GAMETOOLS

ADVANCED TOOLS FOR DEVELOPING HIGHLY REALISTIC COMPUTER GAMES

TEST REPORT ON RELEASE PLUG-INS PROTOTYPES

Document identifier:	**GameTools-6-D6.2.1-02-0-1-Project integration, evaluation and test**
Date: (use "update field" Word function, right mouse button)	**30/11/2006**
Work package:	**WP06: Project Integration, evaluation and test** (e.g.: WP6: Integration, Test)
Partner(s):	**All**
Leading Partner:	**INFOWERK**
Document status:	**DRAFT**
Deliverable identifier:	**D6.2.1**

Abstract: The test report on release plug-ins prototypes covers the matching of the requirements and the functionality, the performance of the plug-ins and the assessment of the increase in realism of the resulting 3D – Applications. Furthermore it gives some insight of the ease of integration of each work package.

Delivery Slip

	Name	Partner	Date	Signature
From	Martin Kolb	Infowerk		
Reviewed by	Moderator and reviewers			
Approved by				

Document Log

Issue	Date	Comment	Author
0-0		First draft	Martin Kolb
0-1		GEDAS second draft	David Gutiérrez
0-2		Third draft including results of DLE	Martin Kolb
0-3		Fourth draft including integration work of GEDAS	Martin Kolb

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files / URL
Word	gametools-6-d6.2.1-02-2-0-project integration, evaluation and test.doc (use “update field” Word function)

CONTENT

1. INTRODUCTION

1.1. OBJECTIVES OF THIS DOCUMENT

This document is the second deliverable of the “Project integration, evaluation and test” work package. Its aims are to summarize the results in requirement fulfilling, performance, increase of realism and ease of integration. These results were originated by each company partner in their own testing conditions, which will be listed in particular.

1.2. APPLICATION AREA

The testing environment at InfoWerk GmbH has the following specifications:

Pentium D 3.2 GHz

GeForce 7900 GS with 256MB

The testing environment at GEDAS Iberia (a member T-Systems Iberia) has the following specification

AMD X2 4800, 2 GB RAM

GeForce 7900GTX with 512Mb

The testing environment at DLE has the following specifications:

P4 3GHz, 2GB RAM

GeForce 7800GT 256Mb

1.3. TERMINOLOGY

Glossary

WP	Work package
OGRE	Open Graphics Rendering Engine
DLL	Dynamic Link Library
SVN	Subversion, a version controlling system
HLSL	High level shading language, a programming language to write certain programs for the shader unit on a graphics card
GLSL	OpenGL shading language, the open source pendant to HLSL
CHC	Coherent hierarchical culling
HPS	Hierarchical particle systems
FPS	Frames per second
LOD	Level of detail
NextGen	Next generation, a catchphrase meaning hardware developed from 2006: Playstation 3, Xbox 360, PC graphics cards with at least shader model 3 and other appropriate hardware
PVS	Potentially visible sets
DLE	Digital Legends Entertainment

2. EXECUTIVE SUMMARY:

All work packages work quite well on their own concerning performance and/or increase of realism. Therefore they can be very useful for the game developing companies. At the other hand it is rather difficult to get the work packages written for the same graphics API to run together. Referring to many conversations with the work packages members this is not developed yet and is an important task for the team, since it is obvious that the game developing companies surely would use all the frameworks together. This requirement is hard recommended.

The next recommendation is connected to the unification of used languages.

In fact some of the packages are written using OpenGL and some of them are written using Direct3D so is impossible to use all of them at once without translate from one language to the other. GEDAS had to translate some effects to use them inside its VRengine.

This table covers the application possibilities for each tested tool.

	Third person	First person	Driving	Sports	Flight simulators	Fighting
CHC	🟩	🟩	🟨	🟥	🟧	🟥
LodStrips	🟩	🟩	🟩	🟨	🟩	🟧
Depth of Field	🟨	🟩	🟨	🟨	🟨	🟨
Environment maps	🟩	🟧	🟨	🟧	🟨	🟩
Glow	🟩	🟩	🟩	🟩	🟩	🟩
Tone mapping	🟩	🟩	🟩	🟧	🟧	🟧
Reflection	🟨	🟨	🟩	🟨	🟩	🟨
Refraction	🟧	🟧	🟨	🟧	🟧	🟧
Caustics	🟧	🟧	🟧	🟧	🟧	🟧
Precomputed light paths	🟩	🟩	🟧	🟧	🟧	🟨
Spherical billboards	🟩	🟩	🟩	🟨	🟩	🟨
Hierarchical particle systems	🟧	🟧	🟧	🟧	🟩	🟧

🟩	Essential
🟨	Useful
🟧	Not very useful
🟥	Useless

3. WP3: VISIBILITY

3.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY

Referring to the results created in Infowerk’s test environment given by the DirectX™ tool PIX and the summary given by the OGRE the matching between requirements and functionality is fulfilled.

The algorithm works properly and exposes its advantages in appropriate environments. A suggestion would be a more flexible algorithm selection mechanism reacting to the environment and the objects present in the view frustum. However, the visibility work package was not working together with the illumination package.

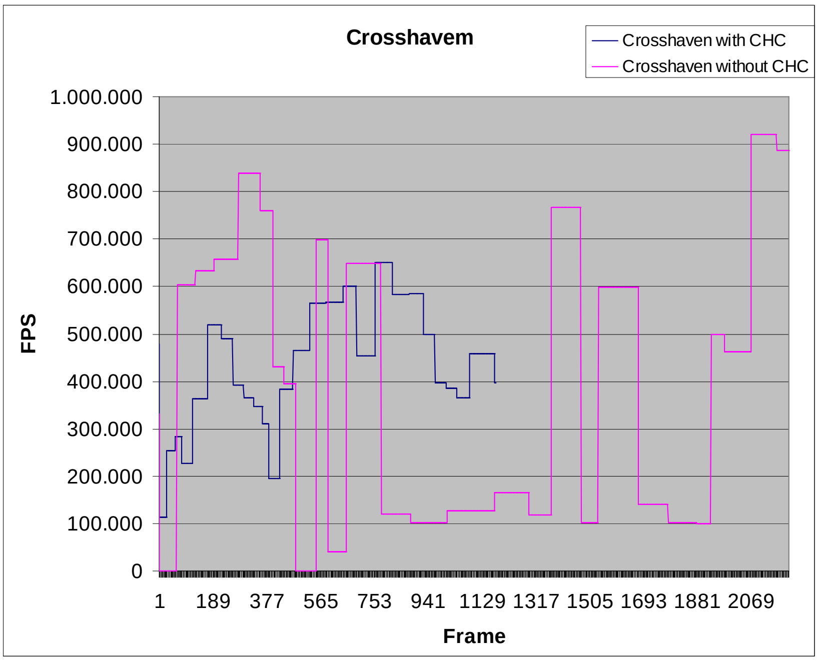
GEDAS has begun the integration of the offline visibility module into their engine. They now export Alias/Wavefront .OBJ files from their editor, and use these files with the stand alone application provided by Vienna to generate the Visibility Cells and the PVS.

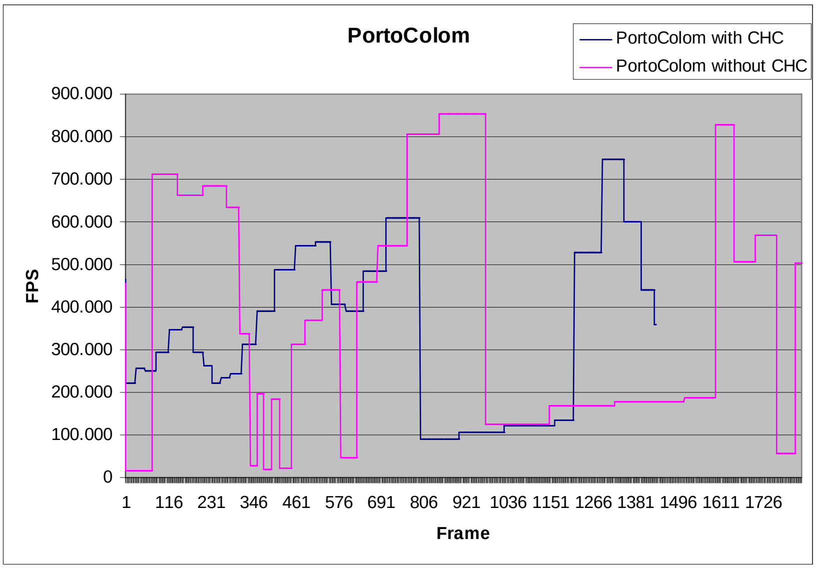
GEDAS has not had any problem integrating the offline visibility module with the other modules for the moment. The offline visibility part is very engine independent, so GEDAS doesn’t expect to have any problems in the future.

