

GAMETOOLS

ADVANCED TOOLS FOR DEVELOPING HIGHLY REALISTIC COMPUTER GAMES

TEST REPORT ON RELEASE PLUG-INS

Document
identifier:

**GameTools-6-D6.2.2-02-0-1-
Project integration,
evaluation and test**

Date: (use "update field" Word
function, right mouse button)

10/04/2007

Work package:

**WP06: Project Integration,
evaluation and test**
(e.g.: WP6: Integration, Test)

Partner(s):

All

Leading Partner:

INFOWERK

Document status:

APPROVED

Deliverable
identifier:

D6.2.2

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.



PLUG-INS

Doc. Identifier:
TGameTools-6-D6.2.2-02-0-
1-Project integration,
evaluation and testTTT

Date: 10/04/2007

Delivery Slip

	Name	Partner	Date	Signature
From	Martin Kolb	Infowerk	04.04.2007	
Reviewed by	Moderator reviewers and	ALL		
Approved by	Moderator reviewers and	ALL		

Document Log

Issue	Date	Comment	Author
0-0	15.03.2007	First draft	Martin Kolb
0-1	04.04.2007	Final version	Martin Kolb
0-2			
0-3			

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files / URL
Word	gametools-6-d6.2.2-02-2-0-project integration, evaluation and test.doc (use "update field" Word function)



CONTENT

DELIVERY SLIP..... 2

DOCUMENT LOG 2

DOCUMENT CHANGE RECORD..... 2

FILES 2

1. INTRODUCTION 5

1.1. OBJECTIVES OF THIS DOCUMENT..... 5

1.2. APPLICATION AREA..... 5

1.3. TERMINOLOGY 6

2. EXECUTIVE SUMMARY:..... 7

3. WP3: VISIBILITY 9

3.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY 9

3.2. PERFORMANCE 10

3.3. INCREASE OF REALISM..... 13

3.4. EASE OF INTEGRATION 14

4. WP4: GEOMETRY 15

4.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY 15

4.2. PERFORMANCE 15

4.3. INCREASE OF REALISM..... 16

4.4. EASE OF INTEGRATION/USE 17

GeoTool..... 17

5. WP5: ILLUMINATION 20

5.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY 20

Obscurances..... 20

Environment Map 20

Glow/Tone mapping 20

Pre-computed light paths tool..... 20

Raytrace effects..... 21

Hierarchical Particle Systems..... 21

5.2. PERFORMANCE 21

Glow..... 21

Environment mapping..... 21

Obscurances..... 22

Hierarchical Particle Systems..... 22

Spherical Billboards..... 22

Environment mapping..... 22

Pre-computed light paths tool..... 23

Hierarchical particle systems..... 23

5.3. INCREASE OF REALISM..... 26

Glow..... 26

Screenshot..... 26

Environment mapping..... 27



PLUG-INS

Doc. Identifier:
TGameTools-6-D6.2.2-02-0-
1-Project integration,
evaluation and testTTT

Date: 10/04/2007

Screenshots.....	28
Obscurances.....	28
Hierarchical Particle Systems.....	30
Screenshot.....	31
Spherical Billboards.....	31
Screenshot.....	32
Screenshot.....	33
Depth of field.....	33
Glow.....	33
Pre-computed light paths tool.....	33
Raytrace effects.....	34
Hierarchical particle systems.....	35
Obscurances.....	36
5.4. EASE OF INTEGRATION.....	38
Obscurances.....	40
Environment mapping.....	40
Pre-computed light paths tool.....	40
Raytrace effects.....	40
Hierarchical particle systems.....	40
Glow/Tone mapping.....	41
Ray-traced Environment mapping.....	41
Obscurances.....	42



1. INTRODUCTION

1.1. OBJECTIVES OF THIS DOCUMENT

This document is the final 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, 2GB RAM

GeForce 7900 GS with 256MB

The testing environment at GEDAS Iberia (a member of 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 very well on their own concerning performance and/or increase of realism. Therefore they can be very useful for the game developing companies. Earlier it was rather complex to get the work packages written for the same graphics API to run together. This requirement was recommended and was completed by the WP teams on time.

