **Defining the presentation strategy**: The game designer must now determine the visual effect(s) that can effectively satisfy the communicative goals. Essentially, specific presentation strategies must be defined—for example, *what* visual cues and effect can best clarify a given part of the game, influence player decisions, or convey object or character value. An effective strategy might emphasize particular objects, or map visual attributes to characters/objects.

(3) **Defining the presentation method**: Given the presentation strategy, the game designer now specifies *how* the visual effect can be realized. For example, a strategy that uses emphasis may be achieved by requiring the scene element concerned to have a prominent size and location in the image, or by using a particular rendering style to distinguish it from surrounding objects or give it particular attributes.

(4) **Image generation**: Based upon the selection of the presentation methods, the specified visual properties can now be applied. This process may involve optimizing the camera position or rendering of various styles.

It is important to realize that although this pipeline is a conception of how to achieve effective communication between game designers, developers, and users, these stages do not necessarily encompass what is currently possible in the game-design process. Specifically, the latter three stages rely on research considerations of user-game interactions to establish effective communication strategies. We must first formally explore, define, and establish the parameters and limitation of player responses to given graphics variables within the game environment before presentation strategies and methods can be effectively employed to achieve a set of communicative objectives. Finally, the first two stages are somewhat confined to the functionality of the tools used to specify presentation methods, which in turn are dependent on algorithms to generate images.

1.2 Objectives

In order to begin providing the necessary concepts and tools for supportive presentation, we must:

- *Formally understand the impact graphics exerts on the viewer in order to effectively support game design.*

  Computer graphics, and non-photorealistic rendering (NPR) in particular, can be used to influence more than just ‘cool graphics,’ holding the potential to uniquely impact game design and subsequent play as never before. This potential is not only defined by the boundaries of technological advancement, but demands a deeper understanding of how computer graphics can influence the viewer. Formalizing psychological knowledge via new research paradigms within computer graphics will promote general awareness of alternative, non-psycho-physical visual influences and thus, increase innovative methods for supporting game design.

- *Allow designers to focus on illustrative intent by visually specifying effects without requiring an in-depth knowledge of the implementation processes.*

  The conceptual process of specifying *what* effects will best convey communicative intent should not necessitate a technical understanding of *how* these effects are achieved. Freeing the designer to focus on effects themselves (and not how to generate them) will facilitate the game design process creatively—opening new visual avenues within supportive game designs.

- *Design and streamline algorithms within a system model which supports a variety of visual specifications and effects, which can then be employed as supportive presentation methods.*

  Algorithms designed with only one specific presentation method in mind remain limited in their scope and future applicability. Combining existing algorithms within a system model that enables easy application is crucial to ensure powerful and flexible systems capable of supporting and combining various presentation methods.

We focus on two central types of presentation methods—*visual content* and *visual appearance*, as both can be applied to more closely correlate game design intentions with visual effects. Specifically, we explore aspects of how the camera captures presentation content, and how rendering variables define the visual appearance of that content.

1.3 Results and Thesis Structure

Below, we present a brief overview of our tools and concepts that significantly contribute to supportive presentation applications in the areas of non-photorealistic rendering (NPR), camera techniques, user interfaces, and action summarization techniques:

- **Empirical psychological evidence for utilizing NPR-specific presentation methods.** NPR offers opportunities beyond photorealism to carry significant implicit information that affects choice and judgment, which we empirically demonstrate to convey levels of threat, character strength, and potential for interaction in terms of navigation, exploration, and object availability.

- **Tools for the creation of NPR styles at all user levels.** We present a modular NPR system based on a first unifying framework for NPR which includes a novel interface for designers to experiment on presentation methods by *creating new rendering styles* without knowledge of dimensionality of data, ordering of modules, or type conversion requirements.

- **Tools for advanced camera control.** Real-time frame-coherent techniques for a virtual camera are introduced whose behavior can be specified from a shot property taxonomy and complemented with methods for automatic feature extraction returning timing information of important events in games.

- **A proposed integrative system that focuses key aspects of game design and development within a basic supportive presentation framework.** A system integrating the above tools is presented with examples for supportive presentation in potential game applications for which we outline required future interdisciplinary research in graphics and psychology to further explore and evaluate the influence of presentation methods.

The following sections detail the thesis contents in terms of individual chapter objectives and how these respective goals are achieved.

Chapter 2: Supportive Rendering

Current presentation approaches in the game industry offer few means of supporting the player, thus we aim to demonstrate the potential of alternative presentation methods to visually influence player response via rendering styles hypothesized to access higher-level cognitive perceptions.

Trends in the game industry are analyzed that reveal evolving problems in game designs, for which we subsequently propose alternative solutions and test 12 specific hypotheses on 156 subjects to probe the effects of *non-photorealistic* rendering styles in influencing human judgment and interactions. We interpret our results and discuss the merits of using these techniques within computer games in terms of supportive presentation.

Chapter 3: Non-Photorealistic Rendering Tools

Our objective is to first demonstrate the necessity for an effective rendering system that allows artists, as well as programmers, to create new visual effects without technical knowledge of how the effects are generated, and secondly, describe the results and applicability of a unifying framework created to this end.

We design a unifying framework for NPR and specify how users of all knowledge levels can interact to create effects. A specialized NPR system, OPENNPAR, is built on this framework that is comprised of modules that can be fitted arbitrarily into a rendering pipeline. We then observe how it can mimic a designer's creative process with a novel interaction method, which works to produce new visual styles. Lastly, we consider how these can contribute to computer game development and supportive presentation.

Chapter 4: Camera Control

The camera should provide a supportive role in capturing objects and associated events, with the goal of enabling the designer to precisely specify when and what content to capture in real-time during the game.

To specify visual content, we construct a low-level shot property taxonomy that is integrated into our camera system, CAMPLAN—demonstrating that consistent results for spatially similar scenes and solutions in unique regions can be found. The shot property taxonomy is subsequently refined and specialized for real-time application in the PREDATOR camera system, wherein novel algorithms for maintaining frame-coherence over time are introduced. Finally, methods are designed for automatically evaluating game variables to detect specific events, used to return time intervals for capturing content.

Chapter 5: Integration and Future Research

An integration of the above tools towards improving presentation methods is necessary to optimize existing supportive presentation variables. Additionally, the inherent flexibility of NPR and thus its psychological functionality, remain largely unexplored. Together, a theory of psychology within NPR and existing tools must emerge to be collectively applied to further supportive presentation within computer games.

A system diagram is constructed which integrates and highlights the use of each system and tool formerly described. Examples are then provided to illustrate the effective cohesion of this model via our graphical presentation pipeline. Directions for future research are discussed for each example, as supportive presentation in general necessitate a deeper understanding of the influence(s) computer graphics can exert on the viewer.

Chapter 6: Conclusion

If we can provide designers and artists the tools to experiment and create both a broader and deeper range of game designs and solutions, developers will have a better chance of creating games that continue to increase in their innovative capabilities.

We summarize the contributions of this thesis, comprising new and advanced presentation strategies and tools that incorporate a variety of new techniques. This chapter concludes by outlining additional future directions for supportive presentation and potential implications in designing new games.

2 Supportive Rendering

Game developers utilize each advance in graphics hardware to create products that are more innovative than previous releases: but such innovation is too often limited to 'visual realism'. In this chapter, we argue that non-photorealistic rendering (NPR) provides an opportunity to influence the game experience in novel ways. However, although the technical side of NPR has advanced significantly over the past decade [Strothotte and Schlechtweg, 2002; Gooch and Gooch, 2001], the understanding of how and why to apply this technology has lagged, and the ideas presented here represent a contribution to closing this gap, primarily: (1) the empirical testing of a number of hypotheses regarding the influence of rendering styles on user response, (2) an analysis of how NPR can be used to affect higher-level cognition, and (3) a discussion of the relevance and applicability of these results within computer games.

In Section 2.1 emerging factors in computer games are outlined in the context of photorealism and the need for supportive game designs that cater for different types of users. Section 2.2 introduces alternative rendering strategies and outlines an approach to build on communication via latent knowledge that will prove invaluable in supporting user assessments and interaction in virtual environments, for which the design of an empirical study is given in Section 2.3. Specific hypotheses and results are reported in Section 2.4, followed by an interpretation of these results in Section 2.5. Finally, a discussion of our achievements and its contribution to computer games, and computer graphics in general, is given in Section 2.6.

2.1 Emerging Factors in Computer Games

Game design has changed dramatically over the years. In the past, games were typically very repetitive due to tight technical limitations such as the amount of information one could fit into memory, processing power, and graphics resolution. Games were often organized around distinct levels that could be completed in a linear fashion, with the player beginning each gaming session by starting at the first level. Failure at a particular task or challenge usually required the player to revisit an earlier stage in the game. Today, however, technological advances allow for enriched game designs that have enabled a more varied playing experience. Rather than requiring a game to be a series of difficult interaction tasks that require dedication to improve skills and progress to the next stages of a game, the playing experience can be much more open. This progression has allowed the introduction of a different class of gamer, those normally not found down at the arcades. As a result, a number of game design issues have emerged that are discussed in the following subsections.

2.1.1 Introducing the Casual Gamer

Adams [2000] identifies two gamer type extremes: the casual and the core gamer. The casual gamer invests limited resources (time, or money!) on games, regarding them as the occasional short-lived entertainment and considers alternatives such as reading a book or watching television, whereas hardened core gamers devote much of their life to games. However, Adams also highlights a more important distinction between these types of gamer: *why* they play games and what they expect from them.

- Core gamers, who were the initial enthusiasts, are compared with athletes: their aim is to win, they approach games as a gruelling, exhausting, sometimes painful, challenge, and they will spend hours learning complex controls and experimenting with gameplay to achieve perfection.
- Casual players, on the other hand, are a more recent phenomenon. These players want to challenge their minds as much as their motor skills and are much less tolerant of frustration in order to achieve something.

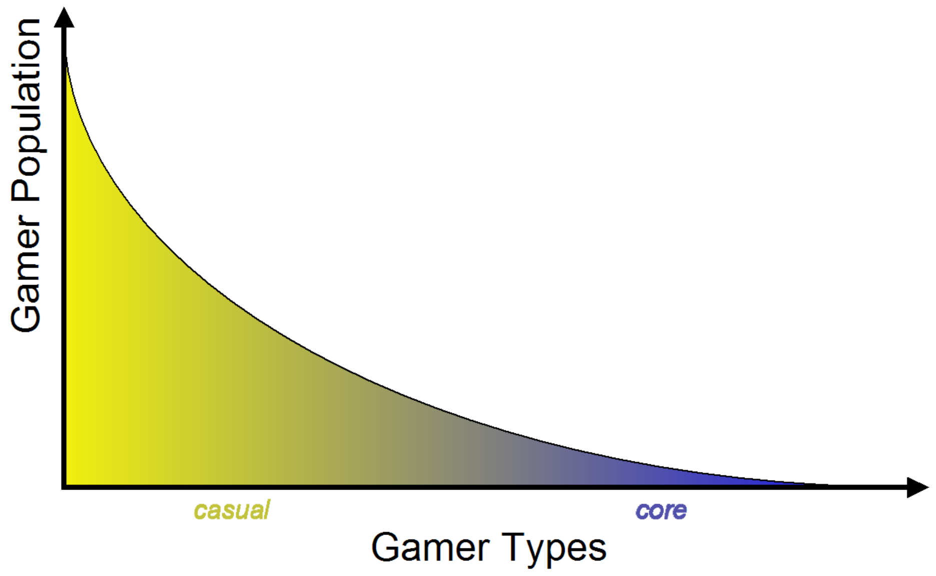


Figure 2.1: Speculated computer game player population distribution: From Casual to Core (adapted from [Ip and Adams, 2002]).

The importance of catering a game for both casual and core gamers cannot be understated. As can be observed in Figure 2.1, the expected distribution of gamers comprises mostly casual gamers [Ip and Adams, 2002]. A goal for the industry is to get these casual players more into the hardcore market to increase the regular game buying population.

2.1.2 Skill and Challenge

In order to reach the casual gamer, game design needs to involve challenges that are not just nasty, mean, and hard to beat, but clever, exciting, and fun to beat [Adams, 2000]. In addition, the casual and core player must be able to play the same game. For producing a successful title (i.e., those that sell well), Shelley [2001] emphasizes providing multiple gaming experiences that cater for different skill levels and challenges. As a result, core gamers will tell others about their favorite games that will get more casual players to buy them too. The casual players might not be able to compete with star players, but by adjusting parameters they can still find the type of game that suits them and have fun. In the end more people are able to buy a game and be happy with it with potential for becoming further loyal customers [Shelley, 2001].

To address the problem of meeting all players with a satisfying degree of challenge, a number of different strategies can be used to regulate the challenge level in a game to support both skilled and unskilled players [Pagulayan et al., 2003]. For instance, the use of rubber-banding[1], or providing the trailing player with tools that have certain advantages. Achieving this successfully is complicated even for single-player games, but becomes very difficult when pitting players of different skills against one another [Pagulayan et al., 2003]. The game designer must come up with ways to maintain challenge, reward, and progress for the unskilled player *without severely hampering the skilled player.*

2.1.3 Guided Interaction

With the advent of the casual gamer, game designers must create an experience that is much more open than the linear challenges of skill that have historically appealed to core gamers. Rouse III [2001] points out that the reason players often fail to finish games is that they become stuck at a particular juncture and thereby can no longer advance. For casual gamers, such hurdles mean that abandonment of play leads to abandonment of the game. Thus, the game designer is to make sure not to beat the player [Bates, 2002, Chapter 2].

This problem can be ameliorated by introducing a range of options that a player can take at any given time. However, introducing such non-linear gameplay presents new problems: players can get lost in the space of options, and then waste time worrying that they are doing the wrong thing. Some kind of reassurance is required, for example incremental rewards as the player progresses towards their goals. They should be gently guided to appropriate locations and challenges, yet also be informed when they are straying from the path [Bates, 2002, Chapter 2]. Or put another way, the designer's job is to take care of the (casual) player, not to be their hidden adversary. Guidance in navigation and exploration are especially

[1]regulating gaming parameters (e.g., maximum speed, acceleration) in favor of trailing players

important when learning how to play a game [Pagulayan et al., 2003]. Creating a suitable game experience in this manner involves consideration of a range of issues, for example narrative, task, and interaction. The concern here, however, is with rendering, and one account of how this relates to casual game play is given in the following analysis, by Bates:

> The single biggest problem of puzzles in action games is that players often don't know where they are. Sometimes a player will slay all the monsters on a level, yet still be left wandering around wondering why he's not done. After a while, he'll either give up in disgust or reluctantly reach for the strategy guide. Only then will he learn that the ledge that looked too narrow can actually be traversed, the jump he tried five times can really be made, or the wall he tried to climb in six places can be scaled from a seventh.
>
> This problem arises from the wealth of realistic graphical detail we can now put in our levels. If we can make the entire level look interesting and beautiful, how do we draw the player's attention to the spots that are actually important? [Bates, 2002, Chapter 5]

Bates suggests providing players with cues, for example: a trail of blood to a particular spot, a glimpse of an adversary making a jump from one ledge to another, or a blatant message over the radio that tells the player where to go and what to do. In certain game genres (e.g., adventures), clues can be given in appropriate locations. For example, a player who is unable to unlock the secret of a room, relevant objects could be highlighted, possibly in a sequence leading to a solution. Another option, suggested by Falstein, is to increase the chance of a player progressing within a game by rewarding task completion with information or material that then supports subsequent tasks [Falstein, 1999].

However neither of these options address the source of the problem, that the uniform level of realism achieved by photorealistic rendering provides little scope for conveying cues about the game situation. In fact, the uniformly photorealistic appearance of objects in virtual environments can often confuse novice or casual users as to which objects are interactive and which are merely 'decoration'. In addition, certain successful game formulas twenty years ago will no longer appeal as casual players often expect real-world logic to be applied to photorealistic environments—they are not motivated, as still required by many modern games, to blow up all wooden crates on a level with rocket launchers and expect to find contained power-ups, such as medikits, that survive the blast.

The way that games are presented should adapt to the evolution of the casual gamers: game graphics need to go beyond images that are merely 'realistic' and engrossing, to images that carry cues and hints that will support and enhance the experience of the casual gamer.

2.1.4 Differentiation and Innovation

Despite certain issues raised by photorealism, with every advent of new technology emerging on the market, there is a persistency for computer game developers to rush in games that attempt to sell on advances in visual effects. This inspired Adams in 2001 to challenge designers to stop relying on these advances in game technology to push games, arguing that too much time was spent on getting used to technological hardware and not enough on game design [Adams, 2001].

This situation is somewhat ironic. One of the biggest imperatives for game design is to differentiate and innovate [Shelley, 2001], such that each game quickly grabs the player's attention and is noticeably different than the last otherwise the player risks losing interest [Pagulayan et al., 2003]. Those games that become relegated to clones because of a lack of distinction are usually commercial failures [Shelley, 2001]. Thus, it seems odd that most games differentiate themselves by converging to the goal of visual realism instead of employing different visual means that actually might be more suitable to the game design itself. Hecker regards that:

> "The game industry is not experimental enough...games cost a lot of money, and publishers want some chance of making that money back, so that leads to conservative and incremental designs...that's not healthy for the medium in the long term...[In other art forms] there are built-in mechanisms for experimentation and exploiting experimentation...Games, as an art form, have hardly scratched the surface of their potential, but [the game industry has] already calcified" (Chris Hecker in [Adams, 2002])

One of the goals of this thesis is to provide tools that allow the creation and application of presentation styles that contribute towards innovative game designs. Thus, rather than using graphics as a medium within itself, we aim to use graphics in a way that supports the intentions of the game design.

2.2 Alternative Solutions in Game Design

The previous section outlined some considerations in game design, but also identified that the market is open to innovation and experimentation—especially one that deviates from expenses of implementing advances in photorealistic rendering. This section proposes the use of alternative rendering styles to convey information in the game beyond the simple object-level model conveyed through photorealism.

The alternative to photorealistic rendering, is *non*-photorealistic rendering (NPR), which continues to provide new tools and techniques for both communication and the experience of synthesized worlds, including aspects of shape and detail in technical and artistic illustration [Strothotte and Schlechtweg, 2002; Gooch and Gooch, 2001]. While many of these techniques have been applied in real-time [Lake et al., 2000; Raskar, 2001; Freudenberg et al., 2001, 2002; Praun et al., 2001] and can be used in computer games, they do not currently contribute more than just a new look. NPR has been known to provide communicative possibilities, and, coupled with its inherent flexibility, it could support many communicative intents behind game design. However, despite this potential in NPR, Section 2.2.1 describes how concrete investigations into NPR research have so far not gone beyond perceptual principles, with only a few works suggesting an actual influence on user responses. Two works in particular, and motivation derived from empirical studies in psychology, suggest that there are qualities of images, independent of overt structure, that influence interpretation and judgement. The theory behind this is described briefly in Section 2.2.2, which opens the possibility of using NPR to communicate at a *non*-structural level, for which we lay out specific questions in Section 2.2.3 to drive an investigation of its potential within computer games.

2.2.1 Communication via Non-Photorealistic Rendering

While algorithmic issues in NPR are of concern (e.g., see [Strothotte and Schlechtweg, 2002]), more pertinent are questions pertaining to the use of such images. The bottom line is the question, which image, or more generally, what kind of non-photorealistic rendition, is appropriate to use in what context.

As early as 1956, a study [Ryan and Schwarz, 1956] showed that cartoons, by exaggerating relevant cues and reducing unnecessary detail, are identified more rapidly than realistic line drawings. More recent work has examined perceptual optimization [Reddy, 1997, 2001], space perception [Gooch and Willemsen, 2002], and the use of line direction [Girshick et al., 2000], texture [Interrante, 1996; Interrante et al., 1997; Interrante, 1997], and non-realistic lighting [Hamel, 2000] to convey surface information. These studies build largely on the established link between rendering and psychophysics, which can be traced back to questions of perceptual acuity and discrimination arising from progress towards photorealism (PR). NPR research also draws on wider sources of inspiration, such as comic art [McCloud, 1994], and analyses of the perceptual merits of artistic techniques [Durand et al., 2002; Laidlaw et al., 1999] to understand the utility of rendering styles in communicating shape or multi-dimensional data.

Communication, however, is about more than just perception. For instance, it has also been shown that variations in sketch rendering parameters can be used to draw attention to particular parts of a scene [Strothotte et al., 1994]. However, while NPR distinguishes itself from PR by *selectively* applying the rules of visual perception, it has yet to begin exploiting the interplay between rendering and other cognitive processes. For example, sketch rendered images of architectural designs—in contrast with PR designs, result in qualitatively different (and improved) dialogue between architects and clients [Schumann et al., 1996]. Sketchiness seems to convey a sense of openness to change and modification. This effect is not just a matter of perception, but also involves levels of cognition connected with judgement and articulation. In psychological terms, the image has an affective content, separate from its structural detail.

The motivation for a further investigation into the potential influence of NPR stems from an early investigation of radar display design which established that affective information (in this case, the sense of 'friend' versus 'foe') can be conveyed quickly through qualities

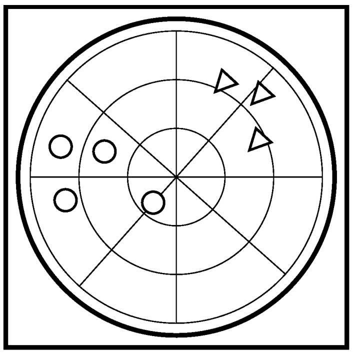


Figure 2.2: Radar display (after Provins, [1957])

of shapes [Provins et al., 1957]. When asked to glance at the display (see Figure 2.2) and immediately identify which group (circles or triangles) of aircraft are the hostile forces, nearly all subjects identify the triangular symbols. Since there is no overt information in the scene that triangles are hostile, this is unlikely to be a reasoned response. The following section argues that instead, such responses utilize latent knowledge.

2.2.2 Latent Implicational Knowledge

It is well known that the brain integrates internal and external stimuli both within and across modalities. Barnard and May [1995] argue that this integration is largely governed by latent[2], rather than overt knowledge to create a coherent understanding of the world. In Provins' radar display (Figure 2.2) the sharp shapes invoke a connotation of threat. In evolutionary terms, rapid reaction to sharp shapes (e.g., in the form of fangs) may have conveyed a survival advantage, and become encoded into the brain. Another relevant example of such integration is captured by Davis [1961] in the following experiment:

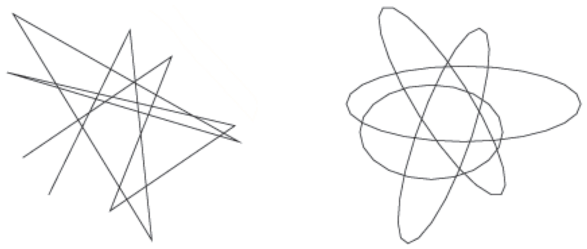


Figure 2.3: Two connotative shapes (after Davis [1961])

When associating the shapes in Figure 2.3 with the words 'Ulloomo' and 'Takete', most people associate Ulloomo with the image on the right and Takete with the image on the left.

[2]Latent knowledge is effectively 'hard-wired' into the brain and utilized without conscious awareness

If a subject is then asked to explain the correspondence between these sounds and shapes, they experience difficulty articulating *why* these qualities match— a response characteristic of latent knowledge. Further, response to latent invariants is extremely rapid, as shown by Guthrie and Weiner [1966]. In their experiment subjects were asked to make character judgements based on an outline image of a seated figure. The subjects were shown an extremely brief, subliminal image of a complete outline, followed, for a longer duration, by an outline with missing segments. The study showed that the figure was judged as more hostile and aggressive when the subliminal image contained angular features, as opposed to curved features. Interactive Cognitive Subsystems (ICS) is a model of human information processing which accounts for this and similar phenomena [Barnard and May, 1995, 2000]. Although the experimental results and analysis given in subsequent sections can be understood without recourse to this theory, a short account of the ICS model provides insight into our approach.

ICS is part of a theoretical movement within cognitive psychology that views human information processing as behavior emerging from the coordinated operation of a highly-parallel, modular architecture (see also work on EPIC [Kieras et al., 1999]). Our interest in the model stems from its distinction between the latent processing of *propositional* and *implicational knowledge*. Propositional knowledge is concerned with referential information or 'facts about the world,' which, in part, processes basic structural information derived from raw visual input. In the case of a computer game, propositional information might be the fact that another player is standing by a door, or that the door is opening. In contrast, implicational knowledge utilizes schematic models to capture the affective content: in gameplay, this might be associating passing through the door with potential danger or safety. Although implicational knowledge is easily abstracted from propositional information, it can also be extracted directly from visual stimuli.

2.2.3 Non-Photorealistic Rendering as a Presentation Strategy

Thus, the focus of this chapter is whether, through NPR, we can manipulate the implicational content of visual input to influence user response. For this first investigation, we refine the question into four specific issues:

- To what extent can we generalize and extend Provins' radar display research to influence latent responses to other images? That is, can more complex images be rendered to effectively convey implicational information regarding object and character attributes, such as danger or strength?

- If sketched images are capable of drawing attention to specific areas, and 'sketchiness' can convey a sense of un-definedness, is it possible to influence latent navigation and exploration tendencies by controlling rendering parameters such as level-of-detail (LOD) and silhouette strength?

- In cartoons, characters perform acts which completely violate normal expectations derived from real-world experience, yet viewers watch such features without a sense of disbelief that might be engaged if realistic characters and scenery were present. In this respect, can NPR be used to convey an appropriate sense of what is possible in an environment?
- Complex environments often comprise large numbers of objects that appear useful, but few of which actually have behavior attached. In realistic rendering, this is problematic: if all of a scene *appears* realistic, should not all of the scene *behave* realistically? In contrast, can the flexibility of NPR styles be used to intuitively distinguish objects that are available for interaction from objects that serve solely as environment detail?

From these questions, we derive 12 specific hypotheses (in Section 2.4) to probe the effects of various rendering styles in influencing human judgment and interactions. In summary, *we hypothesize that it is possible to effectively render images to convey implicational information regarding danger, strength, and interactive potential in terms of exploration, navigation, and object availability*.

2.3 Experimental Design

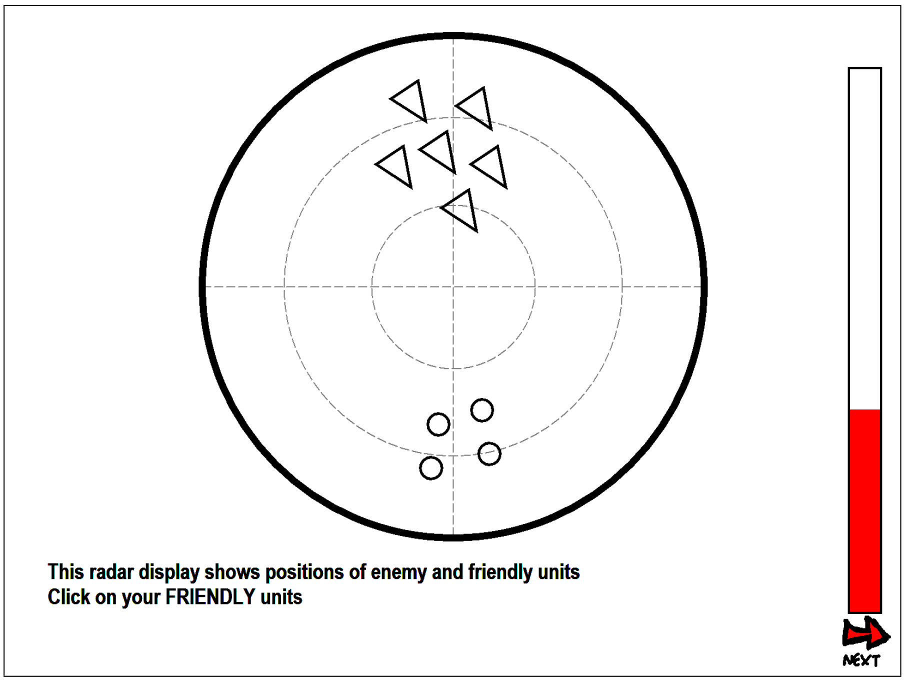
Subjects are tested on two PC laptops with 15” 1600x1200 displays. The overall experiment[3] comprises 12 sequential tests and a post-test questionnaire regarding age, gender, nationality, experience with games, and attitudes toward NPR. Tests are ordered to minimize any undesired priming effects. All test questions are scrolled across the bottom of the screen from left to right at a readable rate (50ms/character). An automatic timer records all subjects accordingly (timing onset: text stops scrolling; offset: subject makes final selection). For each test, a thermometer-style timer is shown on-screen, imposing a 30 second time limit (see Figure 2.4). This rapid-response imperative is intended to increase subject reliance on latent knowledge and reduce propositional analysis of the task and image which may introduce confounding factors such as personal preference or analysis of demand characteristics[4].

Subjects are tested in a sterile environment alone or with one other subject, both are unable to communicate or see each other’s responses. Testers are unaware of assigned test conditions for all subjects, as conditions are automatically generated on each computer using a pseudo-random algorithm. Prior to the actual testing, all subjects are briefed (verbally and on-screen) regarding the format of the test and given two initial un-scored practice tests to familiarize them with testing procedures. Testers are not present during testing, but remain easily available for questioning.

[3]This work was done in collaboration with Mara Mellin and David Duke and will appear as [Halper et al., 2003b]. Selected results can be found in [Halper et al., 2003c].

[4]Any cues indicating the test’s purpose.

One-hundred-and-fifty-six subjects (108 male, 48 female) of 29 different nationalities were tested; none were previously informed of the experimental purpose. Approximately 71% of subjects took the test in their first language (L1) and 29% in a second language (L2). Over 93% of subjects have some affiliation with the University of Magdeburg. Subjects age ranges from 15 to 58, with an average of 26, and a mode of 22. The SPSS statistical package is used to conduct ANOVAs and chi-square tests on all ordinal/scalar and nominal data respectively. The program and images used for the experiment are available on the accompanying CDROM.



This radar display shows positions of enemy and friendly units
Click on your FRIENDLY units

Figure 2.4: Screenshot of the radar test. *Note:* the ‘next’ button (bottom-right) appears after a selection has been made.

2.4 Experiments

The following tests are grouped according to type: (1) *Assessments of danger and safety*, (2) *Assessment of strength and weakness*, (3) *NPR-world vs. PR-world expectations*, and (4) *Goal-directed interactions*. The experimental test questions are presented in their corresponding figure caption. A summary of individual tests[5], hypotheses and relevant results are reported.

[5]The ordering of tests as presented do not necessarily reflect that of the experiment.

Statistical significance is reported when p<.05. The mean response time (MRT) is a secondary measure of performance, defined via individual test times per subject using 5%-trimmed means to reduce effects of extreme outliers. In order to retain statistical validity, MRTs are only reported for L1 unless otherwise stated, as the overall MRTs for L1 (5.3s) and L2 (8.0s) are significantly different (p<.001). In order to extract confounding factors for language comprehension, L2 data is excluded when the following conditions are satisfied: (1) the MRTs of L2 significantly differ from L1, (2) L2 responses differ notably[6] from L1, and (3) L2 responses alone are not significant.

2.4.1 Assessments of Danger and Safety

Our initial test is designed to verify the results from Provinset al. (1957), wherein subjects assess the affective content of simple shapes. The subsequent two tests expand on the logic of the first: objects rendered using sharp-edged or jagged lines may connote threat to the viewer both in terms of danger and safety.

Assessment of Friend vs. Foe

Hypothesis: Subjects shown a radar image depicting circles and triangles (Figure 2.4) and asked to identify their friendly troops, will select circles over triangles.

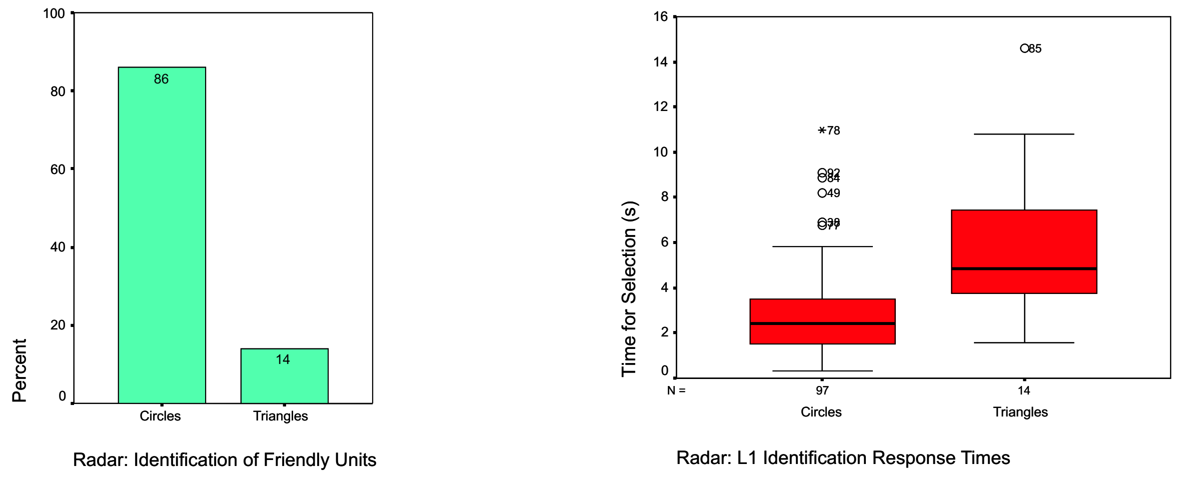


Figure 2.5: (left) A majority of subjects select circles over triangles (χ^2(1,156)=80.41, p<.001). (right) Boxplot of L1 response times for circle and triangle selection. MRTs differ significantly for circle (2.8s) and triangle (5.7s) selection (F(1,109)=21.48), p<.001).

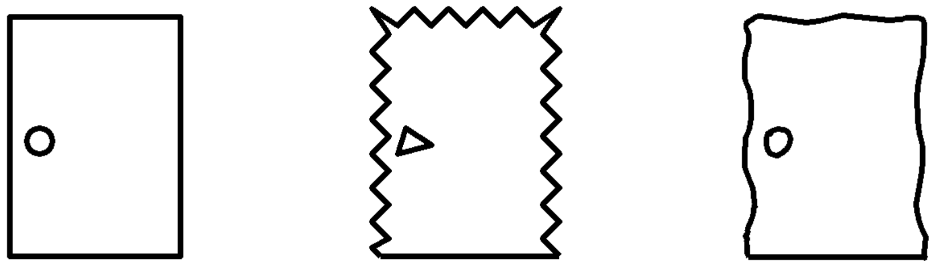
Results: Figure 2.5 shows that a majority of subjects select circles with significantly quicker response times than those who select triangles.

[6]the term ‘notable’ is used when sample sizes are too small to report a potentially significant effect.

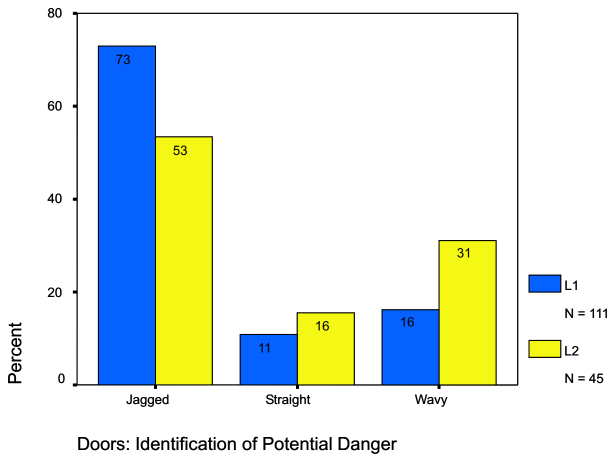
Assessment of Danger

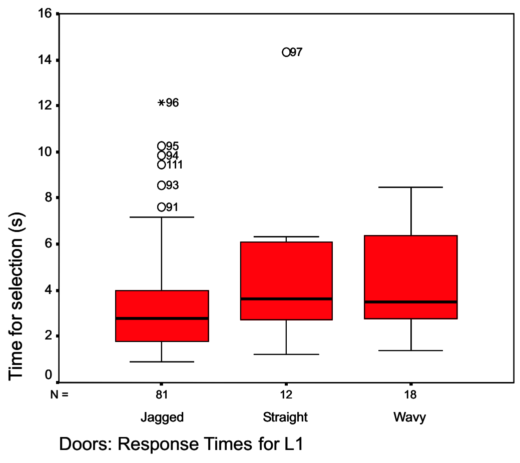
We then attempt to render ordinary objects as threatening and therefore dangerous. Subjects are shown three randomly placed doors rendered with simple, wavy, and jagged lines (Figure 2.6(a)). The 'wavy door' controls for the tendency to respond to objects because they look different from the 'normal' image.

Hypothesis: When asked to select a door containing danger, subjects will select the jagged door over the simple or wavy door.



(a) "Behind one of these doors lies danger. Which one?"





(b) (left) Language groups differ significantly (p=.017): 73% of L1 ($\chi^2(2,111)=78.97$, $p<.001$) and 53% of L2 ($\chi^2(2,45)=9.73, p=.009$) choose the jagged door. (right) Boxplots of L1 response times. The MRT for the jagged door (3.3s) is a notable 20% lower than the MRT for other door-selections (4.2s).

Figure 2.6: Door experiment

Results: A majority of subjects (67%) select the jagged door, while 21% select the wavy door, and 12% select the simple door ($\chi^2(2,156)=82.65$, $p<.001$). Figure 2.6(b) shows both a significantly increased ratio for L1 choice relative to L2, and a lower L1 response time for jagged door selection over other door selections.

Assessment of Safety

If rendering style influences assessments of danger, it may be possible to convey a sense of danger, even in objects typically regarded as safe. All subjects are shown one of three varied images, depicting a house and trees and asked to identify the safest location (Figure 2.7).
Hypothesis 1: subjects shown a simple-house and simple-trees, or a simple-house and jagged trees, will select the house.
Hypothesis 2: When shown a jagged house and simple-trees, subjects will instead select the trees as the safest location.
Results: Figure 2.7(b) shows that most subjects select the simple objects as hypothesized, and significantly faster than for other responses. Moreover, the MRTs differ significantly between genders, with females (2.7s) responding 30% faster than males (3.8s) (F(1,109)=8.48, p=.004).

2.4.2 Assessments of Strength and Weakness

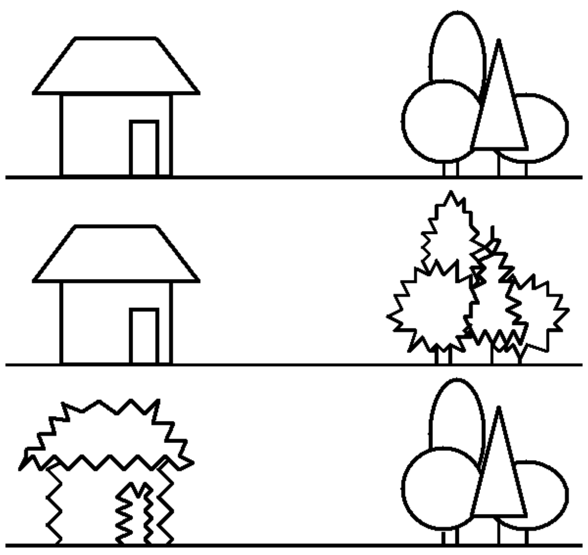
We test the effect of varying silhouette thickness and solidness on assessments of weakness and strength for both basic and complex images. Specifically, a light, broken silhouette might convey a sense of weakness, and thick, solid silhouettes might appear stronger.

Assessment of Weakness

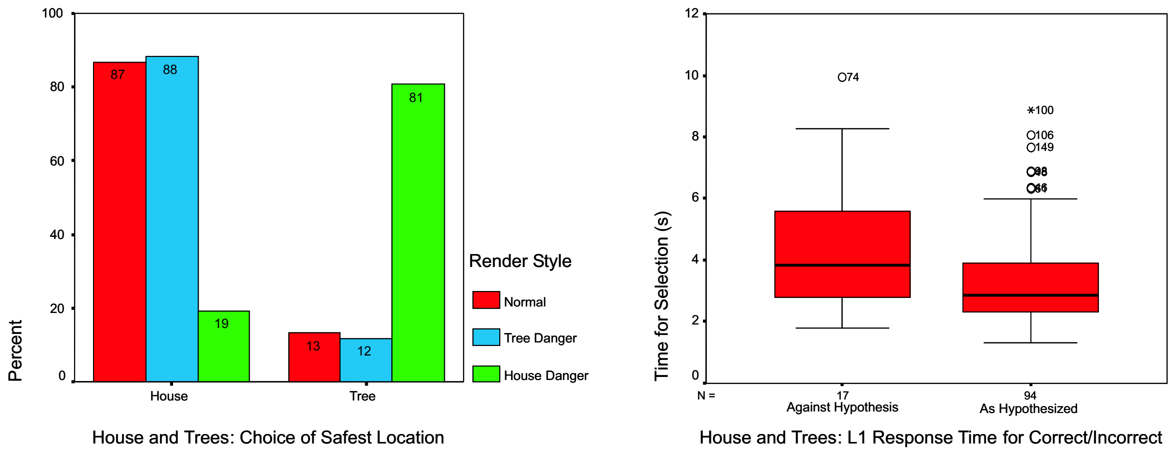
Within the experiment, this test follows the radar test from Section 4.1.1., in order to familiarize subjects with the radar units. Subjects are shown groups of circles and triangles rendered in thick and dotted lines on a radar image, (Figure 2.8(a)). For this test, triangles are not selectable.
Hypothesis: When asked to identify their weakest units, subjects will select the dotted circles over the thick circles.
Results: Subjects (88%) tend to select the dotted circles (χ^2(1,156)=92.31, p$<$.001). If subjects who chose triangles as friendly from the previous test are then excluded: 92% of the circle-consistent subjects respond as hypothesized (χ^2(1,134)=96.99, p$<$.001). In addition, Figure 2.8(b)) shows a higher ratio of selection for L1 relative to L2, with subjects responding about 35% faster for dotted circle selections.

Assessment of Character Strength

To extend assessments of weakness and strength to images of higher complexity, subjects are shown three geometrically-identical men, randomly ordered, and drawn with varied silhouette strength and solidity: thick and angular, light and broken, and a control image with simple lines (Figure 2.9(a)).

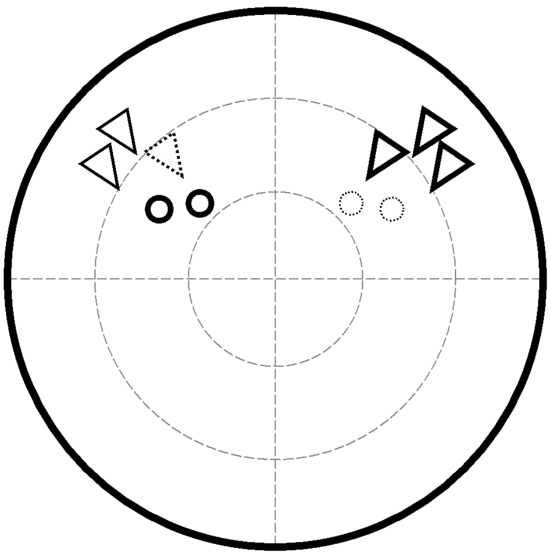


(a) “Click on the safest location”

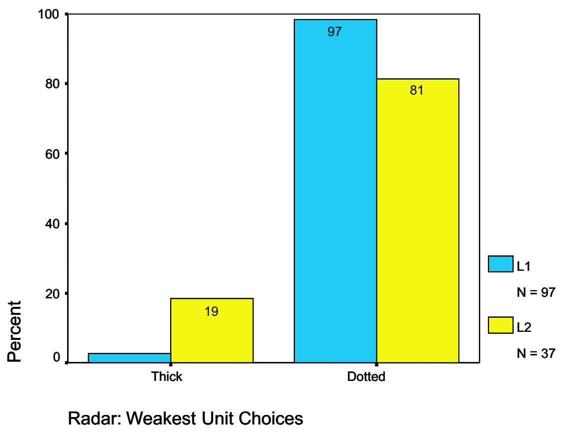


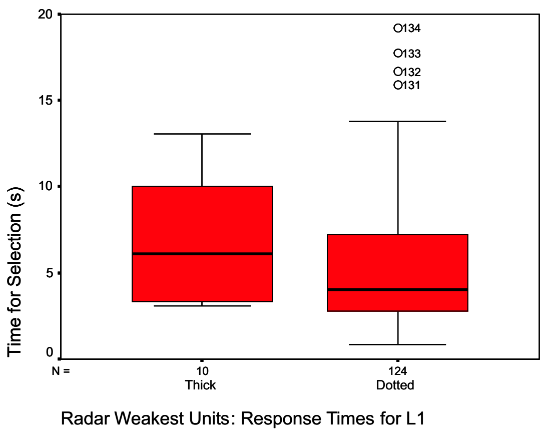
(b) (left) Given the simple-house and trees, most subjects select the house (χ^2(1,53)=28.70, p<.001). Similarly, subjects shown the simple-house and jagged trees tend to select the house (χ^2(1,51)=29.82, p<.001). Lastly, most subjects select the trees over the jagged house (χ^2(1,52)=19.69, p<.001). (right) Boxplot of L1 response times. The MRTs for hypothesized responses (3.1s) are lower than for other responses (4.4s) (F(1,109)=7.41, p=.008).

Figure 2.7: Using rendering style to influence assessment of safety



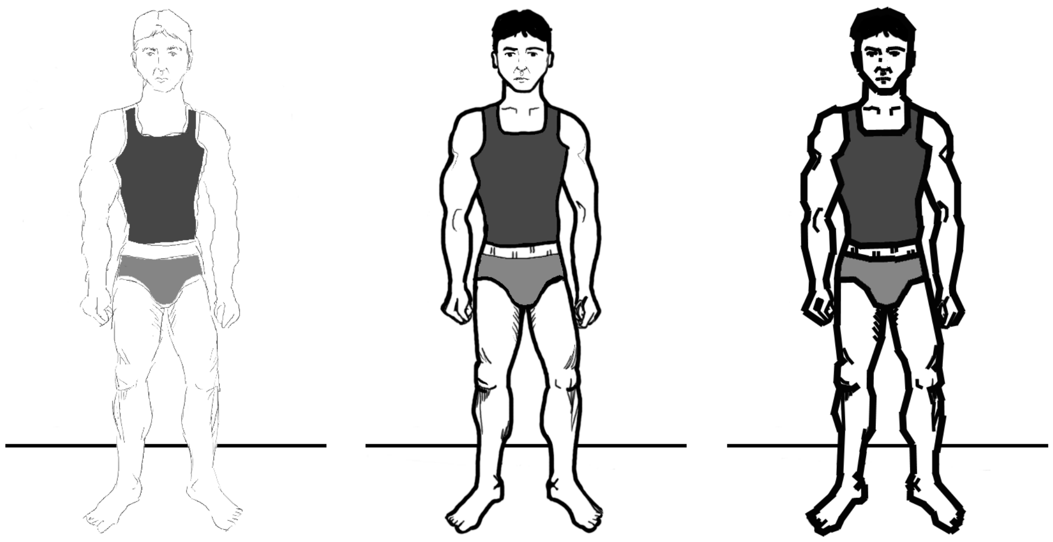
(a) “Your units have engaged in battle. Click on your WEAKEST friendly units”



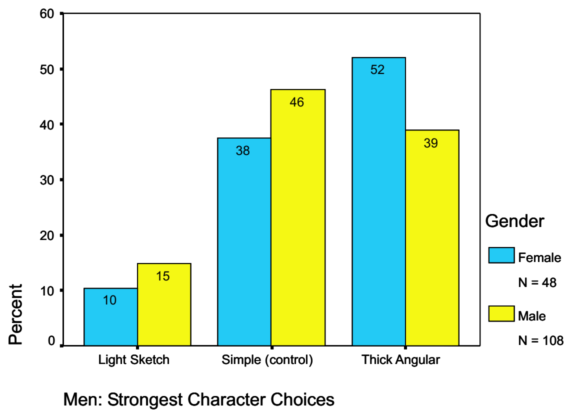


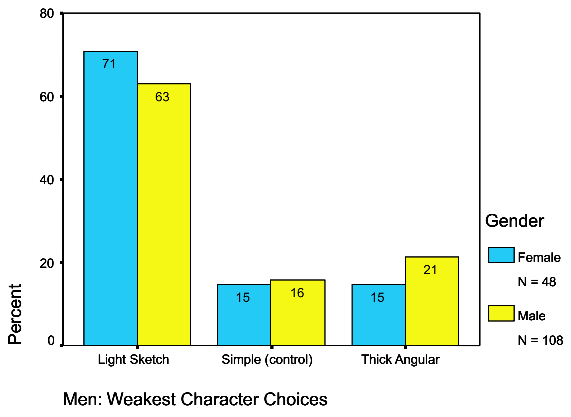
(b) (left) L1 (χ^2(1,97)=85.37, p<.001) and L2 (χ^2(1,37)=14.30, p<.001) selection for weakest units. (right) Boxplot of L1 response times. The dotted circles MRT (4.5s) is significantly lower than the thick circles MRT (7.0s) (F(1,109)=5.40, p=.022).

Figure 2.8: Using rendering style to influence assessment of strength



(a) (i) “Click on the character which appears the STRONGEST” (ii) “Click on the character which appears the WEAKEST”





(b) Results for character selection across gender.

Figure 2.9: Assessment of Character Strength and Weakness

Hypothesis: When asked to select the strongest man, subjects will select the thick silhouette character over the control, and the control over the light silhouette character.
Results: Subjects by in large choose the thick (43%) and control (44%) characters over the light character (13%) (χ^2(2,156)=27.73, p$<$.001). Further analysis reveals notable gender differences for MRT (males 4.8s; females 3.8s) and character selection (see Figure 2.9(b) (left)).