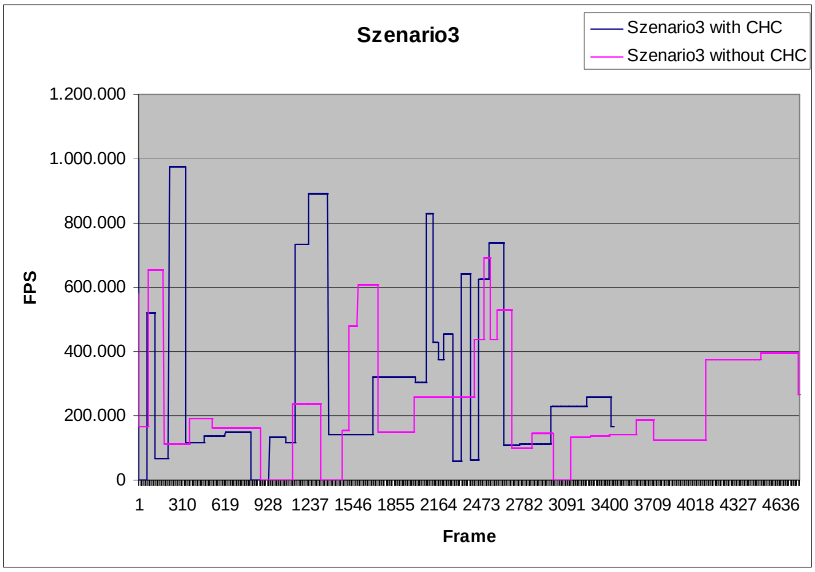
Due to the characteristics of DLE’s demonstrator (character based engine in limited environment) an integration of the visibility work package would not bring any benefit.

3.2. PERFORMANCE

Below some test results are mentioned:







Regarding the deliverable D9.1.1 the test results can be quantified in the following way:

The view-space partitioning module remains untested. However, GEDAS has started to integrate the module in their engine.

The same applies to the PVS computation module.

In the test application of the WP team which provided a sufficient amount of occlusion the frame rate increased by over 50% versus VFC (view frustum culling). In Infowerk's scenes certain framerate drops could be handled more smoothly.

Since none of the testing companies are in hold of a CAVE system there cannot be made a statement about test results.

3.3. INCREASE OF REALISM

Since this WP is fully performance-related the increase of realism is out of the business of this WP. This technique will also help, as any other advanced culling techniques, to use bigger object count scenes. With this technique we will be able to have more number of objects or polygons, so the realism of the scene will increase.

3.4. EASE OF INTEGRATION

Integration of WP3 is quite easy because of the very open SceneManager system the OGRE provides. After compiling the OGRE solution file with the pre-processor flag GTP_VISIBILITY_MODIFIED_OGRE and the changes made in the OGRE rendering system you can go on with integration of the WP in your project. You just have to register your SceneManager plugin per DLL and call Ogre::Root::createSceneManager (const string & TypeName) to plug the OcclusionCullingSceneManager into the OGRE rendering system. Configuring the compilation by setting a pre-processor directive is a little bit inconvenient since you have to recompile every other plug-in concerning vital parts of OGRE since the change in object instance size would result in read-write exceptions in run-time.

4. WP4: GEOMETRY

4.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY

Right now GEDAS is waiting for some final adjustments from UJI to the formula used to calculate which the vertices that have to be simplified are. In the beginning of the project the focus of this simplification was on vertex position, but right now the main focus is on texture coordinates and normals. So in the beginning the simplified objects had a very good simplified geometry but texture coordinates and normals were a bit deformed. Right now textures and normals are practically the same from the original model, but sometimes produce broken objects when the simplification is done. UJI right now is trying to merge the good things from the two formulas. GEDAS is waiting to this final adjustment to generate some benchmarks.

GEDAS has not had any problem integrating this work package with the other ones. As it is only geometry related it works perfectly well with all the other modules.

4.2. PERFORMANCE

Unfortunately we here at InfoWerk weren't able to test the performance in out testing environments, since our meshes were inappropriate to simplify with the GeoTool. The example application worked really well, especially in the LodStrips demo the framerate never dropped below 150 fps.

GEDAS has just recently integrated this work package into their engine. They have no benchmark yet because there have been some little problems with the integration and it has worked just mere weeks ago of the writing of this document. With only the first tests, we can say that for the moment it seems that the raw number of polygons from our scenes will be incremented in more or less 60% without losing any performance.

When DLE analyzed the LodStrips, we observed an excellent frame rate higher than 200 FPS in all tested cases. If DLE compares this performance with their regular engine, they would expect it to bring an increment of the polygon count in an average of 50% to 60%.

Resolution	Worst FPS	Best FPS
1280x1024	175	325
1024x768	214	408
800x600	238	400

These results satisfy the indicators defined in the deliverable D9.1.1 concerning the stripification module and management of about 75% of the whole mesh vertices.

The resulting mesh size is nothing more than 3 times the original model file size, illustrated in the following table:

Model	Original file size	File size after stripification
Athene.mesh	766kB	307kB
Cow.mesh	159kB	181kB
Bunny.mesh	1289kB	2196kB

4.3. INCREASE OF REALISM

Despite this tool is oriented to obtain a higher performance; its use also allows increment the mesh complexity and therefore the level of detail/realism.

Moreover, in the demo DLE hasn’t recognized popping artefacts. It would be interesting to test the tool with meshes with normal maps, to see if the illumination remains equal when changing the LOD level.

The major difference from this technique to ordinary LOD techniques is, that the popping effect with this LOD technique is not as a recognizable as in normal techniques, so the use of LODs will be practically invisible for the user in GEDAS’ opinion.

This technique will also help, as any other LOD technique, to use bigger scenes regarding polygon count. With this technique we will be able to have very big geometric representations of the models when we are near these objects, so the realism of the scene will be greater.

4.4. EASE OF INTEGRATION/USE

GeoTool

The GeoTool is used to generate the triangle strips called LOD technique and to save the thereby generated meshes. Although the GeoTool seemed to work with most of the configurations tested, it sometimes crashed without any obvious reason. Our meshes at InfoWerk were not usable at all, since it is a hard requirement to use manifold meshes only to generate the triangle strips. However, it worked quite smooth with the models deployed by the work package team.

GEDAS has not used the GeoTool package. They have integrated the mesh reduction generation directly into their own editor. It was very easy to integrate into their engine since it was more a work of translation. All the work was about communicating correctly with the library. It was necessary to translate the data to the library structs and translate back the results of the library to the editor classes.

There were many attempts to locate why InfoWerk's meshes were not suitable for triangle strips generation and where the concerning edges are. The test uncovered that most of the meshes were not suitable, however the models which the players drive in were suitable for the LOD generation tool but then not suitable for the demonstrator. The LOD model would not make sense for the player ships since distance to the mesh hardly changes.

Some models which didn't work were sent to the WP team so they can investigate what is wrong in particular with our meshes. In the moment the WP team is working on this investigation.

The LodStrips integration in DLE's engine is complex. At first, it needs porting the GeoTool to their data pipeline. This involves translate their mesh format to another format supported by the tool, and translate again to their proprietary format.

GEDAS is using their own proprietary 3D engine to integrate the GTP libraries. The following paragraph illustrates how the procedure of integrating was done.

For the integration of the library GEDAS included the library 'GTGeometry.lib'. GEDAS uses Visual Studio .NET, and they had to compile the library changing the option under Code Generation -> Runtime Library to MD because they use this option in their project, and if the option is different in the project where you generate the library you can't compile with it.

GEDAS has two options for the simplification, you can generate one LOD with the simplification you want, or you can generate a LOD chain with objects with 30%, 50% and 75% of the geometry of the original model.

In the first step of the simplification model GEDAS has to translate to the data types of the library, so they have to convert all the data they have. They create *Geometry::Mesh*, and fill *Geometry::SubMesh* with all the information. GEDAS uses the type *Geometry::GEO_TRIANGLE_LIST* for the geometry, they don't use triangle strips. Once they filled the *Geometry::VertexBuffer*, they create the simplifier class *Geometry::GeometryBasedSimplifier* with the geometry they just filled and call the *Simplify()* function with the simplification percentage.