Another problem was related to integration into the two main APIs in 3d graphics programming, namely OpenGL and DirectX. Both APIs are used in industrial test sites. Some of the effects were only implemented for one API, probably due to some characteristics of the APIs themselves, because they two differ slightly in some aspects. Nevertheless, the end results of the project are highly sophisticated modules, which are usable in a wide range of applications illustrated below.

This table covers the application possibilities for each tested tool.

	Third person	First person	Driving	Sports	Flight simulators	Fighting
CHC	Essential	Essential	Useful	Useless	Not very useful	Useless
LODStrips	Essential	Essential	Essential	Useful	Essential	Not very useful
Depth of Field	Useful	Essential	Useful	Useful	Useful	Useful
Environment maps	Essential	Not very useful	Useful	Not very useful	Useful	Essential
Glow	Essential	Essential	Essential	Essential	Essential	Essential
Tone mapping	Essential	Essential	Essential	Not very useful	Not very useful	Not very useful
Reflection	Useful	Useful	Essential	Useful	Essential	Useful
Refraction	Not very useful	Not very useful	Useful	Not very useful	Not very useful	Not very useful
Caustics	Not very useful	Not very useful				
Pre-computed light paths	Essential	Essential	Not very useful	Not very useful	Not very useful	Useful
Spherical billboards	Essential	Essential	Essential	Useful	Essential	Useful
Hierarchical particle systems	Not very useful	Not very useful	Not very useful	Not very useful	Essential	Not very useful

- Essential
- Useful
- Not very useful
- Useless



PLUG-INS

Doc. Identifier:
TGameTools-6-D6.2.2-02-0-
1-Project integration,
evaluation and testTTT

Date: 10/04/2007

Having tested all the available modules it comes clear, that many of the effects and algorithms can just be used together, since some of the effects require computation power which is gained with simplification algorithms and culling techniques.

The quality of the developed plug-ins is consistently high; the tests couldn't detect any software errors in the released modules.

Practical application uncovered very sophisticated algorithms and effects which can be used in today's high end video games and simulations. This keeps the European game and 3d graphics community competitive while keeping additional costs low.

Especially small and medium enterprises don't have the financial power to develop similar or equal rendering techniques since it would require more time and personal assignment. That way those SMEs can concentrate on creating the content, which is as important as the used software elements which use that content. Content creation is also considered one of the most time consuming stages in 3d software development.

With the modules developed the effort can be put on content creation again, enabling the companies to keep the quality of their applications high while providing visual appealing virtual worlds with rich detail.



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. Short before end there was an approach developed which could handle scenes which consist of very few occluders. Incompatibilities between this WP and the Illumination WP were recovered after request.