Assessment of Character Weakness

Subjects are shown the same characters as in Section 4.2.2 (Figure 2.9(a)) with re-randomized positions, to further account for any variations in selection, this test is immediately preceded by the 'evaluation of character strength' test.
Hypothesis: When asked to identify the weakest character, subjects will select the light character over the control, and the control over the thick character.
Results: A significant majority of subjects (67%) select the light character (χ^2(2,156)=72.46, p$<$.001) with notable differences in gender preferences for the lesser selected characters (see Figure 2.9(b) (right)). No gender differences for MRT are observed.

2.4.3 World-Based Expectations

If it is indeed possible to convey implicit information via line style variations between objects, there exists the potential for entire environments rendered in a specific style to also convey implicit information. Specifically, it is arguable that an environment rendered in a more cartoon or comic-book style might be perceived as less bound to real-world physics than an identical PR image. In other words, an NPR environment might maintain a wider perceived range of viable options for behaviour between objects. In the following two experiments we measure individual expectations using geometrically identical PR and NPR images, while asking subjects to make scaled assessments of probability regarding the depicted scenarios.

PR vs. NPR Jump

A PR scene is generated wherein a man is shown standing on top of a building in the centre of a cityscape across from another building some distance away (Figure 2.10(a)). Additionally, a geometrically-identical NPR cartoon scene is generated (Figure 2.10(b)). We used a seven point scale ranging from ”impossible” to ”easily,” wherein subjects are shown either the NPR or PR version and then asked to judge the likelihood that the man-with a running start, will make the jump.
Hypothesis: subjects shown the NPR image will make relatively more optimistic judgements than those shown the PR image.

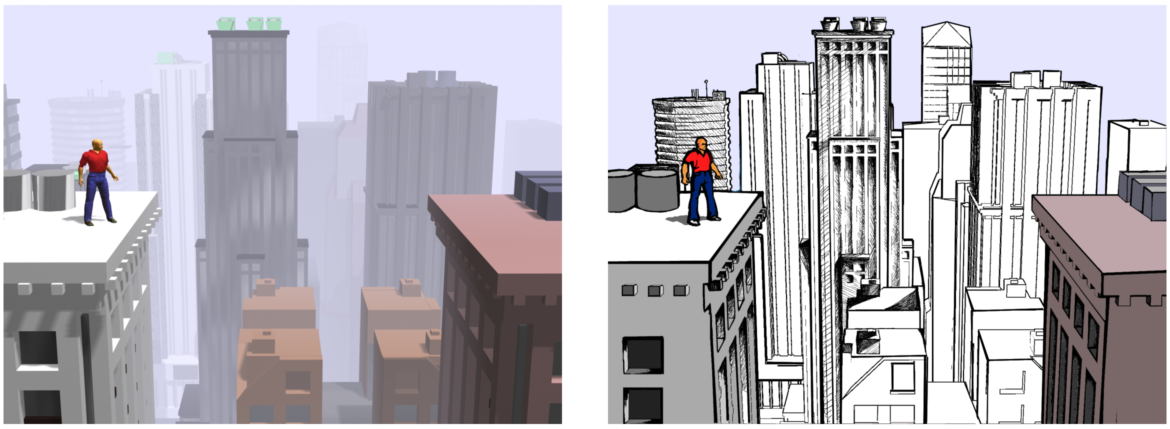


Figure 2.10: “With a running jump, what is your character’s chance of making the jump?”

Results: General pessimism was observed from the distribution of frequencies for responses exhibiting a normal curve peaking at ‘unlikely’. However, we found no significant differences between, within, or across groups.

PR vs. NPR Weights

To further test PR-NPR variations in expectations, a PR scene is shown depicting a muscular man standing in front of a seven weights ranging in descending size and apparent heaviness (Figure 2.11(a)). A geometrically-identical NPR cartoon scene is also generated (Figure 2.11(b)). Subjects are shown either the PR or the NPR image and asked to select the heaviest weight the man could lift.
Hypothesis: subjects shown the NPR image will make relatively more optimistic judgments than those shown the PR image.

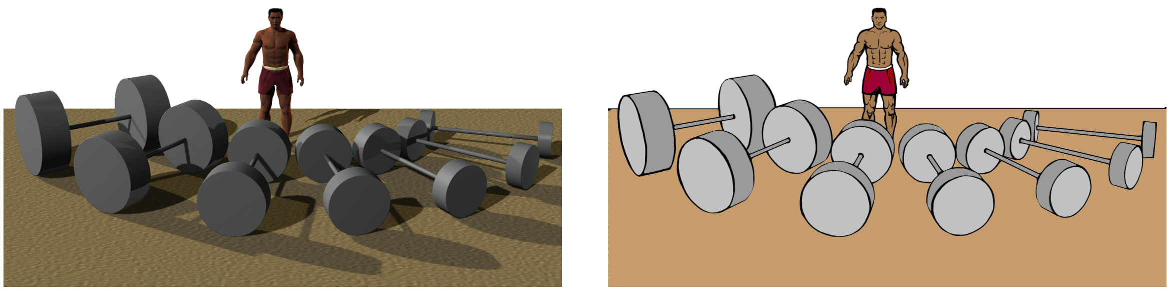


Figure 2.11: “Click on the heaviest weight this character can lift”

Results: There were no significant differences between, within, or across groups. The distribution of frequencies for responses exhibited a normal curve peaking at the 3rd heaviest weight.

2.4.4 Goal-Directed Interactions

The following experiments attempt to measure effects of rendering on exploration, navigation, and interaction. Specifically, LOD is adjusted to test exploratory and navigational decisions. Finally, a variety of objects are rendered in two different styles, distinguishing those objects that are interactive from those that serve as decoration.

Guided Exploration

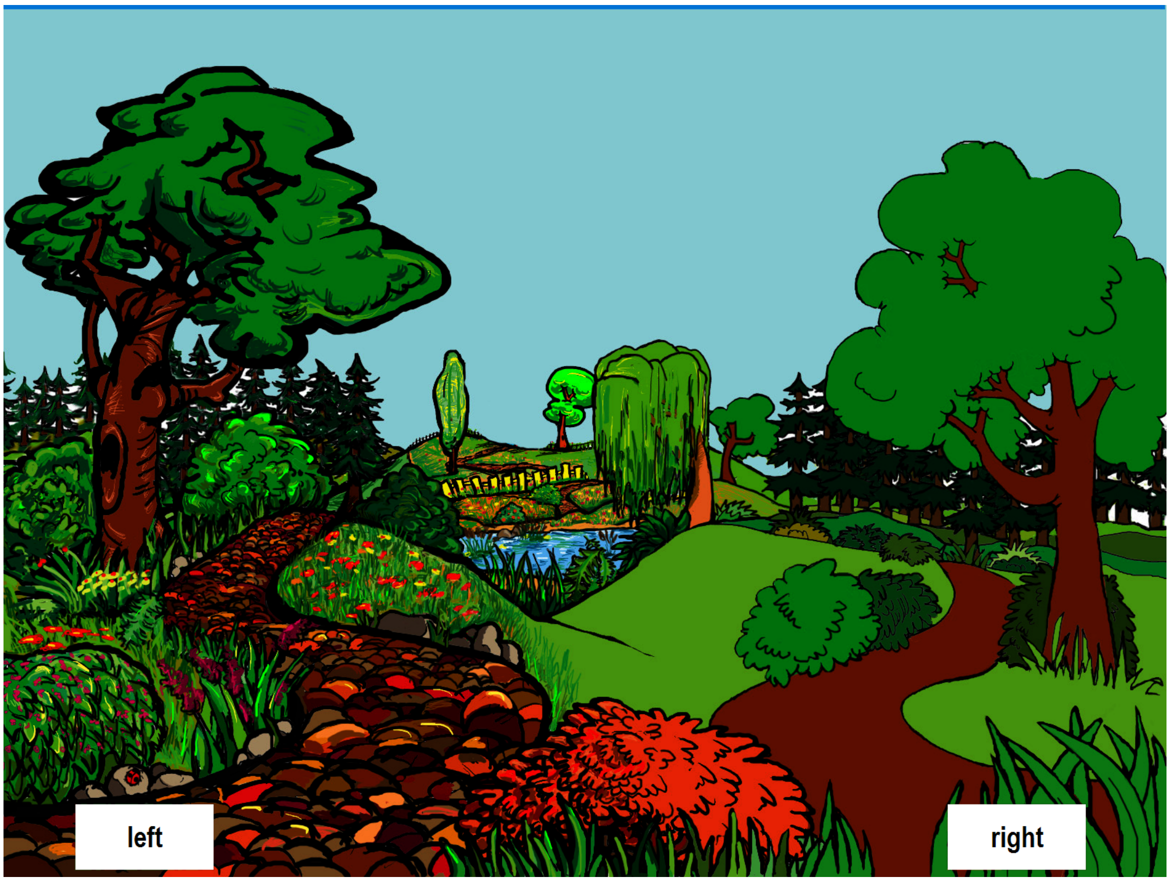
Subjects are shown an image depicting two divergent paths. One group is shown the image with the left path rendered in high LOD (DL_{expl}) as seen in (Figure 2.12(a)), and the other group is shown its mirror image (mDR_{expl}). Image mirroring controls for potential left-right or content biases.
Hypothesis: When asked to select a path to further explore, subjects shown the original or mirror image will select the high LOD path.
Results: Figure 2.12(b) shows a tendency for subjects to select the high-LOD paths, and in particular for DL_{expl} a 35% less MRT than low-LOD selection.

Guided Navigation

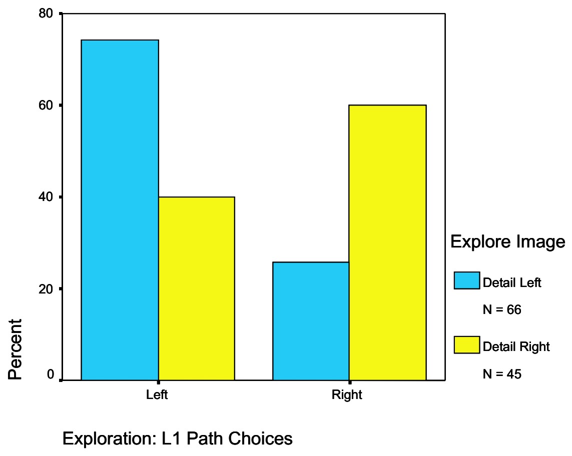
Subjects are shown two divergent paths in the foreground with a single path leading to a house in the background. Groups are randomly assigned to the following test conditions: left path high LOD (DL_{nav}), right path high LOD (DR_{nav}), the mirror image of DR_{nav} (mDL_{nav}), and the mirror image of DL_{nav} (mDR_{nav}). DL_{nav} and DR_{nav} can be seen in Figure 2.13. Image-mirroring and LOD variation controls for potential left-right or content biases.
Hypothesis: When asked to select the path leading to the house, subjects from all four groups will select the high LOD path.
Results: Subjects chose the high LOD path accordingly, as shown in Figure 2.14. Furthermore, a tendency for subjects to select the opposite side relative to the exploration test is observed (F(1,154)=5.30,p=0.023). Figure 2.14 also shows that response times for DR_{nav} and mDL_{nav} are significantly lower than for DL_{nav} and mDR_{nav} .

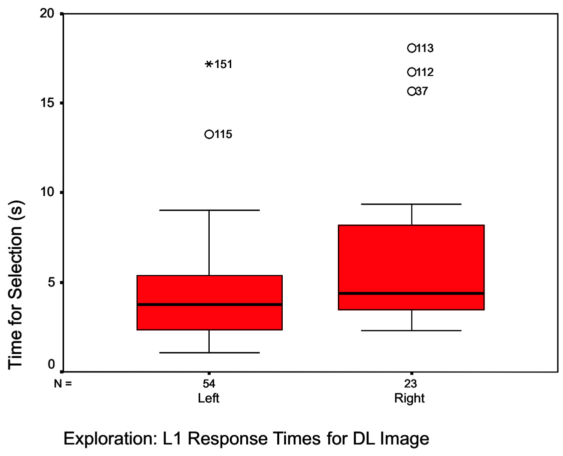
Guided Selection of Objects

Subjects are shown an image of a bookshelf and 20 different objects. Two mutually exclusive test conditions (group A, group B) are assigned, wherein subjects are shown half of the



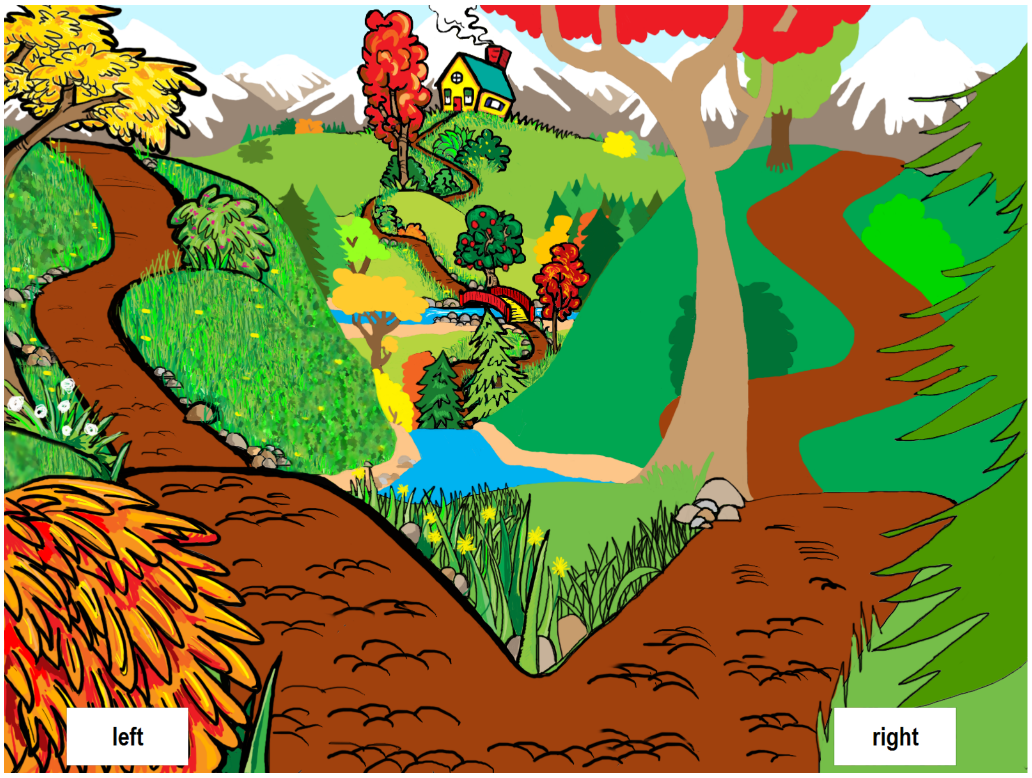
(a) “Choose a path to explore further (click on ‘left’ or ‘right’ box below to pick the path)”





(b) Most DL_{expl} subjects choose the high LOD path (χ^2(1,66)=15.52, p<.001). For mDR_{expl} subjects, a notable majority select the high LOD path. (right) Boxplot of response times for DL_{expl} . The DL_{expl} MRT for the high LOD path section (4.0s) is significantly lower than the MRT for the low LOD path (6.2s) (F(1,75)=6.87,p=.011).

Figure 2.12: Rendering for exploration.



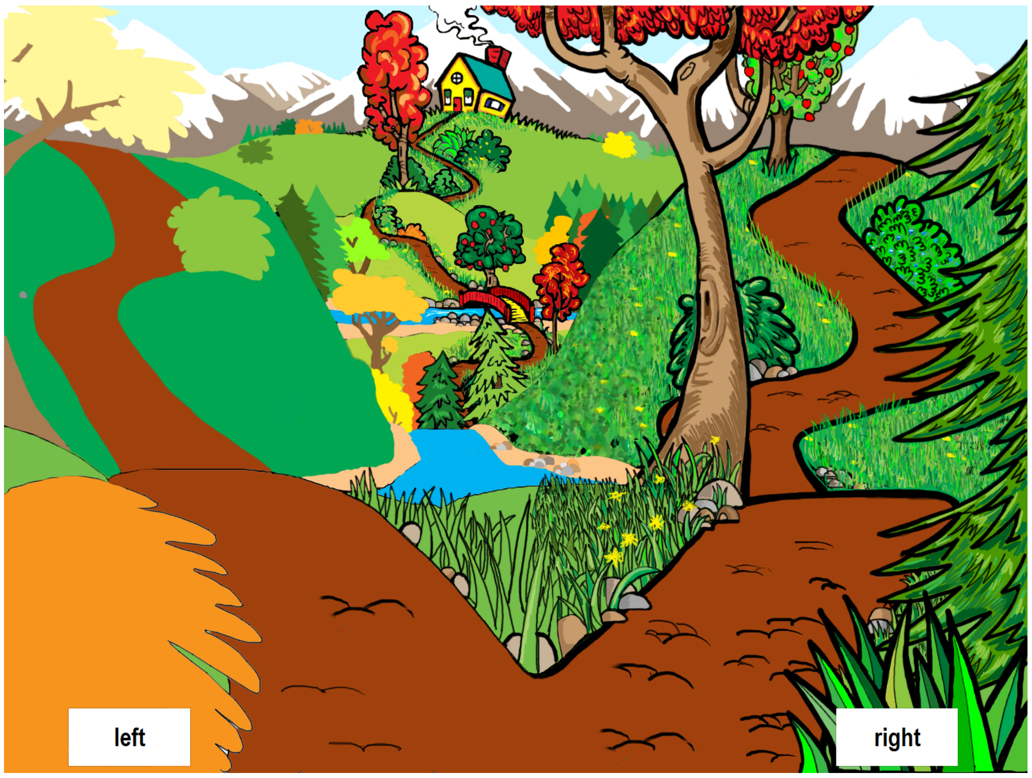


Figure 2.13: “Click on ‘left’ or ‘right’ to choose the path that leads to the house shown in the distance”

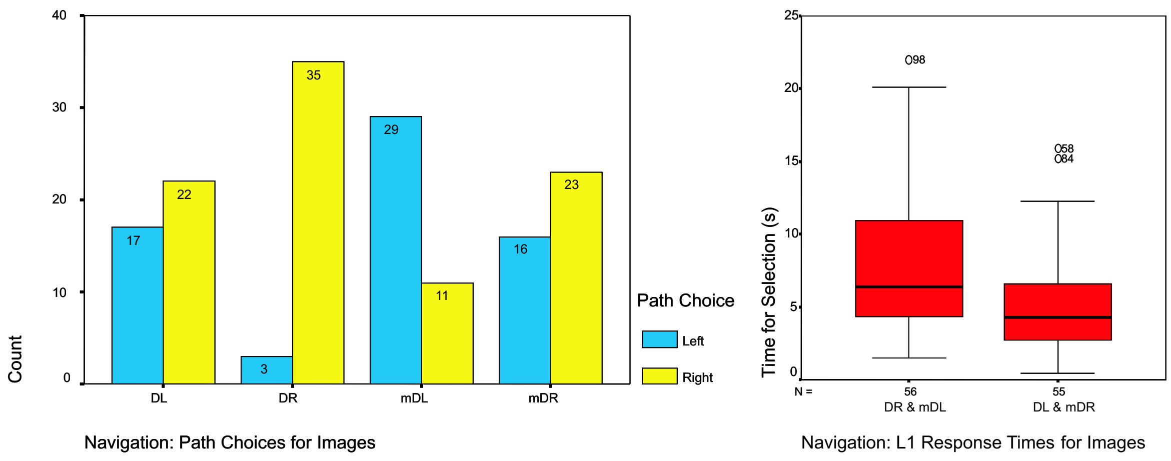


Figure 2.14: (left) Subjects chose: DR_{nav}: 92% (χ^2(1,38)=26.95, p<.001); mDL_{nav}: 73% (χ^2(1,35)=8.10,p=.006); DL_{nav}: 44%; and mDR_{nav}: 59%. (right) Boxplot of response times. The combined MRT for DR_{nav} and mDL_{nav} (4.7s) is significantly lower than the combined MRT for DL_{nav} and mDR_{nav} (7.7s) (F(1,109)=14.12, p<.001).

objects rendered as ‘toon’ to connote interactive qualities, while the other half are ‘painted’ as part of the background (see Figures 2.15).
Hypothesis: When asked to choose three objects, subjects will tend to select toon objects over painted objects, resulting in exclusive choice sets for each groups.
Results: In both groups, subjects show a significant tendency to select two or more toon objects (see Figure 2.16).

2.5 Interpretation

Our results suggest that rendering variables can be manipulated to effectively guide user responses to (1) character and object attributions, (2) aspects of navigation and exploration, and (3) interactive choices. In terms of ICS, we are specifically interested in understanding subject responses to rendering styles used to convey latent affective content, or ‘implicational rendering.’ We accordingly discuss the meaning and potential of our results[7] while considering design limitations within the experiment. Overall test design limitations are: the timer and ‘next’ button, which are consistently viewed on the right of the screen, and may effect mouse placement, image evaluation, and subsequent responses. Since our selectable items are large, according to Fitts’ Law (Fitts 1955) mouse placement and use do not significantly contribute to variations in response times.

Further evaluation of MRT allows for a better understanding of subject responses, wherein

[7]In [Duke et al., 2003] we develop a deeper theoretical analysis of the results in terms of the ICS model mentioned in Section 2.2.2.

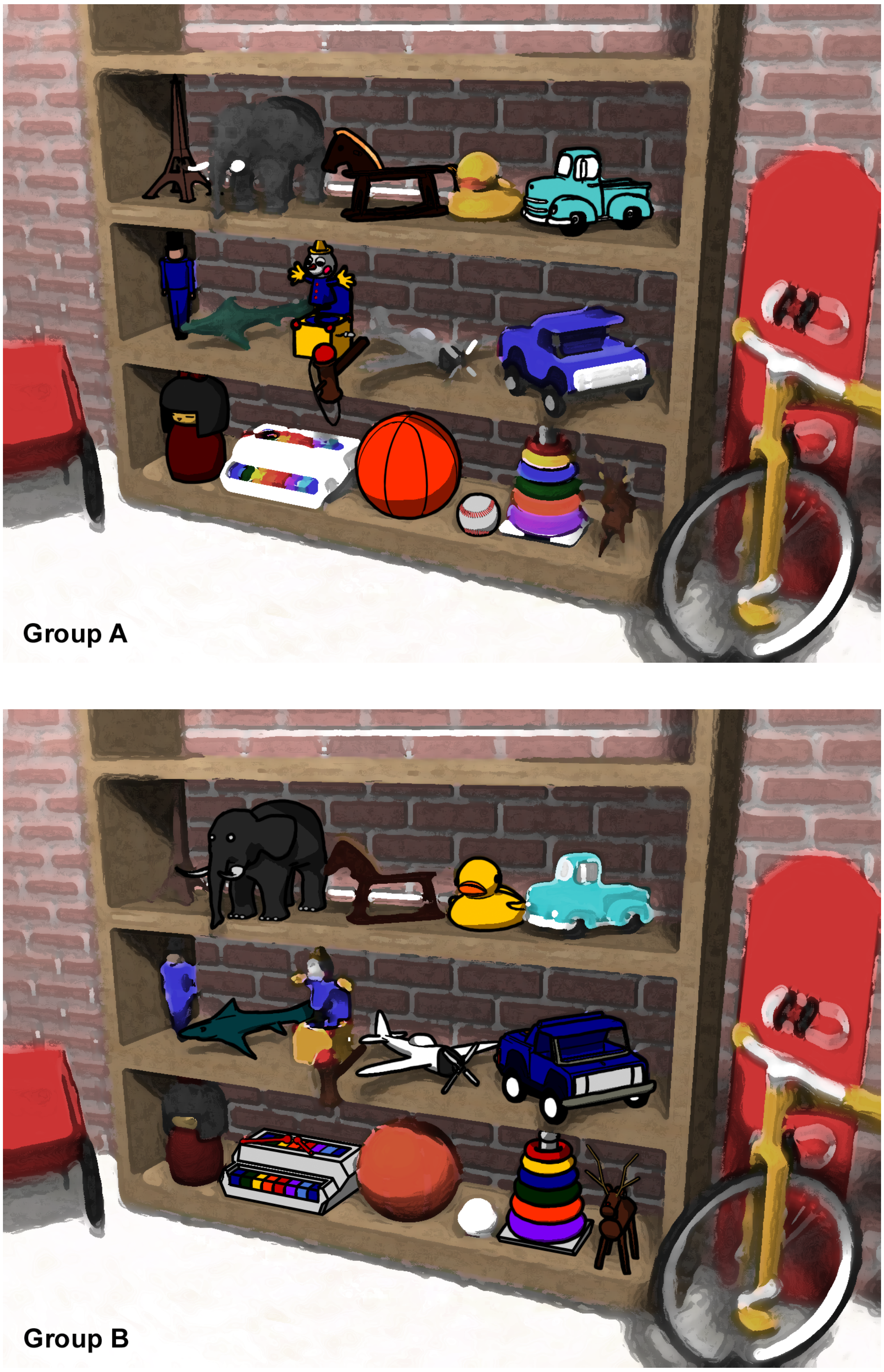
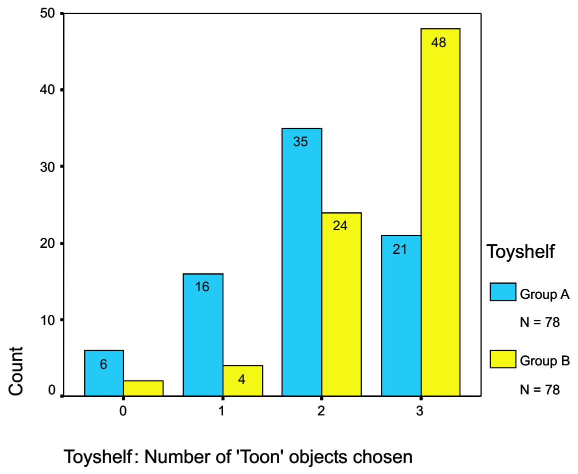


Figure 2.15: “Click on THREE objects”



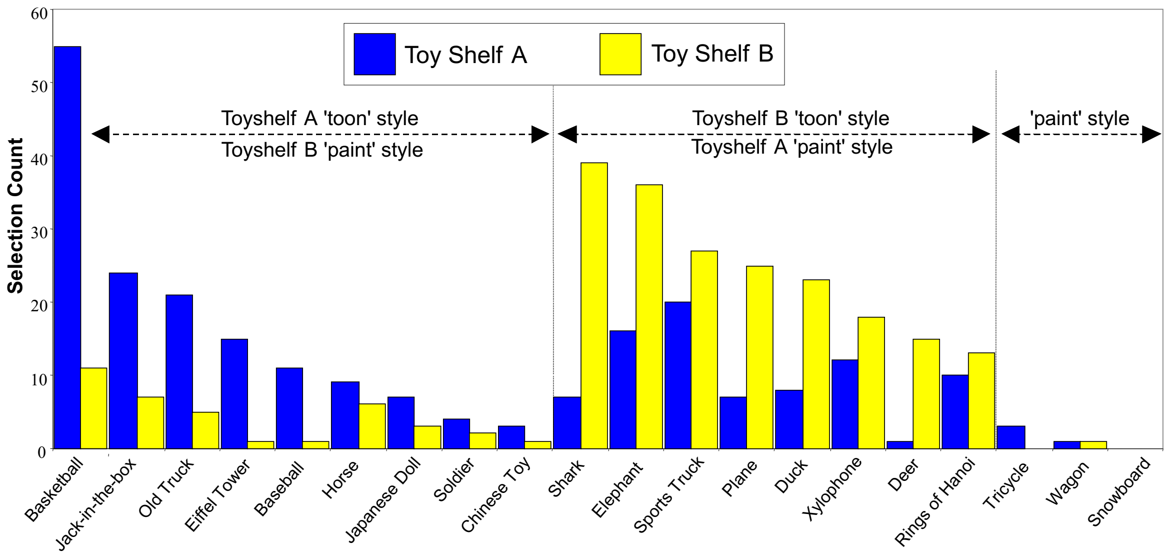


Figure 2.16: (top) Toyshelf results. A significant majority of subjects select two or more toon objects: 72% of group A (χ^2(1,78)=22.41, p<.001); 92% of group B (χ^2(1,78)=55.84, p<.001). (bottom) Object preference is significantly different across testing groups (p<.001)

higher MRTs suggest more time spent analyzing the task and image, whereas lower MRTs indicate less analytical, latent responses. Moreover, the rapid-response imperative induced by the timer pre-disposes subjects to answer as quickly as possible, thereby enhancing latent information processing. Therefore, if implicational rendering is truly ‘tapping into’ latent implicational processes, subjects responding as hypothesized should demonstrate significantly lower MRTs than those responding otherwise. Indeed such an effect is observed within our overall results.

2.5.1 Attributions of Strength and Danger

The overall results for implicational rendering in simple images (radar units, doors, houses and trees), clearly suggests that it is possible to influence attributions of unit ‘friendliness’ and weakness, as well as assessments of danger and safety. The initial radar test reveals that subjects selecting triangles, rather than circles as their friendly units, tend to respond in the minority on all tests, suggesting individual predispositions to further analyze the tests. Gender differences are observed for the tree and house tests, wherein females tend to respond as hypothesized more often and more rapidly than males, though a relatively small female sample and lack of a control image prevents significance. Despite the house and tree not controlling for responses based solely on how different images appear from the simple image, the door test demonstrates that ‘threat-connotative’ implicational rendering overrides this tendency. Responses to the control door may be partially due to cultural or language differences, as L2 selects the control and simple doors more than L1. Because this is a first analysis of implicational rendering, simple images are used to extract the impact of rendering on assessments of danger and strength.

It is possible that due to both the simplicity of images and the relative deviation of rendered lines from the objects’ ‘normal’ physical structure, assessments of danger may be influenced by a changed perception in the physical characteristics of the images. Thus subjects could assess objects based on a literal analysis of its physical structure (e.g., “The house appears physically jagged and therefore more dangerous than the non-jagged image, as jagged shapes hold the potential for inflicting pain”). However, given the fact that the rapid response times in the assessments of relative danger were similar across test types—regardless of the presence of a ‘jagged’ image, it is unlikely that such overt consideration took place, and more probable that subjects are assessing an abstract representation of objects. Thus, at a rather simplistic level, implicational rendering demonstrates itself to be quite effective and necessitates further study of effects in terms of line variables and image complexity, while accounting for potential gender variations.

When assessing strength for the complex image, males respond notably[8] slower and differently than females who respond most to the thick character, and least to the light character, whereas males select the control over the thick. In accounting for these results, further consideration reveals that the thick character’s lines diminish his apparent muscle mass rel-

[8]Unfortunately, low numbers of female subjects could not show statistical significance

ative to the control and light character, thus suggesting that males engage in more overt analysis of the image than females. Subsequent results for assessment of character weakness, while generally supporting our hypothesis, reveal a similar trend for the minority who respond otherwise. Generally, responses to the complex image suggest implicational rendering influences attributions of strength when using extreme line variations to assess weakness or strength. Further work is necessary to test varied rendering parameters, the impact of threat-connotative lines on more complex images, and the impact of image-type on gender.

2.5.2 Expectations in Rendering

In ICS terms, the difficulties with the jump and weight tests (Section 2.4.3) stem from attention being drawn to the structural detail of the scene, and (consequently) focusing awareness on matching the semantics of the scene with the task description. A better approach would be to set up an assessment to draw attention on a secondary task, leaving assessment of the physical circumstances to latent implicational knowledge. For example, the jump test could ask whether the character would be able to retrieve some object (which happened to be located on the other side of the gap). A more in-depth ICS analysis is given in [Duke et al., 2003], that notes this would encourage buffering at the propositional level, allowing direct mode processing of any implicational invariants.

2.5.3 Guiding Navigation, Exploration, and Interactions

Given the DL_{expl} image[9], most subjects select the high LOD path. mDR_{expl} subjects also show a notable preference for the high LOD path, although the sample size is too small to determine significance as L2 results are omitted. Potential cultural or language comprehension effects invalidated L2 results. For the sake of simplicity when discussing the navigation test, ‘DL_{nav}’ and ‘DR_{nav}’ refers to both the original and mirror image. A notable content bias is observed, wherein the high LOD path in DR_{nav} is strongly preferred both independently and relative to the high LOD paths in DL_{nav} .

These differences are best understood in terms of (1) the pre-conditioned test objective and (2) potential biased image content. First, there is a pre-defined goal, wherein subjects are asked to deduce which path leads to the house, thus potentially directing subjects to analyze and respond to the differentially rendered paths in an overt—rather than latent manner according to which foreground path appeared most similar to the background path. Although during informal questioning after testing, a number of subjects tended to admit they did not really notice the relative appearance of the background path. Second and perhaps more importantly, the background path veers to the right as it approaches the foreground, potentially predisposing subjects to the right-hand path regardless of LOD.

In addition, DL_{nav} subjects take significantly longer to respond than DR_{nav} subjects, potentially due to additional image analysis by DL_{nav} subjects to the LOD, in opposition to the

[9]For the exploration test, L2 results are omitted as they satisfy all conditions for data exclusion.

content-bias. If content biases are extracted, results suggest that LOD can influence explorational and navigational decisions. Further tests are needed which control for content and cultural biases, thereby more precisely defining the impact of LOD on general exploratory and navigational decisions.

Results from the ‘guided selection’ test strongly suggest that implicational rendering can influence subject response, wherein toon (active) objects are significantly preferred over painted (inactive) objects. A basic consideration in understanding subject responses is the nature and effects of the specific rendering techniques employed. In this particular case, heavily silhouetting objects (rendered as active in a toon style) may actually highlight the object and any perceived relevance. Thus, it is possible that different rendering techniques, using thinner silhouettes and less distinct color tonalities may lead to a lower selection percentage among viewers. Since the goal of rendering an object as active is to visually distinguish it from merely decorative objects, whether an object is perceived as highlighted or active, is somewhat of a semantic distinction as both terms serve the same end: functionally, the attention of the viewer is inadvertently drawn to active objects relative to inactive objects.

Interestingly, informal post hoc questioning revealed most subjects as unaware of the two different rendering styles, despite selecting a majority of active objects. This may be due to familiarity and acceptance of toon and paint style animation (e.g., Disney Feature Animation films). Furthermore, a small percentage of subjects—claiming to notice the distinct rendering styles, admitted to purposefully select a painted (or inactive) object because they ’knew’ they were supposed to select a toon object. Thus—despite reacting negatively to the demand characteristics of the test, these particular responses imply the effectiveness of implicational rendering given a real-world application. In other words, these subjects both deduce and respond to the images according to implicational rendering styles in a manner that would be effective in goal-oriented games. More sophisticated test designs, perhaps utilizing real game properties would better disguise existing demand characteristics, thus revealing the extent to which rendering objects as active or inactive can actually facilitate user-object interactions.

By in large, object attractiveness, as defined by relative brightness, size, and simplicity, is another notable factor for this test. Those objects that are selected when inactive are generally amongst the more attractive, whereas darker, smaller, and more complex objects are only selected once they are active. In addition, every object is selected more times when active than when inactive. Thus, the overriding factor is not the attractiveness of the object, but whether it is rendered as active or inactive. Although further research might uncover potentially unaccounted for effects, these results suggest that implicational rendering is immediately useful in a variety of interactive graphics settings.

2.6 Discussion

This chapter began by discussing emerging factors in computer games and identified photorealism as a cause for many problems in game design. Game designers strive to make compelling games which emotionally absorb the player by stimulating affective areas of the brain and have generally made the assumption that the more realistic a graphical environment appears, the better the sense of presence and thus arousal. However, little research has been carried out in this context and photorealistic effects are both computationally expensive and require large resources for production.

As an alternative presentation strategy, NPR was considered, but a lack of an effective evaluation of NPR motivated a first empirical study. While psychophysical studies in computer graphics tend to study visual perception in terms of shape or structure, our experiments reveal that other cognitive processes can be influenced by rendering. Specifically, we effectively influence character and object attribution, while guiding interactive decisions via manipulating the latent implicational content of rendering. Because these processes can occur simultaneously with the processing of basic structural information, NPR has the potential at a pre-conscious level to facilitate user interactions.

Our experimental results suggest that game graphics can be extended by using NPR to provide connotative effects and convey cues (particularly to novice players) about possible and/or appropriate actions, specifically:

- relevant cues about the level of threat (Section 2.4.1) or capabilities of characters, objects, or locations in an environment (Section 2.4.2).
- guidance for choosing between options, such as which path to follow or object utility (Section 2.4.4). NPR effects could also be modified based on player progress, and/or changes brought about by other agents. For example, an explored area could be rendered in low detail to discourage re-entry, but if the actions of another player then change the area, detail levels would again increase.
- Our results demonstrate that it is possible to use implicational rendering to successfully distinguish between active and inactive objects. Thus, NPR could effectively cue users to objects and areas available for use, while others objects or areas rendered as inactive still maintain the world's structure and organization by providing critical spatial information or content.

Rendering can be 'tuned' to players based on their performance by turning NPR effects on/off, or (possibly) by varying parameters. This offers the potential to dynamically adjust the balance between player skill and confronted challenge and allow newcomers to become involved in the game without becoming frustrated while more skilled players can still be entertained by the competition. In addition, supportive rendering has the particular advantage of influencing the quality and enjoyment of the game without actually altering the variables for gameplay control and balance.

The range of non-photorealistic rendering effects is very large, and therefore the number of options to influence the player by subtly bringing across messages and information is greatly increased. Photorealism simply does not offer the range of communicative possibilities without actually distracting the player from the immersive qualities of the game. In this respect, implicational rendering styles resonate more closely to elements of play that are fun.

An effective application of these effects requires further investigation using low-level tests to establish precise mappings between observed effects and drawing parameters (for example, line thickness, style, and angle of ‘sharp’ features). Part of this responsibility lies in the hands of game developers and designers and the tools that can be provided. The rendering system and interface discussed in the following chapter allows programmers and designers to experiment with visual styles to support game goals. Later in Chapter 5, we revisit these techniques in the framework for supportive presentation applied to specific application scenarios.

3 Non-Photorealistic Rendering Tools

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. Current NPR applications are costly to develop because their data and methods are not re-used. Consequently, there is a need for an effective rendering system that provides support for multivariate applications.

Despite the plethora of non-photorealistic effects available, there exists a rather limited number of primitives (such as lines, points, surfaces, etc.) actually employed to generate these effects. 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 (e.g., the similarity of their data structures and algorithms). It is then possible to apply specific classes of algorithms to the same sets of data—consequently sidestepping unnecessary data conversions between software projects. Additionally, NPR algorithms can be individually broken down into a set of smaller algorithms, wherein an ‘elementary set’ of algorithms is 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 rendering pipeline for a certain style, and finally those who use the rendering pipelines in order to produce images.

This chapter is structured based on these user classes and we describe the tasks involved at each level of abstraction. Our main contribution is a first attempt to unify many NPR techniques within a single framework. We present the initial system[1], OpenNPAR, which embodies this framework in Section 3.1. Developing extensions to the system is described in Section 3.2, programming using the system in Section 3.3, and using the system to design new effects in Section 3.4. We base our discussion in Section 3.5 on our achievements.

3.1 OpenNPAR

In this section, we outline and motivate our design goals used to structure the basic architecture for our NPR system, OpenNPAR. We first categorize our base classes for algorithms and groups of users on which we base a conceptual framework. We then introduce initial components for the core system and provide examples for their usage.

3.1.1 Classes of Algorithms and Users

The field of NPR contains a large number of different rendering algorithms that come from many areas. This diversity makes it difficult to come up with a single system that encompasses all NPR styles and techniques. However, there are similarities between algorithms that make a classification possible. For instance, Durand [2002] proposes a classification into four parts: (1) the *spatial system* which deals with the spatial properties of the image (e.g., mapping 3D properties to 2D properties via projection matrices), (2) a *primitive system* which maps primitives in object space (points, lines, surfaces) to primitives in image space (points, lines, regions) as in, for example, silhouette extraction, (3) an *attribute system* which assigns visual properties (e.g., color, transparency, thickness), and (4) a *mark system* which is the implementation of primitives placed at their spatial location with corresponding attributes.

Whereas Durand proposes a useful classification of systems for discussion and terminology, there is still no clear direction for a unification of algorithms into a single system. Indeed, there are many open problems and approaches to producing NPR images. In this respect, our basic philosophy is to allow users of the system to define their own approach into producing the final output by making available a powerful, flexible, and extensible set of tools. We aim to achieve this by providing small well-identified modules for better interoperability, centered around a small set of basic primitives. Thus, we focus on the direct relationships between primitives: primitives serve as states in our system described by attributes, and processes describe operations on or between those primitives. In Figure 3.1 we propose a classification of three main categories of primitives on which algorithms can operate. Note that the processes (arrows) in the conceptual diagram can fall into any of the four categories proposed by Durand above. We explain the general use of these classes of primitives below:

[1]Initial developments on the OpenNPAR system were made in collaboration with Tobias Isenberg and Felix Ritter. Parts of this chapter involve contributions from additional co-authors in [Halper et al., 2003a] and [Halper et al., 2002].

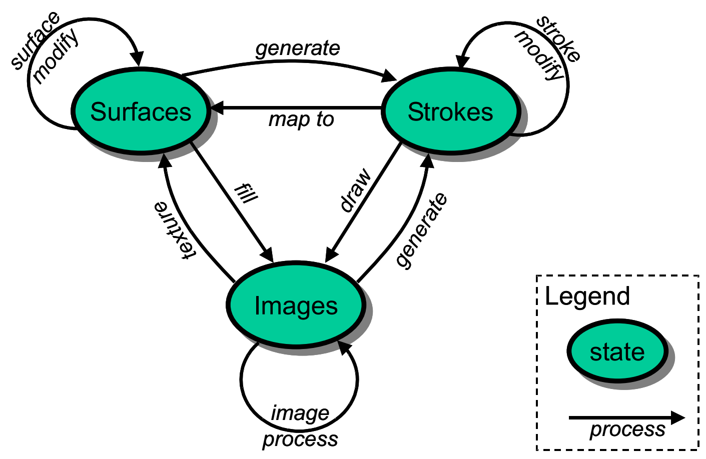


Figure 3.1: Conceptual model for algorithms in NPR

Images store two dimensional arrays of data, such as color, depth, normal data, etc. Eventually, most output is rendered into images either for viewing or for later use in an algorithm. Images can use image processing algorithms to modify their content. Data stored in images can be used to affect surfaces (by defining textures), or generate strokes (as in [Salisbury et al., 1994]).

Surfaces define objects and their appearance in 3D. The way that surfaces 'fill' images is described by components such as coordinates, connectivity information, texture mapping, and so on (see [Lake et al., 2000] for a few examples). Surfaces can also generate strokes, such as placing strokes along the silhouette of an object, or adding 'graftals' (a special kind of stroke) onto the surface [Kowalski et al., 1999].

Strokes are individual primitives that create tone and texture depending on their placement, number, and parameterization. A high stylistic variety of effects can be achieved when strokes are 'drawn' to an image, such as stippling, pen-and-ink illustration, and painting. Strokes can also be mapped onto surfaces indirectly by first drawing them to an image, and then texture mapping this image to a surface (e.g., [Praun et al., 2001]).

Visualizing the classification of primitives according to Figure 3.1 enables us to create building blocks (individual processes) that can be used in various combinations (a sequence of processes) for different rendering pipelines (the collective sequence of processes). In fact, many hybrid algorithms use combinations of two or all three of these classes (for an overview see [Strothotte and Schlechtweg, 2002; Durand, 2002]). Table 3.1 outlines processes between primitives according to our conceptual model for several 'classic' NPR algorithms. Having such basic building blocks, the user can freely combine NPR algorithms in order to achieve a certain kind of image. However, the amount of knowledge involved in the "programming" process for a rendering pipeline is different depending on the level

Name	Description
Saito and Takahashi [1990] Comprehensible Rendering	Surfaces fill Images with shaded, z, normal data; Images process silhouette Image; Image processing (composite shaded and silhouette)
Salisbury et al. [1994] Pen-and-Ink	Surfaces fill reference Image; reference Image generates Strokes; Strokes draw into Image
Meier [1996] Painterly Rendering	Images generate Strokes (particles); Strokes map to and Images texture Surfaces
Kowalski et al. [1999] Graftals	Surfaces fill ID Images and desire Images; ID Image processing; desire Images generate Strokes; Strokes draw into Images

Table 3.1: A selection of classic NPR algorithms and how they relate to our conceptual model.

of abstraction. Hence, a distinction into different types of users is needed. These types are—with increasing level of abstraction—developer, programmer, and designer.

- The *developer* has the scientific knowledge to come up with new algorithms and techniques.
- The *programmer* takes these algorithms and turns them into the aforementioned basic building blocks for rendering pipelines.
- The *designer* knows how to combine these building blocks and thus how to create different rendering pipelines.

At each level of abstraction, certain knowledge is needed; the higher the level, however, the less significant are technical and technological details so that the designer can concentrate on the image generation process. Thus, the goal of OpenNPAR is to embody our conceptual model for algorithms in NPR shown in Figure 3.1 and support user groups of various knowledge levels for creating NPR images.

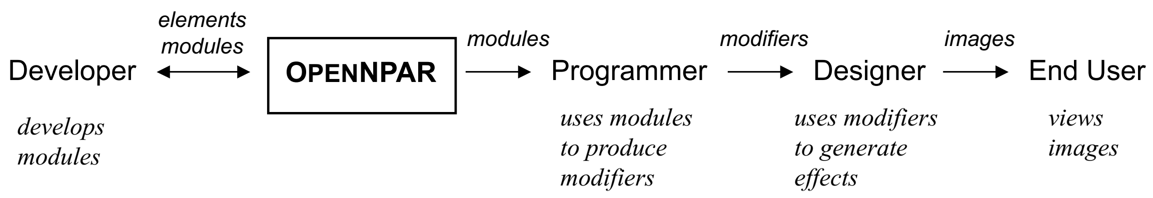


Figure 3.2: Knowledge pipeline for developers, programmers, and designers

In Figure 3.2 we show the knowledge requirements and dataflow between the various user groups and OpenNPAR. Designers are given the task of creating new images by applying a selection of effects onto existing images. Thus, to work effectively designers only need knowledge of what *results* are produced once processes reach the final image. The selection of effects offered to designers are assembled by programmers. Programmers understand the relationships between processes and the various categories of primitives. From this knowledge they can assemble modules into a rendering pipeline to produce an effect. The more modules that are provided to programmers, the more operations and thus effects they can offer to designers. The developer's task, therefore, is to broaden OPENNPAR's base functionality by extending classes of primitives (by creating elements) and adding modular operations that decide what to do with them. Before we go into details for the user categories, we first describe some initial components of OPENNPAR.

3.1.2 Basic Architecture

OPENNPAR is to provide an architecture that supports our conceptual framework with the following requirements:

- *High-Level API:* this would be structured for ease of use in plugging modules into a rendering pipeline.
- *Object-Oriented:* an object-oriented approach is necessary to support re-usable, well-engineered, and extensible code.
- *Cross-Platform:* should be able to support development on a variety of platforms, including Windows, SGI, and Linux.

For these reasons, we choose to base the framework for OPENNPAR upon OPEN INVENTOR, an object-oriented graphics architecture that uses a scene-graph based approach [Strauss and Carey, 1992]. OPEN INVENTOR is a proven and extensible architecture that provides all major graphics functionality and scene-graph management (e.g., interaction, I/O, textures, VRML). In addition, it has a large support group (newsgroups, websites), with both LGPL and commercial licenses available.

Although OPEN INVENTOR provides a standard set of functionality, it must still be able to capture the processes in Figure 3.1. Below, its main features are outlined (based on Wernecke [1994]), which include additional descriptions of its use by OPENNPAR to support our conceptual model:

Scene Database: consists of information representing one or more 3D scenes. It can contain several scene graphs, which can also be read from and written to a file.

Whenever OPENNPAR extends the base functionality of OPEN INVENTOR , these extensions update the scene database. New scene descriptions can then be automatically written to and read from files. Scenes are also capable of containing 2D information, thus we can also include methods to store image primitives.

Scene Graphs: consist of one or more *nodes*, each of which represents a geometry, property, or grouping object. Hierarchical scenes are created by adding nodes as children of grouping nodes, resulting in a *directed acyclic graph* (see Figure 3.3).

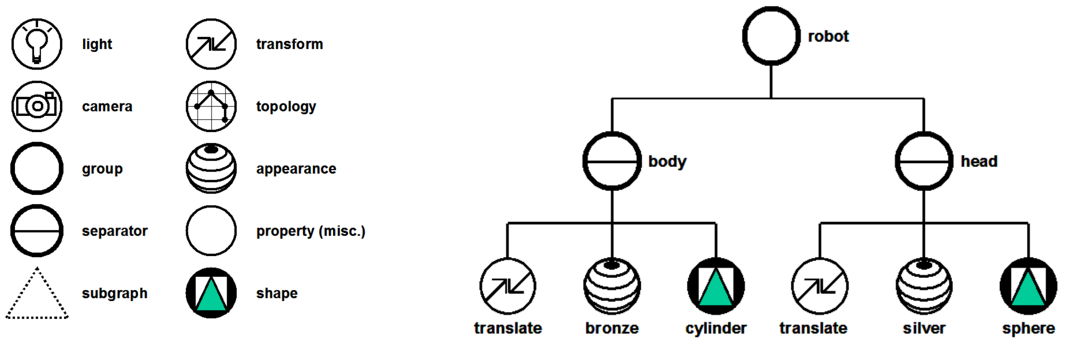


Figure 3.3: (left) OPEN INVENTOR example nodes and (right) a scenegraph that describes a robot (adapted from Wernecke, 1994).