When the function ends, they call the *GetSimplificationSequence* of the *Geometry::GeometryBasedSimplifier* class, and obtain *Geometry::MeshSimplificationSequence*. Here we have all the steps needed to achieve the simplification GEDAS desired for their geometry. This class has a *Geometry::MeshSimplificationSequence::Step* member of the type *std::vector*. GEDAS iterates through all the elements of the vector and for each of these steps they search for the eliminated indices, and change these indices with the new one got from the actual step. Once finished this process they search for degenerated triangles, triangles with two or more indices with the same value, and eliminate all these triangles.

All this process, search for the indices inside the array, will be changed once the face indices of the struct (*mFaces* member) begin to work.

Once the LODs are generated GEDAS adds this geometry to their Object geometry class. Their geometry and render class decides which geometry to draw taking into account the size it will have on screen.

5. WP5: ILLUMINATION

5.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY

Matching between requirements and functionality fulfilled many expectations. An important improvement for the work of the WP team is to put the frameworks to work together.

GEDAS has translated the Glow and Tone mapping effect to OpenGL.

GEDAS will not be able to use this environment mapping technique into their engine, because the effect needs to use the alpha channel of the environment mapping for the depth calculation, and GEDAS use this channel for other purposes (alpha blending and more). Before discarding the effect because this alpha channel issue, they disabled transparency, and translated the effect to OpenGL and tried the effect inside their engine. The results were good, but not good enough for the moment to use it above other cubemap techniques. The framerate drop is too important right now (it will probably change with DX10).

Obscurances

This pre-processing tool developed by the UdG, have been very useful in the ambient illumination pass for DLE, adding a important visual improvement in the characters.

Environment Map

This tool allows to illuminate an object using its environment information, with results seemed to the use of global illumination. Moreover, both diffuse and specular components are computed with this tool. Due to the performance cost, actually the tool is basically applicable to one or two objects. This allows use of the tool in games with third person cameras or fighting games.

It would desirable for DLE to have the shaders as optimized as possible, to do a best performance study of the demo. For example, in the most complex shader, DLE reduced the complexity from 1010 assembler instructions to 523.

Glow/Tone mapping

As Depth of Field tool, this is a needed effect in the NextGen games, useful for all type of games.

Pre-computed light paths tool

This technique computes the global illumination term for the static objects in the scene. Moreover, it is possible to translate and rotate the light, and use of whatever type of light.

Is possible use this technique in different types of games, but probably the most interesting types will be indoor games, like the biggest part of FPS or RPG games.

Raytrace effects

This tool solves the problem of the reflection/refraction. It can be very useful for example for racing videogames, since in this type of games the reflections of the vehicles are very important in order to obtain a realistic result. Moreover, the distances between the reflected object and the reflector can be small. (Two cars together, the car with the track, etc…).

Hierarchical Particle Systems

The primary target of this tool is the volumetric cloud generation, which makes it ideal for flight simulators. However, the algorithm also allows smoke generation, which makes it useful for more game types.

5.2. PERFORMANCE

Glow

Since the glow effect is a post-processing effect, it is not such a big issue concerning performance for regular PC/Console gaming resolutions.

With GEDAS experience it is possible to have some issues when you use very big screen resolutions. In the GEDAS reality centre software has to render at resolutions like 2560x1024, and then, effects like HDR and Glow need a lot of pixel power, and usually produce some framerate drops. In fact with the Glow + Tone mapping effect GEDAS has detected that framerate is 20-30% lower. With only the Glow effect the framerate is only 5-10% lower, so GEDAS thinks that the Glow effect has a far better quality/cost ratio than the Glow + Tone mapping effect.

Environment mapping

Environment reflective and refractive mapping, especially in a dynamic manner, is a really performance decreasing approach to put a realistic impression into practice. However the work package team did a really good job in optimizing performance of this render technique. But also the user can optimise performance with reducing texture size of the cube maps and using material LODs, a feature from the OGRE.

GEDAS has also detected that if the cube maps are dynamic, the performance drop is very important. In fact, they are practically unusable if the scene has more than one reflection at a time. With static cube maps things change, but even in this case memory consumption is important (you don't need float cube maps with standard cube map techniques, neither alpha channel).

Obscurances

Since the obscurances are generated as a pre-process, there is no big issue concerning time generation. To use the obscurances when you draw you only have to modify your draw function to use another texture. It could have some performance drops if the scene use more textures than the RAM of the graphic card, so the performance in this case is memory dependent.

Hierarchical Particle Systems

Hierarchical particle systems are quite fast approach to simulate a vast amount of particles. After integrating the module there was hardly a frame rate drop noticeable. Another concern could be implementation of a LOD technique where distance determines parameters for the big and little particle systems.

Spherical Billboards

The spherical billboards worked very well concerning performance ignoring the fact it didn't work with the terrain scene manager of the OGRE. Reading the depth buffer to get depth information of the terrain should not be a major performance issue though.

Depth of Field

This will be a "must have" feature in the NextGen games in DLE's opinion, similarly as Normal Mapping or Shadows are for current games. This effect allows blurring the image as and reproduces human eyes or video cameras behaviour. Moreover, the use of this effect allows designers to focus player's attention in specific screen zones, as done in cinema industry.

This tool works in two phases:

On the first phase, in the render o ZFill pass, this technique adds an overhead of 3 instructions in the pixel shader, but with practically zero cost in the current GPUs with 3.0 shaders.

The second phase is a post process filter like blur. This is a cheap process for current graphics cards and next gens, but difficult to use in older cards.

Resolution	Worst FPS	Best FPS
1280x1024	60	62
1024x768	90	95
800x600	140	150

Environment mapping

Despite of the incredible results of the tool, your cost is relatively high. For this reason it is difficult to apply this technique with the current hardware. However, with a good optimization of the shaders, or for the next graphics cards, this technique will be very useful for main characters and objects.

	Classic Method	Specular	Specular Texel only	Diffuse	Fast Diffuse
800x600	175	46	167	13	75
1024x768	175	35	168	9	60
1280x1024	175	30	168	7	45

Glow/Tone mapping

Due to the lower cost of this tools, is possible its use with graphics cards using shaders 2.0. Probably, the only handicap is the requirement of use floating point render targets. This is a performance problem with older graphics cards, but is less important in current and next generation hardware.

	FPS
800x600	70
1024x768	43
1280x1024	28

Pre-computed light paths tool

Despite of the algorithm cost, it is possible to use in the current hardware generation. In the demo application, the pixel shader is 128 instructions long. This cost is a bit high for the first SM 3.0 cards, but more useful for currents.

	FPS
800x600	58
1024x768	36
1280x1024	30

Raytrace effects

Despite of the complexity of this shader respecting the original algorithm, in our test machine the frame rate applying reflections with 5 iterations never drops to lower than 100PFS.

Hierarchical particle systems

Although in a first look the algorithm seems slow due by the necessity of sorting the particles two times in each render, the use of depth impostors increases the relation between the number of particles and performance.

Resolution	Worst FPS	Best FPS
1280x1024	5	60
1024x768	6	60
800x600	20	60

Relating to the deliverable D9.1.1 the following results can be outlined:

- Using less than 20000 surface elements the environment mapping technique performs at real-time framerates.
- Using less than 20000 surface elements the obscurances module performs at more than interactive framerates.
- Global illumination with stochastic iteration was not tested as it revealed as inappropriate at current graphics cards. Maybe it can be used in the oncoming generation of hardware.
- Photon map based global illumination or pre-computed light paths performs at real-time frame rates with limited inter-object light transfer and less than 20000 surface elements.

- The “Local illumination with physically plausible material models and direct lighting and shadow computation for point and area light sources” (spherical billboards) technique performs at real-time frame rates using less than 20000 surface elements.
- Shadow computation for point and area light sources performs at real-time framerates using less than 20000 surface elements.
- The image based rendering techniques (hierarchical particle systems) performs at above interactive frame rates for scenes with less than 20000 surface elements

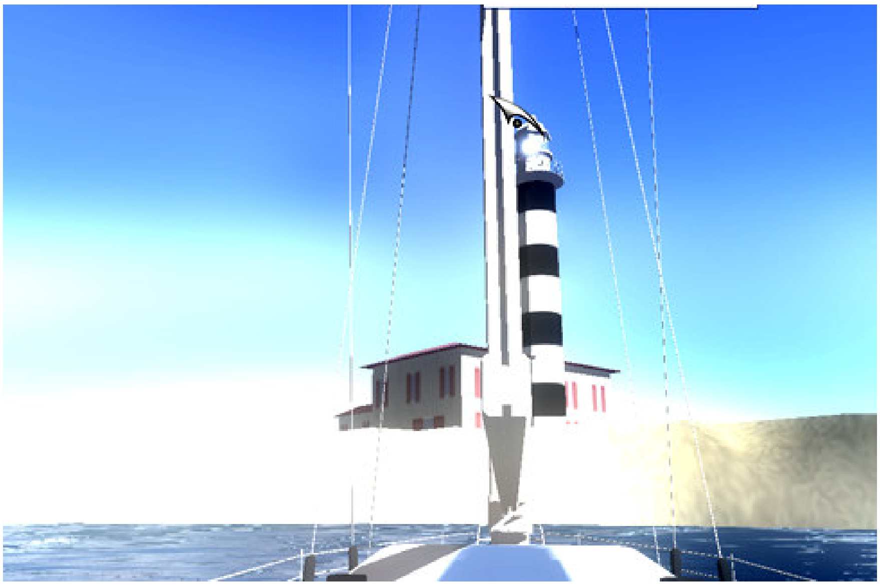
5.3. INCREASE OF REALISM

Below there are a few screenshots of our demonstrator application using some of the effects developed by this WP. By using approximate respectively physical models for calculating the output textures the effects are convincing and the increase of realism can be - depending on the artwork of course – very high.