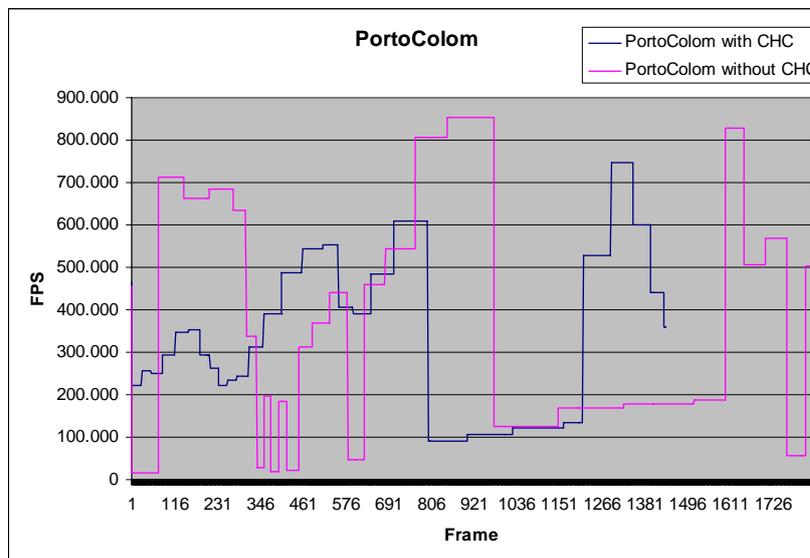
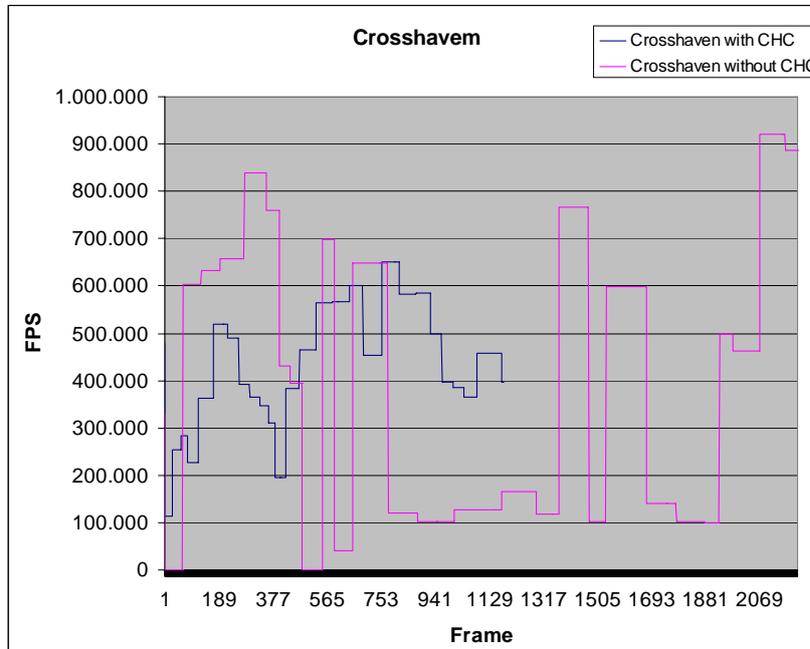
GEDAS has begun the integration of the offline visibility module into their engine. They now export Alias/Wavefront .OBJ files from their editor, and use these files with the stand alone application provided by Vienna to generate the Visibility Cells and the PVS.

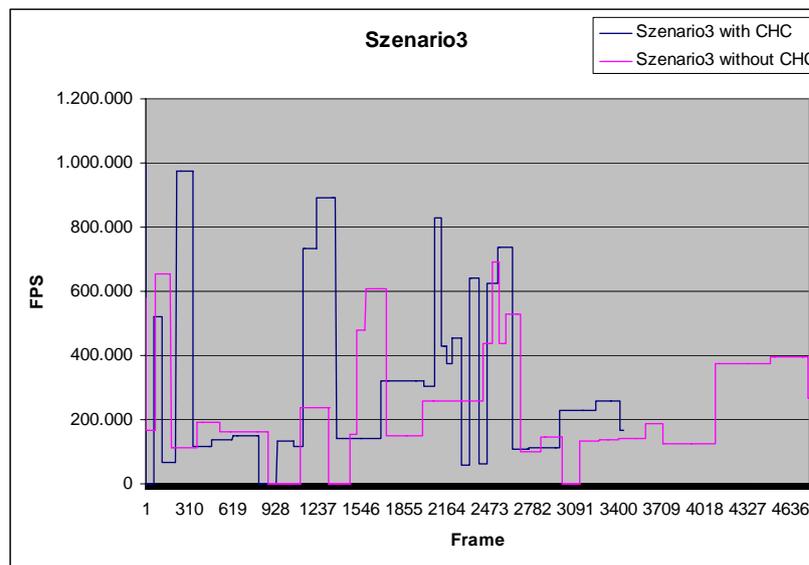
GEDAS has not had any problem integrating the offline visibility module with the other modules for the moment. The offline visibility part is very engine independent, so GEDAS doesn't expect to have any problems in the future.