In our conceptual diagram processes may be iteratively applied between primitives, which suggests the use of undirected cyclic graphs. Fortunately, OPEN INVENTOR does not enforce a policy on the organization of a scene database, thus nodes can be organized into structures that are not graphs. In addition, fields (see later) can propagate information inside and between scene graphs.

Nodes: A *node* is the fundamental element of a scene graph. It contains data and methods that define some specific 3D shape, property, or grouping. Three basic types[2] of nodes are: (1) *shape nodes*, which represent 3D geometric objects; (2) *property nodes*, which represent appearance and other qualitative characteristics of the scene; and (3) *group nodes*, which are containers that collect nodes into graphs.

In Figure 3.3 the bronze and silver nodes describe material properties for their subsequent cylinder and circle shapes respectively. Therefore, property nodes are capable of changing the appearance and rendering style of objects. Thus, they play an important role in OPENNPAR in defining the properties and appearance of primitives.

Actions: When an action is applied to the scene graph, each node encountered in the graph implements its own action behavior. For each action, the database manages a *traversal state* (similar to rendering state in OPENGL), which is a collection of *elements* (or parameters) in the action at a given time. Typically, executing an action involves traversing the graph from top to bottom and left to right. During this traversal, nodes can modify the traversal state, depending on their particular behavior for that action.

[2]these groupings are not strict and are only used to highlight different Inventor classes

In some cases, a particular type of node does nothing for a particular action. For instance, a material node does nothing when a bounding box action is applied, but will set material elements when a rendering action is applied.

All processes (arrows) in our conceptual model (Figure 3.1) are performed within the render action procedure of a node. However, to maintain efficient use of modularity care must be taken to ensure that the render action procedures do not describe more than one process. For instance, in Figure 3.3 the 'myCylinder' and 'mySphere' are shape nodes that both describe 3D data *and* render 3D data during the traversal of a render action. Thus, these describe a process that modifies a primitive (looping arrow) and a process that transforms data to another primitive (an arrow drawn from one primitive to another, in this case the rendering process from surface to image). Whereas efficient for photorealistic purposes, these two stages must be separated so that the description of a scene can be accessed and modified by NPR algorithms (which often need access to normal and coordinate data) to influence output. Thus, all scenes in OpenNPAR are described using property nodes (such as coordinate and normal nodes that describe their respective coordinates and normal data), and all rendering is performed by shape nodes that do not encapsulate any additional data.

Elements: The rendering traversal state consists of a set of elements, each which can be altered by a given class of nodes. When a rendering action is applied, each element is used and interpreted in a specified manner. A few of the elements in the traversal state include: current geometric transformations, material components, lighting model, drawing style, coordinates, normals, lights, and viewing specification.

In OpenNPAR, primitives are represented by elements. Each element may be used to describe aspects of different primitives. For instance, coordinate and material elements can describe properties of both surface or stroke primitives.

Fields: Fields are structures that store parameters for nodes. They are always contained within nodes. However, they have additional functionality to simple parameters, since they: (1) provide consistent methods for setting and inquiring values; (2) provide a mechanism to detect changes to the database; and (3) can be connected to fields in other nodes.

Fields provide a powerful means of capturing processes between primitives. For instance, an image primitive may be connected to an image field in a node that describes a texture property for a surface primitive. Thus, when operations are performed on the image primitive, the field connection automatically captures the process of mapping the image primitive to properties of the surface primitive. In addition, field connections can propagate data back to an earlier part of the scene graph for iterative processing. Finally, fields allow for multiple inputs that come from different sources (e.g., *n* outputs from *n* nodes in different scene graphs can be connected into input fields of a single node for processing).

In summary, primitives are composed from elements, whereas processes are performed in the render action procedure in nodes during a rendering traversal. Nodes operate on primitives by accessing and modifying elements and fields. OPENNPAR's operation of OPEN INVENTOR limits each node to perform one individual process only, as depicted in Figure 3.1, during a rendering traversal. This results in small well-identified operations that improve the inter-operability of nodes. To distinguish typical node operations in OPEN INVENTOR from the disciplined use of nodes to support the conceptual framework for OPENNPAR we introduce the following definition:

Module: A module represents the use of an OPEN INVENTOR node that performs only one individual process on or between primitives during a rendering traversal.

In keeping with design goals of inter-operability, elements should be as generic as possible. Modules, on the other hand, can be numerous but succinctly defined. This allows groups of modules to operate on similar sets of elements, thus facilitating their combination.

3.1.3 Initial System

In this section we motivate the use of some initial components of OPENNPAR that form part of its core system for NPR algorithms. In addition, we provide examples for how elements and modules are added to the system and how they interact with each other in the system data flow.

An Initial Set of Elements

As stated in our design goals earlier, the number of elements should be kept to a minimum but provide a wide range of generic functionality in describing primitives to promote inter-operability between modules. We implemented a few initial extensions and additions to elements in OPEN INVENTOR to provide data structures that support our various primitives:

- *Winged Edge Data Structure:* provides surface connectivity information used to describe surface primitives and covers a large, and often very efficient, generality of use [Baumgart, 1975]. Almost all current NPR algorithms that compute on object data can be implemented using this structure.

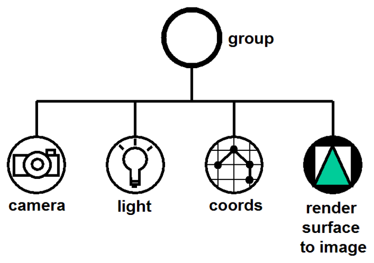
- *Images:* two-dimensional storage of pixel information. OPEN INVENTOR includes functionality for storage of byte data (e.g., used to store RGB values). Many NPR algorithms require storage of additional data in image buffers. Thus, we extend OPEN INVENTOR's image class to include floating point data so that it can store normals, depth information, and other *G*-buffer data [Saito and Takahashi, 1990].

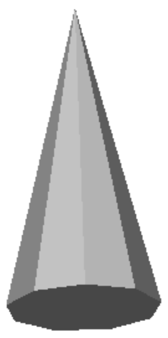
- *Strokes:* coordinate, material, and normal elements in OPEN INVENTOR can be used to describe points along a line or curve. However, strokes are more than just lines. Thus, we add elements that store thickness and orientation values to describe a stroke primitive. Existing elements like textures and texture coordinate data can also be used to define the stroke.

An Initial Set of Modules

Adding new elements to extend primitives requires the use of modules to either: (1) generate these elements from existing data, or (2) define data directly for pushing into the element state. Thus, in this section we describe how an initial set of modules useful for a large number of non-photorealistic algorithms are incorporated into OPENNPAR. In addition, simple examples are provided that highlight the interaction between modules and elements.

- *Render modules:* A standard means of viewing primitives is to render the various classes of primitives using OPENGL. Thus, we extend OPEN INVENTOR to include specialized OPENGL render modules for surfaces[3], strokes, and images.

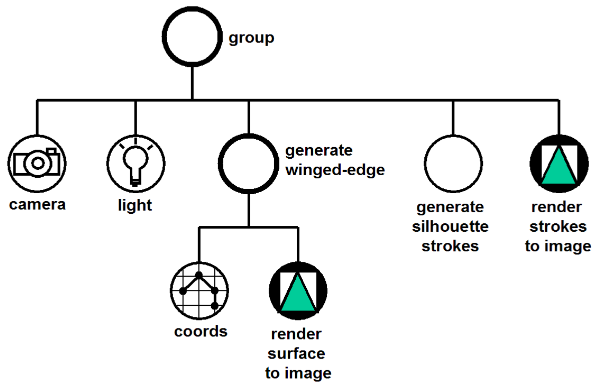


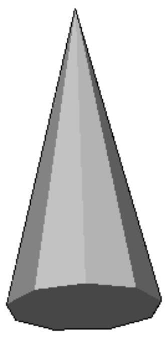


In the example above, a scene graph is created containing a camera defining a viewing specification, a light at the camera position, and a geometry module that defines surface coordinates and connectivity information. The surface render module is appended at the end of the scene graph. During a rendering traversal, the first three modules set their respective elements in the traversal state and the surface rendering module then reads the current state of surface primitive data and viewing specification to render the surface to an image.

[3]Standard shape nodes in OPEN INVENTOR already perform OPENGL rendering, but these are not modules (see Section 3.1.2 for details).

- *Surface modules:* OPEN INVENTOR includes many nodes that can be treated as surface modules. However, we add a module to generate generic descriptions of surfaces in winged-edge element data. This allows additional surface modules to be implemented that require connectivity information. The most widely used NPR algorithms that require such information are silhouette[4] generation algorithms (see [Isenberg et al., 2003]). Thus, we include a silhouette module that reads winged-edge element data to generate stroke element data.

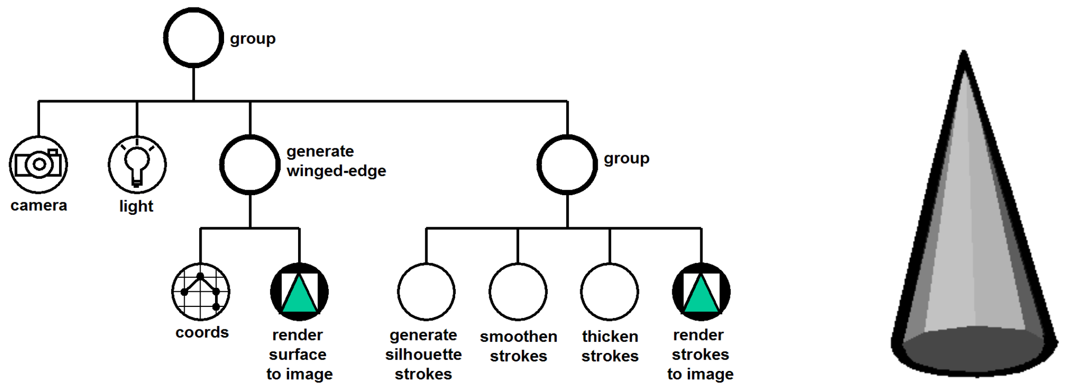




We modify our previous scene graph by grouping surface modules under a winged-edge group module. The winged-edge module generates winged-edge elements that describe the surface of all of its children. A silhouette module is inserted that accesses the generated winged-edge elements and the viewing specification to generate stroke elements that describe the silhouette of the surface. Finally, a stroke rendering module is added that renders the newly generated stroke primitive state to the image giving the final output.

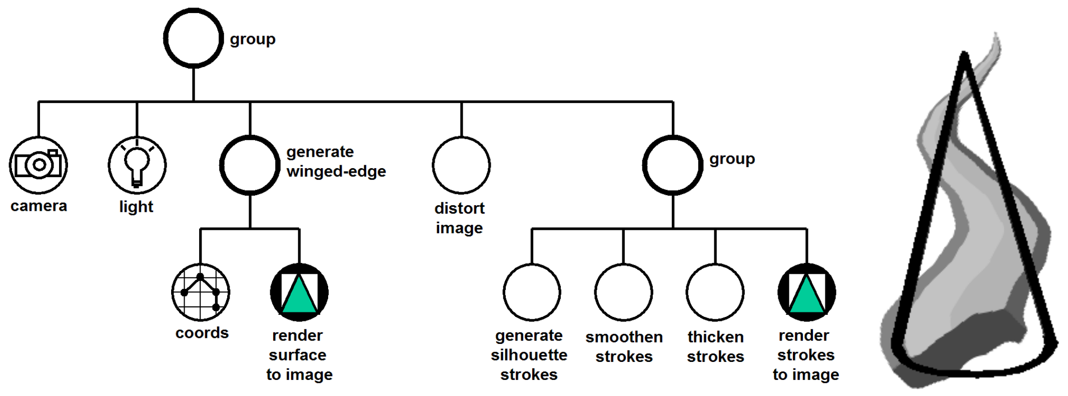
- *Stroke modules:* Strokes offer great potential for an expressive variety of styles. To offer this kind of variety, stroke modules are introduced that modify the stroke primitive state. Moreover, the effects of subsequent strokes modules can be cumulative. The initial set of stroke modules include: thickener modules (which introduce and modify thickness elements by either a constant value, depth, lighting, etc.), smoothen modules (which add coordinate elements along a Bézier path), stroke orientation modules (which add orientation elements that can affect thickness direction, such as calligraphy styles, or placement of points in smoothing a stroke), and perturbation modules (which influence coordinate elements to distort stroke paths).

[4]By silhouette we mean those edges of a polygonal surface that share a front and back facing polygon from a particular viewing direction.



In this example we add modules to modify the stroke primitive state. First, all stroke modules are grouped under a group node. Two stroke modules are inserted after the module that generates silhouette strokes—the first adds coordinates elements to smooth the stroke along a Bézier curve, and second then adds thickness elements for each coordinate element. Thus, the render stroke module subsequently reads stroke primitive data and outputs a thicker, smoother silhouette.

- *Image modules:* OPEN INVENTOR offers image classes to support surface texturing, but does not feature any image processing capability. We add image modules that comprise image input fields, on which to process data, and output fields to which the data is transferred. The output is computed during rendering only when the input field has actually changed value. If an image field is not connected for an image module, then the frame-buffer (to which strokes and surfaces are rendered) is assumed.



In example above, an image module is inserted in the scene graph before the stroke modules. The image module distorts the image present in the frame-buffer (since its input and output field is not connected) which at this stage comprises only the rendered surface of the cone. Thus, the stroke render module draws its strokes as before to the image, which results in a distorted surface rendering underneath the original surface silhouette.

3.2 Developing OpenNPAR

The developer's job is to add functionality to OpenNPAR by extending or creating new elements and modules. A primary challenge for the developer is to support inter-operability between modules and encourage their re-use. This can be done by adding a great number of modules, but constraining the functionality of each to compute a single, specific task. In contrast, elements in the system should be kept generic to maintain a small set that covers a broad range of applications. In this manner, a variety of modules can operate on the same set of elements which aids the interchange of data and resulting flow of computation. We now demonstrate these principles by showing examples of effective contributions to OpenNPAR.

3.2.1 Modularizing Stroke Operations

Our first example demonstrates how breaking down algorithms into elementary tasks adds flexibility and maintains modularity in OpenNPAR. Each elementary algorithm is encapsulated inside a module that can then be used independently or in combination with other modules.

We want to add functionality to OpenNPAR that enables the use of 3D stylized strokes within rendered environments. To do this, there are a number of problems to overcome. First, we require that stylizations be applied across long, smooth strokes. Second, we experience stylization artifacts when strokes are projected to the viewport from 3D. Our final problem is that stylization modules can potentially alter positions of strokes that then interfere with the scene.

We solve each of these problems with singular modules: (1) a module that connects strokes sharing common vertices; (2) a module that filters stylizations artifacts in projected strokes; (3) a module that performs fast hidden line removal (so that strokes could be rendered 'on top' of scenes). Details of these algorithms can be found in [Isenberg et al., 2002]. In terms of Durand's system classification [Durand, 2002], these three modules each comprise part of a primitive system.

By keeping basic functionality in separate modules, we can apply them in a flexible variety of combinations in addition to the problem for which they were implemented. Furthermore, they are simple to use since the results of each module are tightly defined. One or more of these modules are used in each of our programming examples in Section 3.3.

3.2.2 Modifying Elements for Surface Shaders

Often developers can add modules to manipulate elements in unconventional ways. As a result, existing modules designed to use these elements will produce output differing from their initial intentions.

For example, surface shading[5] is generally achieved by modifying elements so that subsequent rendering modules produce a different result. From a developer's perspective, a surface shader module simply sets values in elements used by rendering modules. For instance, we can add a module to modify texture coordinate elements based on lighting conditions. When inserted along with a texture module containing a two-tone image before the actual surface rendering module, we achieve the cartoon effect in [Lake et al., 2000].

We also take advantage of modularity present in modern graphics hardware programmability. To implement hardware features, we add elements that indicate current hardware options to use, and modules that load vertex programs and texture combiners to the graphics card. We leave the programmers to actually define what the hardware should do by allowing them to place code into a text field that is then compiled by the hardware modules. In this case, it is the hardware configuration that influences the subsequent output of rendering modules.

3.2.3 Re-using Elements for Skeletonization

As new algorithms are introduced, developers will often be required to support these using additional data structures. However, rather than adding new data structures for every new algorithm, algorithms can be mapped onto existing ones. This increases potential for alternative uses of elements and re-use of modules.

We demonstrate this by adding support for skeletonization in OpenNPAR, which is used in many NPR techniques (e.g., [Deussen et al., 1999; Raab, 1998]). The skeletonization process collapses edges in a surface definition. To do this, we make sure to load surface definitions into the winged-edge element using an existing winged-edge module. Now, we can add a skeleton module that computes using the winged-edge element. Rather than adding a new element to store the skeleton data, we leave the skeleton as represented by the winged-edge. These 'skeleton' elements can now be accessed by any subsequent modules in the scene graph. In addition, a separate module was included to generate a set of strokes from the winged-edge data. Now we can also view and manipulate the skeleton using available stroke modules. The stroke generation module can also be re-used on 'regular' winged-edge surface definitions. This would now produce strokes from the wire-frame of a surface with equal opportunities for manipulation and stylization.

[5]by surface shading we mean the combination of modules to create effects when the surface is rendered.

3.2.4 Extending Modules to Encapsulate Interaction

Developers are not constrained to extend OPENNPAR in terms of rendering only, but can also extend OPEN INVENTOR 's functionality for interaction. The functionality of OPENNPAR can also be extended beyond the representation and rendering of primitives. For instance, we can extend OPEN INVENTOR 's interactive functionality to incorporate a novel form of interaction for NPR by adding a surface module that evaluates whether or not object shadows on a plane are touched by the mouse pointer. To pass information about the shadow plane down the rendering pipeline, we added an element encapsulating the coefficients of the plane equation. Hence, the surface module can test for an intersection of the ray from the mouse pointer to its shape in the shadow plane. An application that uses this interaction method is discussed in Section 3.3.2.

3.2.5 Polymorphic Rendering

Developers can present a variety of modules used for the same intent, but which produce results tailored towards a specific application requirement. For example, different rendering requirements can be supported by OPENNPAR by increasing our choice of rendering modules that act on the same element data. In terms used by Durand, this would be different implementations of attribute data in the mark system [Durand, 2002].

In this case, we can implement a stroke rendering module that computes a particle simulation of paint along the course of a stroke (e.g., [Curtis et al., 1997]) and another that employs a hairy brush algorithm (e.g., [Strassmann, 1986]). These can take the same stroke elements as suggestions for using varying brush widths and sizes, speed, pressure, and so on. Furthermore, output does not need to be constrained to the visible frame-buffer. A module that takes strokes and generates a Postscript file is possible by translating the stroke coordinates and thickness elements into the required format. As another alternative form of output, we have implemented a module with an input image field that consequently produces a video file. This allows renderings to be captured ‘live’ or sequenced into a steady animation.

3.3 Programming with OPENNPAR

The programmer accesses the functionality of OPENNPAR with the understanding of how modules can be placed into a rendering pipeline to produce desired results. Currently, a programmer can construct content by editing a scene-graph description in a text file and viewing the scene, or by calling OPENNPAR’s API directly within an application.

In some cases, there are dependencies between modules and care must be taken to ensure they are placed in an appropriate order with their field connections properly setup. However, since modules in OPENNPAR are sufficiently succinct, relationships between them can be easily identified. Therefore, the real task given to the programmer is to exploit OPENNPAR’s range of effects and, at times, define new algorithms by coming up with novel ways of ordering modules and interchanging data. In the following subsections, we look at how a few examples of these are constructed through different means of using the OPENNPAR API.

3.3.1 3D Painter

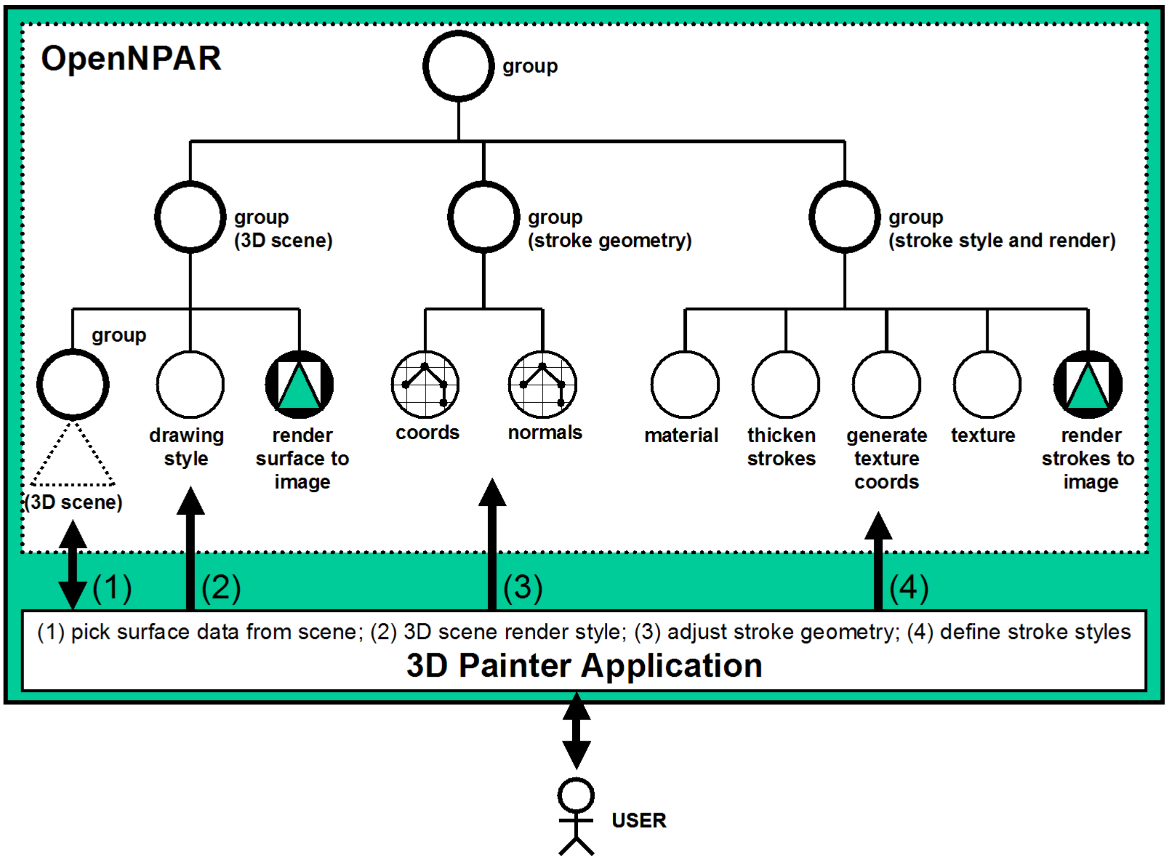
Our first example is simply an exercise in using the OPENNPAR API directly within an application. This program allows a user to import any 3D scene and directly 'paint' on its surface. We use OPENNPAR to render the painted strokes as well as accessing many of its standard operations such as picking, interaction, and shading methods. Figure 3.4 shows how the scene graph was composed of three main subgraphs: (1) the original scene on which to pick points and influence interaction presentation, (2) the stroke geometry that is constructed based on user interaction and picked points, and (3) a group which defines and renders strokes. The interaction technique used is much like the algorithm shown in [Kalnins et al., 2002] where strokes are formed in 3D by 'drawing' on the surface of an object. The difference here is that each point picked on a surface maps its surface coordinates, normal, and color (from material or texture) to the stroke point. Additionally, the application allows for an undo-history of stroke operations by directly accessing the field data of the relevant coordinate, normal, and material modules for stroke elements.

As a consequence of using OPENNPAR, the entire C++ code for this application (including comments) is less than 1000 lines. Despite this remarkably small program, it allowed an animation of the painted scene in Figure 3.4 to be created by a user in a very short frame of time when given the model.

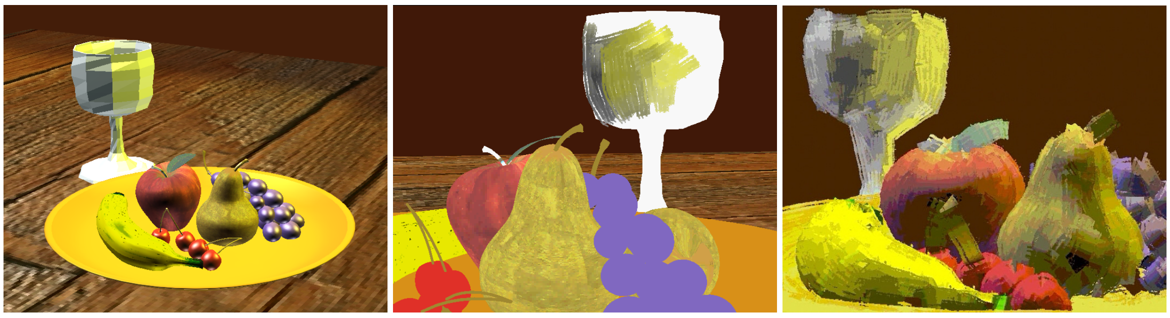
3.3.2 Interactive Illustration

Here, our computer generated illustrations make extensive use of non-photorealistic abstraction techniques to reduce the complexity of depicted structures. As the user interactively explores relationships in the scene, relevant details are emphasized whereas less important aspects are deemphasized or omitted to guide the focus of the viewer.

Figure 3.5 depicts the application of OPENNPAR modules to illustrate the current interaction context. Additional information about correlations between structures of a 3D model are displayed in shadows to enhance a user's contextual understanding. In addition to photorealistic modules, the programmer made use of OPENNPAR's modules that cast individual shadows on a plane to enable a special kind of interaction (see Section 3.2.4). We also see two alternative uses of modules: first, a silhouette module placed so that stokes are aligned around the outline of shadows, rather than the actual structures, and second, modules to derive skeletons were used to guide placement of annotation anchors. In addition to color, line styles applied to the strokes of the outlines emphasize the relevance of important structures.



(a) 3D Painter application system view



(b) View of original scene (c) Flat lighting during stroke input (d) View of fully painted scene

Figure 3.4: *3D Painter Application:* Textured strokes are used to 'paint' over models in this still life scene

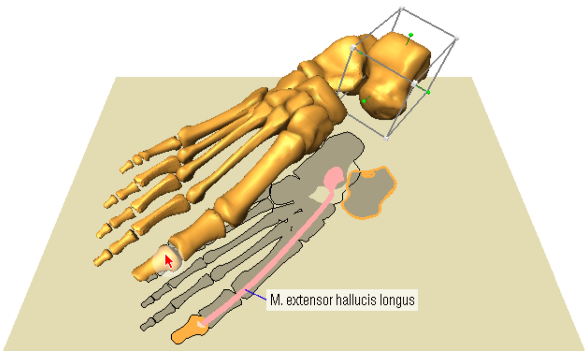


Figure 3.5: Illustrative Shadows (courtesy of Felix Ritter).

3.3.3 NPR in Games

Figure 3.6 shows a sample scene from a game prototyping tool in OpenNPAR. Since games demand real-time, the programmer makes optimal use of the programmability features of hardware surface shader modules. We see examples of cool-to-warm shading, cel shading, stroke textures, and colorized hatching.

The cool-to-warm shading [Gooch et al., 1998] is achieved by defining the vertex programming module to honor material colors of surfaces. As a result, this can be combined with a material module and is used for the teapot and environment. Colored hatching also reflects an object's color. However, a texture combiner module is added so that the interpolated diffuse color is blended with white based on a gray-scale image defined by a texture module. The cel shading is achieved by instructing the vertex programming module to generate texture coordinates for a one dimensional gray-scale texture based on the angle between a surface normal and light vector. This is combined with a texture combiner module programmed to use this texture to darken or lighten the object's base color. The black-and-white stroke textures are rendered as described in [Freudenberg et al., 2002].

3.3.4 Using an External Application for Stippling

Programmers of external applications can still use OpenNPAR's features. We take the case of frame-coherent stippling, designed and implemented outside of OpenNPAR. This is a stippling technique where most of the computation occurs as a pre-processing stage that structures a point hierarchy before the points are selectively rendered at run-time.

The point generation is done off-line independently from OpenNPAR. The output of this stage, however, can be written to a file which complies with one of OpenNPAR's file formats that defines the stipples as 'point' strokes. While the stipple generation stage still lies on the side of the producer, the rendering part now relies on OpenNPAR. Thus, potential for additional rendering features in OpenNPAR, such as its silhouette generation modules, can be used to generate the final image (see Figure 3.7).

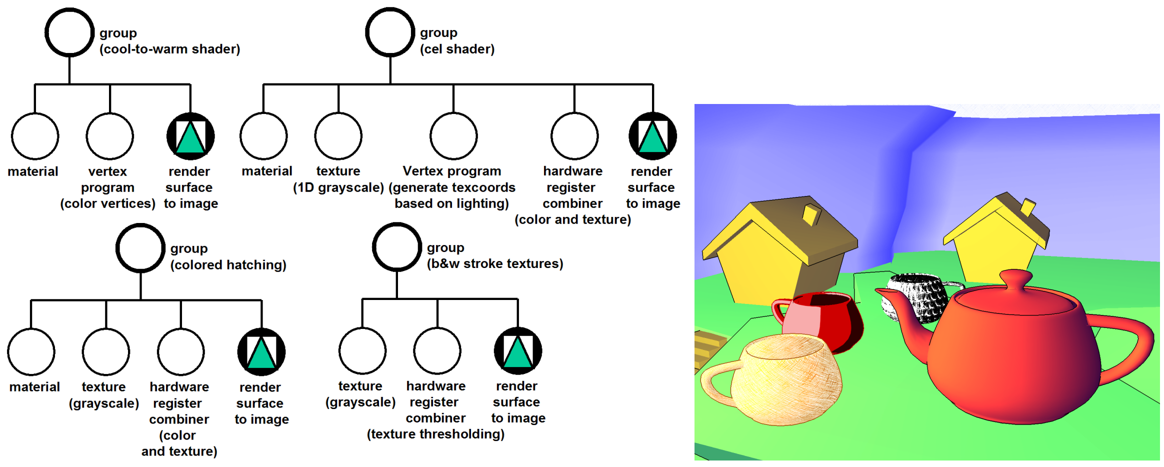


Figure 3.6: Modular surface shaders integrated into a game environment (image courtesy of Bert Freudenberg).

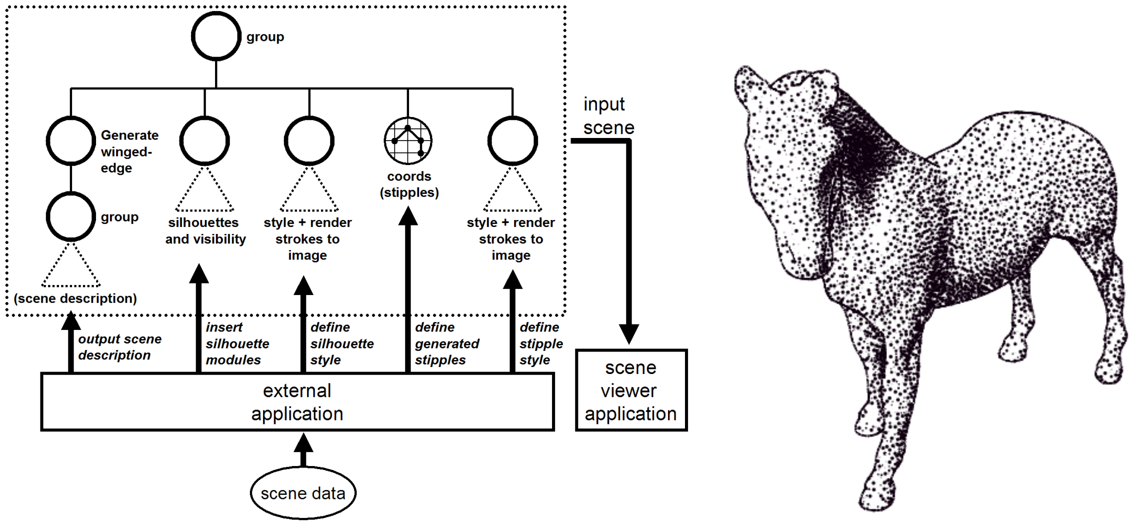
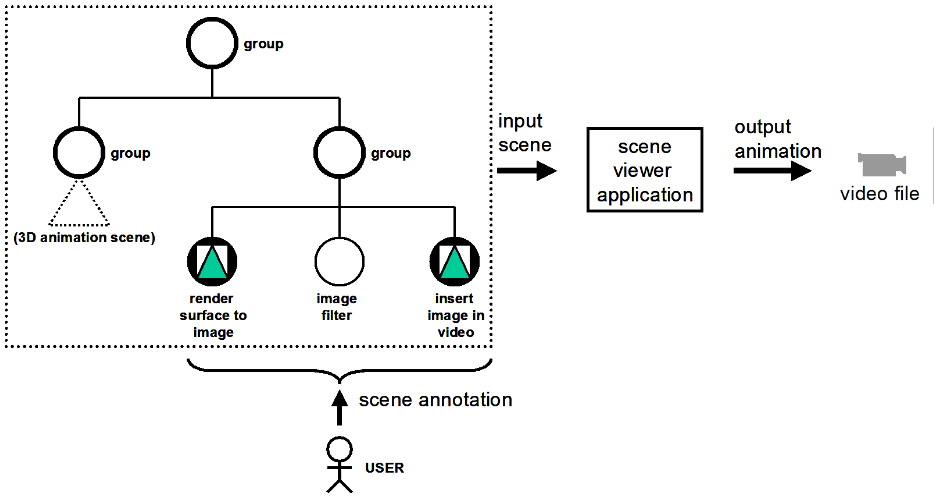


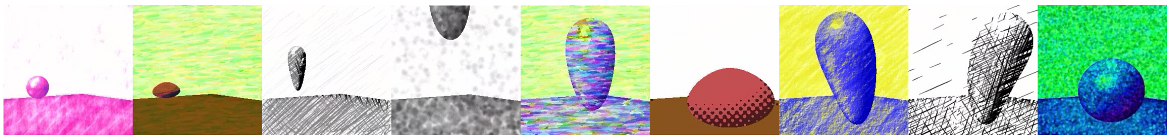
Figure 3.7: The external application outputs a scene description and stipple points in OPENNPAR's file format. A viewer application then renders the stipples and silhouettes (image courtesy of Oscar Meruvia).

3.3.5 Editing an Animation Description to Export Filtered Video

An external 3D application can output an animation to a file that is then marked up by the programmer to produce a video including a few special effects from OPENNPAR. No compilation is necessary—a simple text editor and knowledge of OPENNPAR's API format is required.



(a) Annotating a scene description text file to output video



(b) An animation played back with nine different image filters for artistic effect

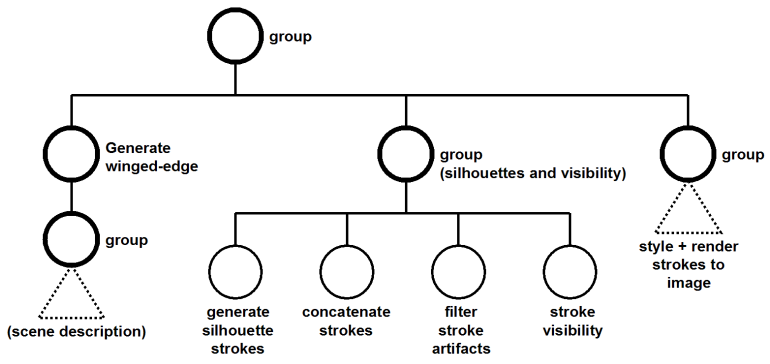
Figure 3.8: Using OPENNPAR to produce filtered video effects for a 3D scene animation

OPENNPAR is able to read in VRML files that include additional animation nodes (currently we support vertex interpolation and group translation/rotation). The programmer edits the VRML scene description to add an image module at a specific location to read in contents of the frame-buffer once the initial rendering of an animation frame was complete (see Figure 3.8). A variety of image processing modules were then appended to the scene graph. The effects for the image processing modules are controlled by their input image fields—in this case by typing in external image file names to use as filters—and connecting their outputs and inputs to propagate results. The last image processor module's output was connected into the image field of a video module, which was parameterized to insert images at specific time intervals to a video file. Producing this file was now simply a matter of using an available viewer application to read in the scene which would automatically run the animation.

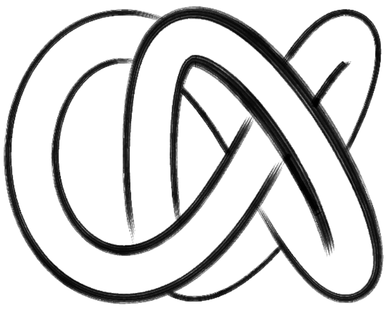
3.3.6 Creating Level-of-Detail Silhouettes

Clever use of modules often allows programmers to define new algorithms. For this example, we construct a silhouette algorithm to generate stylized silhouette strokes at interactive rates that also uses a level-of-detail technique. As in the previous section, this can be achieved entirely through the use of a text file without need for recompilation.

We first insert the silhouette generation module that fills the stroke pipeline—the stroke elements—with silhouette edges. The stroke rendering module, at this moment, would simply produce thin lines for the silhouettes since no stylistic elements of the strokes have been used. To this effect, three stroke modules to support the 3D stylization of strokes (that we introduced in Section 3.2.1) were added directly after the silhouette module (see Figure 3.9(a)). Now, we can add stylization modules to affect the visual appearance of strokes and render these by disabling the z-buffer field in the surface renderer so that clean strokes are drawn ‘on top’ of the scene. The stylization achieved in Figure 3.9(b) combined stroke modules to influence stroke thickness elements, texture modules to read RGBA images from an external file, and a texture coordinate generator module to define the mapping of the texture onto each stroke.



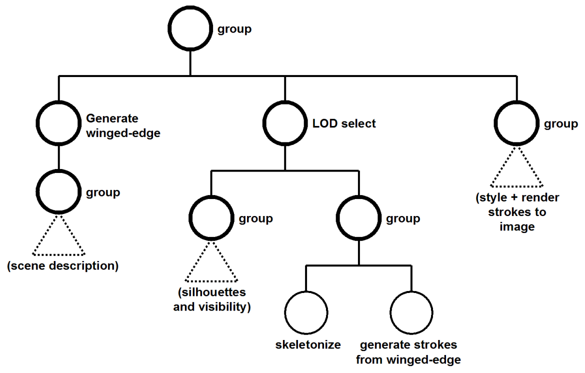
(a) Scene graph for stylized silhouettes



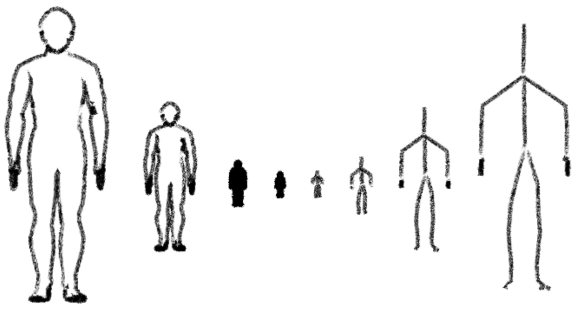
(b) Output of scene graph

Figure 3.9: Visible silhouettes rendered in real-time with oil paint texture and depth cuing on thickness of strokes

For reduced level-of-detail (LOD), we simply replace the silhouette generation module with a skeletonization module and load the stroke pipeline with the skeleton data (see Section 3.2.3). This would similarly concatenate the skeleton edges to strokes and apply stylization to it. We notice that the skeleton is effective in conveying a good idea of the shape of an object when the object is far away (see comparison in Figure 3.10(b)). In addition, it proves computationally efficient due to viewpoint independence and typically generates less lines to convey the whole object than the full silhouette. To combine these two results, we add a standard Open Inventor LOD group node that selects between silhouette generation and skeletonization modules depending on the projected size of objects (see Figure 3.10(a)). Thus, when this scene is now loaded into a viewer application, the stylized silhouette of an object is replaced with its skeleton as it recedes into the distance.


group
Generate winged-edge
LOD select
group
group
(scene description)
group
group
(silhouettes and visibility)
(style + render strokes to image)
skeletonize
generate strokes from winged-edge

(a) Scene graph for LOD selection of silhouettes or skeleton



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

Figure 3.10: Combining silhouettes and skeletons for real-time stroke representations of objects

3.4 Designing with OpenNPAR

Designers are given the task of creating visual results that carry a desired communicative intent. Whereas the programmer has the technical expertise to experiment with the system at a modular level, the designer is much more productive when part of an entirely visual and iterative creative process.

However, OpenNPAR at the modular level still imposes knowledge requirements about the inter-operability of modules which can potentially hinder the creative process. For instance, knowledge of the order in which modules can be inserted into the rendering pipeline and type conversions between field connections, in addition to modules relying on others to be previously inserted.

As a consequence, we devised an interface to overcome these impositions on designers by mimicking the designer's creative process in coming up with new images [Halper et al., 2002]. The contributions to the interactive design of effects are as follows:

- A *method of interaction* which leaves the user unaware of the dimensionality of the data and type-conversion being used to create a given special effect, and
- A *method of computation* which assembles a unique pipeline of graphical operations to achieve the desired special effect.

This gives designers freedom to experiment interactively with images through the use of *modifiers*. In Section 3.4.1 we look at how the designer can use modifiers at a level that allows them to intuitively create a broad variety effects that encapsulate entire algorithms. This in turn outlines requirements for modifiers in achieving the functionality and ease-of-use desired by the interface. Thus, modifiers abstract technical knowledge from designers about module use in OpenNPAR, which is detailed later in Section 3.4.2.

3.4.1 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.

The Designer's Approach

Some of the key issues in the design process that will influence the user interface 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 designer's 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.

The General Interface

Here we look at which important features a system interface must comprise in order to stay close to the designer's 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.

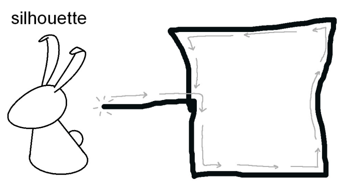
An effect or variation is something that when applied, either adds to or alters the rendering style. Throughout this paper, these are also referred to as *modifiers* that a user may work with to influence the rendering output.

As a general rule, we do not want to take the users attention away from the sketchpad and break the fluidity of the interaction. Overlaid 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.

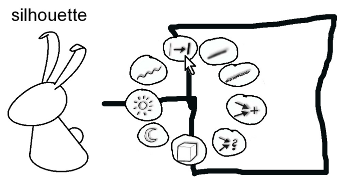
Interaction Tasks

The designer can insert a scene or image into the sketchpad by clicking on any empty space. An import modifier is then added to the sketchpad at the appropriate location. 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 3.11(a).

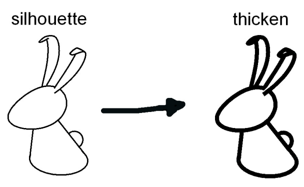
The approximate size of the box determines the size of the new image to render. If no box is sketched, then the same size as the previous image size is taken. Once the user releases the pen or mouse button, she may decide which effect or variation to apply to the image, as shown in Figure 3.11(b). Here we use pie-menus [Hopkins, 1991] for several reasons. The number of available options to the user can be large, from anywhere between



(a) Sketching a new variation. The gray arrows indicate a users path of motion from the point of pressing the button to its release



(b) Selecting a variation from a pie-menu.



(c) Final result of applying a variation

Figure 3.11: Interaction Method of Applying Modifiers

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 [Callahan et al., 1988]). Figure 3.11(c) then shows the result of these simple steps.

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 (see Figure 3.12).

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

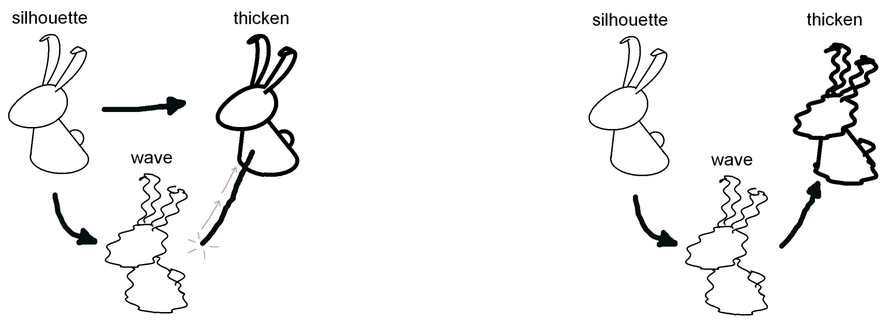


Figure 3.12: Changing an input by sketching the new connection as indicated by the gray arrows (left). The old connection is automatically removed (right)

current state to a previous one. Many design books (e.g., Shneiderman [1992]) 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). Such a visual representation of the history of actions can greatly facilitate the user in making appropriate changes or fine-tuning results [Kurlander and Feiner, 1992].

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 re-using 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, [Schumann et al., 1996]). The connections created by our system are also abstracted to the user so that the visual display of the data-flow to the user is not necessarily the same as that undertaken by the system.

For example, in Figure 3.13 we see an applied sequence of modifiers. The visual results of each modifier are consistent with designer expectations. However, from an algorithmic point of view we notice that certain computations in this order are actually infeasible. For instance, the computation of the base color surface modifier cannot be done given a 2D image result (from the lighten modifier) as an input. In the next section, we show how this is made possible.

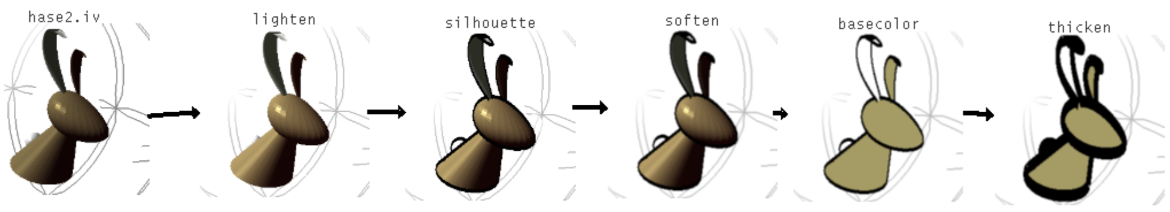


Figure 3.13: Designer's view of piping modifiers.

3.4.2 Supporting the Designer's Interface with OpenNPAR

As discussed in the introduction, the tools that are created by programmers for designers are called *modifiers*. In this section, we detail the implementation of modifiers and their integration into the interface that serve to abstract technical knowledge about OpenNPAR from the designer.

Modifiers

A modifier *manipulates*, *adds*, or *removes* modules in a scene graph. In order for a modifier to adapt to our interaction method, 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 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.

In this respect, modifiers provide a simple means for designers to interface OPENNPAR, whereby the modular components have been abstracted by the programmer into effects that produce *results*. In addition, each modifier is limited to use only its class of modules (e.g., a modifier cannot affect both a surface-shading module and an image module in the scene graph). We do this because: (1) this makes implementing modifiers quick and easy; (2) elementary building blocks are vital for flexibility and power in more complex structures.

Here, we give a few examples of simple modifiers that use various modules:

- *Surface Modifiers:*
  - *Light Intensity Filter:* Parses an input scene and sets all material modules to use a white, non-specular material. On its own, this renders a gray-scale image corresponding to scene illumination.
  - *Base Color:* Changes the lighting module in the rendering pipeline to use base color illumination. On its own, this would effectively render colors based only on material diffuse properties.
- *Stroke Modifiers:*
  - *Silhouette Generation:* Adds a silhouette module that computes a set of silhouette strokes.
  - *Stroke Thicken:* Adds a module to influence stroke thickness.
  - *Stroke Waviness:* Adds a module that applies a wave function over 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 defining a dither-matrix and source image, and applies half-toning to produce an output image comprising only black and white pixels.

  - *Selector:* Takes as input three images. One input image defines threshold values that select corresponding pixels from the other two images in generating the output image.

Piping Modifiers Together

Each modifier that the designer applies is computed by first structuring a *graph of modifiers* from its inputs. Then, the applied modifier is inserted into the appropriate location within this graph of modifiers. Its field connections into other modifiers are then organized (or re-organized). If field connections must undergo conversion into the correct types, we make sure that the relevant conversion modules are present in the scene graph and use them. Thus, each modifier maintains an internalized ordering of the application of previous modifiers that serve as inputs to it.

The rules for inserting modifiers into a graph of modifiers are outlined below:

- Import modifiers are placed first in the dataflow, or after any already existing import modifier.
- A surface-shading modifier is appended after the last surface-shading modifier in the dataflow. In the case of no previous surface-shading modifiers in the dataflow, it is placed directly after the import modifier.
- A stroke modifier is appended after the last stroke modifier in the dataflow. If there is no stroke modifier already present, it is placed after the last surface-shading modifier if there is one, otherwise it is placed after the last import modifier.
- An image modifier is appended at the end of the modifier dataflow.

The end result is a dataflow of modifiers that start with import modifiers. These introduce a scene graph that is first filtered through surface-shading modifiers and then stroke modifiers before converting 3D data into 2D images for input into subsequent image modifiers.

In Figure 3.14 we see the actual translation of the designer's visual dataflow (from Figure 3.13) to the underlying system's dataflow. Whenever the designer applies an image modifier to the sequence the system first checks if 3D information must be converted to an image. For instance, when the lighten modifier is added, a conversion module is inserted that renders the imported scene into an image. Stroke modifiers are placed after surface modifiers so that when the designer applies a silhouette modifier to generate strokes from the lightened result, the system places the silhouette modifier directly after the last available 3D information in the pipeline (which is after the imported scene), and then a re-conversion to a 2D image is necessary to be fed back into the lighten modifier to give the appropriate combined result.

