Dynamic Visual Effects for Virtual Environments

Matthias Haringer
University of Hamburg, Germany
haringer@informatik.uni-hamburg.de

Steffi Beckhaus
University of Hamburg, Germany
steffi.beckhaus@uni-hamburg.de

ABSTRACT

We introduce a high level interface capable of instantly adding and manipulating a multitude of visual effects for any object or area in VE scenes. This enables a controlling person or an automated system to react to the in scene situation and the user's actions by dynamically changing the scene's appearance. Such a high-level scene and effect access is also a powerful tool for story telling engines and scene authors to support the intended plot or impact. To achieve this smoothly and effectively, our interface allows fading of effects to generate soft effect transitions and grouped effects for controlling complex effect combinations in a simple way. We also describe our fully functional implementation and the changes necessary to realize our concept in a scene graph based VR-system. We further introduce a simple script interface to add new effects to the available effect pool. We present, but are not limited to, three visual effect types: shader effects, post-processing effects, and OpenGl based effects. Our implementation supports multi-pipe displays, multi-pass rendering, and an arbitrarily deep per-object post-processing effect graph.

Keywords

Effect control, dynamic effects, post-processing, shaders, virtual environments

1 INTRODUCTION

Visual effects are nowadays commonplace in VR applications and games. Up to now their usage is mostly predefined by scene authors, but in principle many of those effects are accessible and modifiable at run-time. The effects we look at are not only high level effects like depth of field, heat haze, high dynamic range rendering, or lens flare, but also simple changes of brightness, contrast, color balance, and basically everything to change the appearance of the scene. The potential of controlling such effects dynamically in an application is rarely used, because there is no universal and real-time interactive access to most of them. Authoring tools available for games and some VR-systems allow scene creators to place and test effects of a scene, but do not provide for a general and intuitively controllable effect integration. They help to create scenes and introduce some dynamics via scripts, but they do not provide run-time access and effect control for the scene.

An analogy to our effect control concept from theater and stage performances is the light designer and the light board operator. The operator is able to control lighting and special effects with sliders and buttons. He can use one slider to control multiple effects, preprogram spot locations, and fade between effects. Usually a fixed schedule is prepared by the light designer. But at live performances the operator has to instantly react to the stage and audience and activates appropriate effects. As we look on interactive VE's, this live performance case is what we aim at.

Our vision is to create an interface capable of influencing the scene's appearance via many implemented effects. Operators of this interface only have to know what the effects can do, how they can be influenced, which user reactions can be expected, and which mood they might create - like in the light board case. Such an interface is not only useful for human control; it can be as well used for automated story engines or can be connected to user assessment systems to automatically react to the users state. Our system is not limited to visual effects, but as the visual effects were the first to be fully implemented, this paper concentrates on the visual part of the work.

The reduction of the brightness of the sky, for example, is achieved in our system by selecting the sky object, adding a brightness effect from the list of available effects, and reducing the brightness value via a slider. Other examples, which can be applied in a similar way are: making waves and clouds move faster, fading to night lighting conditions (by moving one slider which controls many effects), introducing fog, and correlating the inverse distance to a house to a weather change or a local glow effect on that house.

This paper presents our concept and implementation of a system for such a dynamic effect based scene manipulation. We introduce related work in Section 2 and give an overview over the whole system in Section 3. Section 4 introduces different types of effects and in Section 5 we present the implementation of the system. Section 6 lists some performance measurements for different effect usages, and in Section 7 a summary and outlook is given.

2 RELATED WORK

Various fields and methods informed the concept and the resulting system presented in this paper. The most important of them are discussed in the following. A major feature of our approach is the manipulation of different kinds of effects using a general interface. Conrad et al. introduced a tuner application for virtual environments to be able to manipulate several aspects of virtual environments via a GUI [2]. Modifiable parameters were object transformations, material properties, light source properties, and selecting the navigation metaphor (fly, drive, and walk). Conrad et al. use a fixed set of modifiable parameters. In our approach we want a more general and extendable solution, where new effects can be added easily.

To add effects at run-time, we have to be able to access the scene's objects. There exist many approaches, which allow such an access on the authoring level. [5] introduces an in-scene immersive authoring tool, where authoring takes place in the running system. [9] describes an easy to use rapid VE development environment. Multimodal authoring approaches like [1, 4] further allow the composition of multimodal scenes. As we plan to use effects of other modalities, we adopted the concept of a single object keeping the information for all modalities from these approaches.