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 frame rate 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.

GEDAS also integrated Online Visibility Culling and can give some informal results to their tested scenes with and without CHC:

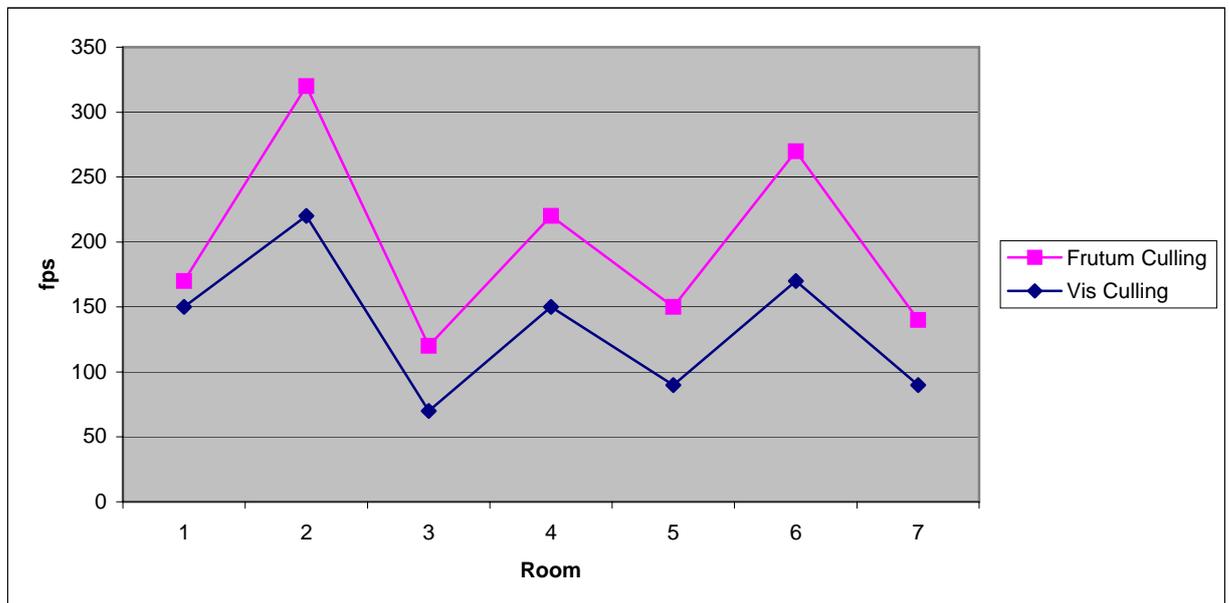


PLUG-INS

Amphitheatre:

Amphitheatre					
	Vis Culling	Frustum Culling			
Entrance	150	20			
Trap Room	220	100			
Armory	70	50			
Bridge	150	70			
Throne Room	90	60			
Amphitheatre	170	100	Min FPS	20	100
Amphitheatre	90	50	Max FPS	70	220