Once the modifier pipeline comprises each of the modifier classes, we simply append new modifiers to the last modifier in its class, undergo conversions for the subsequent classes,

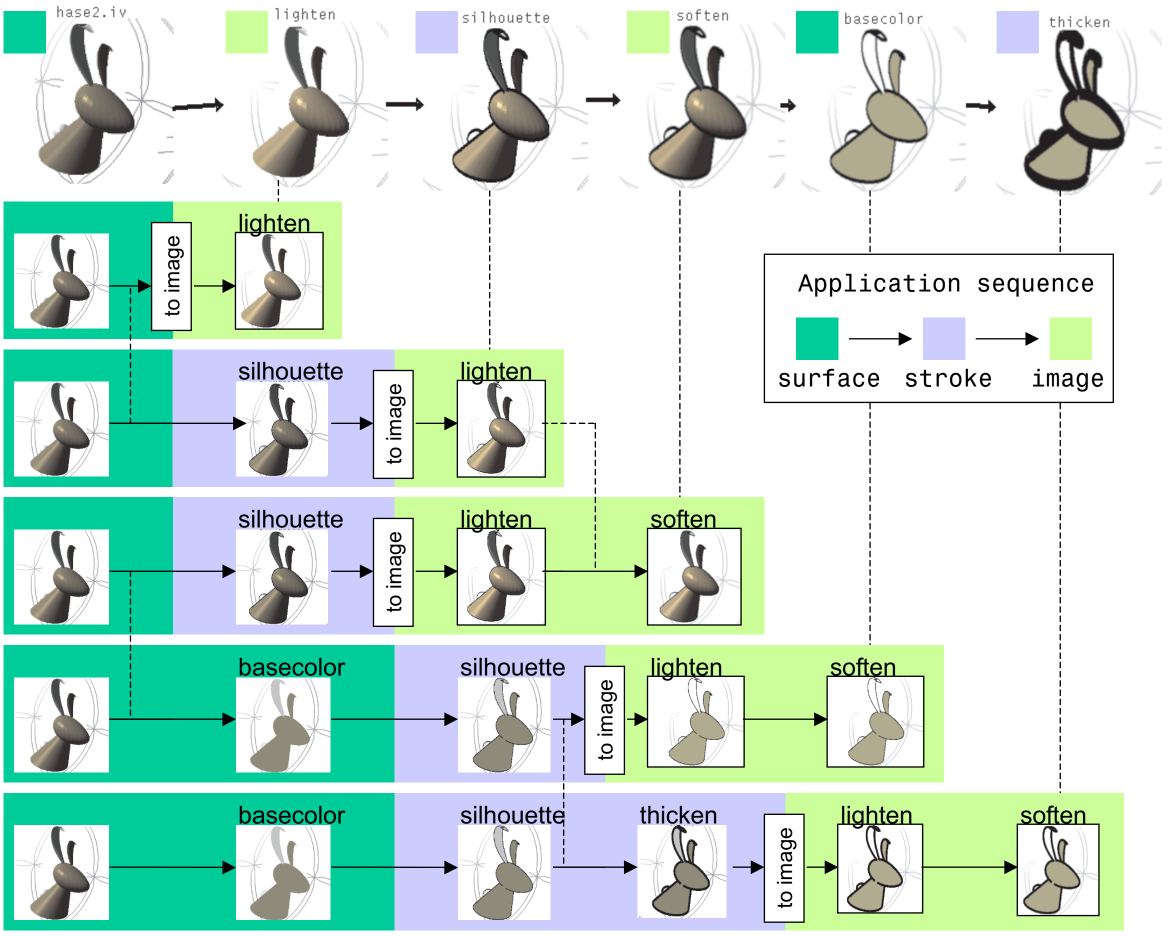


Figure 3.14: System view of piping modifiers. Each of the displayed results at the top store a graph of modifiers (in this example a linear sequence) applied to the original scenes. Computational short-cuts are indicated by the dotted lines between successive modifier graphs.

and recompute the modifiers that lie later in the sequence if necessary. The dotted lines in Figure 3.14 from a previous pipeline to the next indicate a computational shortcut for improving efficiency. For instance, in the case of applying the stroke thicken modifier, we can start the new computation after the silhouette modifier from the previous modifier graph; thus, we only need to re-convert the data to an image and reapply the lighten and soften modifiers.

3.4.3 Interaction 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 can be created.

An Interaction Sequence

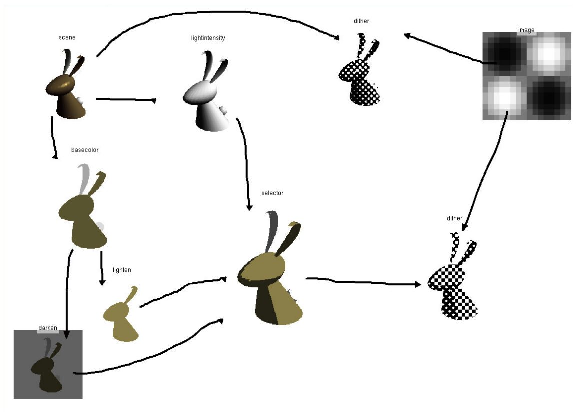
In Figure 3.15(a), the user has imported a 3D model that is rendered using PHONG shading and displayed in the upper left corner of the sketchpad. From this, the user has found a way to create a cartoon effect by using a selector that takes the light intensity of the model to choose whether or not to 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.

In Figure 3.15(b) 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 immediately 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.

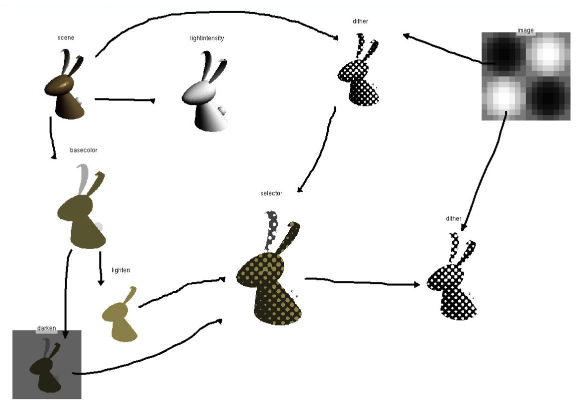
The user has continued with the same sketchpad and played with the interface to give a variety of results as shown in Figure 3.15(c). The modifiers from the first two examples can still be seen in the upper left section of the sketchpad. Note that in all these examples, all results are produced by only using one imported image (the dither matrix), one imported scene, and nine elementary modifiers.

Comparison of Interaction Methods

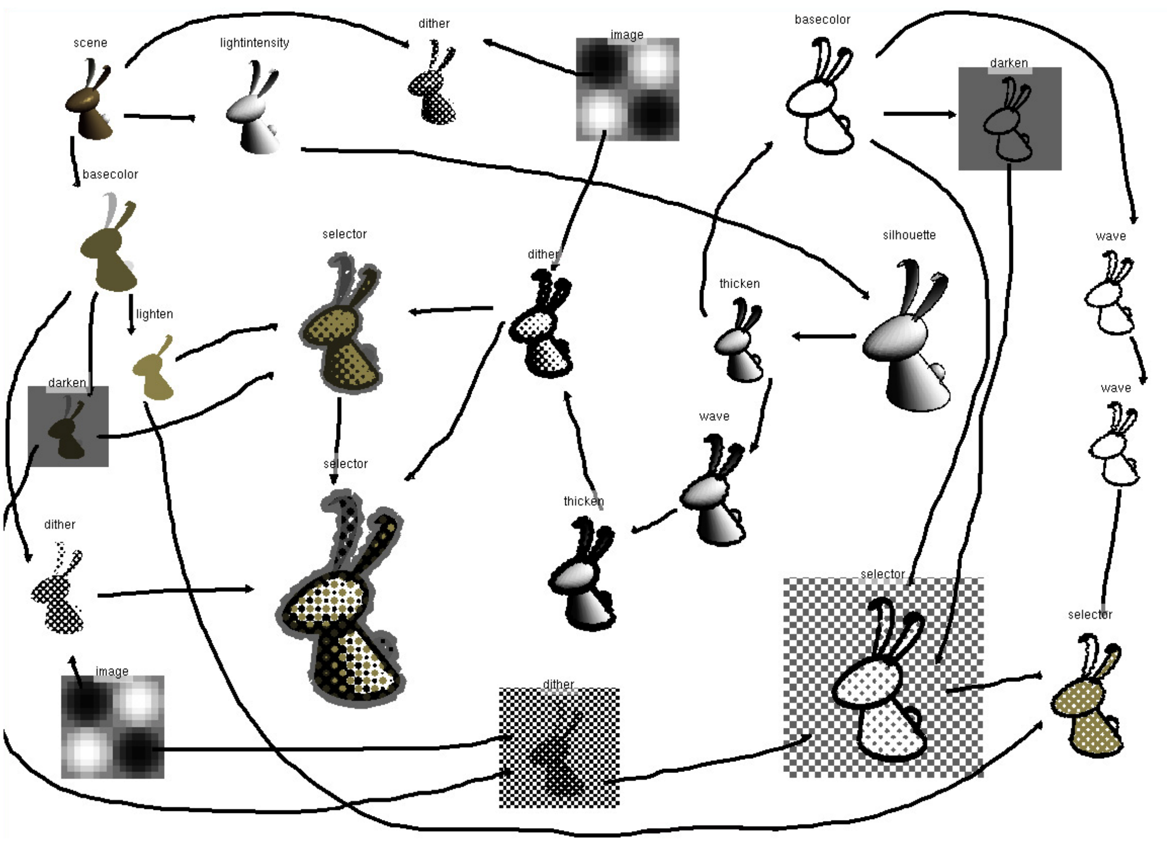
In Figure 3.16 we see a comparison of the sketch interface with and without the abstraction of modifier application order from the underlying system. In each of the sequences, the *introduction* order of modifiers was *the same* (i.e., in both sequences the scene was first inserted, followed by the dither modifier, followed by use of the base color modifier, etc.).



(a) First stages in the development of effects

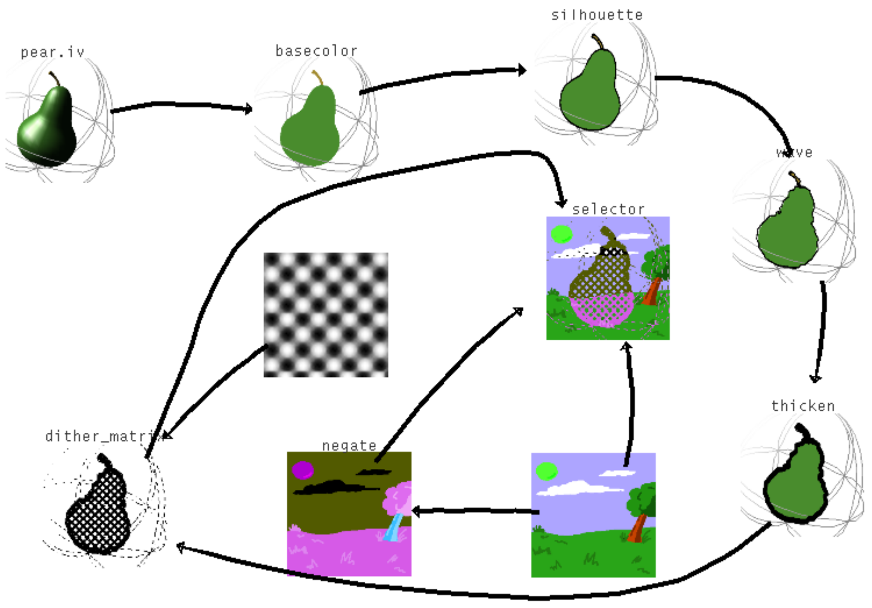


(b) Changing an input and propagating effects

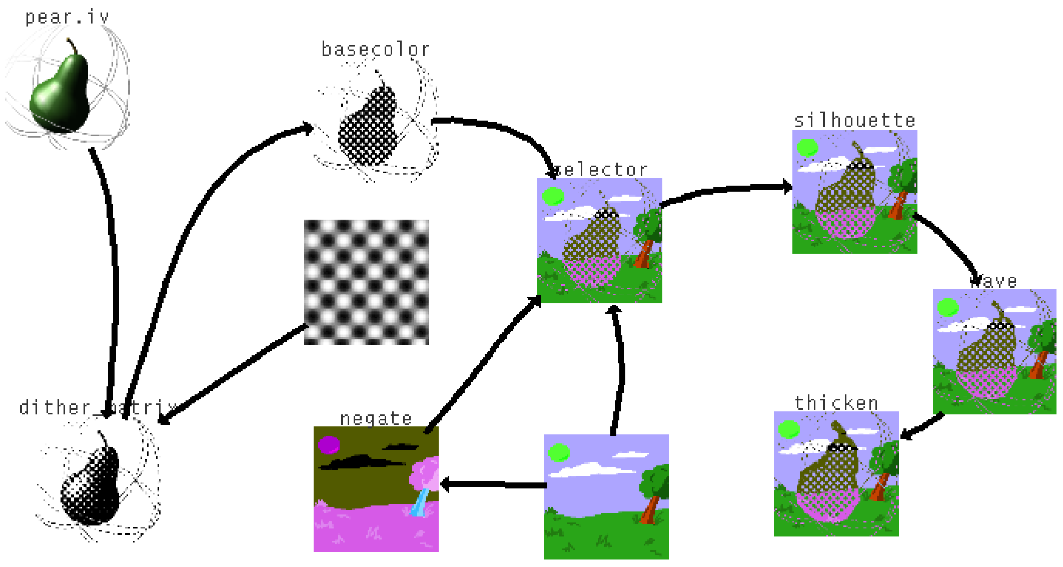


(c) Zoomed out view of the sketchpad after the user has played with it for some time

Figure 3.15: An interaction sequence of applying modifiers



(a) user must be aware of application order of modifiers



(b) interface maps user intention onto the underlying system order of application of effects

Figure 3.16: Comparison of two interaction results given the same introduction sequence of modifiers. Notice that below, we see a linear progression of effects, whereas above, the user has to backtrack to apply effects at appropriate insertion points.

Figure 3.16(a) shows a result in which the user must be aware of the application order of surface, stroke, and image modifiers. Here, the user often had to find a correct insertion point in the flow of modifiers, especially when modifications to the silhouette style were desired. Furthermore, whenever a modifier was inserted additional overhead was required to connect its output into the rest of the sequence. Eventually the spatial organization of modifiers in the sketchpad forced the user to start adding modifiers around the final image of interest (the select modifier in the middle). In contrast, in Figure 3.16(b) the interface maps the user's intention onto the underlying system order of application of effects (as detailed in Section 3.4.2). Here we clearly see a *linear* progression of effects—the silhouette modifier was applied after the selector modifier, and its silhouette line stylization appropriately followed in sequence. This example is best visualized with the accompanying video.

Creating Complex Effects from Simple Modifiers

Programmers can implement a very limited set of modifiers in a short space of time. Yet even with this limited set, enough functionality is provided so that designers can come up with interesting and diverse effects. 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. This is made possible by the modular capabilities of OPENNPAR.

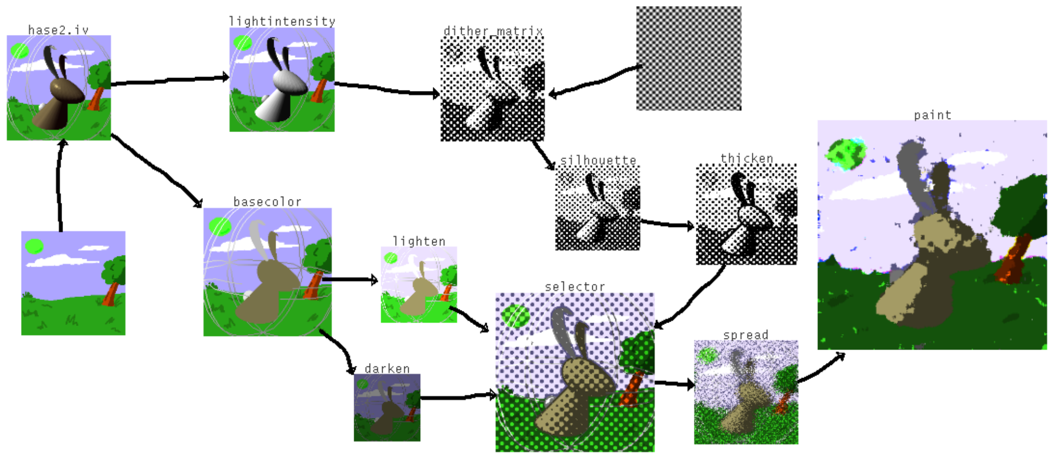


Figure 3.17: Producing complex effects from modifiers that manipulate modules in a rendering pipeline

For example, the programmer introduces two new modifiers: an image modifier that randomly spreads pixels over the image, and an image modifier that 'paints' by evening clumps of colors in the image. These took little time to implement. Nevertheless, the designer can quickly experiment with these new modifiers to come up with a 'sponge-painting' effect shown on the right of Figure 3.17.

3.4.4 Re-Use of Graphs of Modifiers

Our examples have shown that modifiers can produce a variety of results. In this section we provide a means for saving these results for re-use. In essence, modifiers are self-contained effects that encapsulate a history of operations as a graph of modifiers. This lets us easily export and import combinations of modifiers, called *compound* modifiers, for re-use. We demonstrate this with a few examples that combine both 2D and 3D effects.

In Figure 3.18 (left) we show how to construct a composite effect from an image modifier (*select*) and a surface modifier (*black*). Note that output from the import modifier could be any class of modifier. In Figure 3.18 (right) we collapse the circled modifier and rename it to *composite*. Its inputs are then mapped as *foreground* and *background* into the appropriate dataflow.

In the next example (Figure 3.19 (left)) we combine surface and image modifiers to create a cartoon effect. Circling the modifier indicates that we want to save this effect into a new modifier, that we rename to *cartoon*, which is re-used in Figure 3.19 (right). Finally, Figure 3.20 shows the combination of both the composite and cartoon modifier that integrates all the 2D and 3D effects.

To compute the saving and renaming of modifiers into new modifiers we count the number of unique import modifiers that enter the modifiers' dataflow encapsulation. For instance, the composite example (Figure 3.18 (right)) has two unique import modifiers that form part of its encapsulated dataflow, therefore the newly created composite modifier has two inputs that can be renamed (as they are in the example). The cartoon shader (Figure 3.19 (right)) has only one unique import modifier entering its dataflow, therefore requiring only one input.

3.5 Discussion

We have presented OpenNPAR, a versatile and flexible system that addresses a major gap in the creation of NPR and animation—namely a unifying framework that combines many NPR techniques accessible by all user levels. The underlying intent behind NPR is to optimize the communication of a visual goal in a specific context—the scope, thus, is virtually endless. 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. Consequently, OpenNPAR offers potential for defining an effective method within such a scope. This was made possible by structuring OpenNPAR onto a conceptual framework for NPR that categorizes algorithms and primitives to support the interchange and re-use of data.

A 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. However, we hypothesize

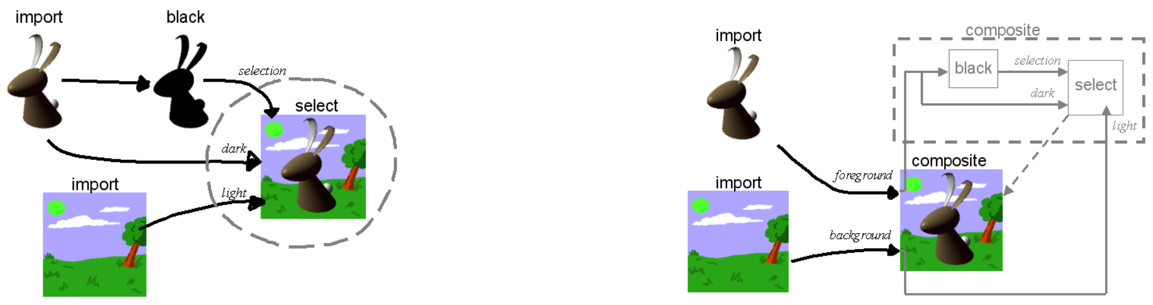


Figure 3.18: (left) Combining surface and image modifiers for a composite effect. (right) Encapsulating dataflow and remapping inputs to produce a compound modifier called *composite*.

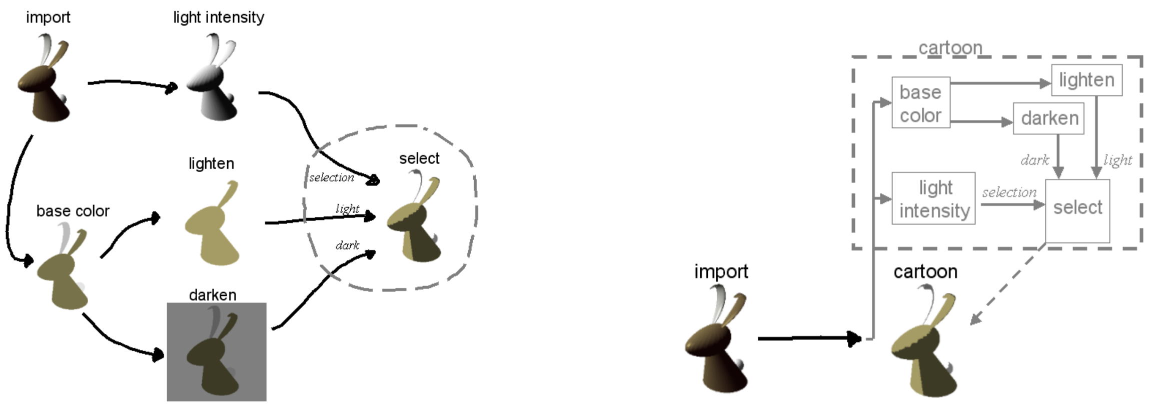


Figure 3.19: (left) Combining surface and image modifiers to produce a toon effect. (right) Encapsulating dataflow in the select modifier and remapping inputs to create the compound modifier *cartoon*.

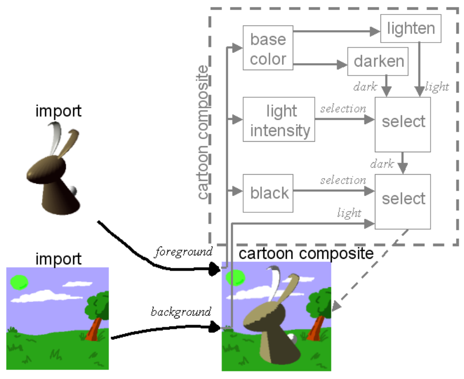


Figure 3.20: A *composite cartoon* modifier which combines the *composite* and *cartoon* modifiers.

that any existing NPR algorithm 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.

Development principles have been outlined and demonstrated using examples to effectively extend OpenNPAR's functionality. Programmers can access this functionality either by linking an application directly to OpenNPAR or through textual descriptions of modules in a rendering pipeline. Due to its modular structure, pre-defined effects can be reproduced or entirely new ones created through the manipulation of interchangeable modules. These modules need to be correctly arranged in the rendering pipeline to comply with the system requirements of dataflow. In the past, designers of special effects have had to manage the data with respect to its dimensionality and handle the sequence of operations explicitly. In this chapter, we have given a detailed account of a new method of achieving visual effects. It relies on the programmer to define a *graph of modifiers*. The designer can then experiment with NPR effects by editing the graph. The key to the method is that the dimensionality of the data no longer need concern the designer. This is achieved by a method of computing a unique pipeline of graphics operations for each node in the graph.

The method relies on a dataflow system but with two major differences:

- Computation may be carried out in different order than the dataflow diagram would visually suggest;
- Modifiers presented to designers as images encapsulate entire dataflow system processes which can be re-used by applying them on new inputs for animation or alternative scenes.

Using this method, we have provided a user interface that enables and encourages designers 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. The designer may take any image in the creative process, and start a new direction of ideas. Possibilities are left open to reduce limitations imposed on designers—modifiers can be linked together in arbitrary ways, regardless of how inappropriate the connections may seem. Results may not quite be to the designer's liking, it might not even be what was expected, but we decided not to give the power to the programmer to constrain how the designer chooses to combine effects. Instead, designers are left to make their own mistakes, with the hope of providing inspiration throughout the course of the creative process.

In conclusion, OpenNPAR holds vast potential as a tool for the development, augmentation, and creation of NPR. Further information about features in OpenNPAR as well as animations and sample applications can be found at `www.opennpar.org`. Many questions and directions for future work arise from the work shown in this paper. We outline a few of the main areas below.

3.5.1 Significance to Computer Game Development

OpenNPAR offers a top-down approach to the creation of NPR effects by giving designers control in defining which algorithms should be used, and offering programmers and developers the option of optimizing the implementation for the results. Designers are able to declaratively specify effects to apply without concern for algorithmic type restrictions or organizational issues of controlling dataflow. Whereas this interactive method sacrifices some control, we do propose that designers, at least in the experimental stage, should be unencumbered by interaction overheads. In practice, there is also a lot of cross-over and communication between user groups. Thus, we suggest allowing more control in subsequent stages of designing effects. For instance:

1. Designers come up with the initial look, working with the range of available effects and combining them in ways that give effective results, with a primary emphasis on experimentation. In this stage, designers can independently concentrate on visual goals.
2. Additional control for effects in the created pipelines are given. This would allow tuning of visual results and using functions to map effects onto more complex structures. Depending on the experience of designers, they can either continue to work independently or with more technical knowledge support from others.
3. At this stage, a pipeline has been created with configurable parameters built from a set of simple modules. These modules can then be integrated into an optimized pipeline (e.g., combining a number of effects in one pass). This is left to the experts of the system.

As a result, programmers define *what can be used*, and designers tell programmers precisely *what should be used* and *how*. Thus, designers are no longer inhibited to work with rigid rules, and programmers take their knowledge of how things really work to structure a more efficient rendering pipeline.

3.5.2 More Powerful Modifiers

Modifiers could be extended to not only allow a variation to be applied to a certain state but also provide more powerful features. For instance, a modifier could compute based explicitly on its inputs, such as connecting two entire separate sequences of modifiers. Such a combiner modifier could, for instance, connect a cartoon-shader and a pencil-sketch shader as shown in Figure 3.21. This might also require an intelligent selection of parts of an image or scene to which the effect can be constrained.

Another powerful feature would be to be able to directly interact and sketch ‘into’ each presented image at any given stage in the sketchpad, as in, for example, the WYSIWYG system by Kalnins et al. [Kalnins et al., 2002]. This would let us vary parameters, or

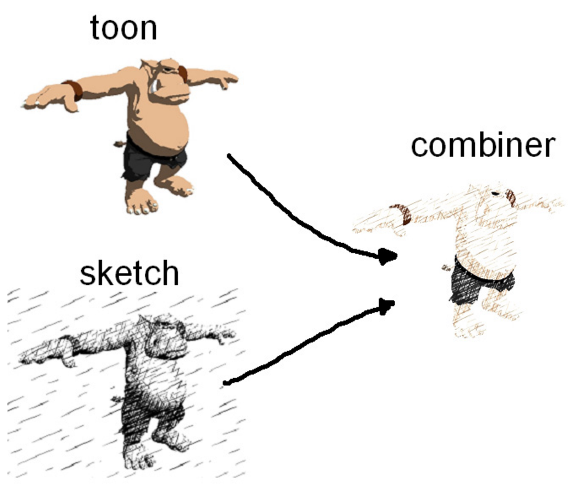


Figure 3.21: An intelligent combiner modifier

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 the user can 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 which opens potential for distraction. A direct manipulation by, for instance, sketching the desired result of a parameter change, would greatly enhance the interface.

3.5.3 Improving Scalability

Many interactive systems experience problems with scalability. We currently provide only two methods to address scalability issues in the design of effects: (1) the feature of sketching out from any image to directly apply further effects, and (2) the grouping mechanism.

Problems sometimes arise when combining many multiple-input modifiers. In this case, it is not clear where new modifiers should be inserted into which previous chains of modifiers. Appropriate insertion points can also be dependent on the actual multiple-input modifiers used, so the addition of self-defined insertion rules for modifiers may produce more appropriate cumulative effects.

Unfortunately, at this time a full implementation of the grouping mechanism was not complete for evaluation. Grouping components in an interface raises many questions such as how to re-use and combine groups as well as accessing internal components of the group. A number of papers have approached this issue in more depth and may provide fruitful insights [Herman et al., 2000], in particular the image graph technique that represents not only results but also processes of data visualization [Ma, 1999].

3.5.4 Generalizing the Sketch-Interface Approach

The interaction method used for the sketch-interface addresses the fundamental issue that the complexity of interfaces is rising because of the complexity of the algorithms that operate on the underlying data structures. Finding ways to coalesce data structures so that they can work for different user interface requirements is a step towards providing a solution for overly cluttered or too complex user interfaces.

Our approach can be generalized to other systems that follow rigid pipelines, in particular for rapid-prototyping applications. For instance, we can create a process for users to come up with new shaders with more specialized modifiers that manage low-level tasks such as hardware register combiners or vertex programs for the creation of real-time special effects. Here we have a fixed pipeline whereby vertex programs are modified first, and then data is sent to register combiners for shading effects that must also follow certain dataflow rules. However, since vertex programs are independent from register combiners, we could find ways to translate user input for use in register combiners, and simply pipe that after the modified vertex programs. This would allow any user to experiment with very little knowledge of graphics hardware the visual effects that can be achieved.

In this respect 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 (as with register combiners). Due to a limited number of such hardware resources, we may extend the interface to provide some sort of resource management, such as suggesting better paths through modifiers, or highlight paths that use too many resources.

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. Thus, 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.

4 Camera Control in Computer Games

The current use of camera techniques in computer games is comparable to the situation in the early days of motion pictures. Back then, actors and directors were used to the theater stage and the viewer had only one perspective for the whole performance. There was no moving camera, no cuts, and only one shot size. Over the following decades, cinematography developed and people became accustomed to its language. Today, camera techniques form an important part of the story-telling process. The right use of a camera can enhance a viewer's experience as much as poor camera handling can destroy it.

In the past, games provided a first-person view or allowed the gamer to choose from one of several predefined camera positions including one or more over-the-shoulder views. The use of first-person views places the gamer inside his player character and has the fortunate side effect of freeing the game developer from any serious camera work. This is a valid approach for the type of first-person shooters that have dominated the real-time 3D game sector, but there is now a need for a more sophisticated camera handling. Indeed, camera settings can reveal information to the user in a subtle way. Third person views, for instance, are often associated with higher player-character identification.

We begin in Section 4.1 with an overview of camera planning problems and related work. Section 4.2 describes a declarative camera planning approach in which an initial camera system is built that constructs a set of shot properties used to specify visual goals. This system is used to demonstrate the versatility and power of declarative specification methods to capture content in a precise yet natural manner. Section 4.3 uses such a declarative approach that is optimized to find frame-coherent solutions in real-time based on a simplified subset of camera operations. This camera engine can be directly applied in computer games. Thus, the developer is in a position to specify visual goals that the camera can then satisfy. However, there are often situations in computer games where it is desirable to automatically generate visual goals to capture events that occur during the course of a game. In Section 4.4, we provide methods that return keyframe and timing information of appropriate events in a game that can be given to the camera for capturing of live content or post-game summarization. Finally, we conclude by discussing achievements and future extensions in camera planning.

4.1 Previous Work

Much work has been done on enabling users to directly manipulate the camera [Hanson and Wernert, 1997; Mackinlay et al., 1990; Ware and Osborn, 1990; Ware and Fleet, 1997] or on providing camera assistants [Phillips et al., 1992; Gleicher and Witkin, 1992; Jardillier and Languénou, 1998]. However, relatively little has been done in immersing the user in an environment by controlling an object or character, whereby the camera process should be invisible to the user. Nevertheless, allowing communication of important visual goals such as navigational information, or notable features in the environment, should be enabled so that the player is not left running into the unknown. In this section we look at camera techniques that can be applicable in the context of computer games.

4.1.1 Real-Time Constraint Satisfaction

Most real-time games solve camera positions procedurally, using specialized camera routines adapted to the design of each level. This makes for an inflexible camera engine, and often leads to situations where the camera is not showing the best view for the user. In Tomb Raider, for example, the camera can produce awkward views in closed spaces when Lara Croft (the heroine) is backed up against a wall, because the camera computes without explicit consideration for visual properties in the view.

Blinn [1988] describes how vector algebra can be used to position a camera given the desired position of two objects in the viewplane, and this method was used in the compiler for the Declarative Camera Control Language (DCCL) which allowed the specification of cinematic idioms in camera planning for animations [Christianson et al., 1996]. The limitation of this approach is that it uses point abstractions of the objects, and therefore cannot account for the range of visual effects that arise from the fact that real scene elements have finite extents (e.g., occlusion between scene elements).

Drucker et al. [1992] and Drucker and Zeltzer [1994, 1995] set up an optimal camera position for individual shots subject to constraints. The camera parameters are automatically tuned for a given shot based on a general-purpose continuous optimization paradigm. These methods proved effective and allowed the design of camera modules. However, the camera modules, such as used in the CINEMA system, experience problems combining and constraining multiple procedures, requiring specifically tailored procedures to be setup. [Bares et al., 1998; Bares and Lester, 1999] have developed CONSTRAINTCAM, a real-time camera visualization interface for dynamic 3D worlds. CONSTRAINTCAM allows the specification and real-time solution of three classes of constraint: viewing angle, viewing distance and occlusion avoidance. Although the expressiveness of this set is limited (e.g., it is not possible to locate objects at particular positions in the image) the strength of the system is the utilization of solution techniques that allow real-time satisfaction of constraints sets. However, neither of these systems provide predictive analysis for their interactive environments, nor do they offer the ability to impose constraints based on existing camera situation and movement. This in turn means that they cannot achieve a high level of camera frame-coherence.

Jardillier and Languénou [1998] have reported an approach which uses interval methods to find camera paths which yield sequences of images fulfilling temporally indexed image properties. Although the properties are once again very limited, and the method computationally expensive, the interval-based approach has the advantage of guaranteeing the maintenance of visual properties for the duration of their temporal indexing. This contrasts favorably with techniques which rely on only a sample of the positions along a camera's path.

4.1.2 Visibility

Few interactive systems propose methods that effectively avoid occlusion. This is surprising considering the importance of maintaining an objects' visibility in shot. Christianson et al. [1996] check for occlusion given a camera shot by testing against overlapping bounding spheres and incrementing an occlusion counter, but do not adjust camera state to accommodate visibility. Halper and Olivier [2000] (see Section 4.2.2) use image precision calculations to assess visibility, but employ offline genetic algorithms for optimal camera state generation. Tomlinson et al. [2000] test against occlusion in their interactive environment by shooting a ray to the object in question, and adjust the camera vertically only, although this method is mostly suited for use in sparse outdoor environments.

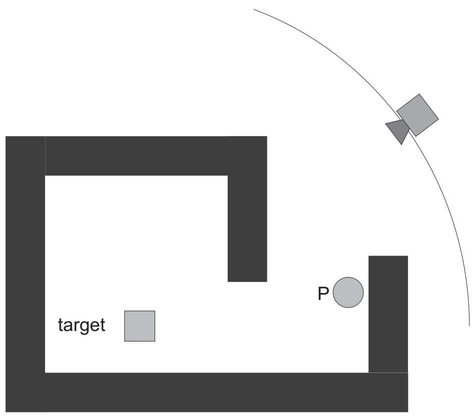


Figure 4.1: Current visibility algorithms do not process depth information across other angles of visibility, such that a viable point P can be returned.

Phillips et al. [1992] developed an algorithm based on the hemicube [Cohen and Greenberg, 1985] to select viewpoints and viewangles to ensure that a manipulated object is not obstructed by other objects. They use a cube centered around the origin of the object being manipulated, and orient it towards the camera position, whilst projecting the geometric environment onto each cube face to achieve a visibility map. If the camera is obstructed, they look in the neighborhood of the direction of the camera for an empty area in the visibility map, which suggests a location of the camera from which the object will be visible. Drucker [1994] processes the visibility map to create a potential map, which is followed from the initial location down to the nearest location in the visible region. Phillips et al. [1992] acknowledge that their algorithm will often fail in an enclosed environment without the use of depth information in the visibility map. Bares and Lester [1999] accommodate this fact when their optimal vantage angle is occluded by decreasing the distance of the camera to the object until it is placed in front of the nearest obstacle. However, none of these techniques can compute a position such as P shown in Figure 4.1, and can only resolve visibility for a single point. Our algorithm described in Section 4.3.3 is able to find points such as P, and resolves occlusion constraints for an arbitrary number of points.

4.1.3 Computer Games and Cinematography

We agree with the game developer Barwood [2000], who notes that

> "[computer games] are as different from [movies] as movies are different from theater"

This means that camera techniques used in future games will have to develop and employ languages and rules on top of cinematographic techniques. Differences arise from the fact that computer games are highly interactive (unlike film or theater). This makes it impossible for any director module to stage and rehearse actions beforehand. Therefore, even if computer games' camera modules would possess the cinematic knowledge of directors, they could not apply it since these techniques depend to a great part on trial and error. Also, when shooting a real movie, directors are at liberty to reposition actors and change the scene, even the script. In contrast, a camera module is typically given the scene as it is. If, due to geometric constraints, a good camera position can not be found in the scene as it evolved up to that moment, a less-than-satisfactory camera position must be used. Thus, encoding cinematographic techniques into idioms provide opportunities for predefined film scenarios (e.g., the Virtual Cinematographer guarantees results in a common shot form [He et al., 1996; Christianson et al., 1996]), but lack camera setups to convey visual goals that aid interactivity for the player.

Game companies such as Lucasarts have a great series of adventure games in 3D, but have rigid fixed camera positions, so that if not everything is going to plan or not all characters are in place, it will not be shot right. Therefore they must limit their interactions based on pre-defined scenarios.

Systems such as CATHI [Butz, 1997] or by Karp and Feiner [1990, 1993] encode idioms in the form of film grammars, using a top-down approach in generating a sequence of shots. They are able to produce camera paths that achieve certain visual goals, but use timing information in the animation and therefore cannot be directly applied to reactive environments like computer games.

Tomlinson et al. [2000] propose a behavior-based autonomous cinematography system by encoding the camera as a creature with motivational desires, focusing on camera movement styles and lighting in order to augment emotional content. This is a step in the right direction for autonomous camera agents in interactive worlds, as their camera creature attempts to capture sequences that are of interest to the viewer. However, their constraint solver is computed procedurally based upon requirements for their system, losing the flexibility desired for developer-defined specifications.

The use of the camera in a movie is highly dependent on the current situation. To approach the quality of film's camera techniques in computer games, games can classify each possible situation during the game into a (hopefully) small number of variants. Each of these can be associated with a number of techniques describing how to shoot that scene, where to place virtual cameras, how to follow an actor and when to cut (switch between virtual cameras). For instance, in a dialog situation, the camera can focus on the participants alternately as they speak, starting with an establishing shot framing both speakers. When exploring unknown territory, the camera – which normally follows the player character – can provide hints for the gamer by looking elsewhere if there are important clues to be found (or missed). In interactive applications, all shots are presented to the viewer immediately without any editing and montage of the raw footage, therefore a director engine should also know what is likely to happen next so that it can plan shots and their transitions (panning or cuts) in advance.

4.2 A Declarative Camera Planning Approach

In order to be able to communicate certain goals, we need a way of specifying how we want to communicate those goals. In this section we look at a viable approach towards specifying and achieving visual goals that can be captured with the camera. Our initial goal is to incorporate a camera planning subsystem of Seligmann's four-stage architecture (see Section 1.1), which resides within the latter two stages. We now impose a division of camera planning into three sub-problems:

1. *specification of shot objectives*: shots are specified not only in terms of explicit spatial relationships between the camera and scene elements, but also in terms of the objectives (visual and spatial properties) of the desired image;
2. *evaluation of objectives*: for any position of the camera, each objective must be well defined and efficient to evaluate with respect to the underlying graphical modelling paradigm (i.e., geometric abstraction of the graphical model of the scene should be resisted);
3. *acquisition of a camera state*: there must be a mechanism by which the camera state (location, orientation and field of view) can be established such that the fulfilment of the specified objectives is maximized.

Each of these issues are resolved in the following subsections. We begin by creating a taxonomy of shot properties in Section 4.2.1. These properties are implemented into a presentational graphics system called CamPlan, described in Section 4.2.2, that acquires a camera state by evaluating shot properties. CAMPLAN is then used to demonstrate the viability of a declarative camera planning approach in Section 4.2.3.

4.2.1 Specification of Shot Properties

In this section we establish a set of shot properties which the user can select for a presentation. The elements of the set of shot properties can be classified according to their type, quantitative character (absolute or relative) and the reference frame with respect to which they are characterized.

Studies of human cognition yield several possible reference frames, for example, relative to which spatial references (e.g., the use of spatial prepositions) may be made. In general these are the intrinsic (object centered), deictic (viewer centered), and allocentric (world centered) reference frames. Whilst a camera is essentially a deictic point of view, specified by its orientation, position in space, and field of view, the properties of a shot (for example, the position of an object in a shot) for a camera state can be further characterized with respect to the viewplane, viewport and viewpoint of the camera:

Viewplane: the surface which extents infinitely in all directions perpendicular to the view direction of the camera. Often it is useful to specify objects in off-screen space (e.g., relate part of an object outside the screen with those in the screen).

Viewport: subset of the viewplane with a defined bounded region for the view with centre-point of projection (0.0, 0.0) and upper-left corner (-1.0, 1.0). This specifies the view and comprises the image output.

Viewpoint: the location of the camera. Camera placement can be controlled in an environment by using properties defined over elements surrounding the viewpoint by specifying positional measures such as angle or distance.

Specification of shot property values and tolerance settings can be made either explicitly or implicitly to these reference frames. Hence, two additional characteristics of shot properties:

Absolute: The shot property types are defined explicitly, as an absolute relationship (e.g., object A is to be projected 20% the size of the viewport).

Relative: The shot properties for objects are defined by reference to the properties of other objects (e.g., object A is to be projected 20% bigger than object B in the viewport).

A type of image property is that which is most characteristic of the constraint imposed on a scene element. For an object in a scene this includes its position, size, visibility and orientation. Ideally, these types should be independent of knowledge about the topology of objects and the spatial arrangement of the scene so that they can be applied generically.

Position: the location an element of the scene, with respect to the reference frame (viewpoint, viewport or viewplane). Thus, a user may characterize both the physical location of an object (position with respect to the viewpoint), visual location of an object (position with respect to the viewport) and the off-screen visual location of an object (position with respect to the viewplane).

Size: how large an object is to appear. The projected size of an object is dependent on the actual physical size of the object, the field of view of the camera, and the distance to the point of projection. This also provides an indirect, but often more natural, means for the user to specify distance to an object and its visual impact.

Orientation: each scene element can have a specified orientation in the final image. Orientation shot properties enable the user to view certain sides of an object that might be difficult to specify otherwise.

Visibility: the requirement that some portion of an object must be viewable in the image. Visibility properties may be used to specify the general visibility of an object, or that two or more objects are in a specific occlusion relationship (thereby indirectly characterizing the orientation of a scene to a camera).

A taxonomy of image properties was created by considering permutations of the image property types with relationships in the various dimensions (viewpoint, viewport, viewplane, absolute and relative). Tables 4.1 to 4.4 show the current set of image properties implemented. In addition, the graphical models allow the specification of the part-whole structure of each scene element, thereby allowing the specification of scene properties over named parts of scene elements. The user may also apply properties across a group of specified named parts. Furthermore, the user must be able to specify tolerance on these properties.

4.2.2 The CAMPLAN System

CAMPLAN is a system that extends ideas from Drucker and Zeltzer [1994] in an attempt to address the principal short-comings of the approaches outlined in Section 4.1: the restriction placed on the range of image properties that may be specified and the unrealistic point-based characterizations of scene elements. Each of the shot properties defined in Tables 4.1 to 4.4 has been implemented into CAMPLAN.

In the following subsections we briefly describe the operation of CAMPLAN for its evaluation of shot properties (specified in the previous section) and its acquisition of camera state. The strength of the system is in planning expressive static visual shots of scenes intended to achieve a specific visual communicative goal. Readers interested in a full characterization of the implementation of CAMPLAN are referred to [Halper, 1999].

Position	Viewplane	Viewport	Viewpoint

Absolute	`CenterX/Y/XY`
	`BetweenX/Y`
	`Behind/InFrontOf`
		`CenterX/Y/XY`
		`BetweenX/Y`
		`Behind/InFrontOf`
			`AngleX/Y`
			`BetweenAngleX/Y`

Relative	`CenterX/Y/XY`
	`BetweenObjectsX/Y`
	`CloserThan`
	`ExtentsX/Y/Z`
		`CenterX/Y/XY`
		`BetweenObjectsX/Y`
		`CloserThan`
		`ExtentsX/Y/Z`
			`AngleExtentsX/Y/Z`
			`AngleX/Y`
			`Distance`

Table 4.1: Position properties. Gray text indicates identical viewplane and viewport functions. In the case of `BetweenX/Y`, the extent values are clipped to the viewport.

Size	Viewplane	Viewport	Viewpoint

Absolute	`ProjectedArea`
	`ProjectedLengthX/Y`
	`SpanXorY`
		`ProjectedArea`
		`ProjectedLengthX/Y`
		`SpanXorY`
			`ProjectedArea`
			`ProjectedLengthX/Y`
			`AngleWidth/Height`

Relative	`ProjectedArea`
	`ProjectedLengthX/Y`
	`SpanXorY`
		`ProjectedArea`
		`ProjectedLengthX/Y`
		`SpanXorY`
			`ProjectedArea`
			`ProjectedLengthX/Y`
			`AngleWidth/Height`

Table 4.2: Size properties. Viewport evaluates only clipped view portions of projected area, whereas viewpoint evaluates areas projected onto a sphere surrounding the camera.

Orientation	Viewplane	Viewport	Viewpoint
Absolute	`Angle`	`Angle`	`Angle`
Relative	`N/A`	`N/A`	`N/A`

Table 4.3: Orientation properties. Viewplane and viewport describe identical image properties. No relative relationships exist since orientation of scene elements relative to one another is independent of camera position.

Visibility	Viewplane	Viewport	Viewpoint
Absolute	`Occluded`	`Occluded` `InViewport`	`Occluded`
Relative	`Occluded` `OccludedBy`	`Occluded` `OccludedBy`	`Occluded` `OccludedBy`

Table 4.4: Visibility properties. The viewport gives clipped view evaluations of occlusion, whereas the viewpoint computes occlusion by projecting onto a camera surrounding sphere.

Evaluation of Shot Properties

CAMPLAN implements evaluation methods over polygonal representations of scene elements. The models themselves allow the specification of the part-whole structure of each scene element which enables the specification of scene properties over named parts of scene elements. For polygonal models, occlusion constraints are evaluated in a two stage manner, first over a bounding sphere approximation of the polygonal object, followed by an adaptive scanline visible surface algorithm for which the resolution is dynamically assigned depending on the tolerance of the specified occlusion property. For a more complete account see [Halper, 1999; Halper and Olivier, 2000].

Acquisition of Camera State

Given the selection of the shot properties, we must produce an image that maximizes the fulfillment of these objectives. This is cast as a standard optimization problem. At present only the camera state is modified to achieve the final output. Possibilities for modifying lighting or introducing cutaway shots of objects (to achieve further visual properties) can be applied in a future extension.

CAMPLAN uses a genetic algorithm (a nondeterministic optimization method) to find the optimal camera position whereby all seven elements of the camera state vector (position/3, angles/3 and field of view) are encoded in the chromosomes of each gene. A population of cameras is randomly distributed in a pre-specified bounded search space. A fitness value, derived from a linear combination of normalized values returned from the evaluation of each fitness function, is computed for each camera state. The top 90% of the population survive to the next generation, and the remaining 10% are re-generated by random crossover and/or mutation of chromosomes. Each successive generation is produced until the user aborts the search or an optimal combination of chromosomes is found for the camera state.

The fitness functions pertain to each shot property and are evaluated in sequence given each new gene (camera state). In order for a gene to survive to the next generation it must give a higher fitness value than the existing top ten percent of the population. This allows the program to check the current accumulated progress of the shot properties for the gene and if it is not possible for the gene to survive to the next generation then all remaining shot evaluations may be evaluated under approximate, less expensive algorithms. In addition, shot properties can be arranged such that the most computationally expensive evaluations are executed last (e.g., `OccludedInViewport`).