The Listen project [3] introduces manipulating auditive parameters in an audiovisual scene graph for restricted zones in the scene. This enables authors to create different auditive spaces in different parts of a real or virtual room. We use this concept to apply our effects to run-time definable spatial areas in the scene. As we intend to use semantically meaningful objects in our scenes and the objects can contain semantic information, we oriented on current findings of this area for structuring the scene and its semantic information. [6] and [7] present examples of semantically extended scenes.

Another closely related field is game authoring and in game effects. Most game engines support object and special effect access on the authoring level. Some game engines allow dynamic effect behavior for some effects via scripts. Game effects are generally not run-time changeable. Examples include the Unreal editor for the Unreal Engine (Epic Games), which enables effect types like full-screen post-processing, particle and audio effects. These effects are realized in separate

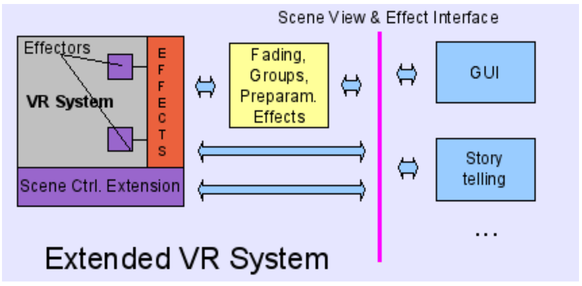


Figure 1: System overview

modules of the Unreal Engine. The effects can be accessed via the scripting language, but lack an uniform and intuitive access. The Source engine [8] uses a two step post-processing system for high dynamic range rendering and a color adjustment step. The color adjustments are made available for authors and are baked into game nodes for final game-play where they cannot be changed anymore. We found that today's game editors and engines do not provide for the generality of effects and run-time accessibility needed for our main purpose: to change the multimodal appearance and impact of the scene dynamically and uniformly.

[10] use non photo realistic rendering (NPR) techniques to create abstract views of 3D scenes. They are able to apply different NPR styles for different objects to emphasize one or more objects, which is comparable to our per object post-processing.

The introduced effects and effect mechanisms in Section 4 are not new and have been used in several games and VE's. Our contribution to using effects is the multi-pipe implementation, the mapping of technical to intuitive parameters, the per object post-processing effects, the general fading mechanism, and the flexible effect assignment and combination.

3 SYSTEM DESCRIPTION

Our System concept can be structured in three parts: The **scene control level** is the high level part, which lets moderators, engines, and authors easily modify the scene (middle and right part of Figure 1). The **effect configuration level** is a medium level, where effects and effect mappings can be defined ("EFFECTS" box in Figure 1). In the low level the effect types and the scene control with the per object effect manipulation are implemented (**implementation level**, see small squares and scene control extension in Figure 1). The scene control level is discussed in Sections 3.1 - 3.4 and the effect configuration level in Section 3.5. In Section 4 the effect types or effectors, as they are called in the following, are described.

The main goal of our system is to enable dynamic runtime assignable and changeable effects on a VE scene. On the scene control level there are three important questions that need to be answered when adding an effect to the scene. First, where do we want to apply the effect to (*object/scene part selection*). Second, which effects are to applied (*effect selection*). Third, how and when do we want to apply them (*effect transition, effect timing*).

To choose where we want to assign the effects to, or which part of the scene should be affected by the effect, the scene has to be accessed in some way. This interface to the scene is called *scene control* in the following. On top of that scene control, we introduce methods to help the easy handling of manipulating complex scenes and effects (box in the upper center of Figure 1). The more important of those methods are fading, preparameterized effects, groups, and a preselection capability. These techniques allow fast and complex effect manipulation and define how and when to apply the effects, which is especially important for live performance and automated scene control. The following sections describe the introduced steps in more detail.

3.1 Scene and Effect control

For intuitive and run-time assignable effects an inexperienced user should be able to select a part of the scene and assign an effect from a list of implemented effects to them. However, for which parts of a scene is assigning effects reasonable? The first case of entities we identified, are semantically meaningful objects in the scene (like sky, buildings, and trees). Which objects are meaningful and which granularities to use, heavily depends on the scene and the application. A second case where applying effects is important in our opinion, are semantic spatial areas in the scene. Such areas are volumes, which do not have a direct visual representation. Examples for effects on areas are a cool scene appearance around a specific house, brightness on the beach, or a greenish look in a forest.