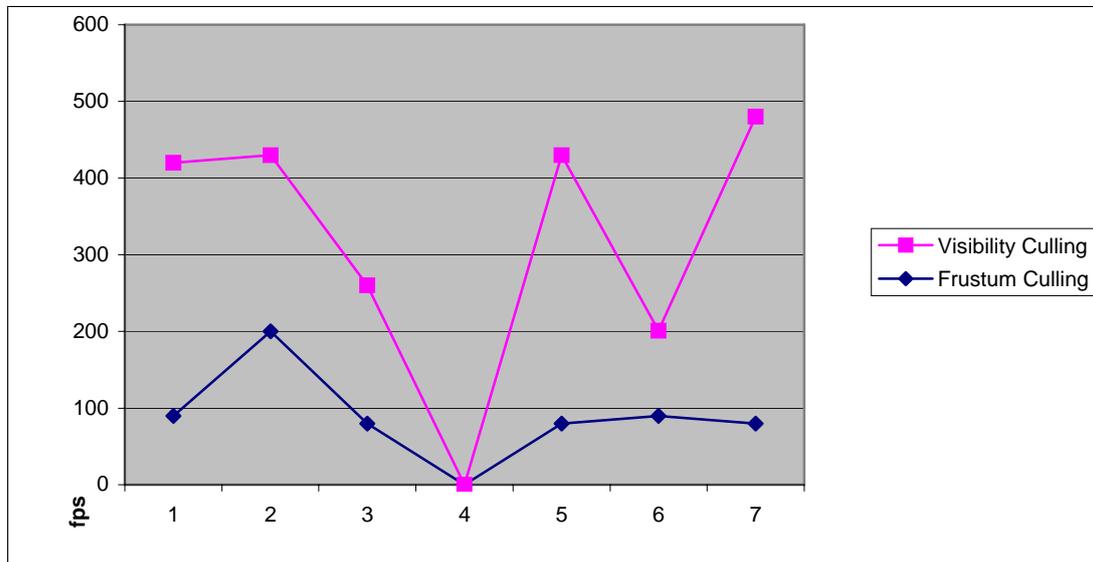
Graph:



Hospital:

HOSPITAL					
	Vis Culling	Frustum Culling			
Entrance	330	90			
Corridor	230	200			
ICU	180	80			
First floor					
Corridor	350	80			
ICU	111	90	Min FPS	80	111
Laboratory	400	80	Max FPS	200	400

Graph:



The WP Team announced a slightly adapted version of the CHC scene manager for more open environments, the Random Update Culling scene manager (short RUC) which should provide better functionality in open environments like in Infowerk's demonstrator.

Below some performance are results mentioned:

Scenario	Frame rate without CHC	Frame rate with CHC	Frame rate with RUC
Pula	55	65	69
Scenario 4	50	70	75

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 one can use the developed classes by the WP team. Register the 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 it is necessary 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

In the beginning the models were simplified regarding their vertex positions. So the simplified objects had a very good simplified geometry but texture coordinates and normals were a bit deformed. Later on textures and normals were practically the same from the original model, but sometimes produce broken objects when the simplification is done. After request by GEDAS there were found solutions which decide which triangle strips can be simplified without destroying the original look of the model.

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 InfoWerk wasn't able to test the performance in their testing environments, since their 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.

When DLE analyzed the LodStrips, they 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

More test results can be seen at GEDAS' results of testing the Illumination package. There was also the Geometry WP used and satisfied the need for higher performance - because of the quite costly effects in the Illumination WP - ideally. There are also some performance graphs drawn.

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. The Mesh simplification process is outlined in the documentation and is noted now shortly:

1. Load the mesh in .obj (Wavefront) or .mesh (OGRE) format.
2. Generate LodStrips in the menu of the same name and choosing the amount of simplified polygons.
3. Build the LodStrips and save them on hard disk.
4. Visualize LodStrips for control of the look of the model.

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.

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 the library is generated, the compilation with it is not possible.



GAMETOOLS

PLUG-INS

Doc. Identifier:
TGameTools-6-D6.2.2-02-0-
1-Project integration,
evaluation and testTTT

Date: 10/04/2007

GEDAS has two options for the simplification: to generate one LOD with the required simplification, or to 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.