4.2.3 Applicability of Declarative Camera Planning

Human photographers and cinematographers will casually satisfy communicative goals such as “locate the door” by applying suitable composition styles or cinematic idioms. However, our approach to camera planning depends on the specified image properties defining sets of images that satisfy the attendant communicative goals. Since the images produced by

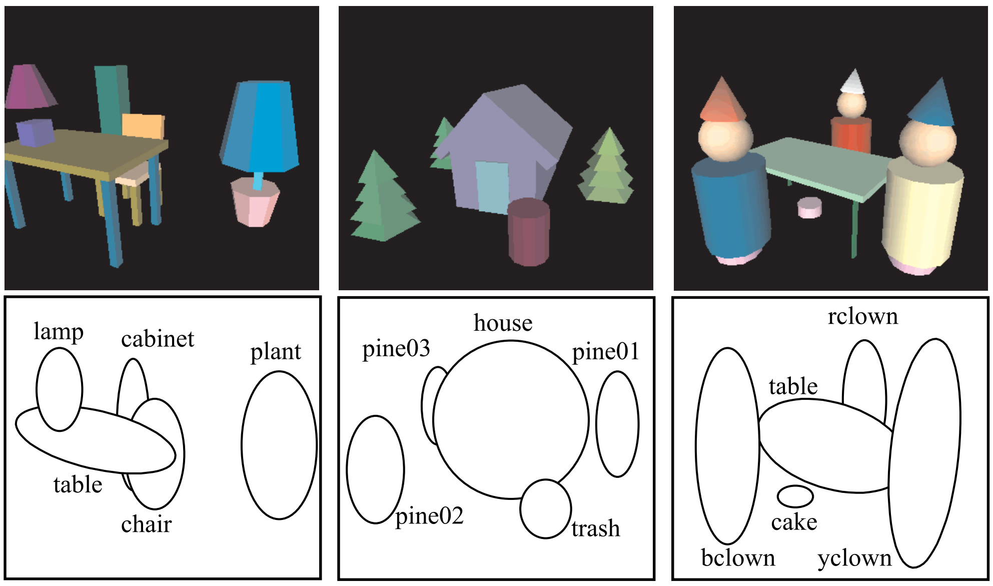


Figure 4.2: Test scenes and names of scene elements.

CAMPLAN are intended to function in the same way as human-composed photographs the appearance of styles when similar communicative goals are applied to similar scenes may be anticipated. Hence we attempted an evaluation of the ability of the system to generate recognizable styles.

The first set of the following examples observe CAMPLAN 's satisfaction of progressively more restrictive sets of image properties for the emergence of common styles. We then demonstrate that CAMPLAN is able to find a solution in a tightly constrained space and that it can provide a natural way of describing a view to generate an image.

Stylistic Consistency

Here we demonstrate that stylistic consistency can be achieved when applying the same set of image functions across object isomorphisms in scenes with similar spatial properties [Olivier et al., 1999].

The same sets of properties are applied to each scene in turn and the results of five random runs are given. Isomorphism between the elements of the different scenes are characterised as sets (see Figure 4.2): *A*={*lamp*, *trash*, *cake*}, *B*={*table*, *house*, *table*}, *C*={*chair*, *rclown*, *pine01*}, *D*={*plant*, *yclown*, *pine02*}, *E*={*cabinet*, *bclown*, *pine03*}. The same objectives are applied across the scenes for isomorphic elements. For example, for every image in which *pine01* is involved in the specification of a property, there are images in which *yclown* and *plant* are identically involved.

Figure 4.3 shows the set of solutions for each scene using the set of objectives given

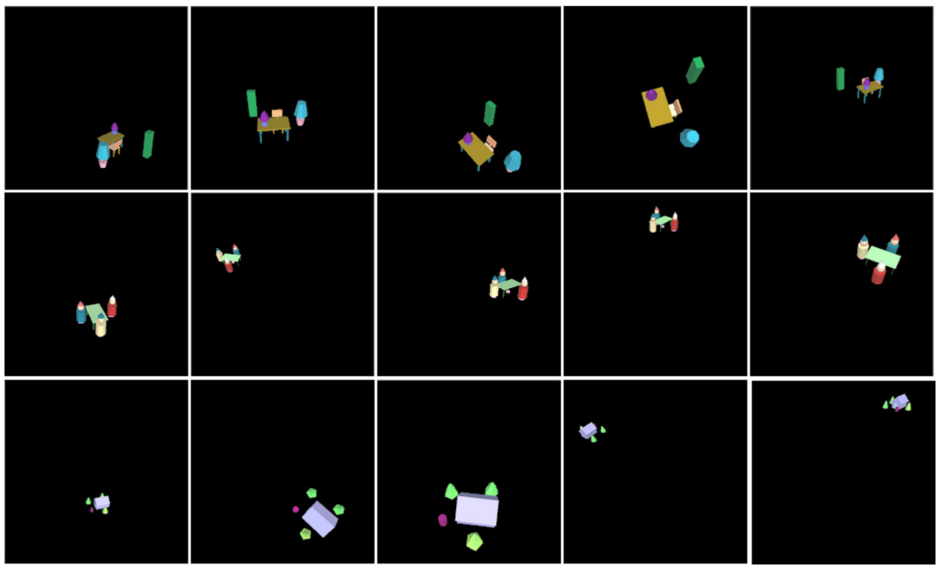


Figure 4.3: Example output under the constraint that all objects must appear in the viewport.

below. These simply specify that all the scene elements must lie in the viewport[1]:

```
EntirelyInViewport objectA
EntirelyInViewport objectB
EntirelyInViewport objectC
EntirelyInViewport objectD
EntirelyInViewport objectE
```

Although the projections of all five objects are required to be entirely within the viewport, the location of the camera may take any value in the range (1000, 0..500, 1000). Since the region of space within which camera location will yield an image where the scene elements are of a reasonable size is small compared to the size of the total space, it is predictable that the resulting images comprise views in which the scene elements occur in the distance.

The undesirably small size of the scene elements in Figure 4.3 can be addressed by constraining the screen size of a particular element (e.g., the largest) to have particular dimensions in screen space. In CAMPLAN the height and width of the viewport is two units, and so the additional constraint in the new set given below requires object *B* (the tables and the house) to be between 40% and 60% of the screen width.

```
EntirelyInViewport objectA
EntirelyInViewport objectB
```

[1]The 'EntirelyInViewport' command defines the 'BetweenX/Y' objectives to be enclosed within the viewport dimensions

```
EntirelyInViewport objectC
EntirelyInViewport objectD
EntirelyInViewport objectE
HorizSize          objectB 1.0 0.2
```

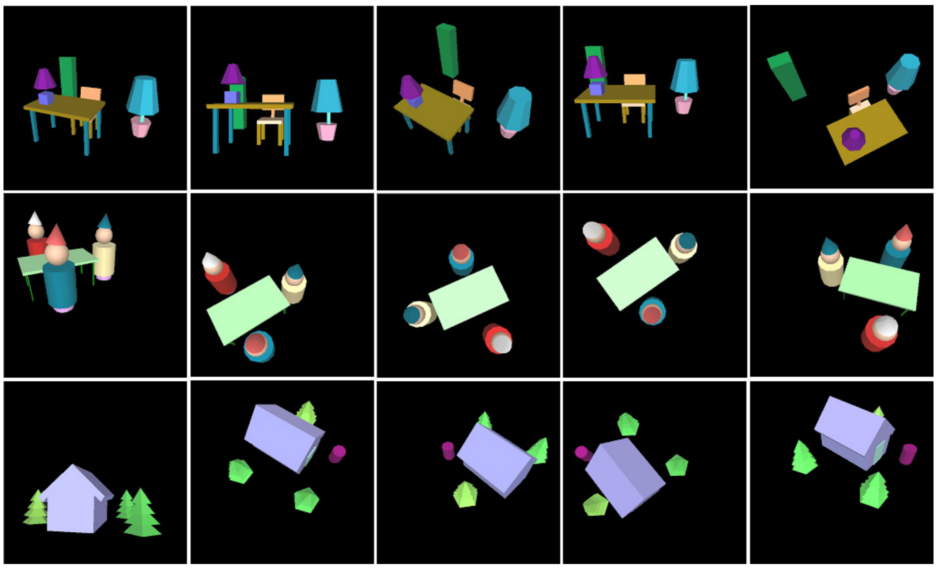


Figure 4.4: Example output after the addition of the constraint that the projected width of object *B* be about half the screen

Figure 4.4 shows example solutions for each of the scenes after the addition of this objective. Although the camera positions are closer to the scene elements than before, there remains a significant variability both in the direction from which the scene is viewed, and in the positions of the scene elements in the viewport.[2]

As with all the previous image objectives, when we place a restriction on the direction from which an object is viewed, it must be as independent as possible of implicit knowledge about the scene.

```
EntirelyInViewport objectA
EntirelyInViewport objectB
EntirelyInViewport objectC
EntirelyInViewport objectD
EntirelyInViewport objectE
HorizSize          objectB 1.2 0.2
```

[2]In the furniture scene (top row), the distance of the cabinet behind the table, and the requirement that all objects are in the viewport, results in views fulfilling this objective having to be shots from in front of the table.

```
ObjectCloserThan   objectA objectC
ObjectCloserThan   objectA objectD
```

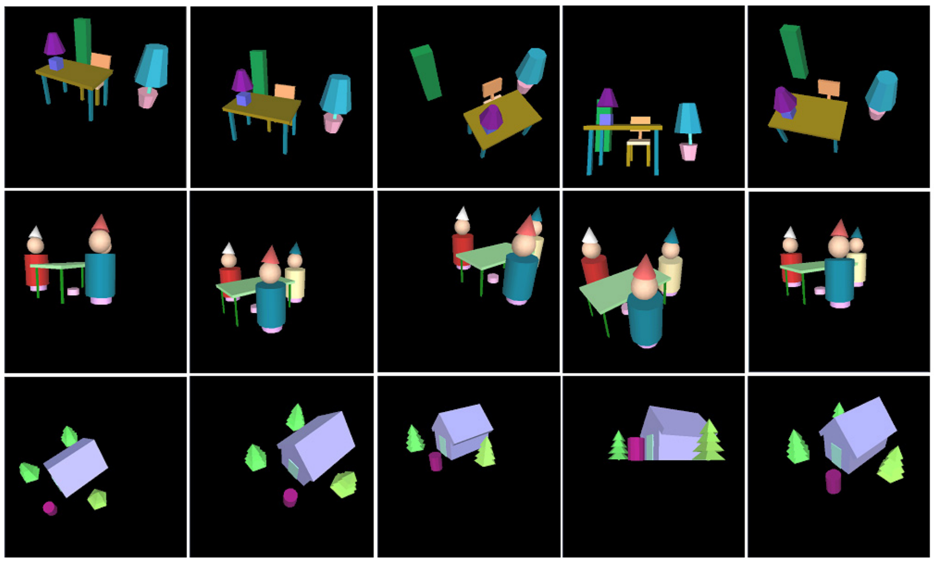


Figure 4.5: Example output for addition of the constraint that *A* is closer to the camera than both *C* and *D*.

Thus, rather than referencing orientation information of any particular object, we restrict the orientation of the view by requiring one of the objects to be closer to the camera than two others. The resulting images are shown in Figure 4.5. A degree of compositional regularity has resulted, although there is still some variation in the position of the elements in the viewport.

The addition of this objective relies on a number of assumptions including the fact that the closer object is not significantly larger than its separation from the background objects, and that the three objects (in this case objects *A*, *C* and *D*) are not collinear.

Restricting the position of the scene elements in the viewport may be achieved by placing a restriction on the image coordinates of one of the scene elements.

```
EntirelyInViewport objectA
EntirelyInViewport objectB
EntirelyInViewport objectC
EntirelyInViewport objectD
EntirelyInViewport objectE
HorizSize          objectB 1.0 0.2
ObjectCloserThan   objectA objectC
ObjectCloserThan   objectA objectD
PositionY          objectB 0.6 0.3
```

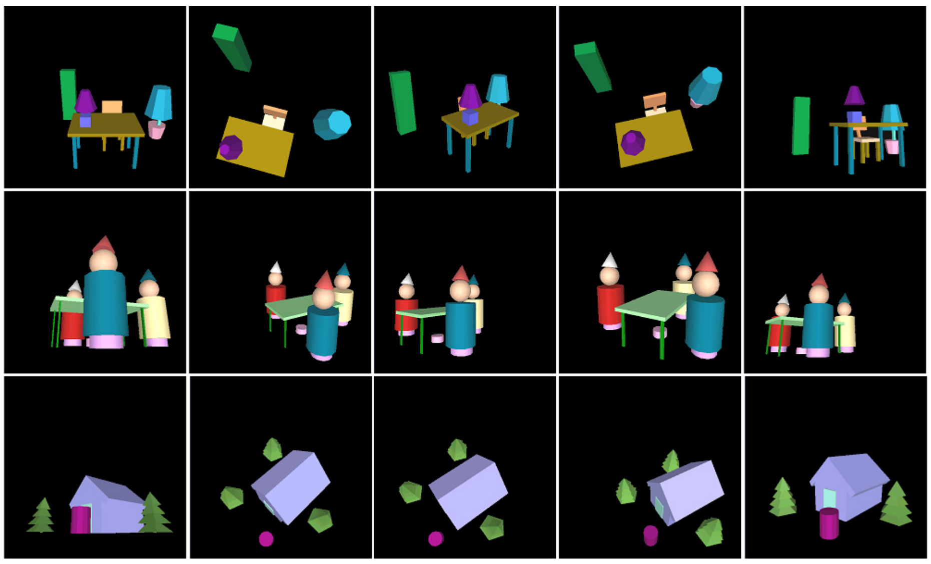


Figure 4.6: Example output for addition of constraint that the position *B* must be in lower section of screen.

In this case the centre of the bounding sphere for object *B*, (the object with the size restriction) is required to lie in the bottom 15-45% of the image. The resulting images are shown in Figure 4.6 and we can observe a further increase in the compositional consistency. However, it is apparent, from the party scene (middle row) and the house scene (bottom row), that objects may fully, or nearly fully, occlude each other.

The final objective is the removal of the possibility that any of the scene elements can be either fully, or nearly fully, occluded by another scene element. Thus we require all of the objects to be at least 20% unoccluded. Examples of the images resulting from the addition of these final objectives are shown in Figure 4.7.

```
EntirelyInViewport objectA
EntirelyInViewport objectB
EntirelyInViewport objectC
EntirelyInViewport objectD
EntirelyInViewport objectE
HorizSize          objectB 1.0 0.2
ObjectCloserThan   objectA objectC
ObjectCloserThan   objectA objectD
PositionY          objectB 0.6 0.3
OccludedInViewport objectA 0.0 80.0
OccludedInViewport objectB 0.0 80.0
OccludedInViewport objectC 0.0 80.0
OccludedInViewport objectD 0.0 80.0
OccludedInViewport objectE 0.0 80.0
```

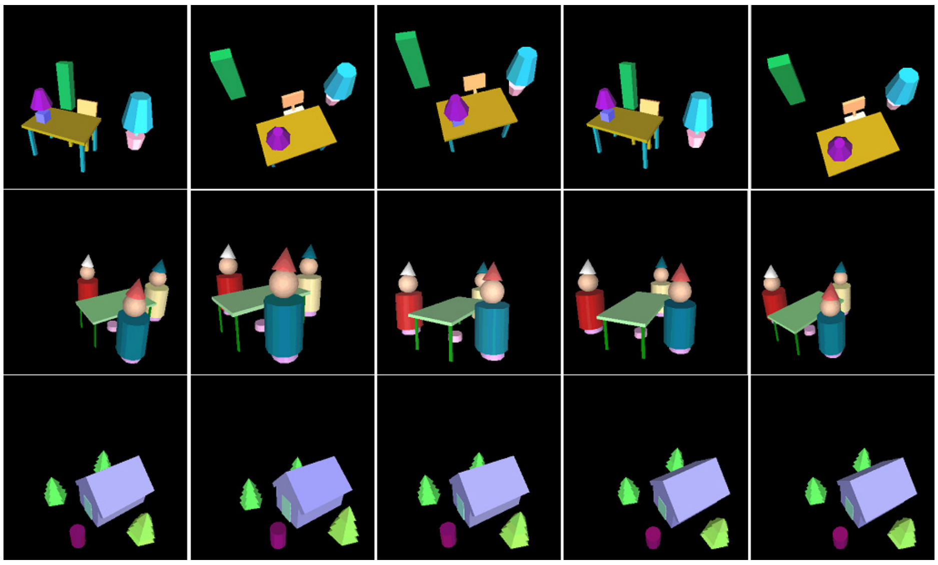


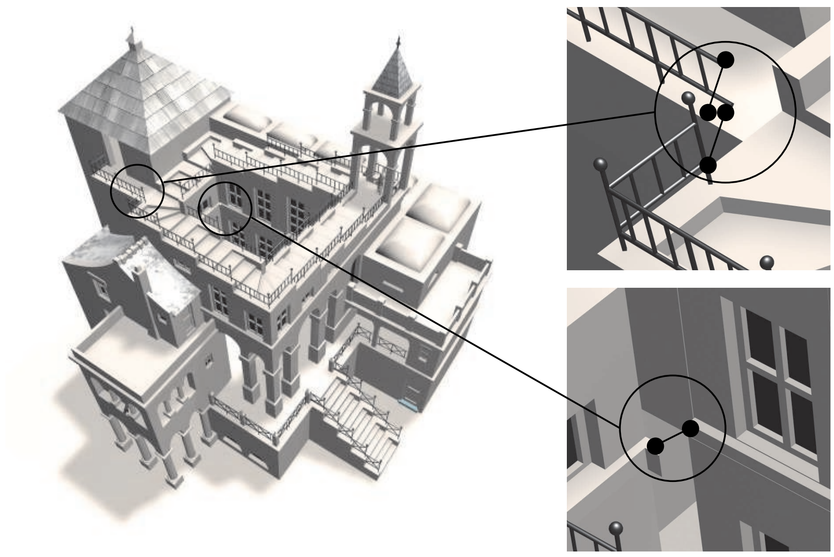
Figure 4.7: Example output after the final addition of occlusion objectives.

Finding unique solutions

We now demonstrate the system's performance on a more complex scene (a virtual reconstruction of Eschers "Ascending and Descending") comprising over 60k polygons. Figure 4.8 shows how CamPlan has found a tight unique solution region over a set of properties designed to produce the correct perspective view of the Escher building by aligning certain visual keys. Note that an additional constraint was used so that the center of the building's bounding sphere is forced to be located in the center of the view. This removes 3 degrees of freedom (the camera orientation is fixed to face a target given its position) and the convergence of the GA towards a solution is improved dramatically. CamPlan is able to find a solution on average within a few seconds. Without forcing the camera to focus on the center of the building the GA converges much slower (taking minutes or hours). This suggests that removing degrees of freedom and introducing methods to restrict the search space will result in significant improvements for the search time.

What-You-Specify-Is-What-You-Get

To show the advantages of a declarative approach to shot specification, and in order to give a clearer shot of the models construction and illustrate how the illusion of the infinitely ascending and descending steps is created, a small number of properties breaking the main

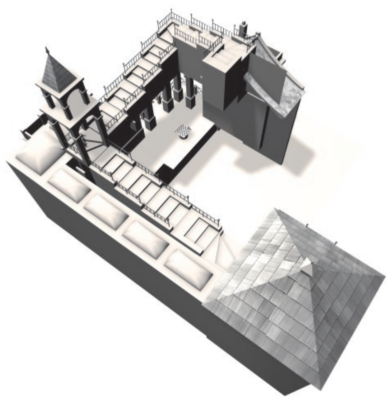


The shot properties are set to align the grid bars such that the vertical and horizontal extents coincide exactly using the relative position properties.

Similarly for the window ledges, their X and Y extents should coincide to start andend at the same point.

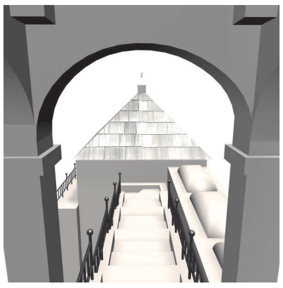
The entire building should also be in the center of the shot and projected at least half the size of the viewport.

Figure 4.8: Camera state found in a tightly constrained solution region.



```
CenterShot "Escher building"
EntirelyInViewport "Escher building"
OccludedInViewport "step high" 0 80
OccludedInViewport "step low" 0 80
NotOccludedBy "step low" "step high"
NotOccludedBy "step high" "step low"
```

Figure 4.9: Highlighting the relationship between the top and bottom step.



```
CenterShot "roof"
Unoccluded "roof" 100 tolerance 0
SpanXorY "roof" 2 tolerance 1.2
PositionRelativeX "pillar01" before before "roof"
PositionRelativeX "pillar03" after after "roof" "grid"
OccludedInViewportBy "pillar01" 100 tolerance 99
```

Figure 4.10: Directly specifying the desired compositional properties of a view.

features of this illusion are specified. The two steps which are disjoint in the cycle should be in some part visible, and that they should not overlap each other. A typical result of the images produced in satisfying these properties is shown in Figure 4.9.

Another set of properties demonstrates how the user can specify a view in a natural way. Figure 4.10 shows a view where the camera was desired to be placed in a position to give an unoccluded shot of the roof of the building leading to the steps as seen between two pillars of the small tower. The model itself has a grid fence which is discontinuous, and the user may hide this by specifying that this grid should be overlapped by the pillar which is to be placed to the left.

4.3 A Camera Engine for Computer Games

In the previous section we have found a viable method of specifying camera goals that can be tied to game goals through declarative specification to satisfy visual properties. In this section we provide new methods for real-time declarative camera specification and show how the camera subsystem can be integrated into the game pipeline [Halper et al., 2001].

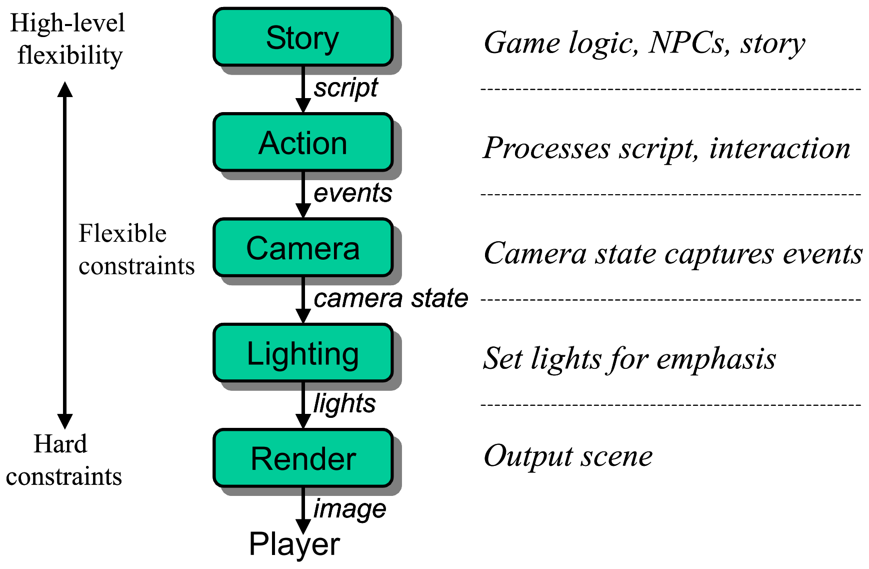


Figure 4.11: The game pipeline.

We propose that a camera module should be a part of the game engine pipeline as shown in Figure 4.11. Each module generates its own output down the pipeline from the given inputs. The story engine drives the motivations for the actions in the game, and is the most flexible and creative part of the project. The action module creates events—interactions from the player with the environment and story-related actions. On the bottom extreme of the pipeline, we find the renderer. This engine must produce consistent crisply defined results. The lighting module is a step higher, but is dependent on adjusting settings to emphasize visual goals depending on the camera state. The camera module finds itself in the middle of the pipeline, a balance between flexibility and hard constraints—it must try to convey visual goals that dramatize the action, but is still confined to properties of the environment.

Two works in camera planning demonstrate two extremes in camera planning systems—the Virtual Cinematographer [He et al., 1996] and a cinematography system based on action-selection [Tomlinson et al., 2000]. Whereas the former uses fixed hard-encoded shot templates, the latter uses an action-triggered function for camera movement. The proposed method will be to use a hybrid-system which allows users to encode their own styles to be applied to scenes through the use of dynamic parameterized camera templates that are solved through a camera constraint solver. We can reduce the camera engine requirements down to three basic requirements:

- **Flexibility:** must use parameterised techniques and versatility in defining constraint specifications in order to adapt to the output of events from the action module.
- **Information:** the more the camera knows about what is going on in the world, the better. We need event calls, information from actors, player motivations, and visual goals.
- **Satisfaction:** a best-fit solution will not always be present. Therefore, we need partial satisfaction solutions, and incorporate adaptive degradation, so that we can at least convey the most important visual cues at any given time.

Note that constraints are based on targeted objects and have to be re-evaluated as they move. In addition, events generated by the story and action modules can change each frame, producing different visual goals and constraint specifications for each situation. A purely reactive application of constraints, such as [Bares and Lester, 1999], will give rise to ‘jumpiness’ as the camera constantly jumps to global best-fit solution spaces, in particular when avoiding viewing obstructions. To address jumpiness we have to maintain frame-coherence. Since frame-coherence makes for smooth camera motion, it must be given priority over strict constraint adherence. A system which addresses these issues is shown in Figure 4.12. Details of the various system components are described in the following subsections: A director module takes events as input and specifies constraints in Section 4.3.1, Section 4.3.2 constructs a constraint-solver pipeline that computes new positions from existing camera state and adds relaxation parameters to the constraints, Section 4.3.3 details the occlusion avoidance algorithm, and Section 4.3.4 uses lookahead algorithms to adjust the camera to future situations.

4.3.1 Director

The director takes input from the action list generated from the previous stage in the game pipeline. The Camera Expert requests settings for a certain time t. Based on input from

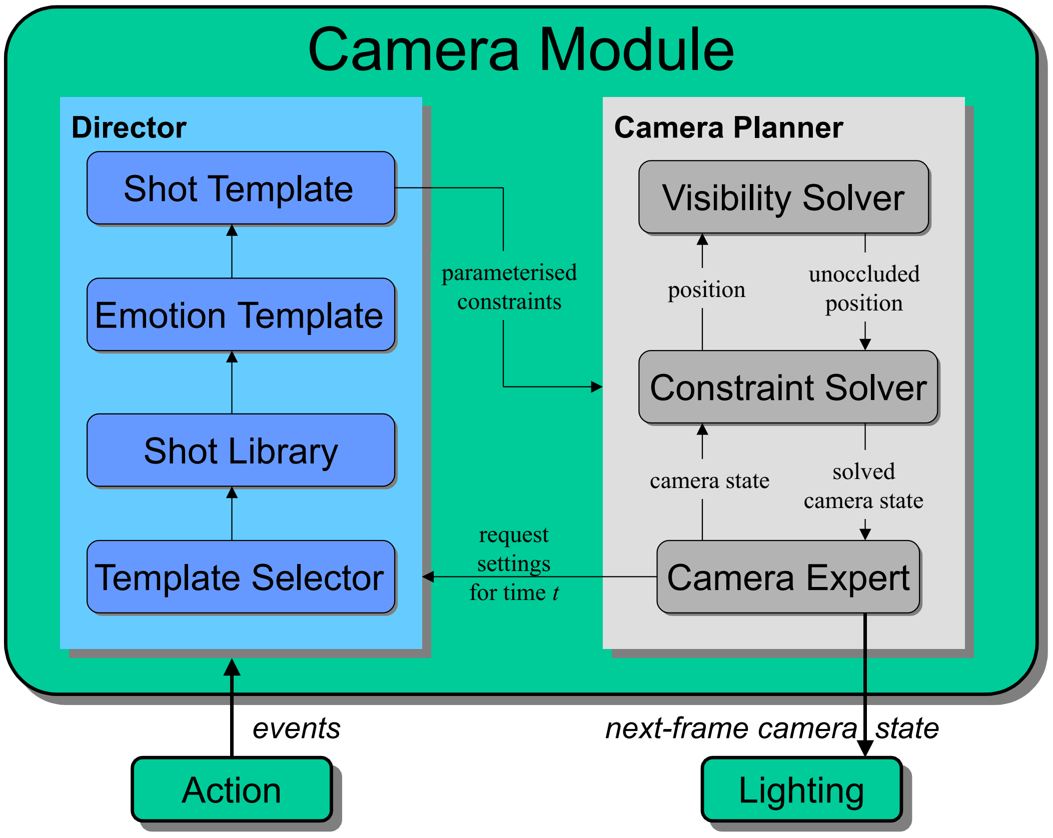


Figure 4.12: The camera module.

the event list, the director selects templates using the Template Selector, and prepares the constraints for the Constraint Solver. The director selects templates available from the shot library, using a number of transition rules so that successive shots *fit* together. In the case that a particular shot may not be satisfiable, the director can try a different shot that might not include all visual goals but one that might satisfy an important subset of them.

Emotion Templates encode various factors that may influence the results of camera shots. For instance, shot specifications can show a certain style of shot, whereas dynamic modifications to those specifications can produce various moods. We can influence the parameterizations of the shot properties defining a shot such that they accentuate a certain effect. For instance, we may add a height angle to the camera, and produce a ‘moody’ effect. The amount of ‘moodiness’ can influence the scale of the alteration to the height angle.

A director does not plan shots by their low-level parameters such as camera position and direction, but by their desired visual properties. Earlier in Section 4.2, we proved that enough flexibility and expressive power is possible by declarative constraint specification and made a first attempt at providing a complete set of workable properties. The class of declarative constraints are refined here—we remove constraints that are either redundant or not useful in the context of computer games. We also find ways to integrate these constraints into a real-time constraint-solving pipeline detailed in Section 4.3.2 that effectively satisfies specified combinations of constraints for arbitrary numbers of objects.

Level At: The camera should be offset at a certain height relative to the object. This constraint often applies to portraits where the camera is facing the actor, or when we want to follow a target.

Angle to line-of-interest: The angle from which to look at the target, specified relative to the line-of-interest, that is usually the line of interaction between characters or directly defined. This constraint is mutually exclusive with the next one.

Facing: Each object has a vector that defines the ‘front’ of the object. Since this direction is tied to the target, the camera moves when the target turns. From this we can also specify informative 3/4 viewing angles to objects, or create over-the-shoulder shots by setting the desired viewing angle to look from behind the object.

Size: This is actually used to control the camera’s distance to the target. Since finding the right distance depends on the targets shape and size as well as viewing angle and focal distance, the camera distance is better specified in terms of the resulting size of the targets projection on screen. When maintaining constraints over time, blind adherance to a size constraint dependent on camera or object orientation can result in oscillating camera movements as the size measures vary (e.g., this would occur if we measure size as the relative number of filled pixels in the view for an object). Therefore, direction invariant metrics such as the bounding sphere radius are a better choice than constraining against more precise measures.

Height angle: Instructs the camera to watch the target at a specified angle from above or below.

View at angle (θ and ϕ): Position the camera so that the line from target to camera has a specified angle to the viewing direction. If set at 0.0, this puts the target in the center of the screen. θ angles other than 0.0 move the projection of the target to the left or right side of the screen; the ϕ angle is used to push the target toward the top or the bottom. For the target to remain on screen, these angles should be set smaller than the camera’s field of view in θ or ϕ. We define position of objects as angles, so that it is possible to specify placements of objects surrounding the camera.

Visibility: The target is to remain visible to the viewer, such that unwanted obstructions from scene geometry between the camera and target are avoided.

These constraints and specifications have been designed so that they can sufficiently define a camera position and viewing direction. Some combinations of constraints will be

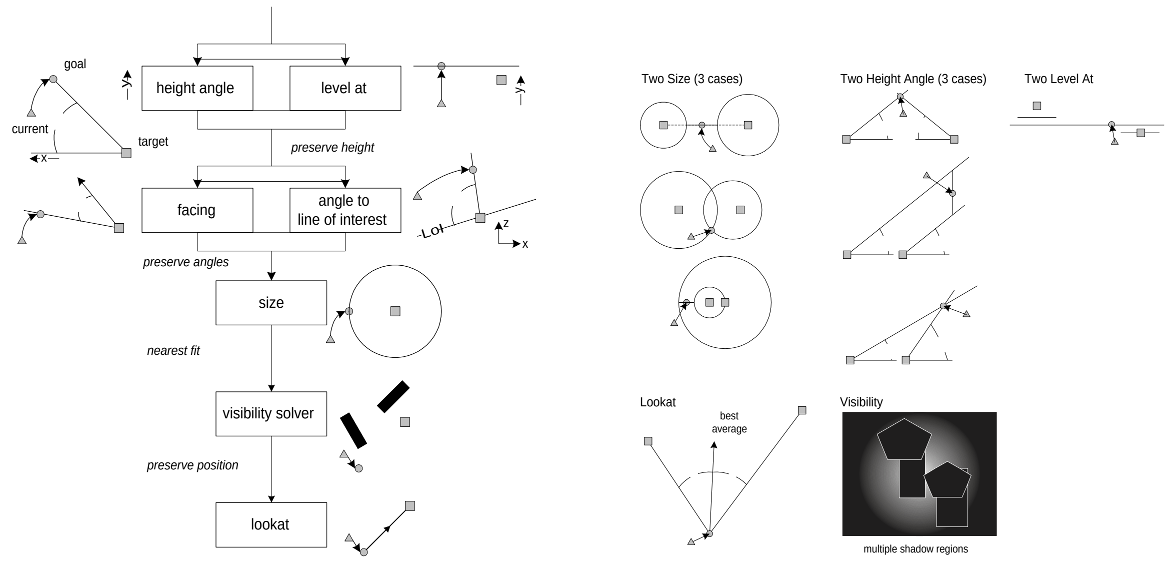


Figure 4.13: (left) The order in which constraints are applied. The current camera state is shown as a triangle, the target as a square and the goal state as a circle. Each successive constraint minimally influences output of previous constraints. For instance, size conserves both the xy and xz-angles created from (height angle/level at) and (facing/angle to line-of-interest). (right) Cases of using 2 targets per constraint. Many of these can be adapted to using 3 or more targets.

impossible to satisfy, especially when constraints are specified across multiple objects at the same time. Therefore, if shooting more than one target, the sets of constraints must be carefully chosen to guarantee a solution. However, the constraint-solver may still try to find a partial best-fit solution for all constraints, as detailed below.

4.3.2 Constraint Solver

To achieve frame-coherence we compute new solutions for camera state based on existing camera state. Thus, we plan the position of the camera for the next subsequent frame. Since not all constraints can be fully satisfied for each change in the scene, we get approximate results by first solving for certain constraints, and then modifying the camera state to accomodate the other specified constraints. The end result is one that best approximates the desired output, and allows a variety of heterogenous constraints to be integrated and put to practice.

Figure 4.13 (left) shows how these constraints are applied. Note that each successive constraint in the constraint-solver pipeline minimally influences the previous adjustments. Figure 4.13 (right) shows what happens when constraints of the same type are used for multiple objects. There we test for cases and adapt to multiple settings and reach approximate results. Alternatively, we get more precise results for specific visual goals by using a special constraint combiner, that explicitly has an algorithm to solve. For instance, we are able to solve for a size and viewport position constraint on one object, plus an additional position in viewport constraint on another.

All constraints cannot be instantly satisfied for each change in the scene, since this produces a rigid camera feel. Therefore, each constraint has an optimal setting (e.g., size 30%) which defines a goal for the camera. A tolerance region is also specified, so that if the camera does not lie in the optimal state, we can compute how far it is from this goal. Depending on relaxation parameters, the camera can be placed a ratio p closer to this optimal state. Applied to each frame in the sequence, this acts in similar fashion to a local optimization search, as the camera fluidly moves to a more satisfiable region. In the case that the camera lies outside the tolerance region for a constraint, then it must be placed at the borders of the tolerance region. If too many constraints are outside their tolerance regions, then the camera may select an additional set of visual properties for the shot, or use a transitional cut.

4.3.3 Visibility Solver

The technique we use to compute a new camera position that provides an unobstructed view of points of interest addresses the shortcomings of the algorithms outlined in Section 4.1.1. A flexible and robust method is introduced that allows user-definable visibility goals to be applied to an arbitrary number of points.

We allow constraints on the camera in the processing of occlusion avoidance by using what we call *Potential Visibility Regions* (PVR). The developer/designer can impose constraints on camera movement for the task of occlusion avoidance by defining a set of geometric constraints using polygons. To denote preference, polygons are shaded a brighter color than those geometric regions of less preference. This geometry defines the PVR to which the camera may move in order to obtain an unobstructed view of the points of interest.

In order to find the best position that satisfies visibility requirements, we first define an offscreen viewport buffer and set its viewing matrix so that it renders from the view of the target object position looking directly at the camera position. Then, we write only depth information from the potential occluders to the buffer. Next, we render the potential visibility geometry in order of most desirable regions (with the brightest colors) to least desirable regions (darker colors), stencilling the regions which are rendered first. The result is an image buffer that represents a *visibility map*, whereby the brightest colors that first pass the depth test are visible. Thus, the brightest color in the visibility map that lies closest to the camera position (center of its viewport) denotes the most desirable position for the camera to move to. Finally, the depth value at this position is read so that we can reverse project from viewport coordinates to get the unoccluded 3D world location. The following pseudo-code describes this algorithm in more detail:

```
// STAGE 1: write occluder depth information to buffer
setViewingMatrix(positionAtTarget, pointToCameraPosition());
clearBuffer();
disableColorWrite();
drawPotentialOccluders();


// STAGE 2: render PVR
disableLighting(); // just want PVR base colors
enableColorWrite();

// all lighter colored PVR geometry will write its color
// and stencil the areas that pass the visible depth test
for lightest_color_PVR_geometry to
    darkest_color_PVR_geometry do:
        render_and_stencil_PVR();

// STAGE 3: extract 3D position of best visible PVR
// find best PVR color and visible pixel location
Scan pixels in buffer:
    best_PVR_position = lightest_color_closest_to_buffer_center();

// now reverse project from position and depth in buffer
// to get 3D world coordinates
depth           = readDepthValue(best_PVR_position);
unoccludedPoint = UnProjectTo3D(depth, best_PVR_position);
```

The flexibility in this algorithm is in design of the PVR. As an example, Figure 4.14 demonstrates how to draw geometry in order to find the closest distance from the camera to an unoccluded view. In addition, we can constrain the camera such that it may move closer to the object or along the world-coordinate y-axis only. The choice of the PVR can be dependent on the tolerance settings of other constraints. If we had a hard constraint on size, we could draw a bounding sphere around the object of interest with radius set to the distance defined by the size constraint, such that visibility solutions will only be found on this sphere, maintaining full satisfaction of the size constraint. These visibility regions also have the possibility to be defined without modifications to the core algorithm—they can be input as geometric data at run-time. Figure 4.15 shows a number of examples.

The PVR algorithm described so far only solves visibility for a single point used to represent the target object. This can produce undesirable effects if, for instance, the camera finds small holes in the scene through which to view a point on the target and still leave the rest of the target occluded. This problem can be ameliorated by rendering occluding geometry somewhat larger than used in the real scene[3].

However, with the introduction of PVR defined by geometry, combined with the advent

[3]This is similar to many collision detection techniques that expand the environment geometry by a factor relative to the size of the 'colliding' object so that point collision computations can be used

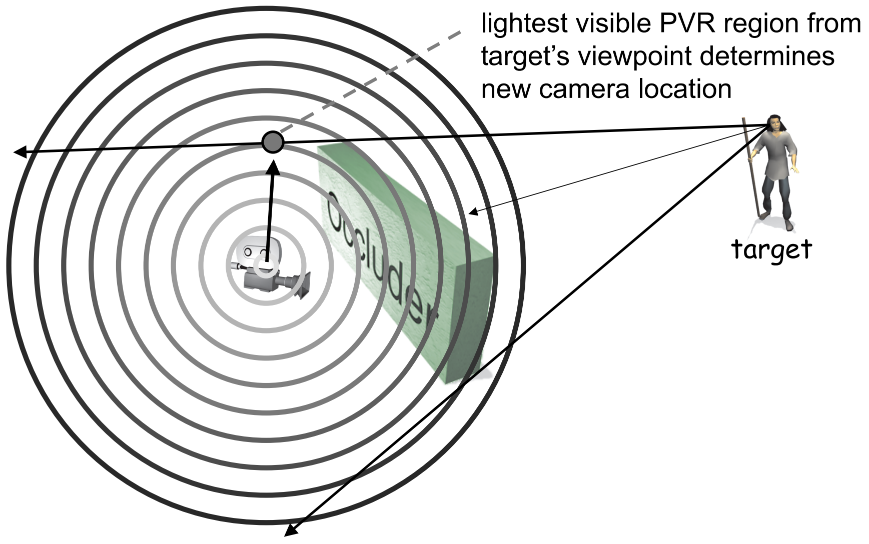

lightest visible PVR region from
target's viewpoint determines
new camera location
Occluder
target

Figure 4.14: Solving visibility constraints with the PVR. In this example, the brightest PVR seen by the target is the closest unoccluded point at which the target becomes visible to the camera.

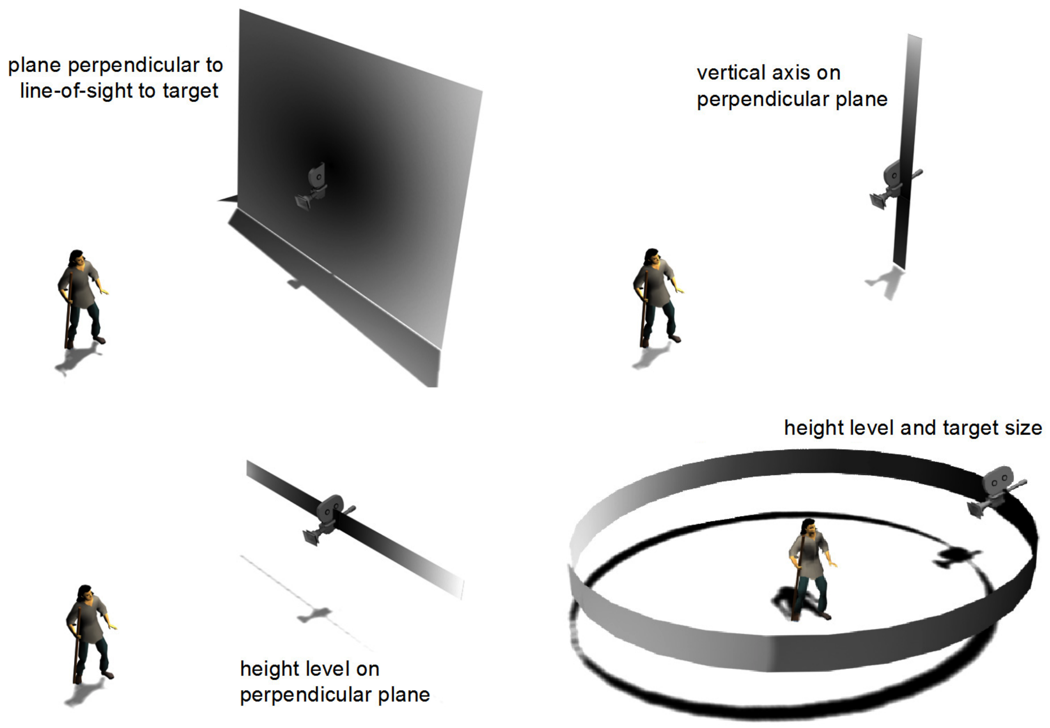


Figure 4.15: Four PVR constraints to find visibility solutions under various conditions.

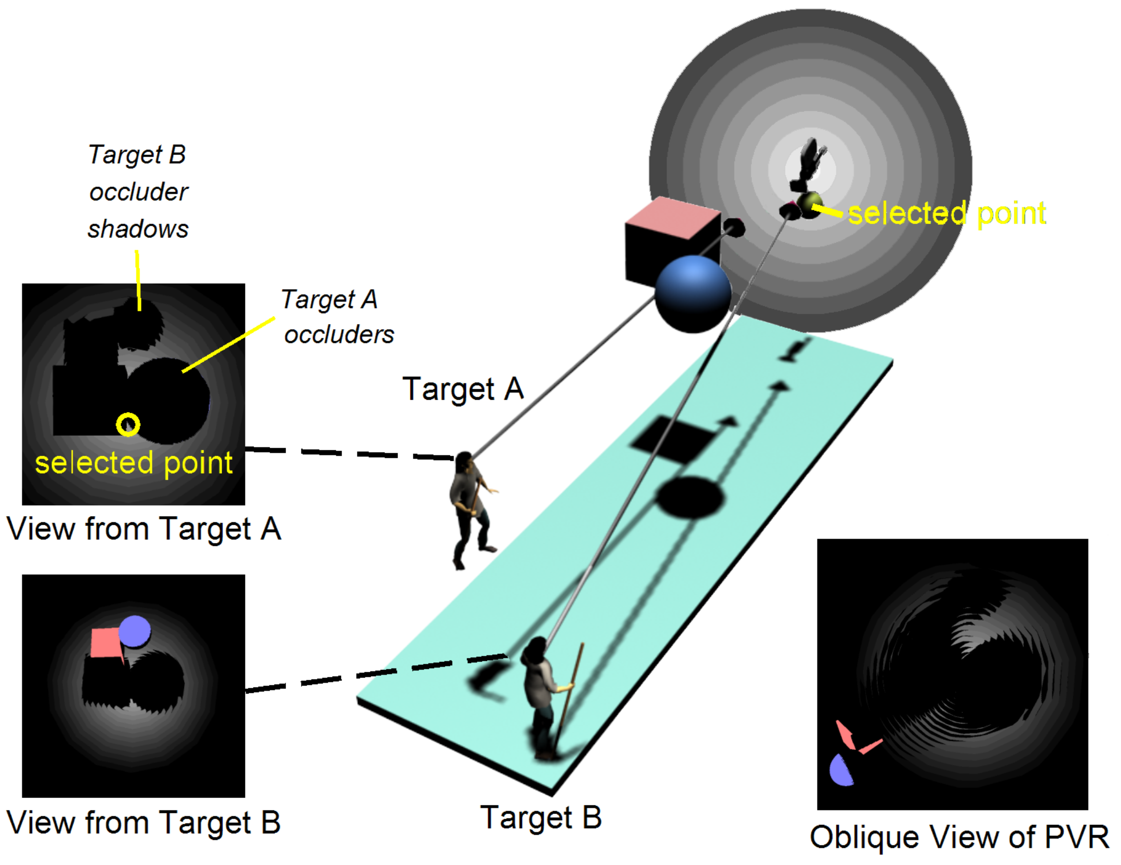


Figure 4.16: The multiple point visibility algorithm: View from Target A shows the visibility map view which includes Target A's occluding geometry and shadows cast from a point light source from Target B onto the PVR. The lightest point closest to the center of the View from Target A denotes the best point on the PVR to which the camera should move to. View from Target B computes the shadow map depth information to cast on the PVR in View from Target A—shadows from Target A and the PVR are rendered for this example only in order to provide a better context. An oblique view of the PVR with the shadows cast from both Target A and B is shown bottom right. Notice how the shadows define the occluded space.

of new shadow casting algorithms and hardware, the visibility search can be extended to return an unobstructed view for multiple points of interest in real-time. This allows a target to be represented with enough points of interest to cover its main features. A projective shadow casting algorithm [Zhang, 1998] can be rendered in real-time using hardware register combiners [Heidrich, 1999]. Alternatively, real-time can be achieved by stencilling shadow regions generated by object silhouettes [Kilgard, 1999]. In either case, shadows can be cast upon the PVR from each point of interest (that are in essence treated as a light source), resulting in a cumulative shadow that describes occluded regions from all points of interest. We render the potential visibility geometry from the point of interest closest to the camera and cast shadows from every other point of interest[4]—this spares performing one full read of the depth buffer and shadow rendering for the closest point of interest. A visual example is shown in Figure 4.16, whereas pseudo-code to detail the algorithm is shown below:

```
// SETUP STAGE: read shadowmap information for occluders
// first find point of interest (POI) closest to camera
closestPOI = POIclosestToCamera();

// then generate shadow maps for every other POI
for i = each POI that is not closestPOI do
    setViewMatrix(POI facing cameraPosition);
    clearBuffer();
    drawPotentialOccluders();
    POIshadowmap[i] = readDepthMap();


// STAGE 1: write occluder depth information to buffer
// using view from closestPOI
setViewingMatrix(positionAtclosestPOI, pointToCameraPosition());
clearBuffer();
disableColorWrite();
drawPotentialOccluders();


// STAGE 2: render PVR
disableLighting(); // just want PVR base colors
enableColorWrite();

// all lighter colored PVR geometry will write its color
// and stencil the areas that pass the visible depth test
for lightest_color_PVR_geometry to
    darkest_color_PVR_geometry do:
        render_and_stencil_PVR();
```

[4]Further investigation may reveal more advantageous views from which to compute the visibility map in terms of precision and consistency of results across frames

```
// configure nVidia register combiners so that black
// color values are only written when a rendered pixel
// is in shadow.
glDepthFunc(GL_EQUAL); // only render if matches depth in buffer
configureRegisterCombinersForShadowAlgorithm();

// Now render shadows cast from the other POIs onto the PVR.
// Each POI sets up texture generation based on its viewing
// matrix. The register combiner setup tests visibility by
// comparing rendered geometry depth against shadow map
// depth values. If a pixel is determined to be in shadow
// it is written in a black color.
for i = each POI that is not closestPOI do
    configureTexGenForPOI(i)
    renderPVRregions();


// STAGE 3: extract 3D position of best visible PVR
// find best PVR color and visible pixel location
Scan pixels in buffer:
    best_PVR_position = lightest_color_closest_to_buffer_center();

// now reverse project from position and depth in buffer
// to get 3D world coordinates
depth          = readDepthValue(best_PVR_position);
unoccludedPoint = UnProjectTo3D(depth, best_PVR_position);
```

The cost of solving visibility using multiple points is linear wrt. the number of points we solve for. For each point, we need to render a view (depth-write only) and read from a depth buffer. This buffer can be set to as little as 32x32 pixels to reduce polygon-fill and read-buffer overhead, and gives approximate results that may produce inaesthetic shadows for visualisations purposes, but serve well for occlusion information. Note also, that the occluding geometry need not be as complex as the actual geometry visualised by the player, allowing further reduction of rendering cost by using coarser occluder geometry as a representation of the actual finer detailed models.