In our system, objects and areas can be structured in a tree hierarchy and each of these objects or areas can hold multiple effects. The implementation of this part has to be able to extract and manipulate objects from the scene, it has to provide areas, and it has to realize the connection between the effects and the objects (see Section 5.3).

After we know where to place an effect, we have to decide which specific effect is to be applied. The control over the effects should be equally high level. For choosing an effect, we only need to know what the effect does (name and description) and how we can influence its behavior (parameters). It is not important at that point whether the effect is realized via shaders, post-processing or manipulating OpenGL states. Our interface for selecting effects is an effect pool with descriptions of effects and parameters. All currently available effects in the system are in that pool and can be arbitrarily added to objects.

3.2 Preparameterized effects and groups

Most effects can be customized via parameters, for example red, green and blue parameters that adjust a simple RGB color balance effect. Because of the lack of time for adjusting the parameters online and the need to access some previously optimized parameters, predefinable settings are additionally required. In our system **preparameterized effects** allow users to store an effect with a specified set of parameter values. The preparameterized effects can be created anytime while experimenting with the parameters of an effect. They can be applied to objects like any other effect.

Our second method to enable an intuitive and quick effect handling are the following grouping mechanisms: **Object groups** allow us to add an effect to multiple objects at once. For example, say we want to modify the color of sky and water simultaneously. An object group holding the sky object and the water object is created and an effect is added to this group. **Effect groups** allow us to add multiple effects to an object or an object group. They make it possible to switch or fade all effect members on and off together. For order dependent effects, like post-processing, an effect order can be defined. **Multi-object effect groups** allow us to assign multiple objects with one or more different effects. They can be used for complex effect settings like to represent different weather situations. A dark rainy day multi-object effect group might combine dark sky, high waves, grey water, dull landscape (low saturation), and rain on all outdoor objects, which can be switched or faded as one. Object and multi-object effect groups are, because of their reference to specific objects, scene dependent. Effect groups and preparameterized effects can be shared between scenes.

3.3 Effect preselection

Not all effects are a good choice at all times or for all places and objects. Therefore, the number of available effects can be reduced per object, area, or by time constraints. This gives a live operator or an automated process the simplification that only effects that make sense at the current place and time are possible to select. Those restrictions and context dependent effects have to be prepared as an authoring step using the scene control, areas, and groups.

3.4 Effect intensity and fading

Being able to switch effects on and off at run-time for arbitrary objects is already a valuable improvement for manipulating a scene. Often, however, a subtle fade in, fade out, or fade between effects is required. When adding or removing effects at run-time, fading is in most cases preferable, as smooth changes are perceived as more natural. One example is slowly introducing a cold scene look by a darker and bluish appearance of the landscape and sky or the simulation of

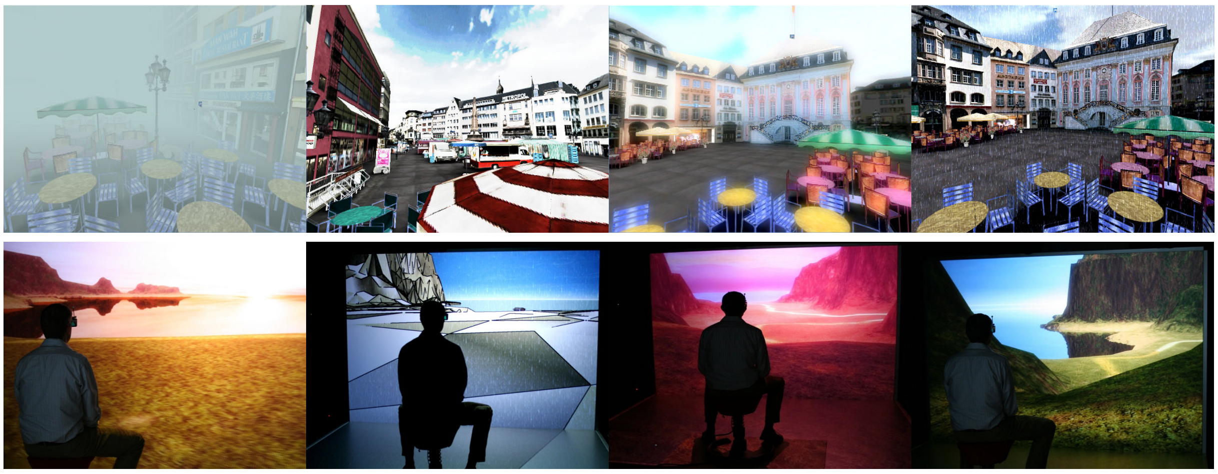


Figure 2: First row: Effects in a marketplace VE scene (left to right): Fog, bleach bypass, bloom, rain. Second row: Island scene (left to right): a) different brightness adjustments for water, land, and sky, color balance on the whole scene, sunset sky; b) no landscape textures, random gray tone per face, edge enhance, rain; c) bloom on landscape, different color balance and brightness effects on landscape, water, and sky; d) daylight sky, adjusted landscape detail shader.

a cloud covering the sun by fading out a glow effect and fading in a low brightness effect. Another application of fading is, when different areas of the scene should have different appearances. When moving over the border, fading between the two appearances is required.