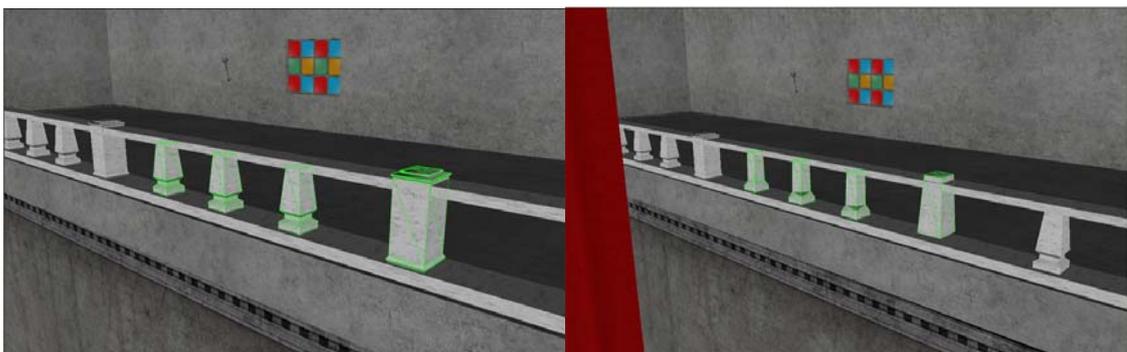
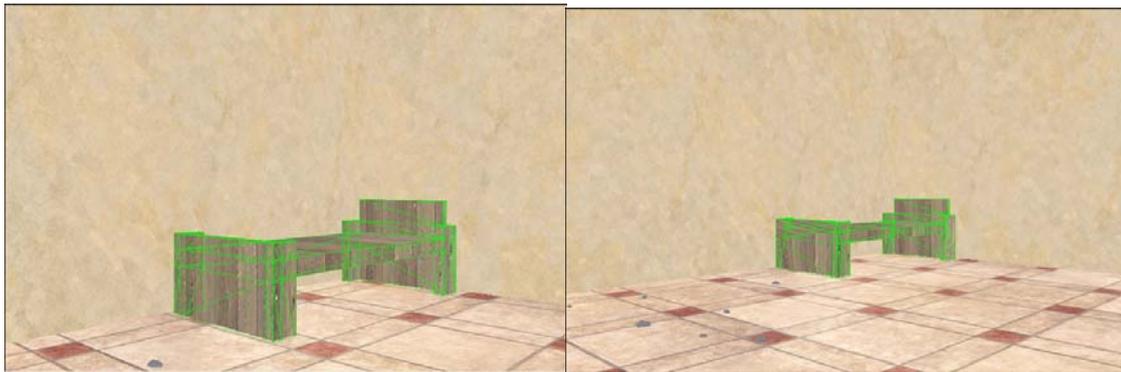
Once the LODs are generated GEDAS adds this geometry to their Object geometry class. Their geometry and render class decides which geometry to draw taking into account the size it will have on screen.

GEDAS has managed to integrate the LOD technique into their engine.

The performance benefit and the visual appealing way how polygons are reduced and increased are necessary to provide the special effects of the Illumination package.

The following screenshots show how polygons are reduced proportional to the viewing distance of the mesh.

Amphitheatre:



(The above screenshots show viewpoint-based mesh simplification reducing polygons to render)



5. WP5: ILLUMINATION

5.1. MATCHING BETWEEN REQUIREMENTS AND FUNCTIONALITY

Matching between requirements and functionality fulfilled many expectations. 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 and GEDAS, adding an 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 be 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.



Ray-trace 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 very big screen resolutions are used. 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 frame rate drops. In fact with the Glow + Tone mapping effect GEDAS has detected that frame rate is 20-30% lower. With only the Glow effect the frame rate 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 by 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 (float cube maps with standard cube map techniques, neither alpha channel are needed).

Obscurances

Since the obscurances are generated as a pre-process, there is no big issue concerning time generation. To use the obscurances when drawing it is only needed to modify the draw function in order 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 is not a major performance issue though.

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



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

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 perform 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
-
-

Raytrace Effects:

Despite of the complexity of this shader respect with the original algorithm, in our test machine the frame rate applying reflections with 5 iterations never drops of 100PFS. Furthermore, use this tool requires computes depth and colour. To do this, we have many options to do this:

Set as render target 0 a RGB8 or RGBA8 cubemap, and a R32F cubemap as render target 1. This is a good performance option, but the antialiasing is disabled.

Set as render target 0 a RGB8 or RGBA8 cubemap, and a floating point cubemap as depth buffer. It is probably the best option, but it is only compatible with Nvidia cards.