4.3.4 Camera Expert

A camera cannot make an intelligent move without at least considering a future situation. In order to achieve higher frame-coherence and smoother camera movement in a reactive environment, the camera is to progress consistently from frame-to-frame, without disorienting the player with rapid swinging camera movements. This is the role of the camera expert.

We are able to make some guess as to where objects and the camera are likely to be a given time t in the future. To do this, we use approximative calculations for future scene and object state based on past trajectory and acceleration information, and solve the camera for that predicted state. The current camera trajectory is adapted so that the camera will be at this predicted position at the given time t, and we compute an estimated camera position for the next frame along this path. Now the camera expert calls the constraint solver on that expected position, to give the solved camera state for the next frame. Slight deviations to the expected may occur, but because recomputations are made every frame, the camera is able to intelligently adjust ahead of time to a large number of situations. The camera expert is summarized in Figure 4.17.

4.4 Capturing Action

In previous sections tools for specifying camera placement have been described. In this section, we look at methods of automatically guiding the camera to specific locations at appropriate times[5]. All kinds of games can be characterized by a small number of "interesting" events in which the players have to make important decisions or execute appropriate actions which lead to significant changes in the game's status. It is often of interest to the players to view, or review, such situations for any one of a number of reasons.

We present an approach for the automatic generation of action summaries from computer games whereby interesting events can be extracted by the analysis of specific game variables [Halper and Masuch, 2003]. Considering information like player status and 3D information allows an elaborate analysis of what events are taking place. On the basis of a game's log our methods propose a selection of events that represent the most exciting action scenes. These scenes can then be presented to the spectator as a sequence of representative images after the game, or given as timing requirements over to the camera for live viewing of action.

4.4.1 Related Work

Many games, especially sports titles, have features for replaying an exciting event shortly after it has occurred. Options typically include a choice of camera angles, and forward and rewind controls. Typically, though, it is left to the viewer to find interesting events manually from a fixed set of camera viewpoints.

In addition, not all games exhibit specific highlights. For instance, whereas sports titles (e.g., soccer) have defined action that occurs (e.g., a goal being scored) they often have subtle measures of good play (e.g., an excellent sequence of passes towards the goal). If a player requests to see a three minute summary of a game, and, for instance, no goals have been scored, what three minutes of play should be covered then? Thus, measures are needed for automatically evaluating quality of play.

Rather than replaying an event, the player could also choose to follow a game as a spectator. In this case, the camera should present the most exciting action to view. Thus, spectators who wish to attend interesting events during a game can be guided to the places and times where action takes place. In realtime spectator modes the challenge is in viewing the most

[5]We gratefully acknowledge the help of Nico Flohr whose earlier work [Flohr, 2001] and ideas contributed to much of this section.

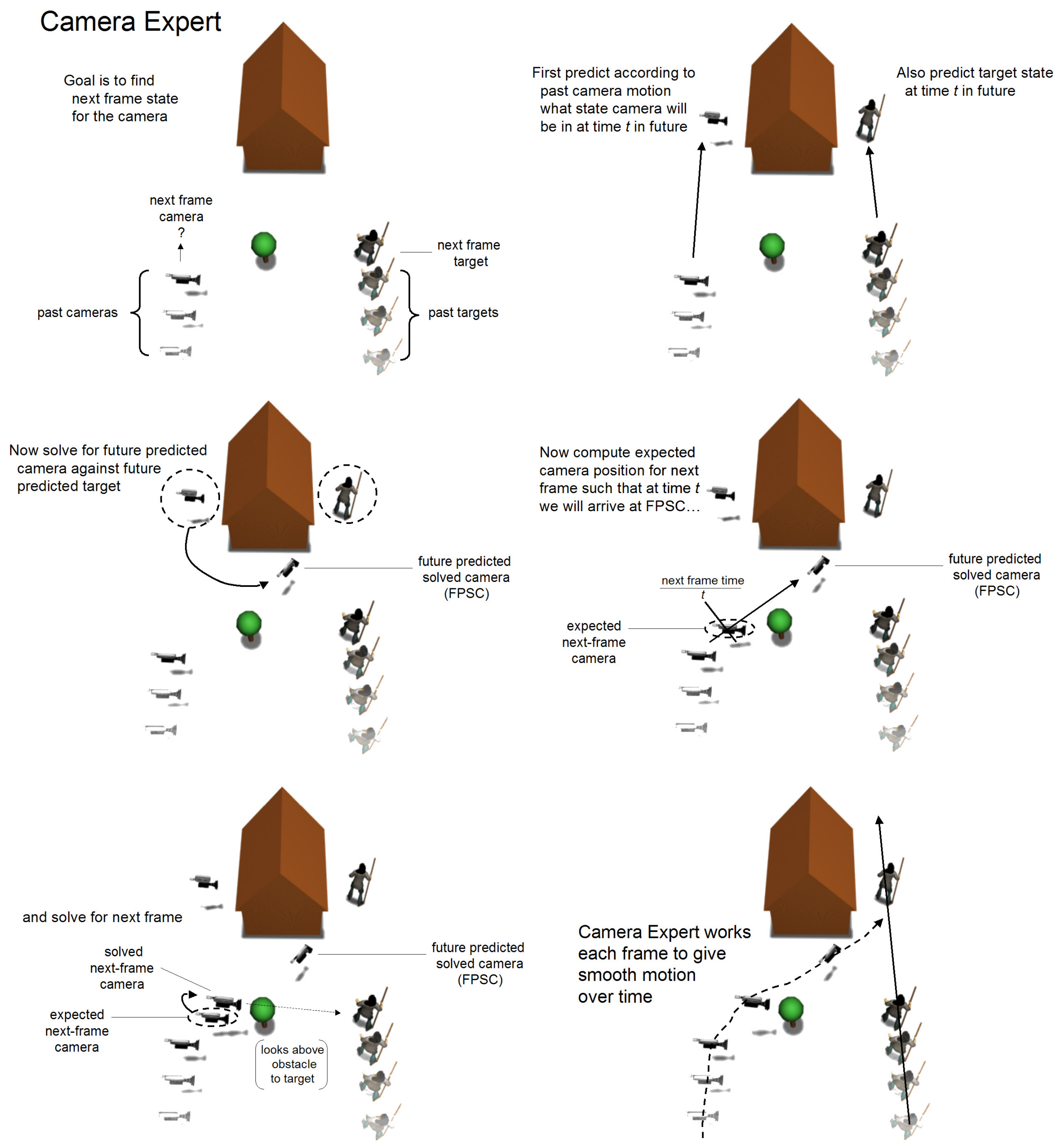

Camera Expert
Goal is to find next frame state for the camera
next frame camera ?
next frame target
past cameras
past targets
First predict according to past camera motion what state camera will be in at time *t* in future
Also predict target state at time *t* in future
Now solve for future predicted camera against future predicted target
future predicted solved camera (FPSC)
Now compute expected camera position for next frame such that at time *t* we will arrive at FPSC…
next frame time *t*
expected next-frame camera
future predicted solved camera (FPSC)
and solve for next frame
solved next-frame camera
expected next-frame camera
looks above obstacle to target
future predicted solved camera (FPSC)
Camera Expert works each frame to give smooth motion over time

Figure 4.17: Stages in the Camera Expert

interesting events when there are several events going on simultaneously. Thus, a fundamental question arises: how can we determine what makes an ‘interesting’ event? Half-Life Television (HLTV) was a big success at a game tournament that captured action during live games [HLTV]. However, there remain problems in setting up an appropriate context for the viewer—the camera would often jump to ‘death scenes’ only a few seconds before they would happen, but this would leave the viewer disoriented because not enough time was given to establish a context rising to the climax of the action. In multiplayer shooter games, the rising action, arguably, is more interesting than the actual instance of the death.

Video summarizations have the task of selecting a number of representative images from n frames of an image sequence (e.g., [Brand, 1997; He et al., 1999]). Techniques deal primarily with automatic image recognition of 2D images from a sequence of frames. Key frames are detected by looking for cuts in the film, camera movements, or blending in successive frames [Ahn and Oh, 1995; Patel and Sethi, 1997; Stringa and Regazzoni, 2000]. However, statistical information for each frame can be generated, but little contextual information about events within the frames can be considered, and thus are not suitable for action summarizations of computer games. Games, especially action games like *First Person Shooters* (FPS) usually do not comprise ‘cuts’ in the action either (e.g., game play is linear in time, whereas movies typically edit time). In this respect, games can have long time spans of uninteresting action (e.g., when all players are lurking, waiting for someone to move first). Others attempt to summarize movies semantically and apply similar techniques [Vasconcelos and Lippman, 1998]. In order to structure films into logical components, only information about the film’s contents or the analysis of frame information can help [Hanjalic et al., 1999]. Further applications of video summarizations can be found in [Gunsel and Tekalp, 1998; Wactlar, 1999]. In addition, it is possible to extract new information about subsequent images through interpolation. For instance, a number of computer vision techniques and hidden-markov models can be applied to 2D images to describe a sequence [Brand, 1997]. However, good results are only achievable under a controlled lab environment. Eickeler et al. [1998] try a similar approach, but limit it to recognizing human gestures.

These works show that information about the contents of frames is important for the evaluation of film, animations, and computer games. The key is to extrapolate relevant data and its contribution to contextual information. Computer games can be thought of as a type of digital film—including the need to find relevant data contained within each frame in order to be able to extract a kind of summarization from it. Therefore, for a function representative of features or action in a game, a certain combination of input data can be found to give an evaluation of excitement, or interest, within the game.

4.4.2 An Evaluation Framework

Our goal is to provide a general framework for multiple game genres to support (1) live viewing of action, and (2) action summarization. In contrast to extracting information from videos and images, computer games have the significant advantage that all necessary information for summarization is already contained and readily accessible within a closed and defined world. On the other hand, games do not only comprise a singular sequence of events and typically have many things happening at the same time. Thus:

- Contextual information can be directly generated from the game world.
- Large amounts of data that comprise the game have to be filtered to only those relevant for action extraction.
- Multiple independent events occurring at the same time in a game require efficient computation and methods for comparison.

Thus, data must be efficiently stored in some kind of 'game log' that represents a game's history readily available for efficient computation and comparison between multiple events. For this purpose, we introduce an evaluation function over time that returns how relevant, or interesting, a particular moment is at time t. We name this function, the i-function (for 'interest' function). An example of its possible output is shown in Figure 4.18. From this, peak moments of interest can be found (local maxima), and also whether an event is becoming more interesting (positive slope) or less interesting (negative slope).

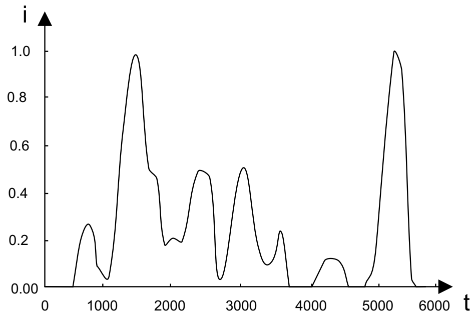


Figure 4.18: An example of an i-function (x-axis is time, y-axis shows interest value)

For fast storage and extraction, the evaluation data is sampled into arrays. The time-intervals for each computed value by the i-function are kept regular to represent the same evaluation space. The fidelity of the sampled data is determined by the *sampling rate*—lowering this rate reduces storage, but also reduces accuracy of representation. However, sampling rate can be considered high enough when maxima and minima in the evaluation representation can be found within a certain accuracy based on how *quickly* we must react to changes in action. For instance, in a fast-paced action game (like QUAKE) changes in game status are occurring all the time, thus our sampling rate should be high enough to capture this. In strategy games, like COMMAND AND CONQUER, even though action is constantly present, changes in maneuvers and other events could be sampled every second. Hardware constraints may also limit sampling rate, such as network data transfer rate in networked games.

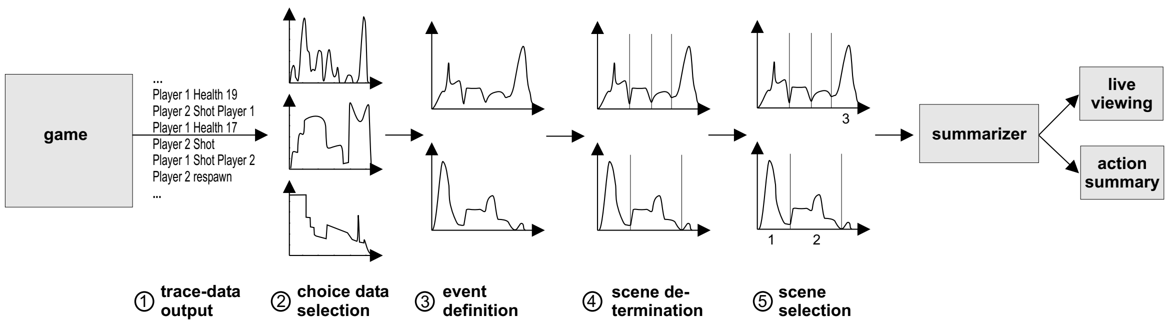


Figure 4.19: Framework for our action extraction system

In Figure 4.19 we show our evaluation framework for action extraction. The first stage outputs data from the game relevant for action extraction. This data, or *trace data*, can be stored whenever it *changes* with an associated time stamp, thus providing an efficient signal-based means of storage. Specific variables from its contents that potentially comprise part of some action are then processed. In most cases, many different variables contribute to perceived action. Thus, in stage 3 results are combined that define action events. Since action is normally perceived as a *sequence* in time, events are split into scenes in stage 4. Scenes can then be weighted according to certain criteria (stage 5) in order to aid the selection of particular action sequences either for summarization or live coverage (stage 6).

The first three stages deal with extracting and converting game variables into $\imath$-functions, and are covered in Section 4.4.3. In Section 4.4.4 we detail stages 4 and 5 that group action into scenes. Section 4.4.5 then covers options of using this information for capturing action as summarizations or live coverage.

4.4.3 Defining Action

Action typically occurs when the status of a game or player has significantly changed state that contributes to the overall perception in achieving goals in a game. For instance, FPS games concentrate on the elimination of opponents. Reduce an opponents energy (hitpoints, armory or whatever) enough times to zero, and your victory ensues. Every time you successfully manage to target your opponent under the shot of your weapon, a degree of satisfaction arises—something has happened, you have managed to hit him, and his energy has changed state. More than that—this was the point in time where you gained the lead. Your number of ‘frags’ just exceeded your opponents’. But wait, one additional factor – it was your last shot of ammunition, and you pumped out those bullets faster than ever before. *This* would be an interesting scene. Strategy games, on the other hand, exhibit a more linear increase in the progression of the game status as actions in general have more long-term consequences. This makes it more difficult to define the *i-function*, although we hypothesize that there are successful combinations of genre-specific data that lead to action. In the following subsection we introduce how game variables can contribute to perceived action. However, game variables are only pieces of the action, thus we also describe how 'real' action can be found by analyzing combinations of variables over time.

Choice Data

A first step in detecting action is in determining which game variables contribute towards an interesting event—we call these variables *choice data*. Typical variables that contribute to action are attention focus, visibility information, positional information, player status (hit points, ammunition etc.), and game status. For instance, in the multiplayer example above, FPS usually exhibit very abrupt and extensive changes in the games status such as player health, rate of fire, and the moment of victory in a deathmatch—all of which contribute to the game goal of eliminating opponents.

Usually, choice data related to game state is interesting to evaluate in terms of changes to this data. For instance, in our FPS example, changes in a player's health. The actual value of a player's health is not particularly exciting – more exciting, is when this value drops to indicate that the player is involved in some kind of dangerous activity. Thus, for this kind of choice data, we indicate how it affects action by computing the derivative of its data $vs.$ time function. In this manner, the times when data is changing is represented as well as the magnitude of the change. The amount of change is likely to correlate with the degree of action.

However, computing a derivative to discrete changes in data results in sharp peaks of the curve. Since action is perceived over a duration of time, and not in split seconds, we compute a moving average over the curve. Applying the moving average over greater time-spans places more emphasis on consistent sequences of changes to choice data, giving higher values when more changes happen *over time*, whereas a short moving average results in more interest being given to individual changes in choice data.

Choice data computed from an *evaluation* of a game state can directly contribute to action detection. For instance, the visibility properties of an opponent in a players view—the larger and more visible the opponent is, the more the opponent contributes to a perceived part of the action.

Defining events

A variety of factors may contribute to perceived action. We can define action as a group of particular occurrences over a particular time period. In this respect, choice data is often interdependent in contributing to perceived action. For instance, shooting a weapon may only be relevant to action when additional variables, such as opponent visibility, also have suitable values. We term an *event* as the combination of choice data that indicates *when* and *how* choice data contributes to action over time. Event definition can only be done semi-automatically, as the overall visualization goal has to be initially specified by the user, and these may be as simple or complex as necessary.

As an example, in a racing game for a car over taking maneuver, we can consider position, acceleration, and proximity to other cars. The position of cars detects the actual ‘over-take’ event, whereas acceleration and proximity variables contribute to how exciting the over-take was. Thus, we could apply a short moving average to changes in position (a few seconds) and multiply this i-function to the acceleration and proximity i-functions. The result is an event that reads zero when no over-takes are occurring, but which rises to values representative of our various action criteria when there are.

4.4.4 Determining Action Scenes

At this stage, we have several i-functions that represent levels of action for events. These can be referenced to determine which periods of time for which events are more exciting than others. As action rises and falls, the task is to find sequences of frames that cohere to a single *scene*. Despite the lack of a precise definition for a scene within cinematography, it is largely accepted to be a complete unit of film narration that comprises a series of shots or single shot. In addition, each shot is bound by a given spatial context—whether an event, a physical location, or a group of characters engaged in some activity.

The particular context for a scene within each i-function is currently provided by the programmer who defines conditions for input of choice data. For instance, a physical context for an i-function can be established by confining its processing of choice data to a defined area. In this manner, for games comprising multiple locations that segment action, each location could have its own i-function to process all events that occur therein. Therefore, any extracted scenes from those i-functions would be bound to a particular location. In contrast, certain games like multi-player action games involve characters moving across several areas within an environment. The action in this case is not divided into particular locations, but is rather defined by groups of interactions. As a result, i-functions can be dynamically created for each detected group of interactions over a period of time. In the simplest case of a first-person shooter where a number of players compete against each other in a small confined environment, it may be sufficient to have a single i-function represent all the random interactions between individual competitors and their setting.

In addition to providing a context for the action, an i-function defines each of its scenes with:

- A beginning and end point in time
- A unique frame used to represent the scene
- A value denoting its importance

A measure of a scene’s importance is necessary for action summarization as each scene in every i-function must be compared for selection. The calculation of the above points for scene definition are discussed accordingly in the following subsections.

Defining Scene Boundaries

Characteristics of the i-function can be used to split action into various segments. The i-function typically exhibits minima and maxima—peaks of action lie at local maxima which reside between two local minima. Between these points, the curve either slides up or down, indicating rising or falling action. Therefore, scene boundaries can be defined by local minima of the i-function, with the representative frame for a scene, or its *climax*, as the maximum value therein.[6]

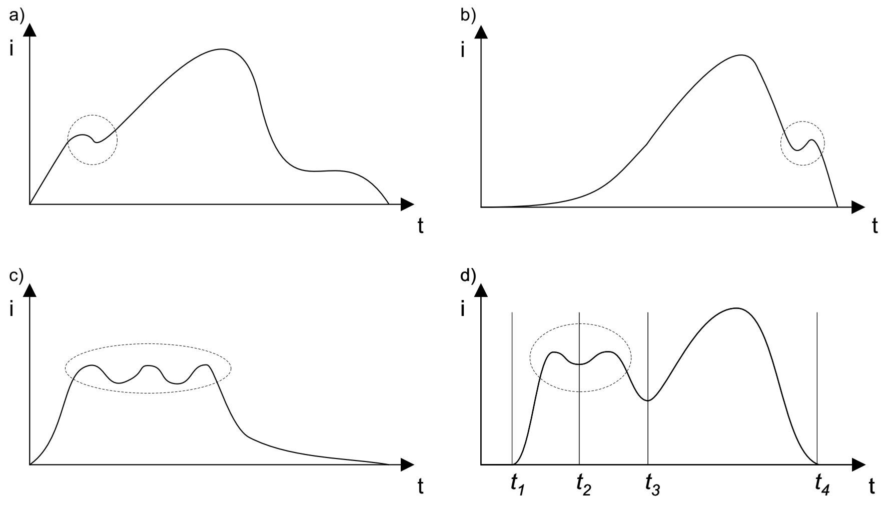


Figure 4.20: Problems with defining scenes between local minima only

However, sometimes a number of local minima can be found close together in time which causes problems in the evaluation of what should comprise a scene. For example, in Figure 4.20 we observe a number of problem cases. In Figure 4.20 (a) and (b) we observe a short fall and rise in the overall slope of the curve, whereas in Figure 4.20 (c) we observe local minima near the 'peak' of interest. These are not points where one would logically split action.

In Figure 4.20 (d) we also encounter an inappropriate scene division. Here, t_1 marks the beginning of a scene, and t_2 the end of it and the beginning of the next scene. However, a spectator would feel that these two scenes should actually form parts of the same scene, and thus would remove t_2 so that the scene sequence is composed of two scenes ($t_1 - t_3$ and $t_3 - t_4$). To compensate these errors, two neighboring scenes are merged whenever the value at their shared boundaries are close in value to either of their local maxima. For example, the scene distance between a local minima at a boundary and a local maxima is shown in Figure 4.21, which is found to be lower than a given tolerance and thus the boundary is

[6]The problem of detecting local and global minimum or maxima is well know in mathematical optimization theory. A number of strategies can be seen in [Chong and Zak, 2001].

removed to produce the merged scene. We continue to do this until no two neighboring scenes have a minima-maxima difference below the given threshold.

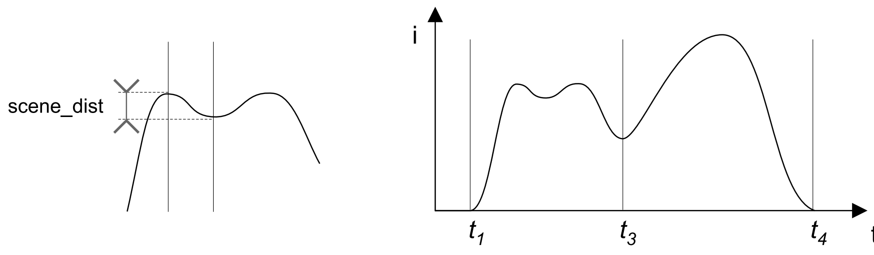


Figure 4.21: Merging neighboring scenes with small differences in interest between local minima and maxima.

Weighting Scenes

In selecting scenes we can observe a time/interest problem that deals with the length of a scene $vs.$ its relative importance. Is a scene more important than another when it is longer or rather, when the peak moment of action in the scene is higher? For instance, in Figure 4.22, given a choice of one scene to use in a summarization, which should be chosen?

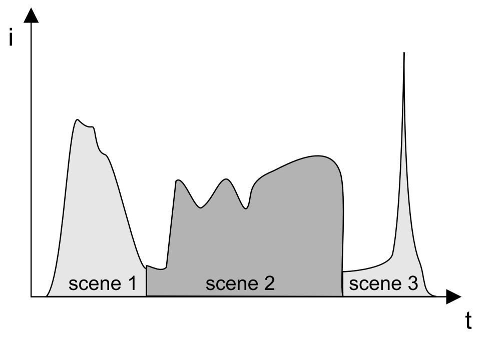


Figure 4.22: Which scene is most important?

To evaluate the relative importance of scenes, the respective merits of both a scene's duration and content (interest values) should be assessed. Below, a number of methods for selecting scenes are outlined and briefly analyzed:

- *Order-dependent selection*: every n^{th} scene is selected. This is the most trivial method of selection that allows action to be summarized evenly across the history of a game, which leaves a good impression of its general scope, although key scenes may be missed.

- *Duration-dependent selection*: scenes are selected based on their time-span. Whereas this can return scenes that match a suitable duration, they might not hold much interest.

- *Representative-dependent selection*: scenes are selected based on the value of their global maxima. The advantage of this selection criteria, is that the n most exciting points in time will be represented. However, short bursts of high action will always be favored over longer scenes that comprise more strategic elements with slightly lower maxima (see Figure 4.22).

- *Area-dependent selection*: scenes are weighted according to the area under their curves. This directly addresses problems with duration-dependent selection (no consideration for the interest value of a scene) and representative-dependent selection (no consideration for duration of a scene). Thus, this method is of good practical use.

The algorithms for scene-based selection demonstrate only a few possibilities, but they suggest important clues for introducing new criteria and extending or combining scene-based selection strategies. For example, the following factors can also be considered: average slopes towards local maxima, number of local maxima in a scene, duration-dependent selection to vary duration of scenes in summarization, and other time/interest correlations. For instance, scenes can be weighted depending on a more time representative criteria for a post-game summarization. Conceptually, a good result would return a distribution of scenes over time and select some scenes that, even though containing less action, would give the player a better idea of what actually happened over the course of the game. For instance, the weightings of neighboring scenes can be reduced according to their distance in time from a selected scene (i.e., its immediate neighbors are greatly reduced in interest because they lie very close in context to the chosen scene).

Weighting Concurrent Scenes

In games we often find a number of concurrent independent events. This requires that each event be evaluated and weighted for selection. For instance, if we consider a car race with three drivers like in Figure 4.23 we simply weight each scene respectively. Determining the order of importance can be done by sorting scene weights across all events.

In some cases, the types of events that are summarized can be varied. This can also overcome potential ‘normalization’ problems with weightings across different event scenes. This can be achieved by dampening each scene weight in an event sequence whenever a scene is selected.

4.4.5 Timing the Action Capture

The weighted action scenes provide information regarding where and when action takes place. In a real-time spectator mode, it would be of little use to detect where an exciting

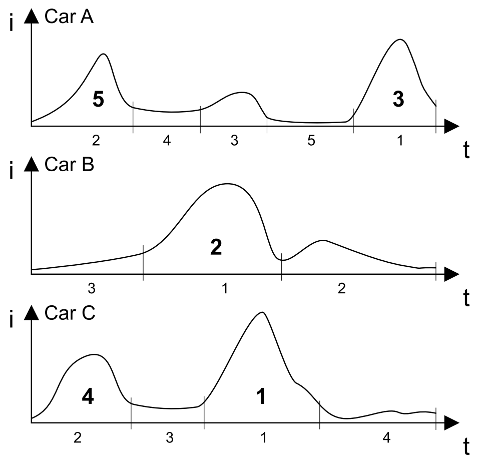


Figure 4.23: Scenes of a car race are weighted in order of importance across three concurrent events that are split into their respective scenes.

event has just occurred only to have the camera jump to the scene of the crime after it has been committed (unless we are talking about a detective game). Therefore, it is necessary to integrate techniques that involve the timing of the camera shot according to interesting events, so that the camera can be positioned in place just as the action is about to happen.

We describe two methods for viewing live action: (1) viewing of the current most exciting scene and persist in viewing until it is over, and (2) viewing the current most exciting scene at every instance. When viewing a scene to completion, we have the advantage that the spectator will observe an identifiable sequence with build-up and denouement of action. However, often action will go missed if another independent event occurred during the viewing of the current one. These situations can be overcome by briefly buffering the action and viewing concurrent events synchronously (i.e., re-viewing an event that was missed at a later time when little action is occurring). This is generally favored over a simultaneous display in a split screen as it gives the viewer a single focus. However, if sequentially queuing action is not an option, then jumping to a more exciting scene as it occurs can be adopted instead. We discuss both methods in the following subsections.

Viewing a scene to completion

To view a scene until completion, we must first consider when it is appropriate to start viewing the scene. In Figure 4.24 we see three currently available scenes and their respective weightings. If we simply select the highest weighted scene at time t we would select scene *1*. However, t lies after the peak value in *1*, suggesting that most of the action has already been missed.

A viable alternative is to select a scene to view based both on its weight and whether or not we still have the climax of the scene to come. In this manner, a worst-case scenario

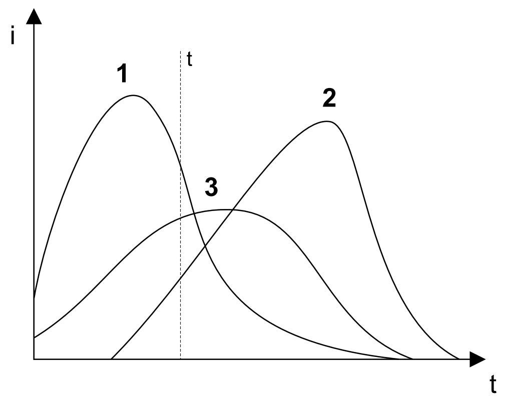


Figure 4.24: Which scene shall we start viewing at time t? Scenes are numbered according to highest evaluation.

can be avoided as shown in Figure 4.25 where each successive scene is less interesting than a previous one. For the selection strategy of choosing the best scene at a given time when we just finished watching a scene, we get the timing results shown with line *A*. For instance, once we finish viewing *1*, we then choose *2* to view next since it has the highest scene weight from the available choices. However, *2* has passed its climax, but we view it to the end. Once we finish viewing *2*, *3* is the next best scene, but again, we have missed its climax. Thus, we view each scene in turn, but miss each scene's climax. Compare this to the selection strategy *B*, where we favor scenes that still have their peak to come. In this case, we are able to view most of the action in *1*, *3*, and *5*. The price, however, is that we have missed *2* and *4* entirely.

Viewing the most exciting scenes all the time

Given a time t, which scene should we be viewing at that time? This differs slightly from selecting a scene to view until the end, because we are in a position to jump between scenes before the scenes are over. Thus, we need to make sure that we are actually jumping to view more interesting situations, rather than jumping out of them.

Here are a number of viable arguments and we will scrutinize their implications and results. Our points below follow the curve and results shown in Figure 4.26.

- *Method A*: view the scene comprising the highest i-value at time t. Thus, we view the most exciting moment at the given time. However, curve values can change quickly, and thus, the problem of jumping back and forth between scenes, as shown by the relatively rapid change-overs in Figure 4.26 from *1* to *2* briefly and to *3* before returning to *2* again. This also does not consider that the peak action might already be over, thus, even though the value of the curve is high, the actual value of interest may be low.

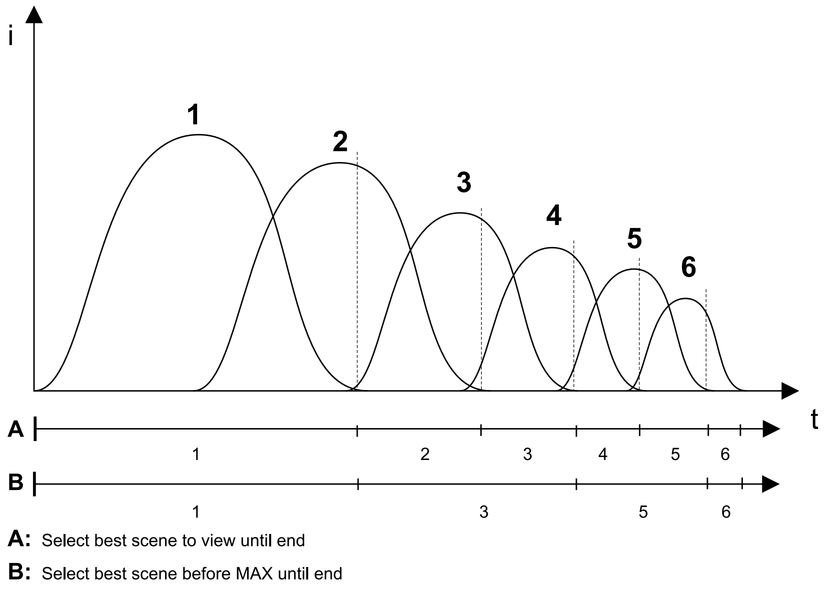


Figure 4.25: A worst case scenario for starting to view new scenes: each successive scene is less exciting than the previous one. The two lines under the graph show the selection strategy results if started viewing at time t.

- *Method B*: the scene with the highest value at time t, with preference for scenes with t before their maximum value. Similar to method *A*, except scenes are disregarded that have passed their climax.

- *Method C*: take the best scene evaluation present at time t, thus always viewing a portion of the most overall exciting scene. However, problems with the worst-case scenario occur as shown previously in Figure 4.25, where we will often switch to a scene whose peak interest is already passed.

- *Method D*: take the best scene evaluation present at time t, but favor those scenes that still have their peak value to come. A consequence is that sometimes we may be viewing the best scene, but at the point after its climax we may potentially jump to a less interesting scene.

- *Method E*: take the best scene evaluation present for time t, but favor those scenes that still have their peak value to come *and* their current value at t is higher than the best scene evaluation value at time t. This overcomes most of the problems of all previous methods. The best scene is viewed until its peak interest, and thereafter we only jump to another scene if that scene's interest value at the given point in time is higher than the one we are currently watching.

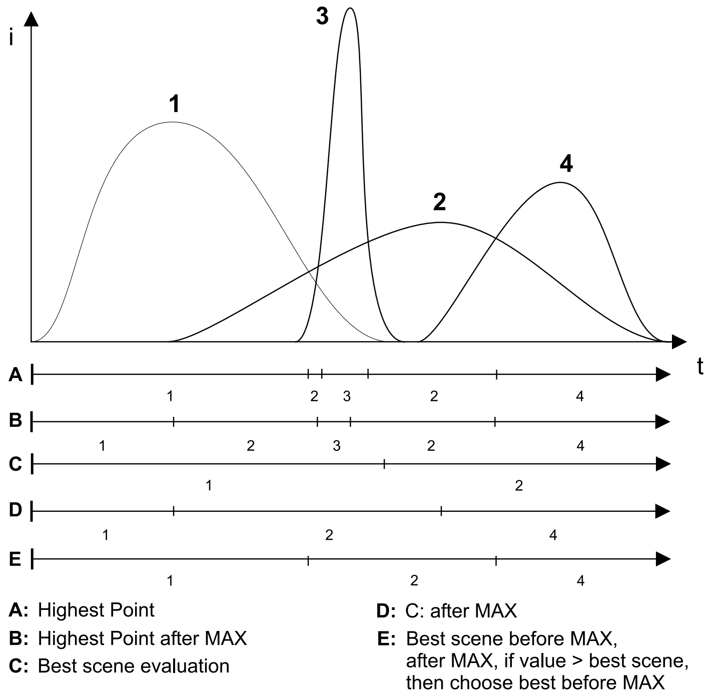


Figure 4.26: Four independent scenes and the results of different selection strategies for viewing them in a linear fashion

4.5 Implementation and Evaluation

The camera techniques from Section 4.3 are implemented into a real-time camera engine called the PREDATOR[7]. The more successful the camera engine is, the more subtle its satisfaction of visual goals will be. For this reason, we find it best to evaluate camera movement over specific scenarios in the following subsections. We begin by applying a 'camera-chase' shot specification on three different scenes. Look-ahead algorithms and relaxation of constraints are important for smooth frame-coherent results, which we demonstrate by discussing the effects of removing key features in the PREDATOR system. To improve computational performance in highly detailed scenes, reduced LOD can be used in PVRs for visibility satisfaction. PVRs also find an alternative use in assisting pre-defined camera paths. In addition, the PREDATOR is tested in a real-time multi-player game environment which allows the user to tweak visual specifications. Selected videos of these results can be viewed on the accompanying CD or at the following website: `http://wwwisg.cs.uni-magdeburg.de/~nick/cameraEngine`.

Finally, the algorithms from Section 4.4 form the SAMURAI system for action extraction, applied to a popular commercially available game in Sections 4.5.6 and to the author's game for post-game summarization in Section 4.5.7.

[7]PREdictive DeclarATive frame-cOherent Response camera system

4.5.1 Exploration

To allow the player to explore her environment, we constrain the camera to focus on the object using a *size*, *visibility* and a *lookAt* constraint. This gives the effect that the camera 'chases' the player, maintaining an unobstructed view at all times.

We have tested the exploration template on three different scenarios: (1) A teapot flying through a highly cluttered attic; (2) a helicopter flying through a city; and (3) a human figure exploring inside a medieval building. Sequences of images representing the teapot and helicopter animations are shown in Figures 4.27 and 4.29 respectively. The camera performs remarkably well in all three cases, avoiding obstructions and collisions, being capable of adjusting appropriately to many situations despite the varied spatial arrangements and complexity of each environment.

4.5.2 Effects of Predicting Camera and Environment State

Here we show and evaluate stages in the evolution of our camera engine. Diagrams of typical results are shown in figure 4.28.

A naive implementation uses just the hard constraints without visibility solving. The results are smooth, but collision through objects and obstructed player views are frequent. A better version uses constraints including visibility computation, but without tolerance settings or predictive measures. We expect that these results are similar to the work of Bares and Lester [1999], causing 'jumping' artefacts and poor frame coherence. Next, we introduce relaxation of constraints. The camera movements are smoother, but still have problems with frame coherence when jumping out of visibility traps. After applying inertia to the camera, so that it prefers to continue along the path it came from, results are less jumpy than before, but we get a ping-pong effect from the change of directions when resolving visibility positions. Finally, estimation routines adjust the camera movement to solved predicted camera states. Now the camera is able to adjust ahead of time to players moving around corners and through objects, and accommodates to situations where the player enters a room, providing a view of the door and its contents before the player is fully inside. In our implementation, we achieve surprisingly good visual results from state predictions based only on past motion data. In a computer game, the prediction accuracy would depend on what information can be supplied by the action module and simple extrapolation of past data would only be a worst-case scenario.

4.5.3 Using Level-Of-Detail Geometry

The effects of using coarser geometry for visibility constraints depend on the game scene and interactive freedom of the player. Our helicopter animation that flies through a city allows buildings to be represented as bounding boxes for use on the PVR, creating a substantial frame-rate increase, without detriment to the camera movement. However, those objects which the player may move through (e.g., moving under a table or through a hoop),

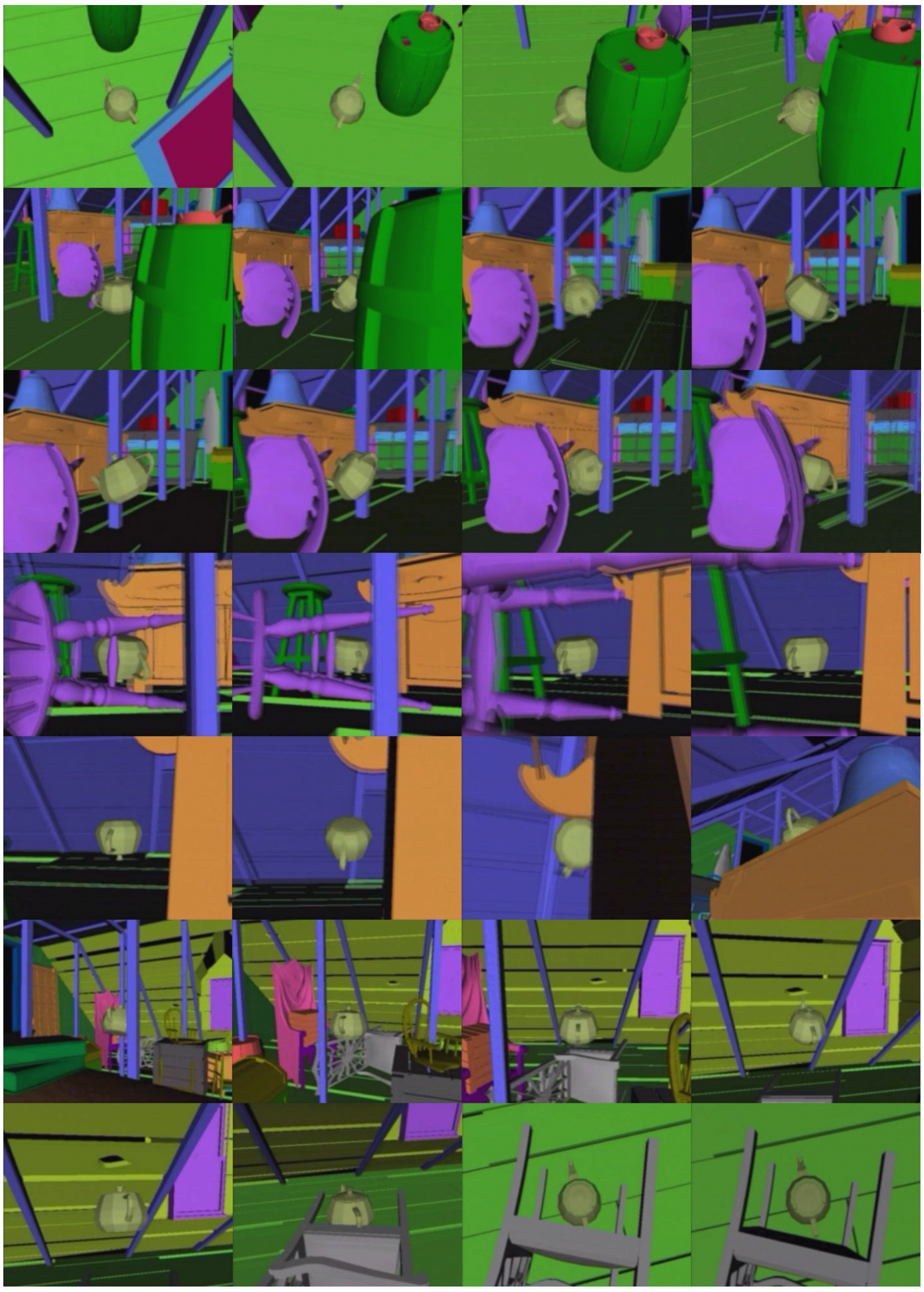


Figure 4.27: Frames from an animation sequence in which the camera chases a flying teapot through a cluttered attic.

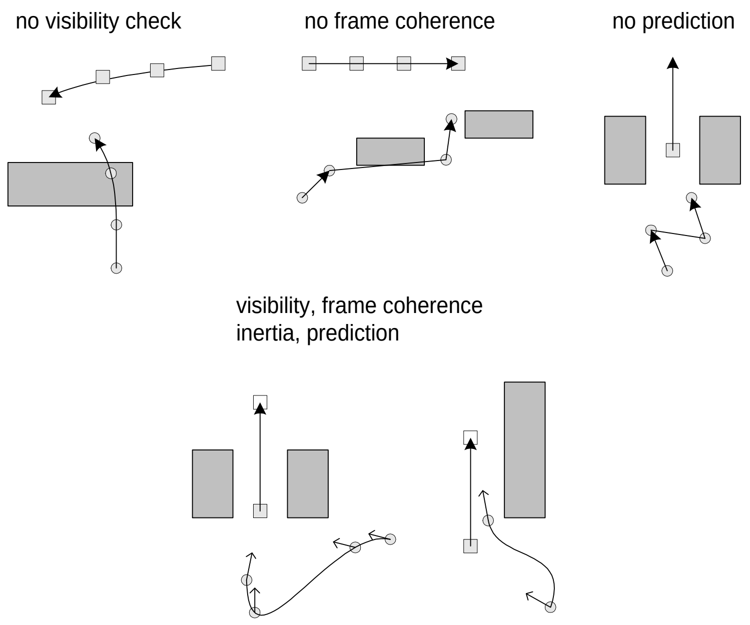


Figure 4.28: Common artefacts of reactive camera planning, only with the introduction of predictive camera planning can we react to avoid certain situations.

not only need to share strong similarities in topology, but must also guarantee that the simplified object's volume is a superset of the original object, which prevents the camera from 'colliding' with the visualized geometry. Large supersets can also be used to make the camera take a wide berth around certain objects.

4.5.4 Fixed Camera Paths

In some cases prior knowledge about a scene is known. There might be a cut-scene in which the player is known to travel along a certain path, or a small enclosed room to which the player is constrained to a limited area. A cut-scene artist can then plan a fixed camera path to express certain visual goals or create dramatic views and effects manually. However, there may be certain aspects of the scene that have been generated at run-time (such as additional characters present) and could not have been accounted for in the game design stage.

In such cases, the camera must adjust for additional possibilities of occlusion, and we may place PVR along the preset camera path or fixed camera setups. For instance, we adjust the camera along the camera path whenever occlusion occurs, or cut to a new pre-setup camera position also defined in the PVR.

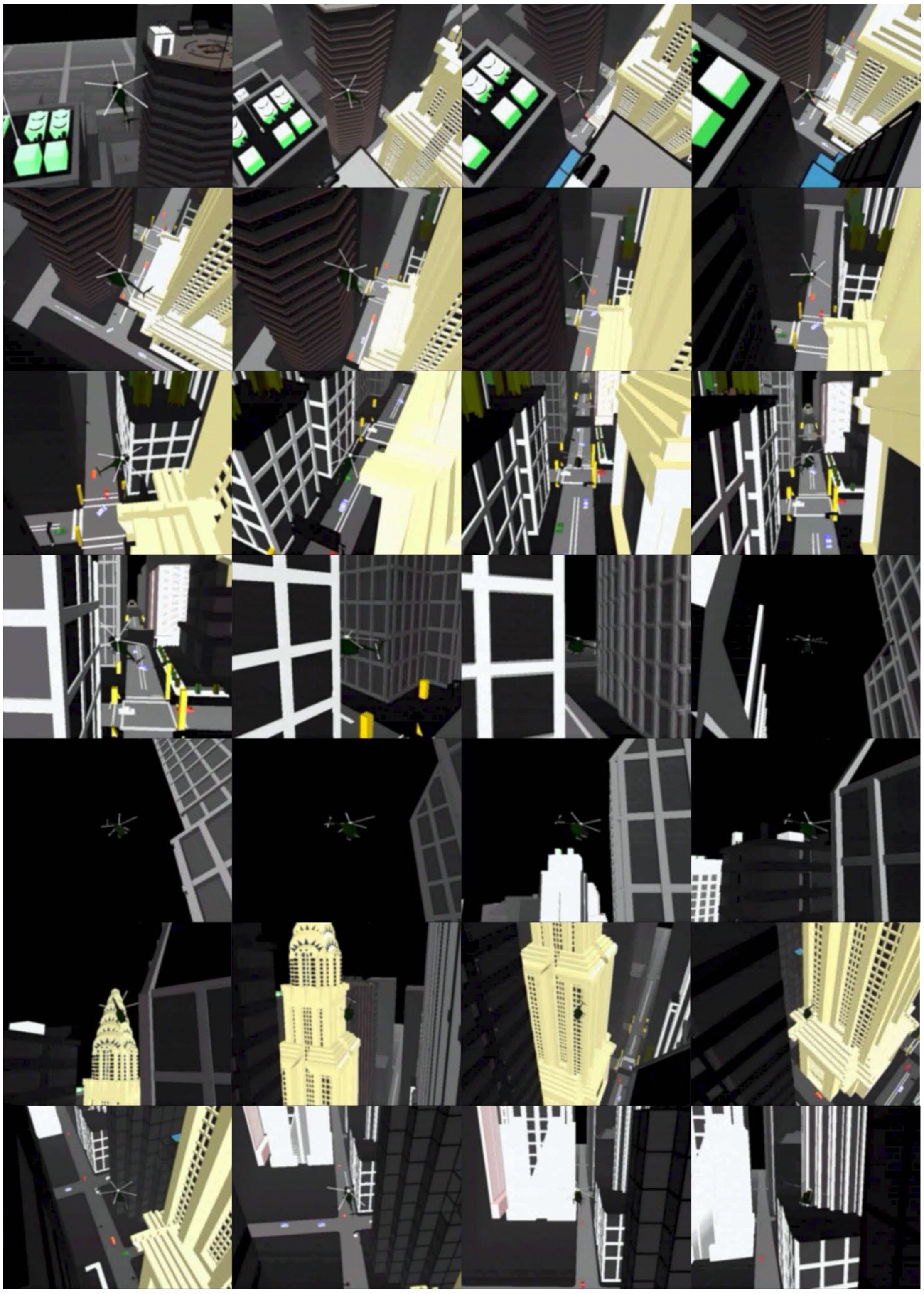
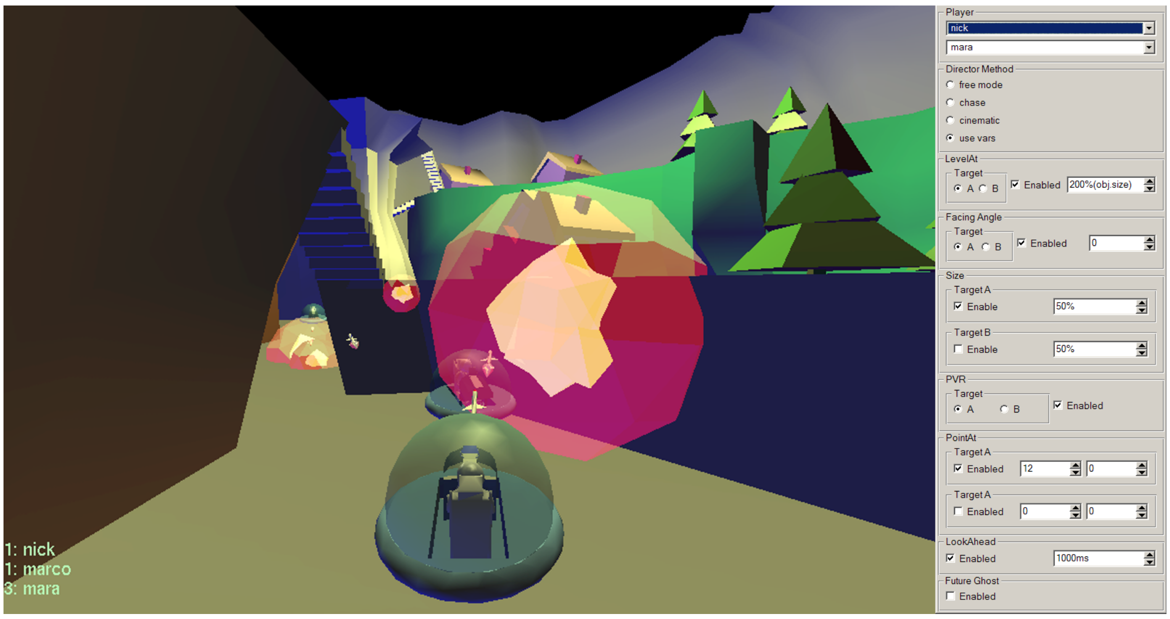


Figure 4.29: Following a helicopter through a city.