Implementing the fading capability involves being able to manipulate an effects intensity without altering the other effect parameters. The effect intensity defines the range of the full effect to some zero effect. Directly fading between two effects is only possible, if the two effects can exist simultaneously. For this kind of fading the intensity of the first effect is lowered and the intensity of the second effect is raised over a defined time interval. Fading is simple to realize for post-processing effects and some object shader effects, because they can easily be mixed with the original scene to an adjustable degree. For other effects the intensity mapping can be more complicated, but the generic fading capability for all effects exceeds the effort spent here.

3.5 Effect definition and mapping

The configuration layer in between the scene control interface and the implementation has two sub-layers: the **definition** of the effects to be used in the interface and the **mapping** of the parameters of the implementation to high level parameters. The mapping also introduces descriptions of effects and their parameters.

The effect definition holds resources like shaders and textures and combines them to form an effect. All necessary functions to easily create and configure effects are provided by the implementation layer (effectors). The effect definition part requires shader and script programming and is intended for effect artists.

The effect mapping can directly assign, combine, or modify internal parameters (e.g. uniforms) to form the final effects parameters. Descriptions, effect name, and parameter names to be seen in the final effect are supplied here. The mapping part requires only simple script programming and is, therefore, easily adjustable. It is intended to separate the technical formulated and parameterized effects from the final effects, which use meaningful parameters with appropriate ranges for their intuitive control. Effect definition and mapping build together a complete effect, which can be accessed from the effect pool.

4 EFFECTORS AND THEIR EFFECTS

Effectors are the implementations of effects or types of effects with the same implementation base. Effectors encapsulate the system internals needed for an effect type and they have to provide everything to assemble the effects in the effect definitions. The shader effector, for example, provides the interface for all shader based effects. Effectors can control the rendering pipeline, if necessary, and they are aware of restrictions of the system or restrictions regarding the coexistence with other effectors. In this section three effectors for a large number of possible visual effects are introduced. Examples of effects from these effectors can be seen in Figure 2. Effectors for ambient lighting, navigation effects, and auditive effects are currently under development.

4.1 Post-processing effector

Post-processing effects are a very powerful type of visual effects. They adjust the appearance of the scene with image processing techniques. Up to now, post-processing has mainly been used on the whole visible display area. In our implementation, it can also be applied on a per objects basis to better

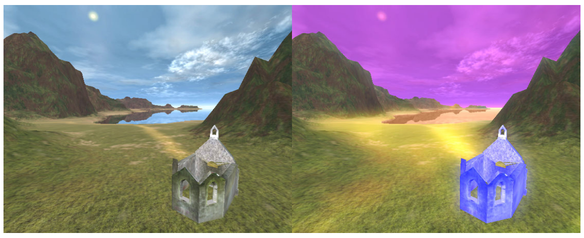


Figure 3: Scene without (left) and with (right) per object post-processing effects for sky, water, ground, and chapel

match our object based effect approach. Allowing different post-processing pipelines for sky, ground, and water surfaces already permits us to refashion the appearance of the scene more efficient than full screen post-processing. Figure 3 shows an example of per object post-processing. Post-processing effects of this kind have the advantage over shader effects that they are universally applicable, i.e. they can be added to any object in the scene and will have the intended effect, where shader effects often rely on specific geometry and vertex properties. The implementation is done via frame buffer objects and specialized shaders and is usable for multiple graphics pipes (see Section 5.6). The passes, shaders, uniforms, and effect parameters are defined and assembled to form the specific effects in the effect definition and mapping.

Our post-processor system is highly flexible. Each object or area can have several post-processing effects and each post-processing effect can incorporate several post-processing passes. Each pass has a post-processing shader, up to six input textures, an arbitrary number of uniforms and a result texture. Passes can be stacked and the results of different passes can be combined, i.e. the passes can be laid out in a directed graph structure, which allows for very complex effects. A simple two pass example is a bloom shader, which extracts all bright pixels in a first step and applies a blur shader to the extracted regions as a second step.