Use a RGBA16F as render target, and use alpha to store depth. We use this option because we need a high dynamic range, it means, a floating point target.

Glow/Tone mapping:

Due to the lower cost of this tool, it is possible to use it with graphics cards with shaders 2.0 or greater. Probably, the only handicap is the requirement of use render targets in floating point. This is a performance problem in old graphics cards, but is less important in current and next generation. Nevertheless, is possible avoid the use of a floating point render target.

To do this, we can encode the output colour in RGBE format, very used to store HDR images. Basically this format encodes the luminance intensity in the alpha channel.

However, this solution has two problems. First of all, if we want use alpha blending, we need render in two passes. Moreover, the pixel shader performance is lower due to the cost of encode to RGBE, and decode from this format to RGB.



Performance of the tool with DLEs hardware

	FPS
800x600	70
1024x768	43
1280x1024	28

Performance in DLEs demonstrator

	No effect	Glow	ToneMapping	Both
1024x768	183	105	90	88

Obscurances:

This is a pre-process tool. For this reason, your use in real time has practically a zero cost. It only adds a texture fech and a multiplication in the pixel shader, moreover, the texture fech can be removed placing the obscurances information into alpha channel of the color texture.

Depth of Field:

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, but it needs to use a multiple render target, that disable antialiasing filters. To solve this problem, we have different options. We can store blur quantity in the alpha channel, but this need two passes for alpha objects, if only need object alpha, or two passes for all objects if we need the frame buffer alpha. Another option is to render the depth buffer to texture, and use it. In our application, we need to do a second pass for all objects to compute depth. Nevertheless, we expect to solve this problem to minimize the number of drawprimitive calls and improve the performance.

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.

To improve blur quality, we apply a Gaussian Blur to a downsampled texture, and for each tap of the DepthOfField filter, modulate between original color texture and blurred texture.

Pixel shader instructions in DOF shader

GT Tool: 60

DLEs technique: 93

Resolution	No DOF	GT Tool	DLEs technique
1051x902	185	73	48

5.3. INCREASE OF REALISM

Below there are a few screenshots of our demonstrator application using some of the effects developed by this work package. 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)