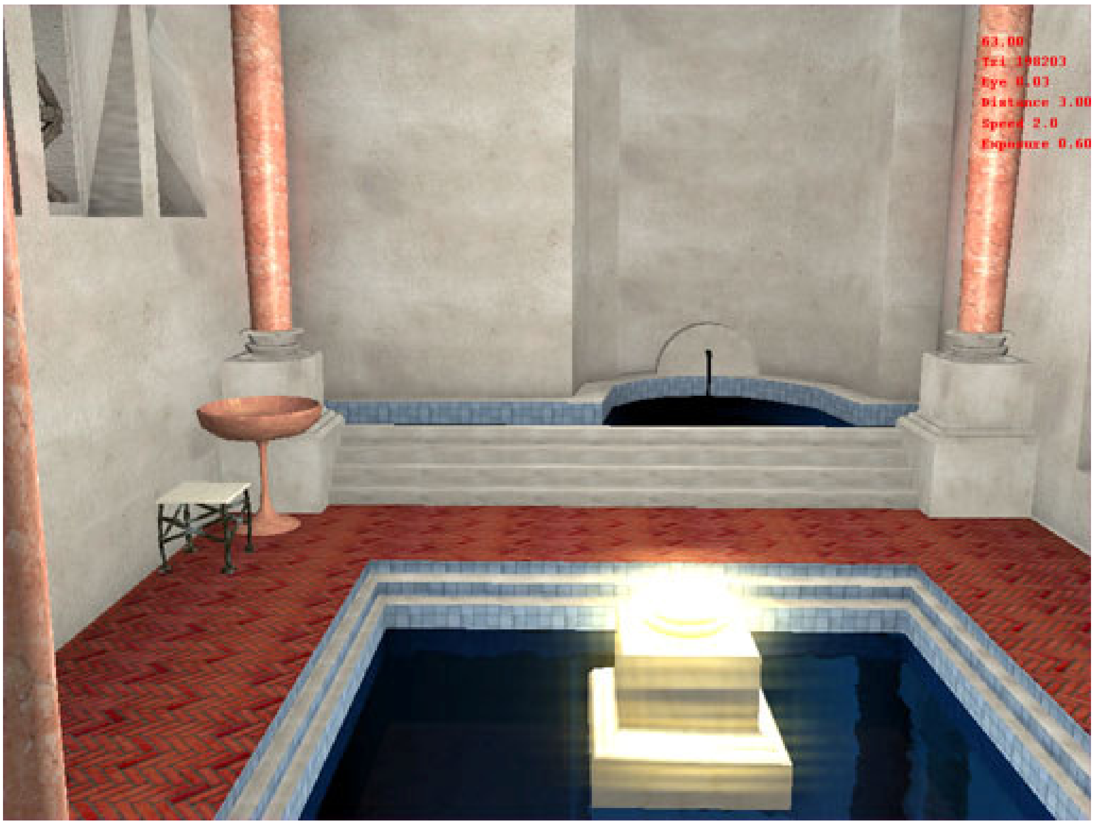
Glow

The glow effect satisfies the demand for “simulating” the reaction of the eye respectively its pupil to light. The effect in WP5 provides a convincing result concerning realism and performance.

Screenshot



Ship scene (Infowerk)

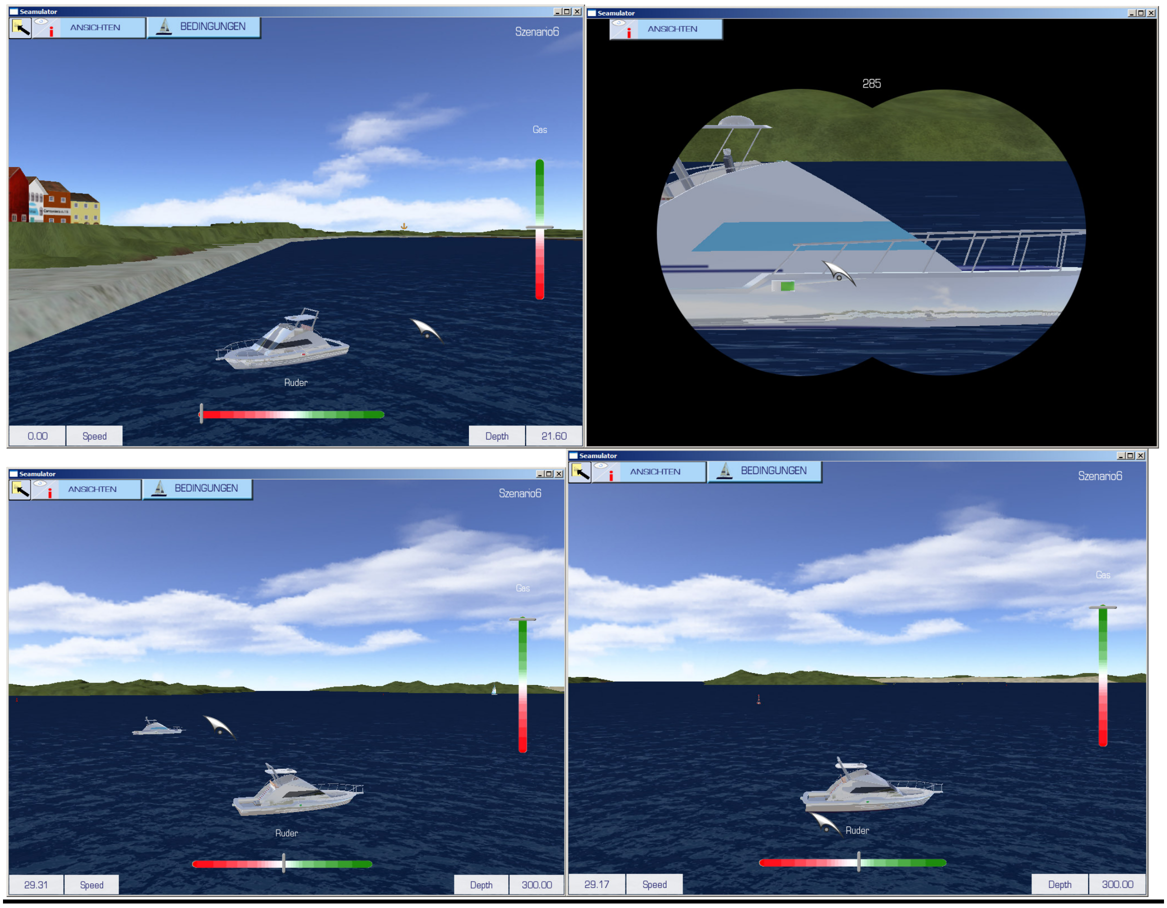


These scenes has glow, but no tonemapping (GEDAS)

Environment mapping

The Environment mapping technique increases realism for reflective and refractive materials, like glass, plain metals and plastic. Since ships and boats in the leisure sector are often made of plain and glossy plastics this technique was quite appropriate for our uses. We had to modify the shader program a little to blend it with the original colour texture. Also a specular map would be usable to define the amount of reflection.