4.5.5 Camera in a real-time 3D Shooter

The author has implemented a prototype Quake-clone[8] game with multi-player network functionality. Players compete against one another in a complex 3D environment with the ability to navigate rapidly under and over bridges, hide behind objects, and shoot at their opponents. It includes a spectator mode used to test the PREDATOR system, shown in Figure 4.30. Here, the user can specify parameters for various constraints. In this case, the target (one of the players) is to be visible and have a predominant size and location in the image, with the camera levelled at about twice the player height, and provide a view of the general direction that the player is facing. The lookahead option is also checked, and in this case is set to view the action one second behind the live events in order to smoothly adapt the camera state to future conditions. Often, there are cases whereby the player turns to go behind an object, for which the camera orients itself ahead of time to cue the spectator as to the player's subsequent movement.



Player
- nick
- mara

Director Method
- ( ) free mode
- ( ) chase
- ( ) cinematic
- [✓ use vars]

LevelAt
- Target: [✓ A] B
- [✓ Enabled] 200%(obj.size)

Facing Angle
- Target: [✓ A] B
- [✓ Enabled] 0

Size
- Target A: [✓ Enable] 50%
- Target B: ☐ Enable 50%

PVR
- Target: [✓ A] B
- [✓ Enabled]

PointAt
- Target A: [✓ Enabled] 12 0
- Target A: ☐ Enabled 0 0

LookAhead
- [✓ Enabled] 1000ms

Future Ghost
- ☐ Enabled

Figure 4.30: Spectator mode for our 3D shooter game. Constraints settings for the camera can be adjusted with the interface on the right, which includes some pre-defined shot templates and options for viewing specific players.

4.5.6 Extracting Action

Although the action extraction algorithms in the SAMURAI system present work in progress, preliminary results are promising. The event detection by our system worked out very well

[8]ID software: see Quake at `www.idsoftware.com`

for our FPS examples. For choice data we used health, movement speed, and shots fired. We tested our techniques with two games, QUAKE ARENA[9] (cf. Figure 4.31) and our own gaming environment where we have better access to all of the essential information. Unfortunately, an implementation that combines both the PREDATOR and SAMURAI systems is not complete at this point and as such we lack a formal evaluation. Interestingly, even given just the first-person views, when the i-function rises it instigates a rising degree of anticipation in the viewer. Thus, we can expect that people like to see not only the exciting event itself but also the action that led to that event, requiring the camera to be positioned a few seconds in advance of the imminent action to provide a context.

One could also notice that action can sometimes be ‘falsely’ detected if players trigger events without any real cause. For instance, in our shooter games, we define damage and shot variables as part of some action. If a player stands against a wall with a rocket launcher and blasts himself to bits, this might be considered more an act of suicidal stupidity rather than something genuinely exciting. On the other hand, this might be occasionally entertaining, especially if a final game outcome is influenced. Frequently repeated events, nevertheless, become repetitive. This suggests that measures should be introduced to lessen the impact of similar events over time.

4.5.7 Post-Game Summarization of Action Scenes

Action scenes can be automatically summarized post-game simply by selecting the top n weighted scenes (see Section 4.4.4). We use our SAMURAI system, shown in Figure 4.32 to adjust the particular weights for choice data in defining events and scene construction on trace data from our 3D shooter game (see Figure 4.30). Typically, the duration of scenes in this example which comprise action typically lie between 5 and 20 seconds (longer sequences are observed in times of little action), which is a comfortable time frame for viewing.

4.6 Discussion

We have presented what we believe to be the first effective frame coherent constraint-based camera engine that allows visual goals to be applied on interactive 3d computer games comprising scenes of arbitrary spatial configurations and complexity.

We have made the following observations:

- Visibility satisfaction should be an integral part of any camera system and should be coupled closely to the satisfaction of camera constraints.

[9]Choice data was extracted using the ARGUS system by Martin Otten. Output from the SAMURAI system was then synchronized with captured video from the two Quake sequences.

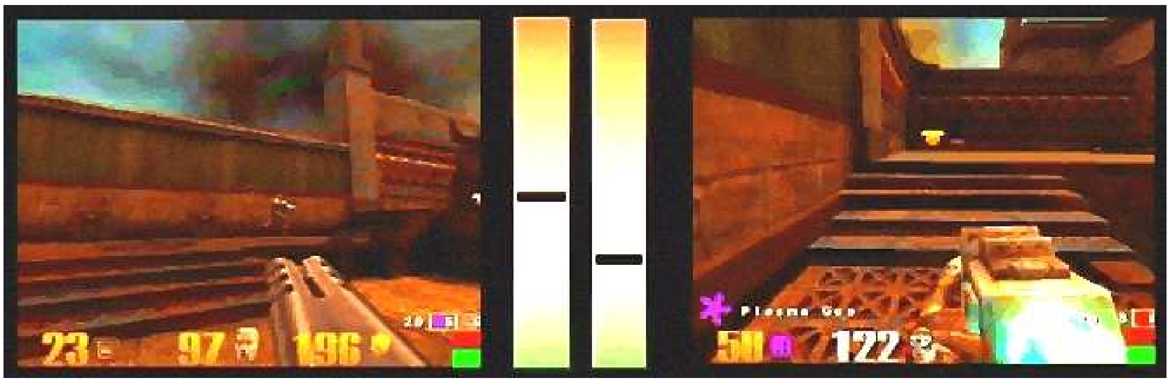
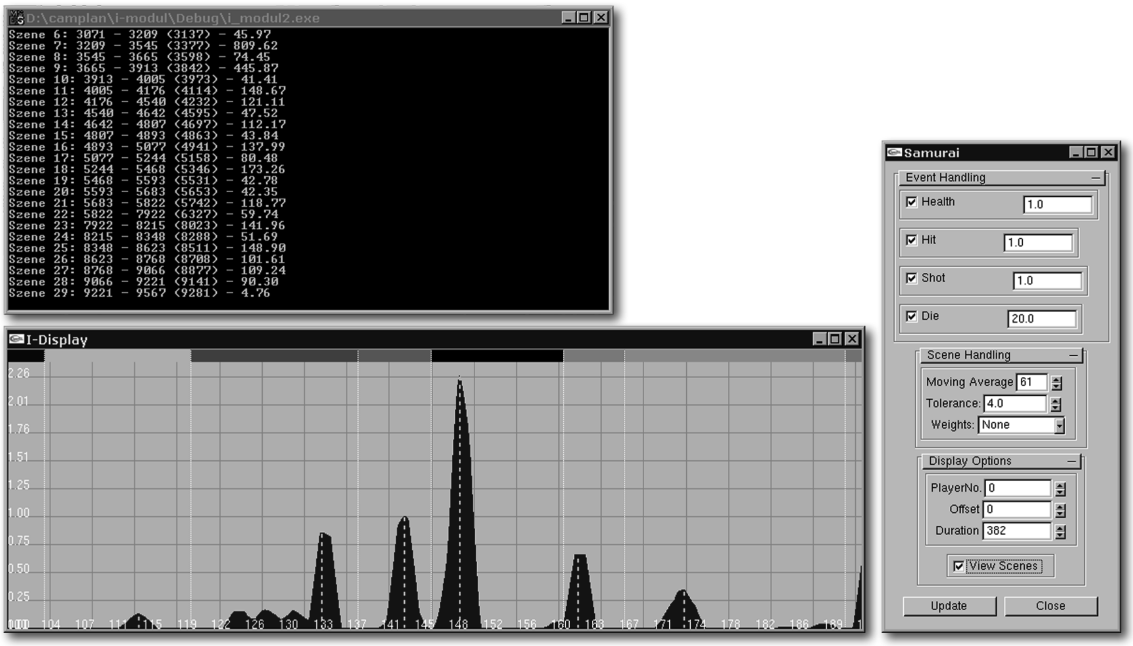


Figure 4.31: The bar on the scales on each side of the player windows represents the concurrent i-function value for each player.



```
D:\camplan\i-modul\Debug\i_modul2.exe
Szene 6: 3071 - 3209 (3137) - 45.97
Szene 7: 3209 - 3545 (3377) - 809.62
Szene 8: 3545 - 3665 (3598) - 74.45
Szene 9: 3665 - 3913 (3842) - 445.87
Szene 10: 3913 - 4005 (3973) - 41.41
Szene 11: 4005 - 4176 (4114) - 148.67
Szene 12: 4176 - 4540 (4232) - 121.11
Szene 13: 4540 - 4642 (4595) - 47.52
Szene 14: 4642 - 4807 (4697) - 112.17
Szene 15: 4807 - 4893 (4863) - 43.84
Szene 16: 4893 - 5077 (4941) - 137.99
Szene 17: 5077 - 5244 (5158) - 80.48
Szene 18: 5244 - 5468 (5346) - 173.26
Szene 19: 5468 - 5593 (5531) - 42.78
Szene 20: 5593 - 5683 (5653) - 42.35
Szene 21: 5683 - 5822 (5742) - 118.77
Szene 22: 5822 - 7922 (6327) - 59.74
Szene 23: 7922 - 8215 (8023) - 141.96
Szene 24: 8215 - 8348 (8288) - 51.69
Szene 25: 8348 - 8623 (8511) - 148.90
Szene 26: 8623 - 8768 (8708) - 101.61
Szene 27: 8768 - 9066 (8877) - 109.24
Szene 28: 9066 - 9221 (9141) - 90.30
Szene 29: 9221 - 9567 (9281) - 4.76
```

Samurai

Event Handling
- ✓ Health 1.0
- ✓ Hit 1.0
- ✓ Shot 1.0
- ✓ Die 20.0

Scene Handling
- Moving Average 61
- Tolerance: 4.0
- Weights: None

Display Options
- PlayerNo. 0
- Offset 0
- Duration 382
- ✓ View Scenes

Figure 4.32: The SAMURAI System: (top) scene frame intervals (1/20th second) and peak values; (right) a simple GUI for specifying weights on choice data; (bottom) graphical output of the action divided into weighted scenes. Dashed lines represent peak values whereas colored bars represent importance of scenes

- The only way to avoid backing out of a dead-end is to look ahead in time before you go in. Therefore, developing algorithms that estimate the future state of an environment are necessary for high quality control of the camera. The more effective these predictive algorithms, the more subtle the camera motions will be.

- Using a global best-fitness camera state is not necessarily the best solution for presenting visual goals. Choosing a partially satisfied camera state that lies closer to the current state often results in smoother and more subtle animations.

- Interesting events can be extracted by the analysis of specific game variables.

We conclude our results as follows:

- We have been able to show how a camera engine is to be successfully integrated as part of the dynamic game environment. Dynamic templates are created from conditions set by events, from which constraint-based specifications are formed.

- We have introduced an efficient and highly flexible way to compute visibility constraints for an arbitrary number of points.

- Our camera system works for arbitrary dynamic scenes and spatial complexities of environments. The camera needs no specialised collision information – this is handled effectively and automatically by the visibility constraint computed from the scene geometry.

- We are the first to provide a constraint-solver based on existing camera state and motion characteristics. The methods are fast, consistent, and robust, producing intelligent ”nearest-best-fit” frame-coherent camera animations in real-time by reacting to future conditions.

- We presented an approach for the automatic generation of action summaries for computer games.

For future work, we are now able to go a step higher in the presentation pipeline. We can solve for visual goals knowing that our low-level constraints will produce coherent results, that they stay in a partially satisfied region rather than jump erratically to global best-fit regions. Using an evolutionary design we can start adding and learning new techniques for camera management, introducing more sophisticated navigational goals and predictive techniques. The goal is to lead to an even more enticing atmosphere created by effective camera work and we conclude this chapter by identifying and looking at a few of the main areas for future research.

4.6.1 Predictive Scripts

An extension to using lookahead algorithms for placement of the camera is simply to ‘know’ what is about to happen. In certain situations we can determine events in advance either by predictive means or by following a generated script. For example, in an action sequence whereby one player comes through a door amidst a conversation between two other characters, the camera could position itself ahead of time to capture the surprise entrance.

4.6.2 Higher-Level Constraints

The evaluation of stylistic consistency in Section 4.2.3 is very limited in its scope. Each scene has the same number of objects, three of which are identical or similar in their spatial characteristics, and one which is significantly larger than the others. Although within these restrictions it is apparent that some degree of stylistic regularity can be imposed, an immediate requirement for research into camera planning is the development of a both a systematic evaluation framework, and a means of eliciting image objectives.

For instance, in Figure 4.33 we notice visual artifacts despite successful solutions to sets of visual properties. Notice, for instance, in Figure 4.33(a), how the head of the foreground character occludes the character on the left (top output), or how the head aligns ‘too close’ to the arm (middle output). In Figure 4.33(b) we again see secondary occlusion and alignment artefacts. Thus, even though we have achieved a degree of compositional regularity, there is no design of aesthetics. Even choosing a specification for a suitable evaluation of a visual property also raises questions, such as consideration of different evaluation results for a size constraint shown in Figure 4.34.

As with existing work [Christianson et al., 1996], one potentially fruitful source of image objectives are the accounts of cinematic practice (e.g., [Arijon, 1976]). Another source of inspiration for the design of image objectives is the visual cognition literature, and we envisage that sets of cognitively motivated constraints will be useful in tuning results initially derived from objectives motivated by insights from graphic design. For example, cognitive theories of recognisability [Biederman, 1987], depth perception [Rolland et al., 1995], and figure-ground separation [Koffka, 1935] offer many insights into how the potential for visual ambiguity can be minimised. We also intend to investigate the feasibility of an algorithmic characterisation of concepts from graphic design [Lauer, 1979] and visual aesthetics [Bethers, 1964], for example, unity, emphasis, balance, contrast, pattern, movement and rhythm.

4.6.3 Camera Language and Rules for Computer Games

Within interactive fiction, much effort has been put into examining whether games and narratives are fundamentally incompatible [Smith, 2000]. Film theory argues that the essence of film is editing [Dudley, 1976]. If we note that it took the film industry 20 years to develop a foundation of cinematographic principles, we can realize that another 20 might be required

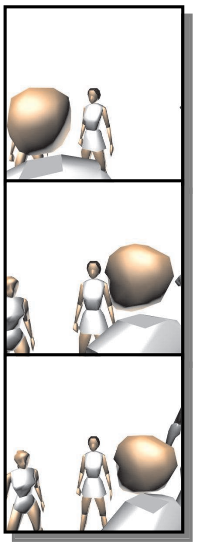
CenterShot "woman01 head"

(PersonTalking and PersonTalkedTo heads entirely inside screen)
PositionViewplaneBetweenX/Y "woman01 head" -1.0 1.0
PositionViewplaneBetweenX/Y "man01 head" -1.0 1.0

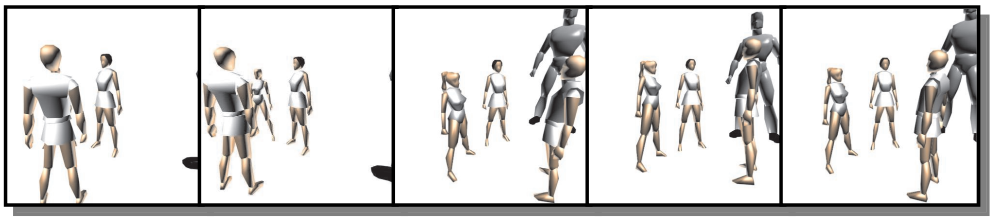
(PersonTalking head and PersonTalkedTo head fully visible)
VisibleInViewport "woman01 head" 100.0 0.0
VisibleInViewport "man01 head" 100.0 0.0

(PersonTalkedTo should be overlapped by PersonTalking but not with his head)
OccludedInViewportBy "woman01" 100.0 99.0 "man01"
OccludedInViewportBy "woman01" 0.0 0.0 "man01 head"

(PersonTalking cut at torso, head quite big and level with PersonTalkedTo head, and in front of PersonTalkedTo)
PositionViewplaneBetweenY "torso" -1000.0 -0.5
PositionRelativeY "woman01 head" x i "man01 head"
SpanXorY "man01 head" 1.0 0.4
PositionRelativeZ "woman01" b b "man01"



(a) Shot properties defining an over-the-shoulder view for conversation.



CenterShot "woman01"

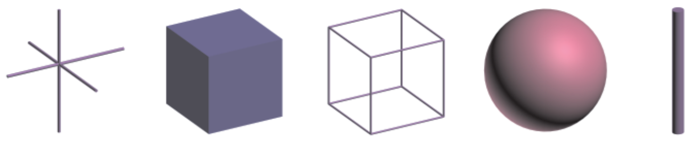
(PersonTalking and PersonTalkedTo entirely inside screen)
PositionViewplaneBetweenX "woman01" -1.0 1.0
PositionViewplaneBetweenY "woman01" -1.0 1.0
PositionViewplaneBetweenX "man01" -1.0 1.0
PositionViewplaneBetweenY "man01" -1.0 1.0

(PersonTalking and PersonTalkedTo mostly visible, with heads totally visible, and PersonTalkedTo not occluded by PersonTalking)
VisibleInViewport "woman01" 90.0 10.0
VisibleInViewport "woman01 head" 100.0 0.0
VisibleInViewport "man01" 90.0 10.0
VisibleInViewport "man01 head" 100.0 0.0
OccludedInViewportBy "woman01" 0.0 0.0 "man01"

(PersonTalking large on screen and in front of PersonTalkingTo)
SizeViewplaneLengthY "man01" 2.0 0.5
PositionRelativeZ "woman01" b b "man01"

(b) Shot properties defining a full view of a conversation.

Figure 4.33: Multiple-shots generated from the same sets of constraints.



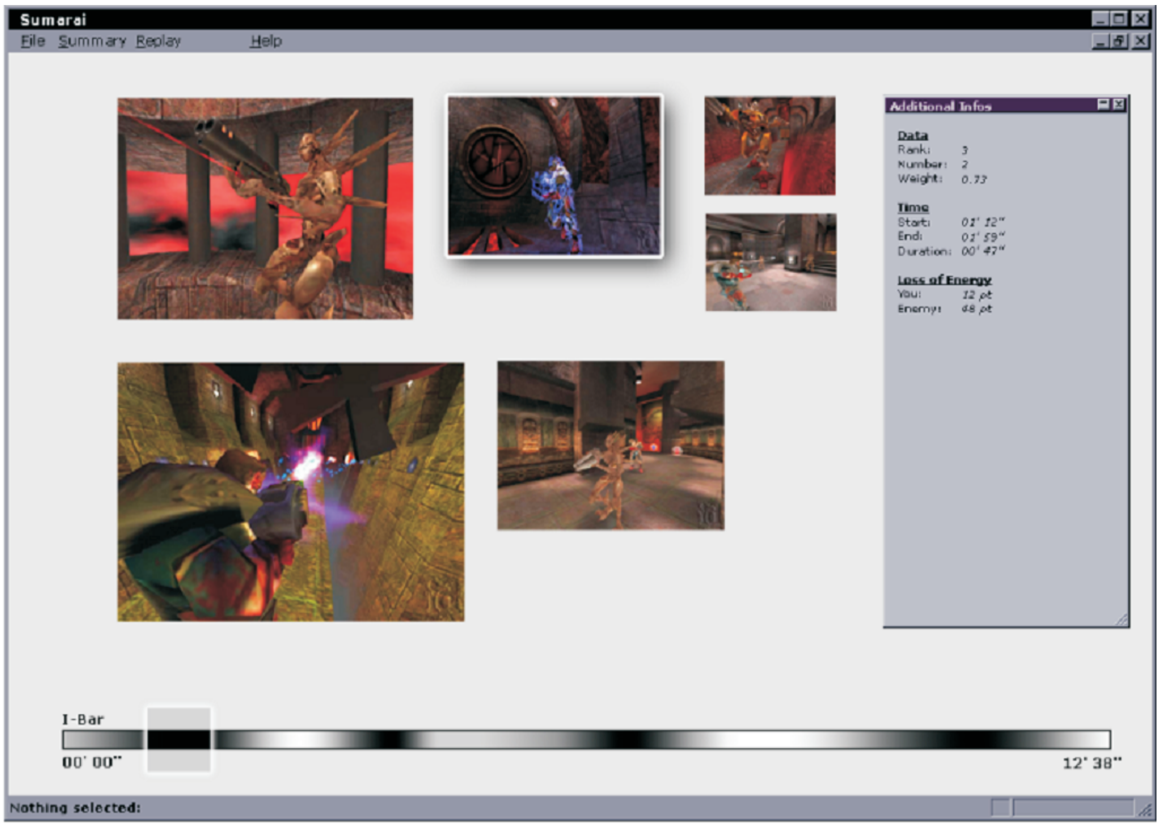
LengthX/Y	0.5	0.5	0.5	0.5	0.02
Area	2%	50%	2%	50%	2%
BoundSphere	0.6	0.6	0.6	0.5	0.5

Figure 4.34: Which is the best evaluation of the perceived size of an object?

for effective camera work in computer games and (to coin a term) ‘interactographic’ principles. With this in mind, games should feel free to try to build camera rules for their own goals in the hope that we’ll see an evolution of techniques and foundations over the years.

Thus, of most importance is to determine what works for the player—this can then guide the use of camera specifications. Pagulayan et al. [2003] performed a case study on camera techniques in “Munch’s Oddysee”. In this game, the camera’s behavior was programmed to create maximal cinematic effects and enhance gameplay. The camera would often provide a specific view with the intent of showing the user what was behind ‘the next door’ or on the other side of a wall whilst still keeping the main character in view. The most common user complaint was that although they often liked the look, style, and behavior of the camera, they wanted more control over the behavior of the camera as other games in the same genre have done. Based on this response, the camera was changed so that users could regain control when they wanted. This resulted in more frequent use of a third-person follow camera. The more advanced camera behaviors are still part of the game but localized to areas where these alternative camera behaviors can only create an advantage for the user.

A potentially invaluable tool for exploring viable camera behaviors would offer a way of creating shot templates for developers to define their own rules and setups in a WYSIWYG[10] manner. The developer could move around in scenes and position objects in front of the camera and take a snap-shot. Algorithms for automatically encoding what the designer sees into a set of constraints could then define the shot. Finally, additional templates that create a certain cinematic effect could be introduced, much like a director’s style. Cinematic templates could alter viewing style by varying parameters such as favoring certain size ranges, viewing angles, camera motion, and frequency of cuts.



Data
Rank: 3
Number: 2
Weight: 0.73

Time
Start: 01' 12"
End: 01' 59"
Duration: 00' 47"

Loss of Energy
You: 12 pt
Enemy: 48 pt

Figure 4.35: Suggested interface for action summarization.

4.6.4 An interface for Summarization

Future work for action summarization will focus on the refinement of the selection process of choice data and presentation of summarizations. Templates for certain game genres could support the user in the process of selecting choice data. Presentation of a summarization would benefit greatly from an interface such as shown in Figure 4.35. Pictures can be shot using the CamPlan system (Section 4.2.2) at the point in time determined by each scene's representative (i.e., maximum value), with size proportional to action and importance. The length and form of these summaries could be specified by the user with parameters for scene weights and could range from a small number of single frames to a set of animation clips.

[10] what-you-see-is-what-you-get

5 Integration and Future Research

As in any field of research, computer graphics cannot truly survive or evolve without input from other relevant schools of thought. More importantly, computer graphics is now technologically ripe to begin considering the psychological repercussions of its art and underlying tools on the user. Indeed, the functional relationships between various visual presentation methods and their viewers must emerge to define the more subtle—though potentially powerful—cognitive, emotional and behavioral consequences of interactive computer graphics. Moreover, the psychology community itself can only indirectly contribute to this understanding, as our theories survive not on understanding how perception works, but how it can be effectively applied within interactive graphics.

Additionally, although rules-of-thumb[1] continue to provide insightful presentation techniques (e.g., [Arijon, 1976; McCloud, 1994; Durand et al., 2002]), their informal approach alone cannot substantiate the implementation of a psychologically driven system model, but may instead highlight possibilities within future, more formalized research. Consequently, an empirical approach that tests core assumptions within computer graphics, and specifically in terms of supportive presentation, is absolutely necessary.

Conceptually, supportive presentation should invoke an effect on the player without necessitating conscious attention to that effect. However, the visual effect itself must also be homogenously inscribed into the image for it to be accepted, as certain styles do not mesh well or effectively convey the desired intent. Thus, the design of suitable presentation methods requires a *creative* mind. Although this type of creativity cannot necessarily be formally itemized or defined, it can be facilitated and even inspired via exposure to and experimentation with the appropriate concepts and tools.

Figure 5.1 highlights how creativity, psychology, and technology interlink to comprise the three fundamental elements necessary to supportive presentation concepts, tools, and applications. Specifically: psychologically-geared research can assess and validate effects from artists while testing environments from programmers; artists can experiment with tools provided by programmers to create images that embed known psychological effects; and programmers can improve technology to accomplish artistic effects and parametric control over images and their influence as defined by psychological research.

The technological component of the supportive presentation triangle is defined and supported in Section 5.1, via the integration of tools as presented in Chapters 2 and 3 into a larger framework. This integration coupled with knowledge gained from our empirical experiments to design effects, allows us to finally apply supportive presentation at all levels.

[1]or ‘craft skills’ within the CHI community

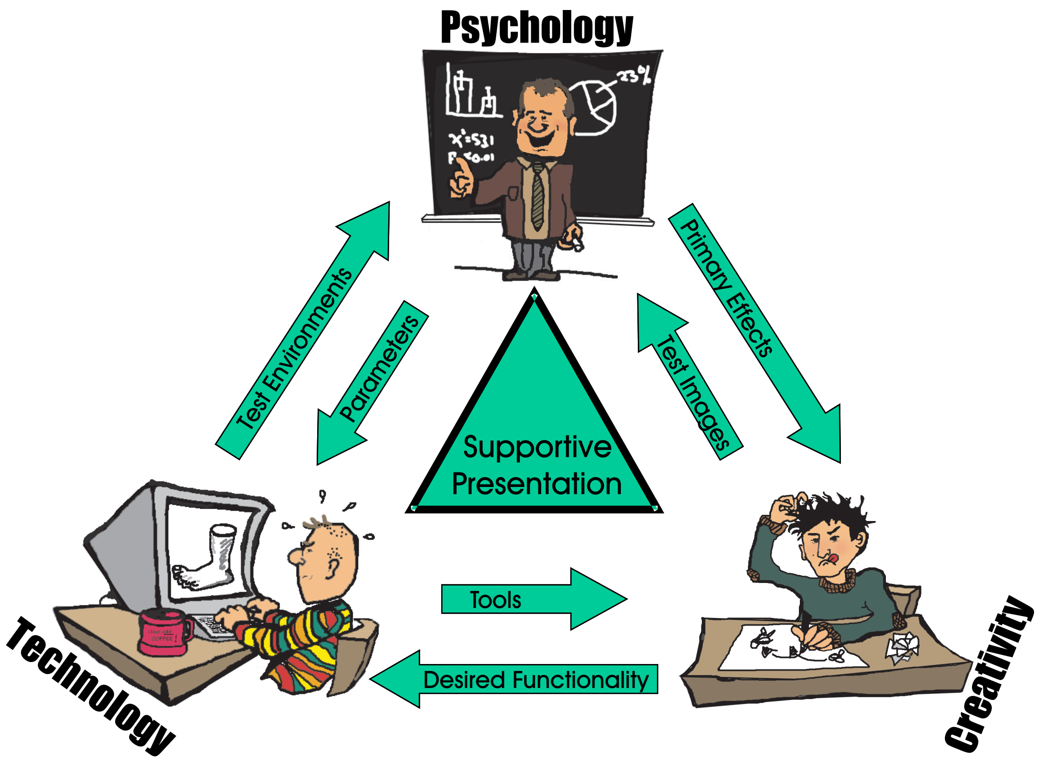


Figure 5.1: The cornerstones of supportive presentation.

Section 5.2 subsequently tackles common issues in game design using three different application scenarios that employ supportive presentation guided by the conceptual graphical presentation pipeline introduced in Chapter 1. Finally, Section 5.3 discusses various factors and future directions in an interdisciplinary approach for supportive presentation.

5.1 System Integration

Figure 5.2 demonstrates how the tools presented in this thesis can be integrated into a larger framework which drives the tools and concepts of supportive presentation. First, the game designer recognizes good elements of play in the game and considers communicative goals for maximizing player enjoyment. Game variables are then used for supportive presentation and passed from the game application to the *Supportive Presentation Manager* (SPM), where the game designer sets conditions for selecting presentation strategies. For assistance, the SAMURAI system (Section 4.4) evaluates action and certain characteristics of the game[2]. Combinations of camera and rendering methods are selected from templates that

[2]In this context, the SAMURAI is considered a presentation strategy

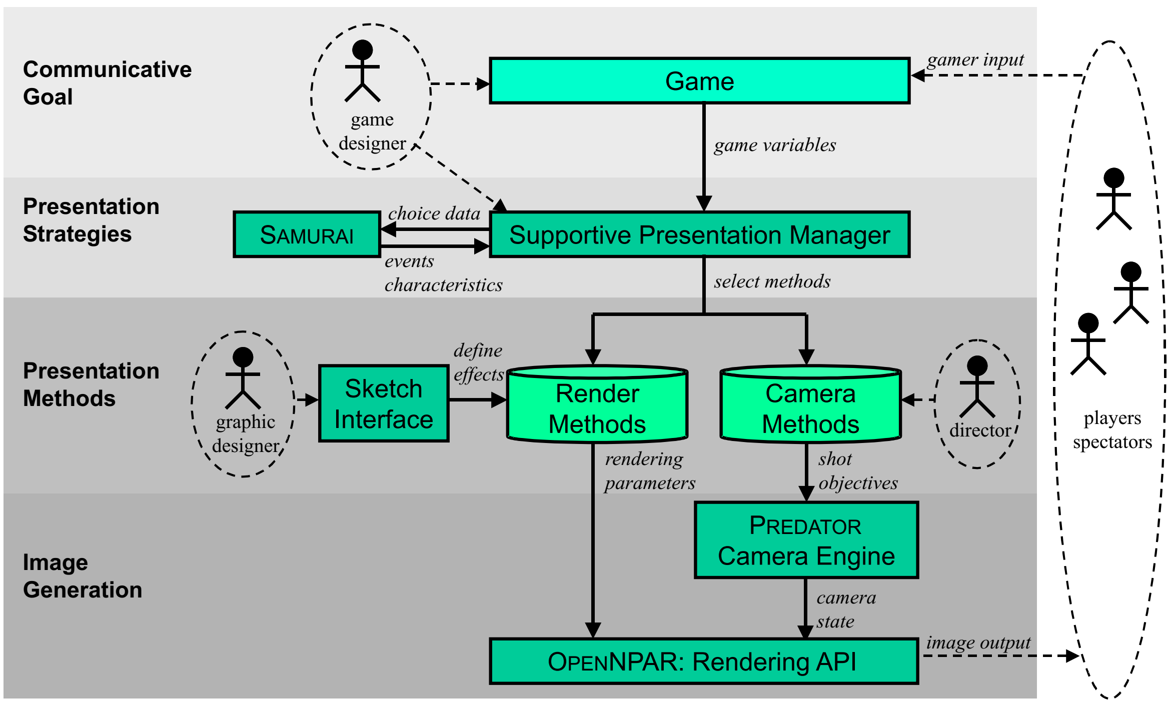


Figure 5.2: Supportive presentation framework.

specify how presentation strategies are achieved. Shot templates for camera methods can be specified by the *director* who visualizes the required shot properties. Similarly, *graphic designers* use the sketch-interface (Section 3.4) to visually specify rendering methods. Shot objectives are satisfied by the PREDATOR camera system (Section 4.3), which then sends the solved camera state to OPENNPAR (Section 3.1). In contrast, rendering methods specify variables and parameters directly with the OPENNPAR rendering API. Lastly, OPENNPAR then outputs the final image to the players and spectators.

5.2 Application Scenarios

This section provides hypothetical applications for supportive presentation in terms of the graphical presentation pipeline[3] introduced in Chapter 1. For each application scenario, we draw upon existing psychological data which indicate or clarify roles of rendering styles on the user. Thereafter, we detail how the SPM would effectively apply those styles in the final application. However, even the simple applications outlined below demonstrate just how limited our understanding of the psychological effects of presentation styles and their

[3]With the exception of image generation, as this is performed automatically without interference from the game designer

potential applicability are. Clearly, the complexity of interactions between presentation methods and the viewer necessitates further controlled experimentation. For simplicity, although we provide examples for both camera and rendering methods, efforts for future research primarily focus on non-photorealistic methods alone. For the interested reader, [Halper et al., 2003c] details further insights into collaborative research between NPR and psychology.

5.2.1 Multi-player Action Games: Balancing Player Skill

In this case study the goal is to cater to a varied range of skill levels within a competitive multi-player game setting, in order to better maintain a satisfying degree of challenge for each player. As discussed in Section 2.1.2, this is a non-trivial task.

Communicative Goals: The overall goal is to involve players in balanced (or fair) confrontations, wherein less-skilled players are given more information to better compete with the more-skilled players. Specifically, we can: (1) warn players of dangerous/stronger opponents, (2) indicate opportunities to lead a successful attack on opponents, and (3) lure players into advantageous or safer locations.

Presentation Strategies: Warning players of dangerous opponents can be achieved by indicating the whereabouts of these opponents and their respective levels of threat. Indicating the strength of an opponent will cue the player to attack those of a weaker disposition. Luring players to advantageous or safer areas can be achieved by emphasizing the appearance of the location such that attention is increasingly drawn to those locations.

Presentation Methods: The director (from Figure 5.2) can specify camera shot properties to provide views around corners to 'hint' at stronger players or traps. Levels of threat can be communicated using the threat-connotative styles from Section 2.4.1, wherein the graphic designer assumes responsibility for effectively mapping threat-connotative effects into a coherent visual style for characters, as is shown in Figure 5.3 with the camera methods.

Tactical locations can be emphasized using the LOD methods in Section 2.4.4 to guide the player along a path to specific opponents or appropriate weapons: Potentially, users view increased LOD as more interesting for exploration relative to lower LOD. Thus, we employ extra shaded lines and thicker silhouettes around doorways of interest, as shown in Figure 5.4.

SPM Application: Communicative goals are employed based on relative player performance, wherein presentation strategies are selected to provide advantages for lower ranked

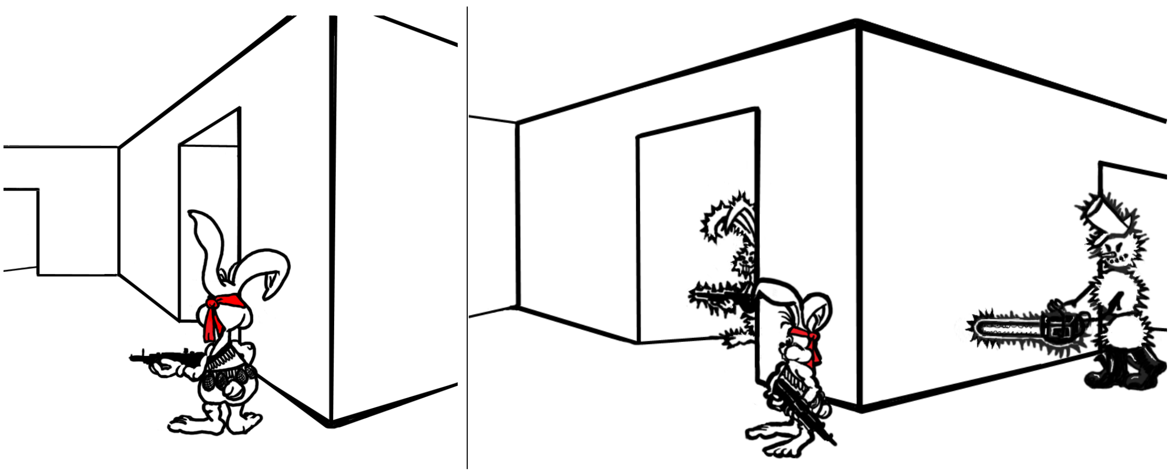


Figure 5.3: (left) Normal player view. (right) The same situation, but the camera has positioned itself to warn the player of potentially dangerous opponents. (Images courtesy of Mara Mellin)

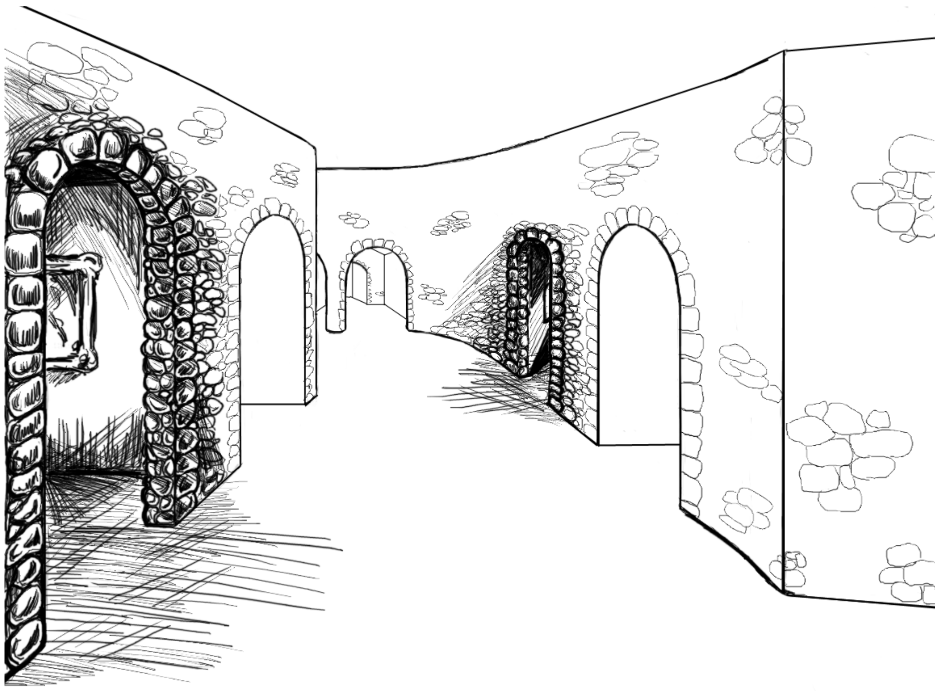


Figure 5.4: Using LOD to guide the player to specific locations. (Image courtesy of Mara Mellin)

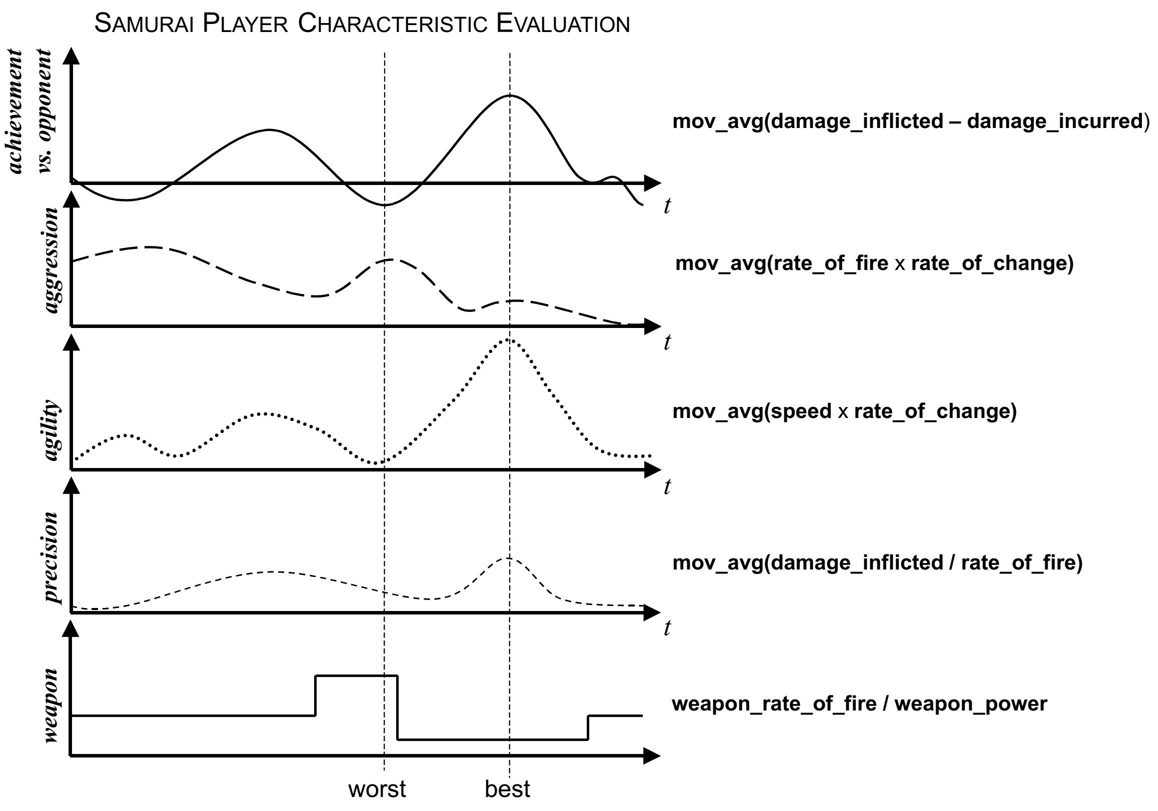


Figure 5.5: Using the SAMURAI to automatically evaluate success strategies by correlating player characteristics with performance. The top evaluation function evaluates player performance relative to an opponent. The middle functions evaluate player characteristics, and the bottom function is a measure of weapon characteristics. In this example, the player is most successful against this opponent when agile and precise in using a high powered weapon.

players against those of higher rank[4].

In this respect, the SAMURAI system (see Section 4.5.6), whose primary function is to evaluate action, can be alternatively used to evaluate player performance since action in multi-player games closely correlates to the performance of players—when players perform well, action rises. Thus, as shown in Figure 5.5, the SAMURAI system can be used to automatically evaluate characteristics of player performance under various conditions.

Based on these evaluations, the SPM applies presentation methods based on the following conditions:

- Opponents with stronger weapons, higher skill, and additional health are mapped with

[4]It could be possible to use *disadvantageous* cues for higher ranked players. However, our objective is to help weaker players rather than hinder skilled players

threat-connotative silhouettes.

- Weak-connotative styles can be applied to those higher ranked players capable of being beaten by a lower ranked player who is capable[5], given the proper weaponry and power, of inflicting the necessary damage to defeat the opposition.
- LOD is used to lure players of equal stature to the same locations. In contrast, LOD is also used to lure players away from others of different skill levels.
- LOD is used to lure the lowest ranked players into locations that offer additional protection or more powerful weapons.
- The camera warns the player with a 'sneak-peak' of lurking higher-ranked players.

Future Work: More complex images, subtler test questions, and an in depth understanding of how assessments of danger, safety, and strength are influenced by visual cues will reveal more definitive insights into how NPR can be used to guide users and their perceptions of images and environments. In particular, low-level tests are needed to establish precise mappings between observed effects and drawing parameters before these can be embedded successfully into fully-featured environments. Other relevant considerations are how levels of line thickness, style, and angles of 'sharp' threat-connotative features, influence graded perceptions of threat. Of additional importance, is the necessity of knowing whether and how these features can effectively be implemented in combination with the following factors: animation, fast camera movement, high color levels, and audio effects.

In Chapter 2, assessments of safety and danger are considered in the context of social judgments, while Provins' [1957] radar research demonstrates that identification of triangles as foes and circles as friendly are pre-conscious processes. Consequently, such assessments of friend and foe could prove effective in facilitating gameplay via rapid character assessment within a game. Further research is necessary to elicit the extent to which triangulation of lines influence both pre-conscious and conscious assessments of danger and how effective they can be within an animated game context.

A second necessary psychological issue to explore is how the effects of *prior knowledge* influence player assessments of opponents, objects, and entire game settings. At a physiological level, increased brain activity is typically observed when subjects are exposed to new stimuli, specifically visual stimuli [Herrmann and Bosch, 2001], thus indicating increased levels of *feature binding*[6] and attention in order to categorize and respond to new images. Images already represented in memory usually require less attention because identification occurs rapidly, whereas new objects require increased attention before identification and categorization occurs.

[5]based on statistics assimilated from the SAMURAI system

[6]the combining of different visual elements (e.g., shape, size, movement) to create a cohesive image(s) for identification and categorization ("that is a dog and it looks dangerous")

Moreover, the type of memory can influence both attention and player expectations, such that a negatively remembered object may increase attention over a new object, whereas an object or character consistently paired with a rewarding experience (getting points, beating an opponent etc.) may later prove to be the downfall of an unobservant player when the previously rewarding object or character becomes threatening. To further demonstrate the possible role of specific presentation strategies in influencing player expectations: if the camera consistently moves to provide views of stronger opponents that hide around corners, the player may begin to expect there *not* to be an opponent around a corner when the camera does not move around to show it. This may, or course, be false if there is actually a weaker player around the corner. Clearly, there exists a number of variables which must be accounted for before psychologically driven supportive presentation strategies can be implemented.

Additionally, even if precise measures are found which convey the desired communicative goal(s), the player remains an unknown. In other words, there exist different types of players, or player personalities: while one player would flee from the conveyed danger or threat, there is an aggressive percentage who would likely choose to take their chances and perceived fighting prowess to attack the opposition. Thus, virtual test environments could be designed precisely to categorize different 'player personalities' and subsequently provide a plethora of useful data for both the computer game industry and psychology proper.

In terms of lower level tests, there is much potential for biological and neurological psychology to physiologically demonstrate just how effective certain presentation methods are. For example, *electroencephalogram* (EEG) measures could be correlated with an eye tracking device to analyze player responses to varied threat-connotative rendering styles. Specifically, such measures could reveal when attention-related increases in brain activity are present for those objects rendered as dangerous relative to those rendered using lesser threat-connotative silhouettes. These experiments could be conducted across a variety of contexts—animated or not, with or without sound, thus, indicating how much influence external variables have on a given presentation method. Additionally, an eye-tracking device could record any differential eye movement patterns and observation times from subjects exposed to mixed and independent PR and NPR images. If such variance does indeed exist, both NPR and PR presentation strategies and tools must then be explored in parallel to uncover their respective optimal applications within animated and gaming environments.

5.2.2 Massive Multi-Player Online Games: Encouraging Cooperation

The success of massive multi-player online games (e.g., Everquest) depends on both the opportunity to socialize with other players and solve tasks together. Primarily they require features that support and encourage reciprocal altruism and ensure that actions have consequences [Smith, 2002]. Moreover, they offer the most potential for non-linear gameplay, since agents and players in the environment can make changes to already explored locations, which may require players to backtrack and complete additional tasks.

Communicative Goal: To help players and support a sense of community, the presentation should: (1) cue players to the relevant groups[7], (2) discourage interactions with 'misfit'[8] characters, and (3) communicate changes and challenges of the environment.

Presentation Strategy: Players and groups can be encouraged to interact with one another via mutual emphasis (i.e., focus is given both to players of potential interest to a given group, and to any groups of potential interest to a given player.)

Social perception and social judgment utilize learned values and behaviors to respond to socially ambiguous situations, wherein interpretation of other people and their expressions—verbal and non-verbal, are necessary [Bente and Krämer, 2002; Dörner and Schaub, 2002]. Thus, simple optical elements may be employed to evaluate 'misfit' characters by using stereotypical visual character attributions when too little explicit information is available (e.g., a character who frequently steals from other players could develop increasingly large hands).

Finally, changes and challenges of the environment can be communicated via a sense of completeness—areas that are more game-relevant should become more interesting or attractive, while areas that are more or less 'used up' or no longer game-relevant, would become less interesting or attractive.

Presentation Methods: The camera could prove an effective presentation method in attracting players to one another, as shown in Figure 5.6. Once again the director from the supportive presentation framework can specify visibility and positional properties in the viewport for relevant characters and groups. Herein, LOD can also detail the given characters of interest.

Although we lack a presentation method to indicate misfit players, threat-connotative styles character silhouettes in these cases may discourage further interaction (cf. Figure 5.3).

In similar fashion to the experiments in Section 2.4.4, familiarity with a specific area may be cued by decreasing LOD, while unknown areas may be depicted with increased LOD to support spatial exploration behavior. LOD may also represent a level of completeness—low levels of detail result in less visual appeal, thus encouraging players to progress towards other areas offering additional challenges.

SPM Application:

- When entering a location, the camera cues the player to interact with any potentially relevant groups, and likewise, players in relevant groups are offered a view of the incoming and potentially relevant player (Figure 5.6).