Some already implemented post-processing effects are color balance, bloom, blur, depth of field, edge detection, edge enhance, tone mapping, rain, and some sketch effects like hatching. An important group of post-processing effects, which can easily change the impact of an object, are color manipulation effects. We have implemented two approaches to modify the coloring of the scene. Our color grading effect assigns each color value a different color via a predefined color map. Combining simple post-processing effects like color balance, brightness, contrast, saturation, and gamma adjustment can produce similar but not as exact results to the mapped color grading approach. The advantage of this approach is the dynamic manipulation possibility of all components.

Realizing effects via post-processing has also some disadvantages. A problem with post-processing for stereo displays is that brought in content like a texture appears only on the screen plane. A rain effect, for example, which is believable on mono displays, can not be used for a stereo setting because of this effect. Our solution is to introduce a pipe dependent shift of the images to produce an eye offset. Using this, the seen plane can be moved before or behind the viewing plane, but it still appears as plane. In our rain post-processing effect five rain planes with different depth are used to simulate the rain in stereo. Each rain plane uses 5 differently sized versions of a rain texture which are moved with different speed over the screen using a post-processing vertex shader.

4.2 Shader effector

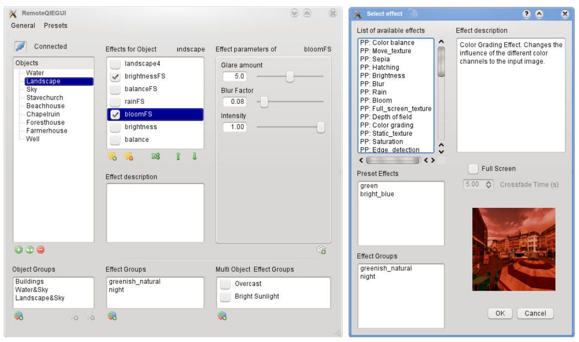
Object shader effects are common in most VR-systems and game engines. They are mostly used to realize specific surface materials and moving surfaces like water and clouds. Many effects which are usually implemented as normal shaders can be created via the per object post-processing effects in our setup. Whenever vertex based information is needed (position, normals, texture coordinates) a conventional shader has to be used. Being dependent on geometry, many shader effects are less generally applicable than post-processing effects. Because of that, shader effects are mainly used for the basic appearance of our scenes, which can be modified by the parameters of the shader effects.

The shader effector provides a means to simply add shaders and supply them with input textures and uniform variables. This makes it possible to easily integrate any existing GLSL shader in a few minutes. Our current implementation allows only one object shader effect per object. Most multi-pass shaders like non photo-realistic shader effects can be created by using one object shader and several post-processing shaders. Currently integrated shader effects are: animated water, sky, and grass, as well as a level of detail terrain shader and materials like plastic and glass.

4.3 GeoState effector

GeoStates encapsulate OpenGL graphics properties for the geometry (GeoSets) in OpenGL Performer style scene graphs. The GeoState of an object's geometry can be overridden with a GeoState effector. The effect definition and mapping defines which properties of the GeoStates are to be overridden and which property values should be applied instead. GeoState effectors can affect the following states: lighting, texture, fog, wireframe, highlighting, material, alpha function, transparency, decals, and light sources.

The fading of most GeoState effects is more problematic than for post-processing and shader effects. When exchanging textures, enabling highlighting, etc., there is no simple way to avoid a sudden change. One possibility, which we use for texture fading, has following



Objects
- Water
- Landscape
- Sky
- Stavechurch
- Beachhouse
- Chapelruin
- Foresthouse
- Farmerhouse
- Well

Effects for Object ındscape
- [ ] landscape4
- [✓] brightnessFS
- [ ] balanceFS
- [ ] rainFS
- [✓] bloomFS
- [ ] brightness
- [ ] balance

Effect description

Effect parameters of bloomFS

Glare amount	5.0
Blur Factor	0.08
Intensity	1.00

Object Groups: Buildings, Water&Sky, Landscape&Sky

Effect Groups: greenish_natural, night

Multi Object Effect Groups
- [ ] Overcast
- [ ] Bright Sunlight

List of available effects
- PP: Color balance
- PP: Move_texture
- PP: Sepia
- PP: Hatching
- PP: Brightness
- PP: Blur
- PP: Rain
- PP: Bloom
- PP: Full_screen_texture
- PP: Depth of field
- PP: Color grading
- PP: Static_texture
- PP: Saturation
- PP: Edge_detection

Effect description: Color Grading Effect. Changes the influence of the different color channels to the input image.