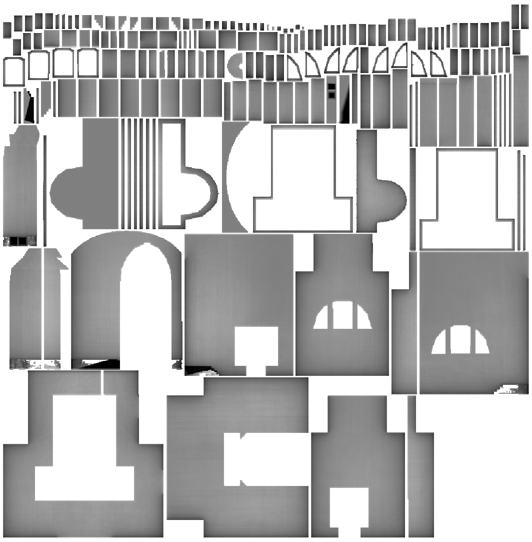
Screenshots

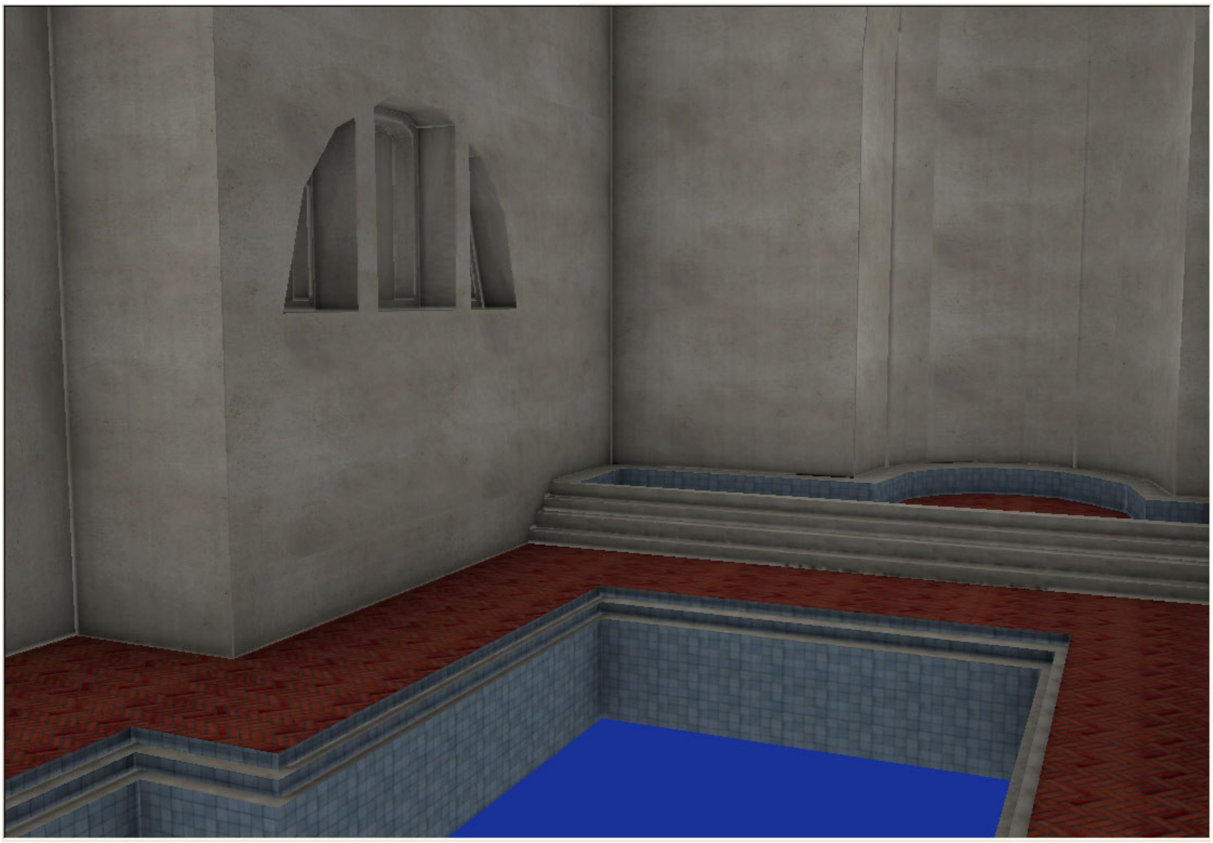


After addressing the problem with the TerrainSceneManager of the OGRE which also concerned depth impostors the look of the environment mapping technique could be improved further.

Obscurances

Obscurances fake the light reflection between all diffuse objects. We use this texture as the ambient element of the scene. Then in a shader we add this contribution to the diffuse component.





In this screenshot we only see the obscurance with the diffuse texture, there isn't any light in the scene.

Tri 367879
Eye 0.03
Distance 3.00
Speed 2.0
Exposure 0.60
92.00

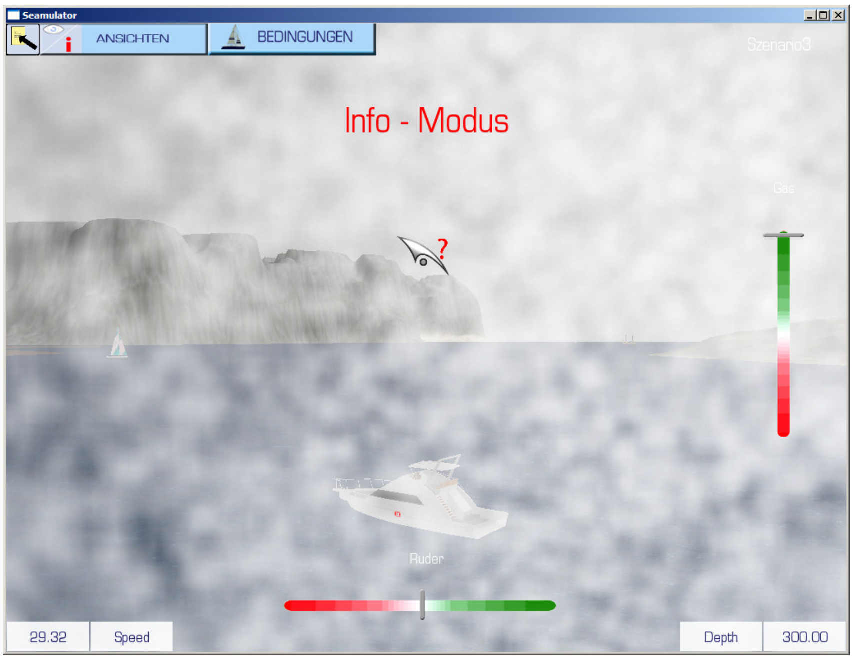
This image has the same scene with some lights.

Hierarchical Particle Systems

Our first attempt for improving the visual appeal of the model used in our demonstrator was integration of hierarchical particle systems since this came first to our mind when investigating the appropriate technique for fog. Hierarchical particle systems are a visual appealing way to simulate a massive amount of particles.

Later we were suggested to use spherical billboards for fog.

Screenshot



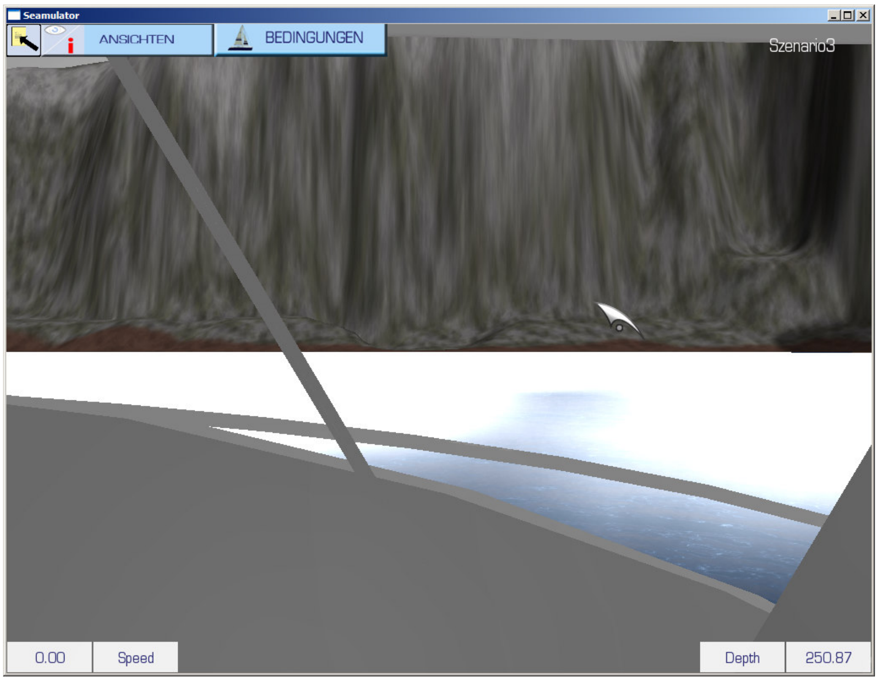
Rain

Because we want to provide a broad variety of situations and sight conditions we tried to implement the rain technique in our application. Because of the issues mentioned later we were not able to integrate it in our demonstrator until now.