[7] those that offer the same level of experience in the game or to which the player could contribute their own set of skills

[8] players who purposely try to ruin the gaming experience for others

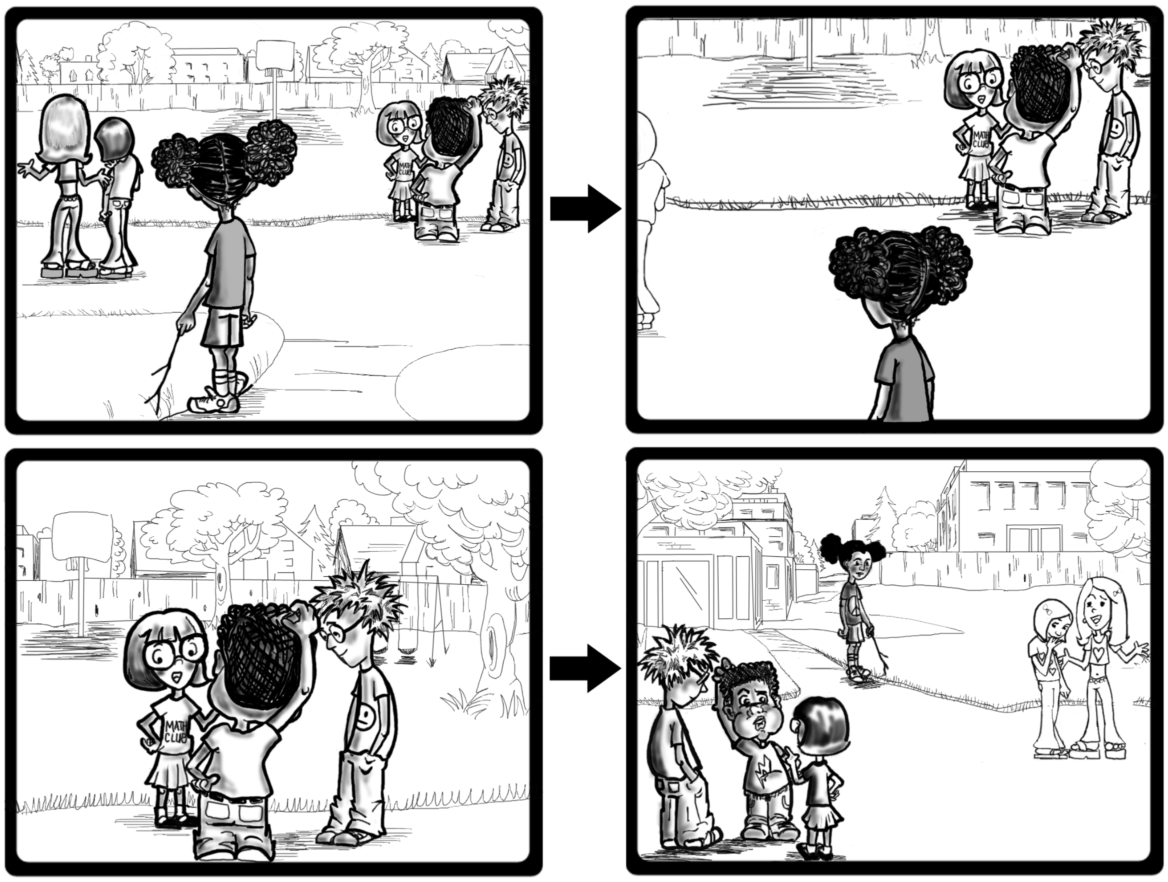


Figure 5.6: Left figures show normal views, right figures show supportive presentation views designed to accommodate players in selecting appropriate groups. (top left) Newcomer sees two groups of people, and (top right) camera and LOD employed to encourage approaching the group on the right. (bottom left) A group member in conversation with two other members, and (bottom right) the camera and LOD suggest welcoming the outside player into the group. (Images courtesy of Mara Mellin)

- Silhouettes for ‘misfit’ players are rendered in threat-connotative styles[9].
- LOD is mapped to buildings, areas, or paths leading to new locations proportional to their relevance in completing another challenge.
- LOD is used on paths that connect relevant groups of players together.

Future Research: Further studies of exploration and navigation can by guided by NPR in conjunction with psychological theories. For instance, the home range concept [van Vliet, 1983] demonstrates that human exploration patterns tend to resemble an ever-expanding circle, wherein space immediately next to the familiar locations is first explored until it becomes familiar before moving outward and so on. Combining experiments that encourage exploration based on visual stimuli could aid level-design to more effectively influence players towards exploring particular areas.

Aggressive and *altruistic* player behaviors may also be influenced by aspects of NPR. For example, NPR and social psychology would benefit from tests measuring the behavioral consequences of negative reciprocity [Mummendey et al., 1984; Patchen, 1993] and whether they vary depending on how a character is rendered (strong, dangerous, etc.). These results might also extend Zimbardo’s [1969] early studies on de-individuation, wherein aggressive, reckless, and chaotic behaviors increased in subjects when their identities were masked. Indeed, game environments might reveal much potential in these areas as online players frequently conceal or change their identities. It is possible that via rendering attributes, which convey more information about the player’s behavior, certain negative behaviors would then decrease as a result of a sort of ‘unmasking’ that takes place when the player is no longer able to completely conceal negative aspects of their behavior. Lastly, game designers could systematically vary characters and scene presentation for their own utility within the game, or create ‘game’ programs employed towards both psychological and gaming goals to better understand the psychosocial impacts of the increasing online game world.

5.2.3 Adventure Games: Assist Player Progression

Adventure games require players to solve puzzles in order to advance to subsequent stages in the game. Modern non-linear gameplay tends to be combined with rather impressive visual realism in complex environments, such that the fundamental challenge for game designers today is to effectively draw player attention to areas that are actually important within these visually fantastic worlds without losing the casual game player in the details altogether [Bates, 2002]. Often, the player will become stuck at one or more puzzles and be forced either to try tedious combinations of object interactions, look at released solutions, or simply give up.

[9]Several measures can be used to detect misfit players. One method, for instance, is to let players vote on characters, as in the www.eBay.com rating system for users.

Communicative Goal: When the player is experiencing difficultly, the communicative goal is to make solutions subtly more apparent when a current task remains unsolved by: (1) guiding the player to relevant locations; (2) directing player attention to particular objects;

Presentation Strategies: Players can be guided to locations and objects by attracting the player's attention. Moreover, to guide user selection to a constrained set of objects, 'active' and 'inactive' styles can be employed (see Section 2.4.4).

Gestalt Law indicates that objects rendered in a similar fashion will be perceived as belonging to the same group, and thus, are considered functionally similar. These concepts can be used to further cue the player to the appropriate objects, their related functions within a given setting, and any potential combinations therein.

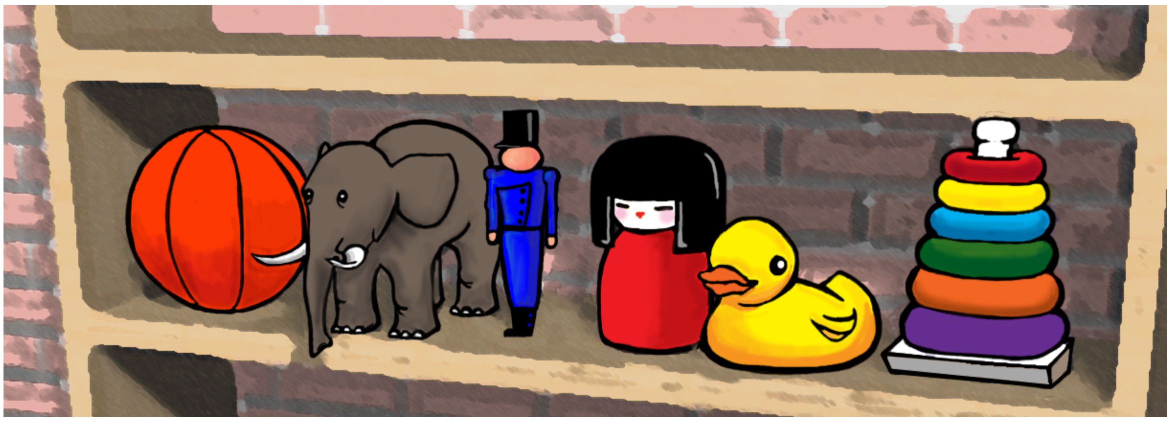
Presentation Methods: Although precise properties of effective rendering styles in representing active and inactive objects are unknown, psychological research reveals that homogenously structured surfaces are easier to categorize as belonging to the same object [Hoffman, 1998]. Therefore, rendering objects using defined 'figure' or 'ground' styles will enable the viewer to more rapidly and accurately assess the status of objects within complex scenes. Feature binding is also relevant within this process—when different parts of an object(s) share the same qualities (in this case rendering style) it is easier for our brain to bind these parts together as a whole object or field of related objects.

Using this information, the graphic designer from the supportive presentation framework can then experiment with designs via the sketch-interface to OpenNPAR, including any variations used in associating different objects. In our example, we chose cartoon rendering styles similar to those in Figure 2.15, and varied the silhouette styles to increase Gestalt mappings.

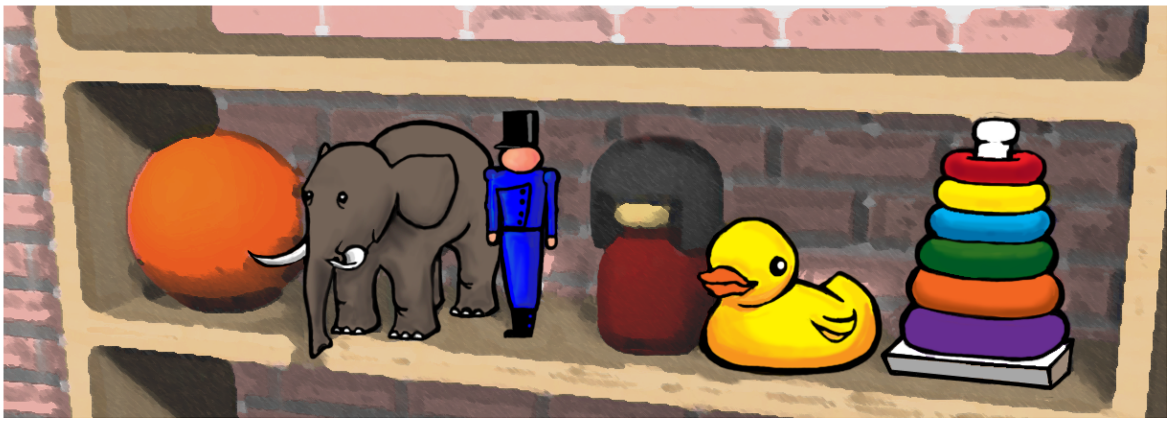
In addition, player attention can be focused by keeping relevant objects and locations in view. In this respect, the director can define view specifications for the Predator system to maintain relevant objects' visibility and placement in the viewport. Finally, the LOD methods from previous examples can again be employed to guide the player to appropriate locations.

SPM Application: The following techniques can be introduced progressively depending on player circumstances:

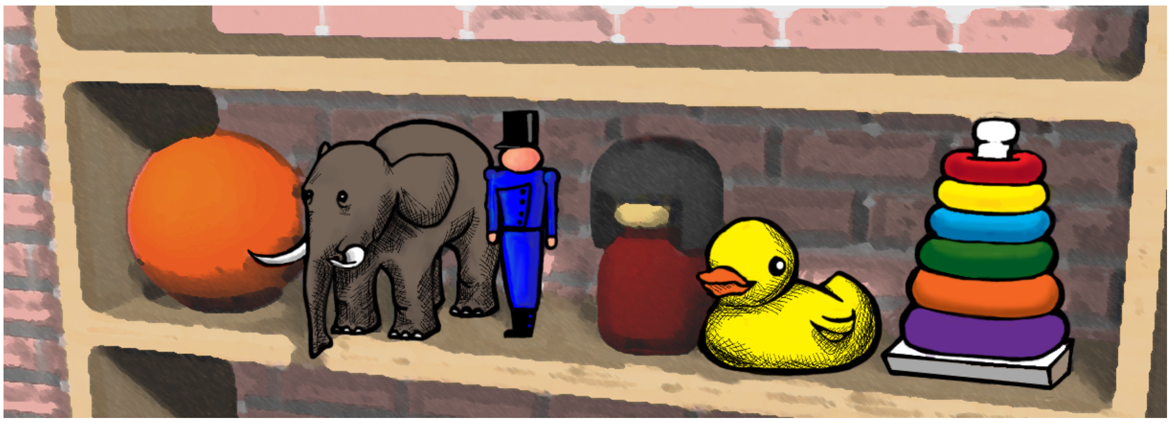
1. All objects with behavior attached are rendered as active; while all objects that serve as background detail are rendered as inactive (Figure 5.7(a)).
2. Given difficulty in solving a puzzle, the player can initially be guided to appropriate locations using LOD.
3. Slowly, more objects can be rendered as inactive, leaving a smaller and smaller set of objects as active (Figure 5.7(b)).



(a) All objects on shelf are active.



(b) Objects not pertinent to puzzle become inactive.



(c) Associating objects with similar shading styles.

Figure 5.7: Using rendering style to visually suggest relationships between objects.

4. The camera can begin to suggest objects of particular importance by keeping them focused in view.

5. Objects that are used with each other in solving a puzzle can be rendered in similar shading and silhouette styles (Figure 5.7(c)).

6. Given continued difficulty in finishing a task, only the objects immediately relevant to solving a problem are rendered as 'active' and with similar rendering styles.

Future Research: The cognitive processes necessary to effective figure-ground segregation via active/inactive rendering require increased levels of attention, such that those objects perceived as foreground logically receive more attention. However, further testing can confirm whether these conditions hold when the overwhelming majority of objects are rendered as active, potentially causing the few remaining background objects to then, in effect, 'pop out'.

A second consideration in rendering objects active or inactive, as seen in Figure 2.15 is that aspects of *associative learning*[10] inevitably emerge. Specifically, if a set of objects is rendered as active and also as belonging to the same group, there is the potential for interference with certain feature-binding processes or more simply put, confusion as what objects belong to what functional group. For example, rendering parameters for a Gestalt group of objects might erroneously be perceived as background or non-relevant, and thus ignored. Consequently, understanding the effects of such possible rendering variables requires additional research into the specific qualities of rendering elements for associative conditioning and the number that can be simultaneously used while still maintaining active/inactive object identification.

Although one intention of the application example above is to eliminate objects that are not pertinent to a puzzle by blending active objects into inactive objects, it is not clear how prior knowledge about selectable active styles will compete with prior knowledge of which objects were previously selectable. In contrast, prior knowledge may reinforce behavior in the guided selection of objects—repeated and consistent object interactions would likely strengthen the influence of rendering style until players cease to interact with background objects altogether. Furthermore, research could reveal that graded levels of active and inactive rendering styles associated with object functionality might reveal different subject responses.

Thus, experiments that measure the extent to which prior knowledge and rendering properties play a role would shape the domain in which styles can be chosen for active and inactive objects. However, even though the use of stylistic variations in NPR overshadows possibilities within PR, its deviation from realism consequently requires evidence that chosen styles can still be understood. In this respect, recent research indicates that objects

[10]when two or more stimuli become associated with each other, such that they are perceived to have a relationship.

rendered in non-realistic styles do not influence primary feature binding processes necessary for basic object identification, which maintains the potential for NPR to vary aspects of identified objects in manners not possible within photorealism. For instance, memory is not dependent on how realistic an image appears and has been demonstrated in infants as young as 11 months—infants tend to attend new or implausible objects longer than previously seen objects, while the duration of object fixation (or attention) was independent of how realistic or simplified the objects appeared (cf. Figure 5.8) [Pauen, 2003].

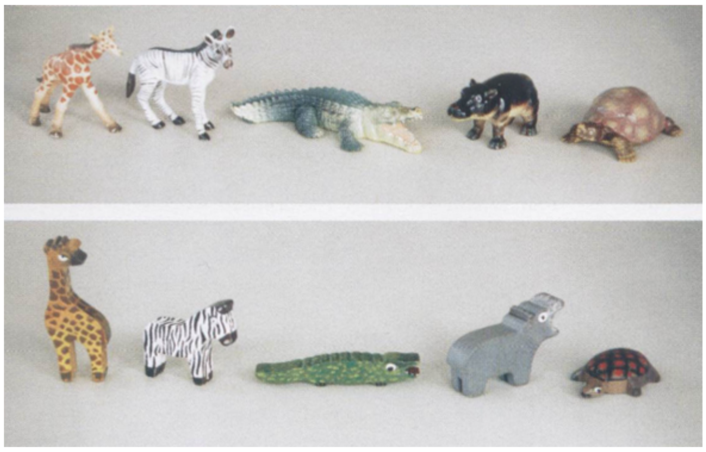


Figure 5.8: Realistic (top) and simplified (bottom) toy animals lead to the same attention levels in 11 month old infants. Memory plays a central role in how images are visually perceived.

5.3 Discussion

The simple scenarios shown above clearly indicate that developments in the application of supportive presentation can only be driven in conjunction with psychological paradigms, which in turn, necessitates both creative design concepts and effects. There are other links between graphics and psychology that have not been touched on here. For instance, temporal qualities of animation and movement are likely to have a strong effect on perceived character—the rate-of-change of coarse silhouettes may invoke a sense of aggressiveness, hostility, or even nervousness as we associate the instability of rendering animation to the character itself (e.g., see the animation examples from [Curtis, 1998]). There are also links to be explored with the use of sound in virtual environments; as the ulloomo/takete example from Section 2.2.2 shows, sound also has affective properties, something which cinema uses to good effect in creating and sustaining emotional states.

The ICS model (also Section 2.2.2) includes acoustic processing, and as such it provides a natural route for exploring the theoretical basis for multimodal blending. It is clear that a model is needed to effectively underpin the development and deployment of NPR in a manner similar to that of psychophysics and models of human vision, which have supported development of realistic rendering. Such a model may remain a long term goal, and our recent paper [Duke et al., 2003] is only a first step towards achieving it. Most importantly, we have demonstrated that the effects we describe are outside of the realm of low-level visual perception, and involve semantics, affect, and other high-level cognitive processes wherein meaning is ultimately conveyed and interpreted. Understanding the meaning carried by rendered images demands a broader approach within human information processing.

In addition, part of our long-term goal is to extend OPENNPAR and the PREDATOR camera system to include psychological methods *directly* within their API. This could significantly ease the configuration of effective supportive presentation to optimize conditions in virtual environments. For instance, OPENNPAR could systematically adjust renditions in the application, based on empirical evidence as to the psychological effect of NPR, to achieve a desired effect on the end-user.

As the application scenarios have shown, it is relatively easy to define a presentation strategy for a game, and yet another to find the right presentation methods to bring it fully functioning in an aesthetic application. In this respect, the creativity of designers (and collaborations with psychologists) will play a major role in defining supportive presentation, potentially driving technological requirements, which utimately will advance the body of knowledge and applications within the field of NPR.

The real question remains: how will players actually react to supportive presentation in computer games? Needless to say, feedback from simple game applications which employ basic supportive presentation will reveal many further directions for interdisciplinary work.

6 Concluding Remarks

Modern technology continues to surpass initial expectations of the nascent years of computer game design. Even so, an observable and specific sort of stagnation is emerging within the modern game industry, wherein game companies—employing evolutionary game designs, largely distinguish individual games via achievements in increasingly spectacular graphical realism. Indeed, one might say that motivation for buying a game these days almost entirely hinges on the evolution of the previous version's graphical improvements within the realm of photorealism. This has inevitably led to an exponentially growing race within the gaming industry, wherein developers tend to focus on visually representing real life more accurately than the currently 'less impressive' versions on the shelf. The resulting fester of eye-candy can leave players, especially novice ones, lost in the details and wondering what to do next. Moreover, aspects of game play may be compromised with the excess of game-irrelevant (but realistic) effects absorbing the player's attention to the point of distraction. Worse though, is the fact that such games simply do not appeal to those people who begin to feel it too much resembles a skewed version of real life, effectively removing that simple feeling of 'play' that is, after all, the purpose of playing a game.

This thesis has introduced the concept of *supportive presentation*. Rather than using graphics to simply enhance visual appeal, graphics can be used to augment the gameplay experience. Unfortunately, although photorealistic mastery has come a long way, tools that allow tailored visual presentations have not. Additionally, we still lack a comprehensive understanding of the effects of existing and potential presentation methods on the viewer. As such, a finished application utilizing supportive presentation is beyond the scope of a single thesis. However, the overall intent—to empower designers to specifically tailor a graphical presentation to carry across a communicative goal, will enable game designers to truly think outside of the box; to use graphics as a medium to support elements of the game in terms of the player(s), thus making the game experience more absorbing and ultimately more satisfying.

The remainder of this chapter summarizes our main contributions: The tools and vision for supportive presentation within game design.

6.1 Summary of Contributions

The main contributions of this thesis and how they relate to supportive presentation are subsequently summarized. It is necessary to note that although this thesis has focused on supportive presentation for computer games, our contributions are not restricted to this domain. In fact, our techniques are suited to all applications that seek to convey user presence within a synthetic environment and engagement with the objects therein. Games are simply a specialized instance of an immersive environment, chosen in this context because their goals are easily identifiable. Thus, many of the concepts and tools for supportive presentation described below are equally relevant to, for instance, virtual reality systems that demand clear communication and high levels of user interaction.

6.1.1 Empirical Evidence for Supportive Rendering

In Chapter 2 options for presentation strategies were introducing via the rendering method, *implicational rendering*: conveying latent affective content through rendering style. A first empirical study demonstrated that NPR can play a subtle yet effective role in guiding judgment and subsequent user interactions, implying that powerful effects are present in human responses to rendering style.

Consequently, we discussed the relevance and applicability of these results within computer games. Although our knowledge and tools are not ripe enough for a full implementation, Chapter 5 describes potential applications using these techniques for supportive presentation. These included use of relevant cues about level of threat, capabilities of characters, objects, or locations, and guidance when choosing between options such as which path to follow or even object utility.

Although knowledge of the effects of rendering are limited and player responses are subject to significant variation, these are amenable to systematic enquiry, wherein appropriate experimental design can further uncover and develop the relationship between rendering style and implicational knowledge. Thus, we outlined how existing psychological paradigms can be utilized to drive further research in NPR by using proper psychological measures ranging from statistical analysis of user selection to lower-level studies of brain activity. While we are far from providing a full theoretical account of the relationship between NPR and psychology, our research results are clearly suggestive of this interdisciplinary potential.

6.1.2 Rendering Methods

Supportive presentation requires visual styles to be tailored towards achieving a specific effect. To this end, we presented the OPENNPAR system, a unifying framework that supports a wide variety of NPR techniques. OPENNPAR is the first system of its kind to allow for a range of different user classes to both reproduce and create algorithms. Consequently, OPENNPAR is an invaluable tool for supportive presentation, offering potential for defining effective presentation methods within the virtually limitless scope of NPR. Developers can extend OPENNPAR's functionality by adhering to the development principles supported by our examples. Due to its modular structure, programmers can create new effects or assemble pre-defined effects via the manipulation of interchangeable modules. Knowledge of module assembly requirements is abstracted in a novel interface which frees designers to test new presentation methods without requiring in-depth technical knowledge.

The ability of OPENNPAR to create visual effects at all user levels is important for computer game development. Designers work at high levels of abstraction to explore presentation methods, which are then fine-tuned by programmers, and finally optimized by developers. This fits into the ideology behind the graphical presentation pipeline, wherein the higher-level tasks are executed first to more closely and effectively define the communicative intent behind supportive presentation.

6.1.3 Camera Methods

To further the ongoing development of camera techniques within interactive graphics, we provide tools that enable the designer control over the camera to more precisely define when and what content to capture.

The CAMPLAN system allows for experimentation with declarative constraints to evaluate their effectiveness in camera shot specification and communication before real-time solutions are worth investigating. Camera objectives are specified using visual properties selected from a complete taxonomy of specifications. We demonstrate that desired views of a scene can be defined naturally with these visual properties to produce results with stylistic consistency whilst also proving powerful enough to control precise viewing instructions.

An optimized subset of the visual properties from CAMPLAN are applied to the PREDATOR camera engine, wherein visual goals can be solved in real-time using novel techniques for interactive 3D games comprising scenes of arbitrary complexity. The PREDATOR system solves visual constraints based on existing camera state and motion characteristics, reacting to predicted future conditions to produce results that are consistent, robust, and frame-coherent. Each of these constraints are specifiable by a user, including the Potential Visibility Regions that offer advanced methods for solving multiple visibility constraints. The camera engine offers potential for a director to declaratively specify shot templates in composing sets of re-usable presentation methods.

To control when particular content should be captured, we outlined how the camera engine can be successfully integrated as part of a game pipeline that generates events, and implemented the SAMURAI system which presents an approach for the automatic generation of action summaries. Herein, interesting scenes are extracted by the analysis of specific game variables, and can be subsequently presented to the player as a sequence of representative images or given as timing requirements to the camera for live viewing of action. Although still in its early stages, the SAMURAI offers additional assistance for the selection of presentation strategies in particular contexts, such as evaluating player performance in fine-tuning presentation methods to players.

6.2 The Horizon

Throughout this thesis we discussed possible extensions or improvements to the rendering and camera tools. Employing these would certainly improve the functionality of tools and thus potential for devising and using a broader, more powerful range of presentation methods. However, there remains much to explore before the potential for supportive presentation can be fully realized.

6.2.1 An Interdisciplinary Approach

As discussed in Chapter 5, creation of effective supportive presentation involves the skills of psychologists, programmers, and designers. As a result, supportive presentation will benefit from bridges of communication with each area. Psychologists will provide insight into parameterizations for API's in system models for effective rendering and camera methods, but must at the same time communicate primary effects to designers. Designers are then responsible for creatively integrating essential visual components into an aesthetic result. Their images must be tested and re-affirmed by psychologists in conjunction with virtual test environments constructed by programmers. If the technological tools are too limited for designers to create desired images to be embedded into these virtual environments, additional functionality must be developed.

Consequently, regardless of where an idea for supportive presentation originates, best results can only be achieved by combining the expertise, imagination, and discipline within these three fields.

6.2.2 More than Just Graphics

The era of employing evolutionary graphical realism as a primary factor in game production is coming to an end. Graphics in games have now reached a maturity whereby any additional visual improvements are simply becoming details. Games can no longer be improved by, for instance, including a more advanced 'hair' shader into a sports game. It is also important to realize, that neither can games be improved by simply replacing realistic effects with alternative visual effects that are just a new look [Rubins, 2003]. We conclude this thesis by quoting the CEO of a game company, one which has traditionally thrived by taking existing game designs and pushed graphics to make the best looking game available, who comments on the issue of using non-photorealistic rendering styles in games:

> "It looks cool the first time, it may actually sell games the first time, but [eventually people will] expect you to do something unique and original with rendering style, just like when you walk into a new art show, you expect somebody to have done something different with the painting... You have to go beyond just doing some new style... You have to say something with the painting, and we're

> going to have to do something with the game...In the past, getting more colors, or more pixels, or more polygons, allowed you to do things that couldn't be done [before]... How is anything relative to cel-shading, giving...the ability to do something in the game, that couldn't have been done using the normally rendered ink?"—Rubins [2003].

Indeed, the answer lies in supportive presentation.

Appendix A: List of Implementations

All implementations were done in C++ and with the OpenInventor API with the exception of CAMPLAN which is implemented solely in C++ and OpenGL. The following applications/tasks were done independently by the author (estimated times are m = man months, w = man weeks, d = man days):

Application	Section	Specific Tasks	Time
OPENNPAR interface	3.4		4m
3D Painter	3.3.1		3d
Animation filter	3.3.5		1d
Camplan	4.2	Extensions from Master's thesis	1m
	4.2.3	Modelling Escher structure for testing	1w
PREDATOR	4.3		3m
3D Multi-player networked game	4.5.5		3m

The author gratefully acknowledges the participation and respective contributions of co-authors in assisting with the following:

Application	Section	Authors	Tasks	Time
Imp. Rendering Experiment	2.3	Nick Halper	Exp. design, image creation, test program, subj. testing, stat. analysis	3m
		Mara Mellin	Exp. design, image creation, subj. testing, stat. analysis	2m
		David Duke	Exp. design	3w
OpenNPAR	3.1	Nick Halper	Framework, line stylization, image processing, video output, animation modules, examples, etc.	8m+
		Tobias Isenberg Felix Ritter Roland Jesse Bert Freudenberg Oscar Meruvia	Please see www.opennpar.org for a full list of contributors and their accomplishments	24m+
OpenNPAR interface	3.4	Jana Hintze	Icon design	1m
		Iryna Davydova	Additional image processing filters	1m
Samurai	4.4	Nico Flohr, Nick Halper	1st version	1m
		Sven, Rico	2nd version (linked to game)	3w

Bibliography

Ernest Adams. Designer's notebook: Casual versus core. *Gamasutra*, August 2000.

Ernest Adams. Dogma 2001: A challenge to game designers. *Gamasutra*, October 2001.

Ernest Adams. Technology inspires creativity: Indie game jam inverts dogma 2001! *Gamasutra*, May 2002.

Hyeon-Soo Ahn and Il-Seok Oh. Fast shot detection from video images using large and adaptable skip factors. In *Proceedings of ACCV 95, Singapore*, pages 489–493, 1995.

D. Arijon. *Grammar of the Film Language*. Communication Arts Books, Hastings House Publishers New York, 1976.

William Bares, Joel Gregoire, and James Lester. Real-time constraint-based cinematography for complex interactive 3d worlds. In *Proceedings of the Tenth National Conference on Innovative Applications of Artificial Intelligence*, pages 1101–1106, Madison, Wisconsin, July 1998.

William H. Bares and James C. Lester. Intelligent multi-shot visualization interfaces for dynamic 3D worlds. In Mark Maybury, editor, *Proceedings of the 1999 International Conference on Intelligent User Interfaces (IUI-99)*, pages 119–126, N.Y., January 5–8 1999. ACM Press.

P. Barnard and J. May. Towards a theory-based form of cognitive task analysis of broad scope and applicability. In J.M.C. Schraagen, S.F. Chipman, and V.L. Shalin, editors, *Cognitive Task Analysis*, pages 147–163. Lawrence Erlbaum Associates, Inc., 2000.

P.J. Barnard and J. May. Interactions with advanced graphical interfaces and the deployment of latent human knowledge. In F. Paternó, editor, *Interactive Systems: Design, Specification, and Verification: Proceedings of the 1st Eurographics Workshop, 1995*, pages 15–49. Springer, 1995.

Hal Barwood. Cutting to the chase: Cinematic construction for gamers. *Gamasutra*, May 2000.

B. Bates. *Game Design: The Art and Business of Creating Games*. Premier Press, Inc., 2002.

Bruce G. Baumgart. A polyhedral representation for computer vision. In *Proceedings National Computer Conference*, pages 589–596, 1975.

G. Bente and N. C. Krämer. Virtuelle gesten: Vr-einsatz in der nonverbalen kommunikationsforschung. In G. Bente, N. C. Krämer, and A. Petersen, editors, *Virtuelle Realitäten*, volume 5, pages 81–107, Göttingen: Hogrefe, 2002.

Ray Bethers. *Composition in Pictures*. Pitman Publishing Corporation, New York, U.S.A., 1964.

Irving Biederman. Recognition by components: a theory of human visual understanding. *Pyschological Review*, (94):115–147, 1987.

James F. Blinn. Jim blinn's corner: Where am I? what am I looking at? *IEEE Computer Graphics and Applications*, 8(4):76–81, July 1988.

Matthew Brand. The "inverse hollywood problem": From video to scripts and storyboards via causal analysis. In *Proc. 14th Conference on Artificial Intelligence*, pages 132–137, 1997.

Andreas Butz. Anymation with CATHI. In *Proc. of the 14th National Conference on Artificial Intelligence and the 9th Conference on the Innovative Applications of Artificial Intelligence (AAAI/IAAI'97)*, pages 957–962, Menlo Park, July 27–31 1997. AAAI Press. ISBN 0-262-51095-2.

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.

Edwin K. P. Chong and Stanislaw H. Zak. *An Introduction to Optimization*. Wiley John & Sons, 2001.

D. Christianson, S. Anderson, L. He, D. Salesin, D. Weld, and M. Cohen. Declarative camera control for automatic cinematography. In *AAAI Proceedings of the National Conference on Artificial Intelligence*, volume 1, pages 148–155, 1996.

M. F. Cohen and D. P. Greenberg. The hemi-cube: A radiosity for complex environments. In B. A. Barsky, editor, *SIGGRAPH '85 Conference Proceedings*, volume 19 of *Annual Conference Series*, pages 31–40. ACM SIGGRAPH, Addison Wesley, July 1985. held in San Francisco, CA; 22–26 July 1985.

Cassidy Curtis. Loose and Sketchy Animation. In *SIGGRAPH 98: Conference Abstracts and Applications*, page 317, 1998.

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*, SIGGRAPH 97 Conference Proceedings, pages 421–430, New York, 1997. ACM SIGGRAPH. held in Los Angeles, 04-09 August 1997.

R. Davis. The fitness of names to drawings. *British Journal of Psychology*, 52:259–268, 1961.

Oliver Deussen, Jörg Hamel, Andreas Raab, Stefan Schlechtweg, and Thomas Strothotte. An Illustration Technique Using Hardware-Based Intersections and Skeletons. In I. S. MacKenzie and J. Stewart, editors, *Proceedings of Graphics Interface'99*, pages 175–182. Canadian Human-Computer Communications Society, 1999.

D. Dörner and H. Schaub. Die simulation von gefühlen. In G. Bente, N. C. Krämer, and A. Petersen, editors, *Virtuelle Realitäten*, volume 5, pages 57–79, Göttingen: Hogrefe, 2002.

Steven M. Drucker. *Intelligent Camera Control for Graphical Environments*. PhD thesis, Media Lab., MIT, 1994.

Steven M. Drucker, Tinsley A. Galyean, and David Zeltzer. CINEMA: A system for procedural camera movements. *Computer Graphics*, 26(2):67–70, March 1992. ISSN 0097-8930.

Steven M. Drucker and David Zeltzer. Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface '94*, pages 190–199, Banff, Alberta, Canada, May 1994. Canadian Information Processing Society.

Steven M. Drucker and David Zeltzer. CamDroid: A system for implementing intelligent camera control. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 139–144. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

Andew Dudley. *The Major Film Theories*. Oxford Press, 1976.

David Duke, Phil Barnard, Nick Halper, and Mara Mellin. Rendering and affect. *Computer Graphics Forum*, 22(3), 2003. Proceedings of Eurographics 2003, to appear.

Fredo Durand. An invitation to discuss computer depiction. In *Proceedings of NPAR 2002, Symposium on Non-Photorealistic Animation and Rendering 2002*, pages 111–124, New York, 2002. ACM.

Frédo Durand, Maneesh Agrawala, Bruce Gooch, Victoria Interrante, Victor Ostromoukhov, and Denis Zorin. Perceptual and artistic principles for effective computer depiction. Course Notes for SIGGRAPH 2002, July 2002. San Antonio, Texas.

Stefan Eickeler, Andreas Kosmala, and Gerhard Rigoll. Hidden Markov Model Based Continuous Online Gesture Recognition. In *Int. Conference on Pattern Recognition (ICPR), Brisbane,* pages 1206–1208, 1998. URL `citeseer.nj.nec.com/eickeler98hidden.html`.

Noah Falstein. A grand unified game theory. In *Game Developers Conference Proceedings,* pages 229–239, San Fransisco, 1999. Miller Freeman.

Nico Flohr. Action summary. Master's thesis, University of Magdeburg, Department of Simulation and Graphics, 2001.

Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Walk-through illustrations: Frame-coherent pen-and-ink style in a game engine. *Computer Graphics Forum*, 20(3): 184–191, 2001. ISSN 1067-7055.

Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Real-time halftoning: A primitive for non-photorealistic shading. *Rendering Techniques 2002, Proceedings 13th Eurographics Workshop*, pages 227–231, 2002.

A. Girshick, V. Interrante, S. Haker, and T. Lemoine. Line direction matters: An argument for the use of principle directions in 3D line drawings. In *Proceedings of NPAR'00*, pages 43–52, New York, 2000. ACM Press.

Michael Gleicher and Andrew Witkin. Through-the-lens camera control. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 331–340, July 1992. held in Chicago, Illinois; 26-31 July 1992.

Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A Non-Photorealistic Lighting Model for Automatic Technical Illustration. Computer Graphics Proceedings, Annual Conference Series, pages 447–452. ACM SIGGRAPH, 1998.

Amy Gooch and Peter Willemsen. Evaluating space perception in npr immersive environments. In *Proceedings of NPAR 2002, Symposium on Non-Photorealistic Animation and Rendering (Annecy, France, June 2002),* pages 105–110, New York, 2002. ACM.

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

Bilge Gunsel and Murat Tekalp. Content-based video abstraction. In *IEEE Int'l Conference on Image Processing - ICIP98,* volume 3, pages 128–132, 1998. URL `citeseer.nj.nec.com/gunsel98contentbased.html`.

G. Guthrie and M. Weiner. Subliminal perception or perception of partial cue with pictorial stimuli. *Journal of Personality and Social Pyschology,* 3(6):619–628, 1966.

Nick Halper, Ralf Helbing, and Thomas Strothotte. A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence. *Proc. Eurographics 2001*, 20(3):174–183, 2001. ISSN 1067-7055.

Nick Halper, Tobias Isenberg, Felix Ritter, Bert Freudenberg, Oscar Meruvia, Stefan Schlechtweg, and Thomas Strothotte. Opennpar: A system for developing, programming, and designing non-photorealistic rendering. Submitted, 2003a.

Nick Halper and Maic Masuch. Action summary for computer games: Extracting action for spectator modes and summaries. In Loo Wai Sing, Wan Hak Man, and Wong Wai, editors, *Proceedings of 2nd International Conference on Application and Development of Computer Games*, pages 124–132, Hong Kong, China, 2003. Division of Computer Studies of the City University of Hong Kong.

Nick Halper, Mara Mellin, David Duke, and Thomas Strothotte. Implicational rendering: Drawing on latent human knowledge. Submitted, 2003b.

Nick Halper, Mara Mellin, Christoph Herrmann, Volker Linneweber, and Thomas Strothotte. Psychology and non-photorealistic rendering: The beginning of a beautiful relationship. In *Proceedings of Mensch und Computer*, page to appear. ACM German Chapter, 2003c.

Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. Creating non-photorealistic images the designer's way. In *Proceedings of NPAR 2002, Symposium on Non-Photorealistic Animation and Rendering*, pages 97–104, New York, 2002. ACM. held in Annecy, France, June 2002.

Nicolas Halper. Camera planning for polyhedral worlds. Master's thesis, Department of Computer Science, University of York, 1999.

Nicolas Halper and Patrick Olivier. CAMPLAN: A Camera Planning Agent. In *Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20–22, 2000)*, pages 92–100, Menlo Park, 2000. AAAI Press.

Jörg Hamel. *Alternative Lighting Methods for Comuter Generated Line Drawings*. PhD thesis, Magdeburg, 2000. Published at Shaker-Verlag, Aachen, 2000.

Alan Hanjalic, Reginald Lagendijk, and Jan Biemond. Automatically segmenting movies into logical story units. In *Visual Information and Information Systems*, pages 229–236, 1999. URL `citeseer.nj.nec.com/hanjalic99automatically.html`.

Andrew Hanson and Eric Wernert. Constrained 3d navigation with 2d controllers. *IEEE Visualisation*, pages 175–182, 1997.

L. He, E. Sanocki, A. Gupta, and J. Grudin. Auto-summarization of audio-video presentations. In *Multimedia'99*, pages 489–498. ACM, 1999.

Li-wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 4-9 August 1996.

Wolfgang Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen, Computer Graphics Group, 1999.

I. Herman, G. Melançon, and M.S. Marshall. Graph visualisation in information visualisation: a survey. *IEEE Transactions on Visualization and Computer Graphics,* 6(1):24–44, 2000.

C. S. Herrmann and V. Bosch. Gestalt perception modulates early visual processing. *Neuroreport*, 12(5):901–904, 2001.

HLTV. Half-life television, 2002. http://www.planethalflife.com/features/articles/hltv/.

D. D. Hoffman. *Visual intelligence*. New York: Norton and Company, 1998.

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

V. Interrante. *Illustrating Transparency: communicating the 3D shape of layered transparent surfaces via texture*. PhD thesis, University of North Caroline at Chapel Hill, 1996.

V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3d line integral convolution. In *SIGGRAPH*, pages 109–116. ACM Press, 1997.

V. Interrante, H. Fuchs, and S. Pizer. Conveying the 3d shape of smoothly curving transparent surfaces via texture. *Transactions on Visualization and Computer Graphics,* 3(2): 98–117, 1997.

Barry Ip and Ernest Adams. From casual to core: A statistical mechanism for studying gamer dedication. *www.gamasutra.com*, June 2002.

Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Transactions on Computer Graphics*, pages 28–37, July/August 2003.

Tobias Isenberg, Nick Halper, and Thomas Strothotte. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum (Proceedings of Eurographics)*, 21(3):249–258, September 2002. ISSN 0167-7055.

Frank Jardillier and Eric Languénou. Screen-space constraints for camera movements: the virtual cameraman. *Computer Graphics Forum*, 17(3):175–186, 1998. ISSN 1067-7055.

Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In *Siggraph 2002 Conference Proceedings*, Annual Conference Series, pages 755–762, Reading, MA, July 2002. ACM SIGGRAPH.

Peter Karp and Steven Feiner. Issues in the automated generation of animated presentations. In *Proceedings of Graphics Interface '90*, pages 39–48, May 1990.

Peter Karp and Steven Feiner. Automated presentation planning of animation using task decomposition with heuristic reasoning. In *Proceedings of Graphics Interface '93*, pages 118–127, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

D.E. Kieras, D.E. Meyer, S. Mueller, and T. Seymour. Insights into working memory from the perspective of the EPIC architecture for modelling skilled perceptuo-motor and cognitive human performance. In A. Miyake and P. Shah, editors, *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*. Cambridge University Press, 1999.

Mark Kilgard. Creating reflections and shadows using stencil buffers. *Game Developers Conference*, 1999.

K. Koffka. *Principles of Gestalt Psychology*. Harcourt Brace Jovanovich, 1935.

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*, Computer Graphics Proceedings, Annual Conference Series, pages 433–438, New York, 1999. ACM SIGGRAPH. held in Los Angeles, August 1999.

D. Kurlander and S. Feiner. A history-based macro by example system. In *UIST '92 (ACM Symposium on User Interface Software and Technology)*, pages 99–106, Monterey, CA, November 15-18 1992. ACM.

D.H. Laidlaw, R.M.Kirby, and H.Marmanis. Visualizing multivalued data from 2d incompressible flows using concepts from painting. In D.Ebert, M. Gross, and B. Hamann, editors, *IEEE Visualization '99*, pages 333–340. IEEE Computer Society Press, 1999.

Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of NPAR 2000, Symposium on Non-Photorealistic Animation and Rendering*, pages 13–20, New York, 2000. ACM. held in Annecy, France, June 2000.

David A. Lauer. *Design Basics*. Holt, Reinhart and Winston, New York, 1979.

Kwan-Liu Ma. Image graphs - a novel approach to visual data exploration. In *Proceedings of IEEE Visualization '99*, pages 81–88, San Franciso, CA, October 24–29 1999. IEEE Computer Science Society Press, Los Alamitor, California.

J. Mackinlay, S. Card, and G. Robertson. Rapid controlled movement through a virtual 3d workspace. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 171–176, July 1990.

Scott McCloud. *Understanding Comics*. Kitchen Sink Press, May 1994.

Barbara J. Meier. Painterly rendering for animation. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 477–484. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 4-9 August 1996.

A. Mummendey, V. Linneweber, and G. Loeschper. Aggression: From act to interaction. In A. Mummendey, editor, *Social psychology of aggression: From individual behavior to social interaction*, pages 69–106, New York, NY, 1984. Springer.

Patrick Olivier, Nicolas Halper, Jon Pickering, and Pamela Luna. Visual composition as optimisation. In *AISB Symposium on AI and Creativity in Entertainment and Visual Art*, pages 22–30, Edinburgh, 1999.

Randy Pagulayan, Kevin Keeker, Dennis Wixon, Ramon Romero, and Thomas Fuller. User-centered game design. *Handbook for Human-Computer Interaction in Interactive Systems*, pages 883–906, 2003.

M. Patchen. Reciprocity of coercion and cooperation between individuals and nations. In R. B. Felson and J. T. Tedeschi, editors, *Aggression and violence: Social interactionist perspectives*, pages 119–144. American Psychological Association., Washington, DC, 1993.

Nilesh Patel and Ishwar Sethi. Video classification using speaker identification. In *Storage and Retrieval for Image and Video Databases (SPIE), San Jose, California*, pages 218–225, 1997. URL `citeseer.nj.nec.com/patel97video.html`.

S. Pauen. Denken vor dem sprechen. *Gehirn und Geist*, 1:45–49, 2003.

Cary B. Phillips, Norman I. Badler, and John Granieri. Automatic viewing control for 3D direct manipulation. In Marc Levoy and Edwin E. Catmull, editors, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 71–74, Cambridge, MA, March–April 1992. ACM Press. ISBN 0-89791-471-6 / 0-89791-467-8.

Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. *Proceedings of SIGGRAPH 2001*, pages 579–584, August 2001. ISBN 1-58113-292-1.

K. Provins, H. Stockbridge, D. Forrest, and D. Anderson. The representation of aircraft by pictorial signs. *Occupational Psychology*, 31:21–32, 1957.

Andreas Raab. *Techniken zur Interaktion mit und Visualisierung von geometrischen Modellen*. PhD thesis, Magdeburg, 1998. Published at Shaker-Verlag, Aachen, 2001.

Ramesh Raskar. Hardware support for non-photorealistic rendering. *2001 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 41–46, August 2001.

M. Reddy. *Perceptually Modulated Level of Detail for Virtual Environments*. PhD thesis, University of Edinburgh, 1997.

M. Reddy. Perceptually optimized 3d graphics. *IEEE Computer Graphics and Applications*, 21(5):69–75, (September/October) 2001.

J.P. Rolland, D. Ariely, and W. Gibson. Towards quantifying depth and size perception in virtual environments. *Presence*, 4(1):22–49, 1995.

Richard Rouse III. Game design: Theory and practice, chapter seven, the elements of gameplay. *Gamasutra*, 2001. http://www.gamasutra.com/features/20010627/rouse_03.htm.

Richard Rouse III. *Game Design: Theory and Practice*. Republic of Texas Pr, 2003. ISBN 1556229127.

Jason Rubins. Great game graphics...who cares? In *Game Developers Conference*. Gamasutra, April 2003. http://www.gamasutra.com/features/20030409/rubin_01.shtml.

T. A. Ryan and C. B. Schwarz. Speed of perception as a function of mode of representation. *American Journal of Pyschology*, 69:66, 1956.

Takafumi Saito and Tokiichiro Takahashi. Comprehensible Rendering of 3-D Shapes. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 197–206, August 1990.

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*, Computer Graphics Proceedings, Annual Conference Series, pages 101–108, New York, 1994. ACM SIGGRAPH.

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*, pages 35–42, New York, 1996. ACM SIGCHI.

Doree D. Seligmann. *Interactive Intent-Based Illustrations: Visual Language for 3D Worlds*. PhD thesis, Department of Computer Science, Columbia University, 1993.

Bruce Shelley. Guidelines for developing successful games. *Gamasutra*, August 2001. http://www.gamasutra.com/features/20010815/shelley_01.htm.

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

Jonas Smith. The road not taken – the 'how's and 'why's of interactive fiction. Technical report, The Department of Film and Media Studies, The University of Copenhagen, 2000.

Jonas Smith. The architectures of trust. Master's thesis, University of Copenhagen, Department of Film and Media Studies, February 2002.

Steve Strassmann. Hairy brushes. *Proceedings of Siggraph '86*, 20(4):225–232, August 1986.

Paul Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. In Edwin Catmull, editor, *Proceedings of SIGGRAPH'92*, Computer Graphics Proceedings, Annual Conference Series, pages 341–349, New York, 1992. ACM SIGGRAPH. held in Chicago, Illinois, July 1992.

Elena Stringa and Carlo Regazzoni. Real-time video-shot detection for scene surveillance applications. *IEEE Trans. On Image Processing*, 9 No.1:69–80, 1 2000.

T. Strothotte, B. Preim, A. Raab, J. Schumann, and D.R. Forsey. How to render frames and influence people. *Computer Graphics Forum*, 13(3):455–466, 1994. Conference issue: Proceedings of Eurographics'94.

Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann Publishers, San Francisco, 2002.

Bill Tomlinson, Bruce Blumberg, and Delphine Nain. Expressive autonomous cinematography for interactive virtual environments. In Carles Sierra, Gini Maria, and Jeffrey Rosenschein, editors, *Proceedings of the 4th International Conference on Autonomous Agents*, pages 317–324, NY, June 2000. ACM Press.

W. van Vliet. Exploring the fourth environment: An examination of the home range of city and suburban teenagers. *Environment and Behaviour*, 15:567–588, 1983.

Nuno Vasconcelos and Andrew Lippman. Bayesian modeling of video editing and structure: Semantic features for video summarization and browsing. In *IEEE Intl. Conference on Image Processing*, volume 2, pages 550–555, 1998.

Howard Wactlar. New directions in video information extraction and summarization. In *10th DELOS Workshop, Santorini*, pages 66–73, June 1999.

C. Ware and D. Fleet. Contex sensitive flying interface. *Interactive 3D graphics*, pages 127–130, 1997.

C. Ware and S. Osborn. Exploration and virtual camera control in virtual three-dimensional environments. *Interactive 3D graphics*, pages 175–184, 1990.

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

Hansong Zhang. Forward shadow mapping. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 131–138. Springer-Verlag Wien New York, 1998.

P. G. Zimbardo. The human choice: Individuation, reason, and order versus deindividuation, impulse and chaos. In W. J. Arnold and D. Levine, editors, *Nebraska Symposium on Motivation*, volume 17, pages 237–307, Lincoln, Nebraska, 1969. University of Nebraska Press.

List of Figures