- [ ] Full Screen

5.00 Crossfade Time (s)

Preset Effects: green, bright_blue

Effect Groups: greenish_natural, night

Figure 4: Scene Control GUI: Left: Object and effect control. Right: add a new effect.

process. We start from a single textured object without a shader. As a first step, we add a shader effect with two configurable textures, the one that was present and a new one. We fade from the first to the second texture via the shader effect's intensity. In the mean time a GeoState effect with the new texture is applied (hidden by the shader) and finally the shader effect is switched off again. We use this technique to fade between different sky textures for different times of day or weather conditions.

5 IMPLEMENTATION

This chapter gives an overview of the implementation of our system. The VR-system dependent parts of our implementation use Avango [12] which is based on OpenGL Performer [11]. Future adaptions for AvangoNG and VRJuggler (using OpenSceneGraph) are planned. Section 5.1 describes the implementation of the scene control interface and Section 5.2 the implementation of the effect control layer. Sections 5.4– 5.7 lay out the implementation of the effectors and necessary adjustments to the render pipeline.

5.1 Interface

The scene control interface is realized with XML-RPC. The implementations of objects, effects, and parameters define RPC functions for all functionalities of the interface. On the remote side, handles of the objects, effects, and parameters are used to access their interface functions. The modules for fading, groups, and preparameterized effects provide their functionalities also as an XML-RPC interface. External applications like storytelling or user state estimation can use the complete XML-RPC interface. The GUI which was implemented for moderators and authors also supports the complete interface inclusive fading and groups. The GUI is intended for controlling the scene during run-time and represents the light control board from our analogy in the introduction. A screen-shot of the GUI can be seen in Figure 4. The scene control GUI has three colums for displaying objects, effects, and effect parameters respectively. If an object is selected, its effects are displayed in the effects column. If an effect is selected or added, its parameters are displayed in the effect parameter section.

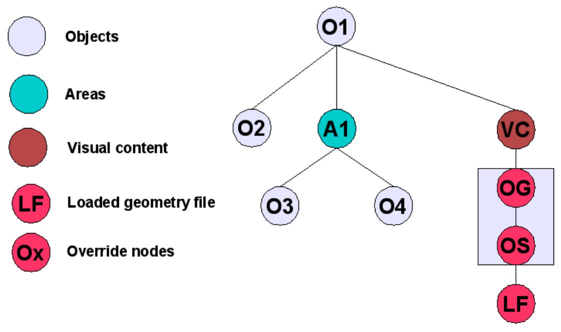


Figure 5: Extension of the scene graph

5.2 Effect definitions and mappings

The effect definitions are currently implemented in Scheme, which is the scripting language of Avango. We plan a reimplementation using Python to be more easily adaptable to other systems. To create an effect of a specific effector type, a C++ instance of this effector is created and initialized by loading the effect definition via its name. Each effect is located in a file system directory containing effect definition, effect mapping, descriptions, and resource files.

5.3 Scene control implementation

The main idea in our scene control implementation is to use the existing scene graph of a given VR-system and extend it to match our scene concept of semantic objects and effects per object. As scene graphs are extensible, the required nodes can be derived from existing nodes. The scene abstraction is mapped to the OpenGL Performer style scene graph as follows: Objects are inherited from transformation nodes (DCS) in the underlying scene graph. Each object has its visual contents as a special child (VC in Figure 5). Our implementation also allows for other content types, like auditive, haptic, or physics based content which can be rendered by the respective engines. The actual visual appearance of an object can be an arbitrarily deep sub scene graph or a loaded model (LF in Figure 5). Child objects and areas are also children of the transformation node (A1 and O2 in Figure 5). Other children than child objects and content are not allowed in the structure. Effects are not implemented as scene graph nodes, they are properties of the object node.

For applying shader and GeoState effects, special override nodes are placed between the VC node and the actual content (OG and OS in Figure 5). They allow to override the GeoState and shader settings of all underlying content geometry. The VC nodes also hold a post-processing ID to be used in object based post-processing (see Section 5.6).

5.4 GeoState effector

The GeoState effector is implemented using override nodes, which partly use the OpenGL Performer GeoState override functionality and in some cases manipulate OpenGL states directly. When a GeoState effect is activated, an override node is placed above the geometry of an object. This allows GeoState effects to change one or more states of this geometry.