Spherical Billboards

Due to the recommendation of the WP team we tried to implement the spherical billboards system for simulating fog in our application. However we encountered some problems concerning the terrain manager of the OGRE.

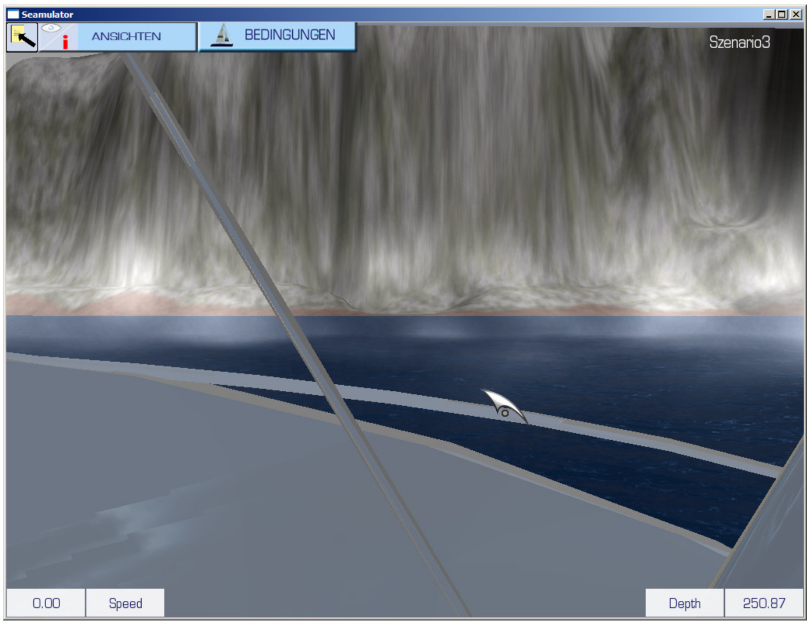
Screenshot



(Note: We didn't create the appropriate texture for fog until now since the problem with terrain rendering occurred earlier and creation wouldn't make sense before proper integration of this feature.)

After reporting the problem with depth information and the TerrainSceneManger the module was changed to work with terrain, which resulted in a much better look of the effect. The technique can be used in our demonstrator application to generate local fog phenomena like sea fog, steam fog, valley fog and ground fog.

Screenshot



(Spherical Billboards using depth information of the depth buffer)

Depth of field

As said previously by DLE, this effect may partially simulate human vision effects, increasing player’s immersion. Crysis™, one of the next technology reference games, use a similar technique.

Environment mapping

The results obtained with this tool provide an amazing result compared with the current ambient illumination from our engine.

Glow

Using glows improves the brightness of the scene, as it affects to surrounding zones.

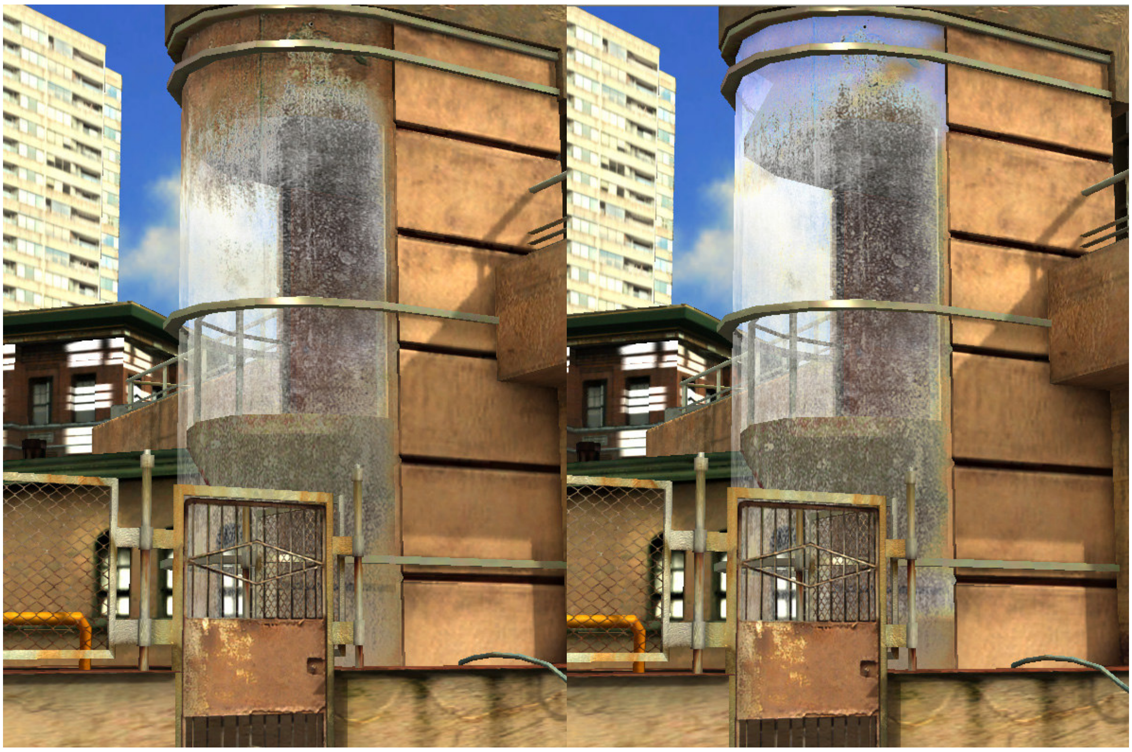
On the other hand, the Tone Mapping tool allows to use higher bright levels in the environments. This is needed to obtain realistic daylight results. Moreover, the technique allows soft changes between bright and dark zones, simulating behaviour of the human eye.

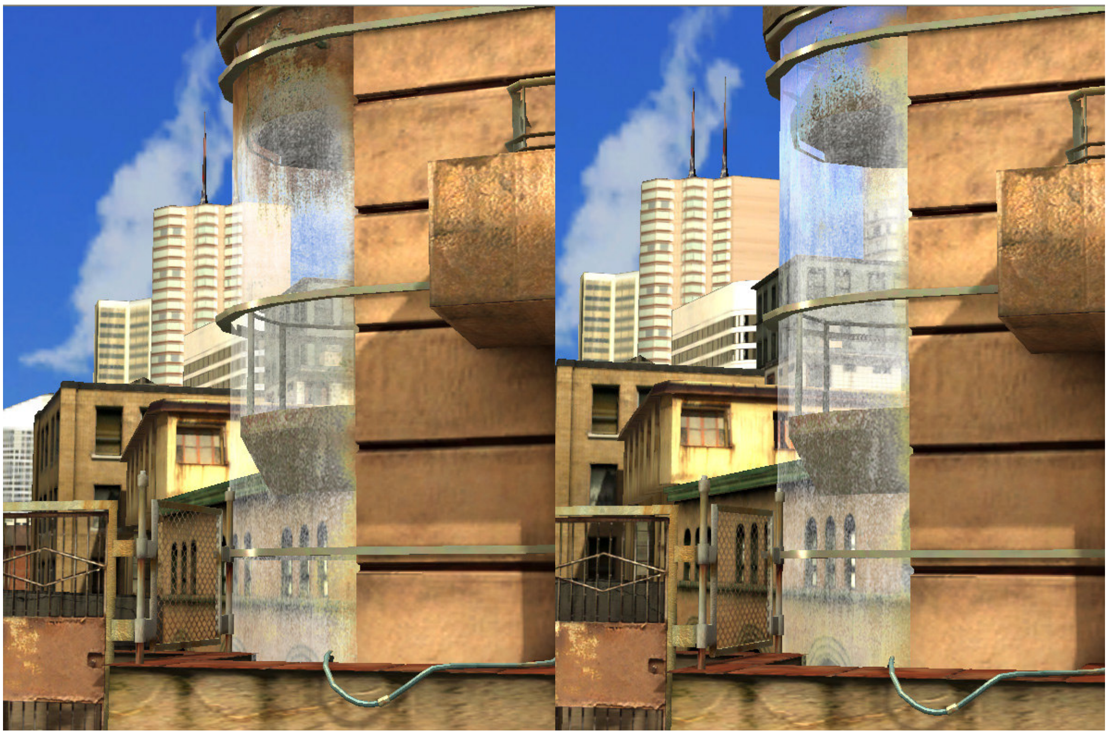
Pre-computed light paths tool

Global illumination is one of the most important goals in the NextGen graphic engines, as said by DLE. The use of this tool improves the quality of render being unnecessary the ambient illumination pass.

Raytrace effects

In many types of materials (metals, polished surfaces, etc.), the reflections are not an option if DLE wanted to reach a minimal degree of realism. This tool provides results closed to work with ray tracing tools. Moreover, since it’s an iterative algorithm, it is possible adjust the complexity based on the user’s hardware.



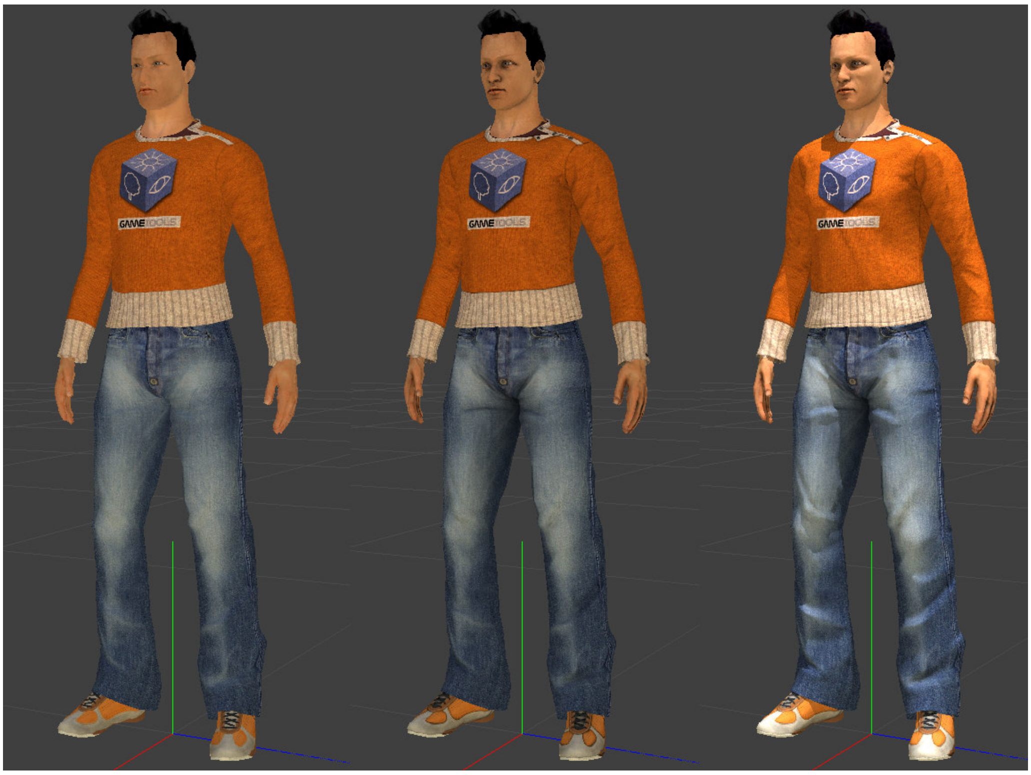


(Differences between surfaces without reflections and using the ray-trace effects: DLE)

Hierarchical particle systems

In flight simulators, realistic clouds are something necessary to obtain a good immersion of the player. Thanks to this tool, we can obtain volumetric clouds probably as well as those used in games like Flight Simulator X®.

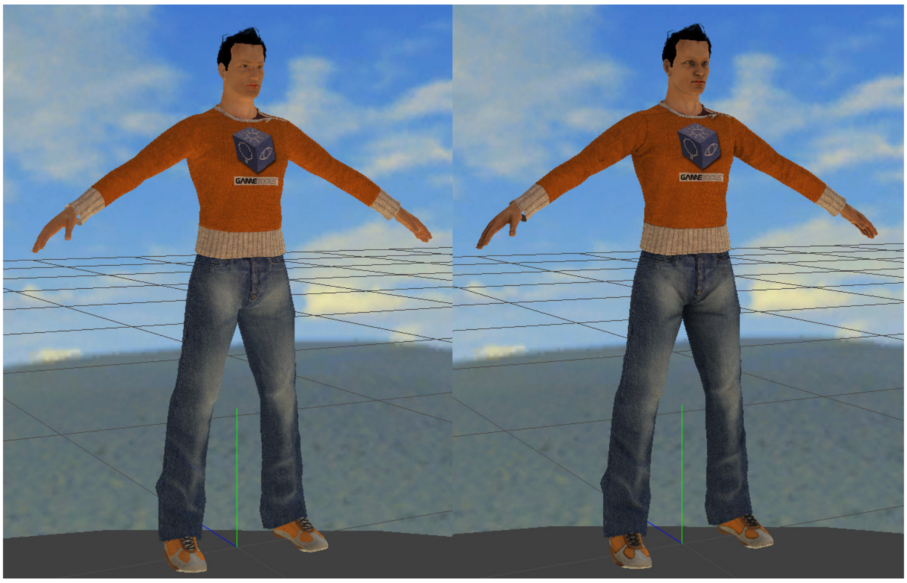
Obscurances



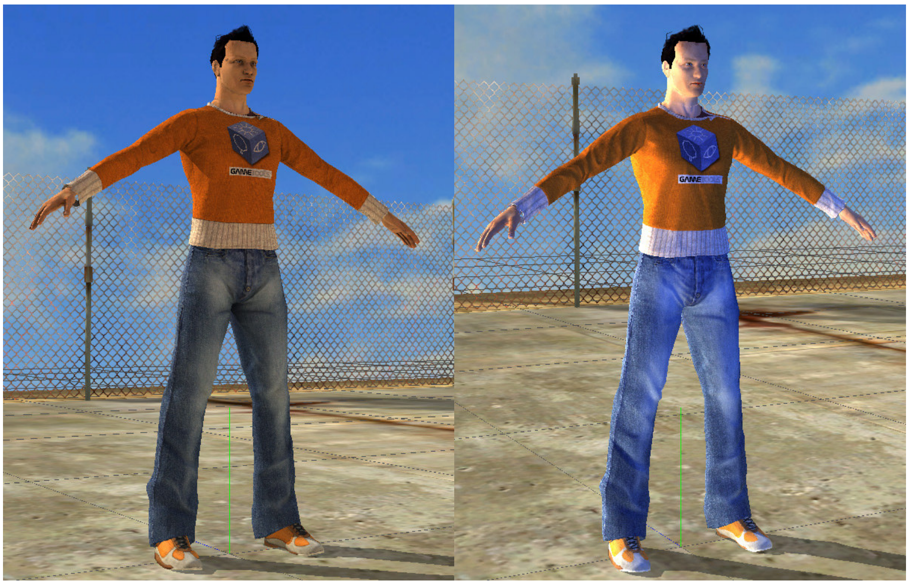
Illumination of the same mesh without any lighting, with obscurances and with obscurances combined with direct lighting (DLE)

The difference in quality when including this tool is very important, since it gives meshes a sensation of greater volume. This is particularly useful in the environmental illumination, where habitually is common to have very flat results. The results improve when using more complex meshes.

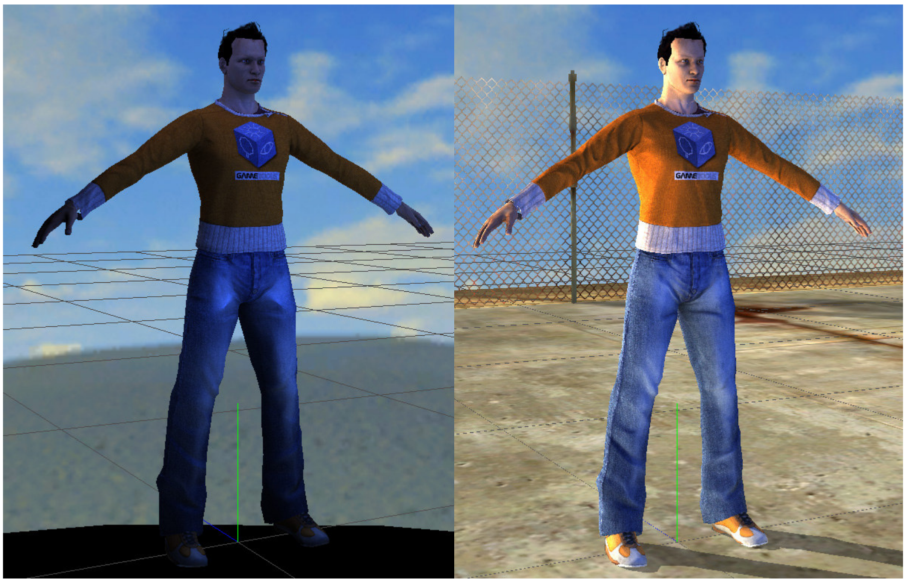
The following screenshots illustrate how the concerning techniques of WP5 can be combined to get realistic results:



(Ambient light, Ambient & Obscurances: DLE)



(Direct lighting & Obscurances, Environment mapping & Direct lighting: DLE)



(Environment mapping & Obscurances, Direct Lighting & Obscurances & Environment mapping: DLE)

5.4. EASE OF INTEGRATION

Since the OGRE has a very flexible material and resource system, it is quite easy to plug new effects into the engine. Compilation of the necessary source files happens like in WP3 with conditional compilation and pre-processor directives. The effects can be easily set up with the .material scripts the OGRE already uses. The glow effect, for example, is a post-processing effect exploiting the OGRE's compositor framework, which are, like material scripts, a convenient method to define effects like bloom, hdr rendering and various filters like an "old tv" effect. The environment mapping technique was also quite ease to integrate since it also uses the common material framework of the OGRE. The shader program needed a few changes to blend the reflective colour with the base texture colour. There was added an "IllumTechniques" section to the material framework.