5.5 Shader effector

The shader effector is implemented using a shader override group, which activates the shader for all underlying geometry. The shader uniform variables to manipulate the shader can be mapped directly or can be further processed to form effect parameters. Standard uniforms variables like time, user position, and light source positions are automatically provided for all shader effects.

Complex material properties like reflection and refraction may need additional render passes. An example is a water shader with water reflection and water refraction, which requires two additional render passes. In those cases, the scene or a part of the scene is rendered before the main render pass. For example, an upside down version of the scene is rendered for reflection which is stored in a texture. The shader effect can use this texture afterwards like any normal shader input texture.

5.6 Post-processing effector

The post-processing effector requires a more elaborate extension to the VR-system. The scene is rendered into a frame buffer sized texture using frame buffer objects (FBO) for each graphics pipe. A shader is then applied on a 2D rectangle holding this texture. The result is again rendered to a texture or, if the last post-processing effect has been reached, the final result is rendered in the frame buffer. In fullscreen mode, the post-processing shader is applied to all pixels of this texture. In the object based case, all objects with post-processing effects are assigned an ID. This number is used to mask the object in the stencil buffer during the rendering of the scene into the texture. The stencil/depth buffer is rendered in an additional texture using frame buffer objects, which can be accessed in the shaders. Only pixels with the according value in this texture are changed by the shaders. This way the shaders only affect the visual geometry of the masked objects.

In each shader pass the first two texture units are assigned to the color texture of the previous pass and the stencil/depth texture. The stencil/depth texture is only used for reading the depth and the object mask values during the post-processing. Six additional textures (assuming 8 texture units) can be freely used by the specific shaders. The number of simultaneously displayed per object post-processing effects is limited by the size of the stencil buffer. As a stencil buffer of 8 bit is available on most systems, this would limit the objects for holding separate post-processing effects in a scene to 255. To relax this limit we dynamically reassign the stencil values for the currently visible objects with post-processing effects. 255 visible objects with post-processing should suffice for most scenes and viewpoints.

5.7 Multi-pass rendering

Shader and post-processing effectors need to modify the render behavior of the VR-system. Each graphics pipe can have several pre-render passes, which render the scene with different settings and one main rendering pass. The pre-render passes are passed as FBO textures. These textures can be accessed by shader effects. The pre-render passes are only activated, if an effect is currently using them. The main render pass incorporates either the post-processing rendering process described earlier or a standard scene rendering. In the post-processing case, the main render pass renders the scene normally, then applies a post-processing pass per post-processing effect and object holding such an effect. The last post-processing pass writes to the frame buffer.

6 SYSTEM PERFORMANCE

Among several test scenes the system has been tested with three larger scenes. Two of them were conventional scenes adapted to the extended system. The adaption needed following steps: Splitting the scene into semantic parts to be used in the scene control, using these parts as visual content of our enhanced object nodes, and adding areas. The whole adaption process of the scenes has been managed in several hours.

An island scene (seen in the lower row of Figure 2) was designed for our system and uses advanced shader concepts and incorporates many areas. The main components are a $2.5km^2$ terrain mesh generated from a 512*512 point height-map and using a 4096*4096 pixel base texture rendered with Terragen, a $5km^2$ water plane, and a sky dome with a 2048*768 pixel base texture. The scene is currently populated with some buildings and billboard trees. The complete scene has approximately 450k polygons. For the landscape a LOD detail shader effect that uses textures for grass, sand, rock, and forest is used. The water is realized via a multi-pass watershader effect.

Our hardware setup is a dual Opteron HP XW9300 workstation with two synchronized Nvidia Quadro 4600 graphics boards. Our L-Shape display is driven by two Projectiondesign active stereo projectors, which are fed with two graphic pipes each.

	1Pipe	2Pipe	4Pipe
1. Basic island scene (Avango)	81fps	62fps	50fps
2. with our extensions	81fps	61fps	50fps
3. 2. + 2 render passes	65fps	35fps	25fps
4. 3. + water & land shaders	60fps	27fps	22fps
5. 4. + medium effect usage	58fps	25fps	20fps
6. 4. + heavy effect usage	55fps	22fps	16fps

Table 1: Performance for 1, 2, and 4 pipes.

Our measurements are taken for a single, a two, and a four pipe setup. A predefined path in the scene is used to generate an average frame rate. Table 1 shows the performance for different configurations from the bare scene to heavy effect usage. The heavy effect usage performance run includes 5 full screen post-processing effects, 15 post-processing effects on currently visible objects (landscape, sky, water), and several shader and GeoState effects. The medium effect usage run is done with 2 full screen and 5 currently visible object based post-processing effects.