Unfortunately the illumination work package did not work together with the visibility work package. It would be a hard request to get both work packages to work together.

In the GEDAS engine we had to program a new post-process class to do the glow effect, as the post-process effect file is not as good as the one from OGRE. As we mentioned above we also translated the fx shader to glsl.

The environment mapping technique has a few drawbacks though:

It isn't possible to simulate self-reflection on the regarding surfaces since this would raise the need for an extensive ray tracing model which is not doable because of performance issues. Dividing the mesh in more sub-meshes would be a reasonable solution but results in more depth cube maps which have to be created. That decreases performance also. The solution best suited for us was dividing the control panel of the motorboat (since this was the object which should reflect the roof of the boat) and the motorboat in two separate sub-meshes and applying material depending on point of view. In helicopter view the ship hull reflects the environment and in cockpit view the control panel reflects the ship hull.

Another issue was reflection of the terrain on the ship hull. Here the same problem occurs like in spherical billboards. There cannot be applied a depth material to the terrain since it would require a recreation of the terrain page(s) which is quite slow. So depth information of the terrain cannot be gathered through a depth program using the shader units on the GPU.

We mentioned a problem earlier regarding compatibility of the spherical billboards effect and the terrain scene manager of the OGRE. The visual feedback was that the billboards rendered behind the terrain. Digging through the algorithm used by spherical billboards uncovered a problem concerning the material system the OGRE uses with terrain. There are no frequent material changes possible since certain circumstances in the terrain rendering API. Because of this the terrain page(s) would have to be reloaded every time the material changed. This would result in a heavy bottleneck in the application since height information is read by a greyscale RAW image used to create the vertices for the terrain. After requesting support by one of the OGRE community over IRC, we came to the conclusion that using the depth buffer for this would also provide the depth information needed for the spherical billboards technique. We forwarded this information to the responsible developer of the WP team and were told the development will continue in this direction.

Integration of the rain module was not possible at all. On one hand there is the fact that it is not deployed in the SVN repository on the other hand it didn't work under quite common conditions. We were sent an archive file containing the module with the promise to put it in the repository as soon as possible, but this has not happened until now.

Testing the module was very difficult either since there was no Visual Studio solution file, just a Makefile which is used normally on Linux based systems. There isn't an import feature for Makefiles either in Visual Studio. The only way to go about it was to convert the makefile into a VS solution file by hand, which is not convenient at all. Another thing is that most games are developed for AND in Windows, so deploying a Makefile is not common at all.

A second problem occurred with the demo application itself since it did not work at once. After several testing attempts we concluded that the demo application only worked the first run after changing the render systems from DirectX to OpenGL. Later runs and starting with DirectX did not work at all. We reported these issues to the WP team and were promised to investigate the problems.

For environment mapping integration GEDAS had to disable the alpha blending of their engine, and added support for float cube maps. In the end this technique was discarded because quality didn't had enough added value from other cube map techniques.

After reporting the problem with depth calculation and the TerrainSceneManager the WP team changed some code in the OGRE to get the depth information of the terrain pages. After recompiling both the OGRE and the illumination module the spherical billboards and the environment mapping techniques worked well.

Depth of field

This is a post process tool, so the integration in the engine is quite simple. Moreover, is easy adjusting performance's cost in base to the user's hardware. This is a necessary characteristic to use the technique in a PC game due to the wide rage of hardware to support.

Obscurances

DLE has integrated this tool easily, due to the great support of the UdG. They gave them a GPU-driven and a software version of the tool.

Environment mapping

Integrate this tools is simple, due to most of the source code is HLSL, easily portable to DLE's engine.

Pre-computed light paths tool

The algorithm works in two phases, pre-process of the scene and the render phase. The integration of the pre-process phase is the most complex, since it requires integration with DLE's pipeline data. Integrate the render phase is more easy, since to port the DirectX source code and HLSL to our engine it would not have to be a too complex process.

Raytrace effects

The integration of the tool is simple; on the one hand it is necessary to modify the generation of cube map to save de distance between the object and the scene, and on the other hand modify the pixel shader in the render phase.

Hierarchical particle systems

The fact that the tool this developing in OpenGL, causes that integrating it in our engine is more complex than other WP5 tools, but we have seen that the source code to port is not too long, and the demo attaches documentation, which would make the process more faster.

The following paragraphs illustrate how GEDAS uses their 3D engine for integration of the GTP libraries.

Glow/Tone mapping

GEDAS translated the effect files from HLSL to GLSL to achieve this effect. In their engine they have a PostprocessorManager similar to the one from Ogre, but they also have inheritances for complex specific effects, and in this case concluded it was best to create a specific derivate for this effect. So they use a *CShaderPostprocessGlowManager* inside their render library, and hard-coded all the steps they need for this effect.

This effect needs seven steps. In GEDAS' engine you can do the Glow effect (3 steps) only or the Glow + Tonemapping effect (3 + 4 steps).

The steps are:

- SceneToSceneAvobeOne: Obtain a buffer with all the colour information of the original buffer above one.
  Shader: *AboveOne*
- BlurTheGlow: Blur the buffer with the above one information. This Function is called two times.
  Shader: *GlowBlurH* and *GlowBlurV*
- FinalGlowComposition: Mix the original texture with the blurred one.
  Shader: *GlowText*
- GetLuminance: Calculate the average luminance of the scene.
  Shader: *Luminance*
- Downsampling: Downsample the luminance texture.
  Shader: *ToneBlurV*

- ToneBlur: Remediate the down-sampled luminance texture.
  Shader: *ToneBlurH* and *ToneBlurV*
- FinalCompostion: Use the blurred down-sampled luminance texture to tone map the result from the FinalGlowComposition.
  Shader: *FinalTone*

Ray-traced Environment mapping

To achieve this effect GEDAS had to use four channel float cube maps, so they have to change the *GL_UNSIGNED_BYTE* format to *GL_HALF_FLOAT_ARB* and *GL_RGB* to *GL_RGBA16F_ARB*. They have a *CubeMapManager* were they create their cube maps, so it was easy to change the format.

The next step was to change the shader were GEDAS used that cube map, the reflection shader, and most important, we had to change all our shaders, because the effect needs to store all the depth information of the scene in the alpha channel of the cube map when the cube map is created. This was one of the big issues of the implementation of this effect.

GEDAS uses the same pipeline when they render a cube map and when they render the final scene, so they use the same shaders. GEDAS has more than fifty shaders in their engine, and in some of them they use the alpha channel, so this was a very big problem.

To evaluate the quality of the effect, GEDAS first changed only two different shaders. They also had to deactivate the Alpha test and the alpha blending, the other big issue of this effect.

Obscurances

GEDAS included inside their editor the *ObscuranceMap* class of the effect. As a prerequisite, all the geometry that is going to generate an obscurance map needs two texture coordinate arrays, one for the diffuse, normal map, etc. texture and one for the obscurance map. This second coordinate array has some restrictions, all the coordinates have to be inside the 0, 1 range, and two different vertices can't share any texture coordinate.

In this effect GEDAS also had to translate their data types to the ones needed by the class, this was done inside the *InitRenderData* and *InitGeometry* functions. This effect has all the shaders hard-coded inside this class and in it is all implemented with OpenGL, so they didn't need to adapt anything regarding shaders. In the end when the texture was generated they only had to convert it to GL_UNSIGNED_BYTE format and assign it to their *CMaterial* class, and of course save the texture to disk, to use it in their VRViewer.