The small difference of the first and second run show the low performance impact of our system extension. The high performance penalty for 4 pipes in run 3 to 6 is mainly caused by the two additional render passes (synchronization issues). To reduce this effect, we intend to introduce optimizations of the graphics pipes, like assigning processes to processors and sharing GL context for the pipes for the left and right eye, to reduce the texture load per PCIe interface. Nevertheless, the relatively small differences of run 4 to 6 show that high quality scenes and a large amount of effects are possible using a 4 pipe display.

7 CONCLUSION

We introduced a system, which is capable of dynamically manipulating a VE scene by adding effects for specific objects, areas, or the whole scene. The implemented system provides an intuitive interface allowing moderators, authors, or automated systems to modify the scene online using all available effects. It also provides an internal effect definition and mapping interface, which makes it possible to easily create new effects for already implemented effectors, like shaders or post-processing, and which maps technical variables to meaningful effect parameters. We introduced and discussed our implementation of the system, currently supporting Avango as a VR-system.

Our implemented visual effects have demonstrated the effectiveness and flexibility of the concept and our implementation. The full potential of the system for multimodal and varying content will show, when more modalities are supported and complex multimodal effect combinations can be generated. Effect performance tests with a complex scene showed that our implementation can enhance scenes with many real-time changeable effects without substantially limiting the performance.

Future work includes effects for the auditive and haptic modality, as well as the addition of several external effects like generating wind and ambient lighting. We are currently working on introducing a mapping of effects to moods they might produce. As a result of that, a target mood selection will cause appropriate effects and parameterizations to be used. As the impact of effects depends on scene context and users, this mapping is a big challenge. Integrating such a system with user assessment systems will make it possible to directly react with specific effects to the current state of the user.

REFERENCES

[1] M. Billinghurst, S. Baldis, L. Matheson, and M. Philips. 3d palette: A virtual reality content creation tool. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 155–156, Lausanne, Switzerland, 1997.

[2] S. Conrad, H. Krüger, and M. Haringer. Live tuning of virtual environments: The vr-tuner. In *Virtual environments 2004. 10th Eurographics Symposium on Virtual Enviroments*, pages 123–128, Grenoble, France, 2004.

[3] G. Eckel. Immersive audio-augmented environments - the listen project. In *Proceedings of the 5th International Conference on Information Visualization (IV2001)*, 2001.

[4] M. Green. Towards virtual environment authoring tools for content developers. In *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 117–123, New York, NY, USA, 2003.

[5] R. Holm, E. Stauder, R. Wagner, M. Priglinger, and J. Volkert. A combined immersive and desktop authoring tool for virtual environments. In *VR '02: Proceedings of the IEEE Virtual Reality Conference 2002*, page 93, Washington, DC, USA, 2002.

[6] E. Kalogerakis, S. Christodoulakis, and N. Moumoutzis. Coupling ontologies with graphics content for knowledge driven visualization. In *Proceedings of the IEEE Virtual Reality Conference 2006*, pages 43–50, 2006.

[7] M. E. Latoschik, P. Biermann, and I. Wachsmuth. Knowledge in the loop: Semantics representation for multimodal simulative environments. In *Smart Graphics*, volume 3638/2005 of *Lecture Notes in Computer Science*, pages 25–39. 2005.

[8] J. L. Mitchell, G. Mc Taggart, and C. Green. Shading in valve's source engine. In *SIGGRAPH Course on Advanced Real-Time Rendering in 3D Graphics and Games*, 2006.

[9] X. Qian, Z. Zhao, and R. Thorn. Rapid development of virtual environments a systematic approach for interactive design of 3d graphics. In *WSCG 2007, SHORT COMMUNICATIONS PROCEEDINGS I AND II*, pages 117–124, W Bohemia, Plzen, CZECH REPUBLIC, 2007.

[10] N. Redmond and J. Dingliana. A hybrid technique for creating meaningful abstractions of dynamic 3d scenes in real-time. In *WSCG 2008, FULL PAPERS*, pages 105–112, Univ W Bohemia, Plzen, CZECH REPUBLIC, 2008.

[11] J. Rohlf and J. Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994.

[12] H. Tramberend. Avocado: A distributed virtual reality framework. In *Procedings of IEEE Virtual Reality 1999 (IEEE VR 1999)*, pages 14–21, 1999.