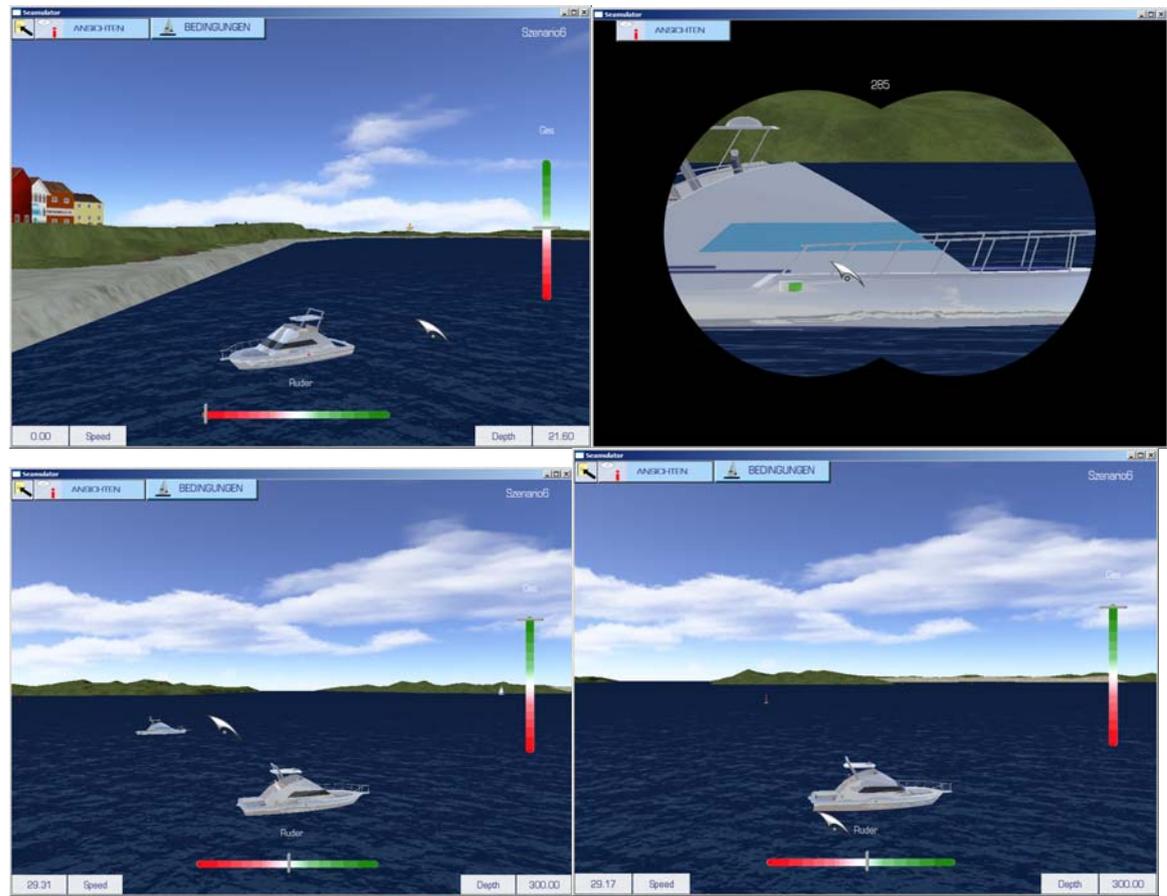


These scenes has glow, but no tone mapping (GEDAS)

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 obscurity with the diffuse texture, there isn't any light in the scene.



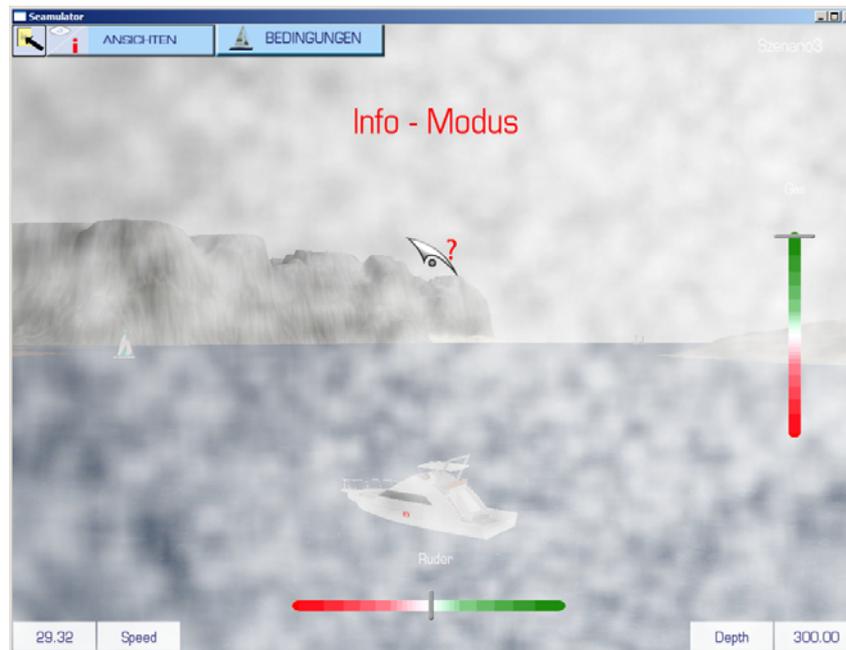
This image has the same scene with some lights.

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 usable for atmospheric effects like clouds. It can be highly recommended in real day light simulated environments.

Later we were suggested to use spherical billboards for fog.

Screenshot



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. Later it was uncovered, that the effect couldn't be applied because the terrain system in OGRE works a bit different than expected. To maintain compatibility the terrain material cannot be changed during run time but would require to rebuild the terrain again which is unacceptable for real-time applications.

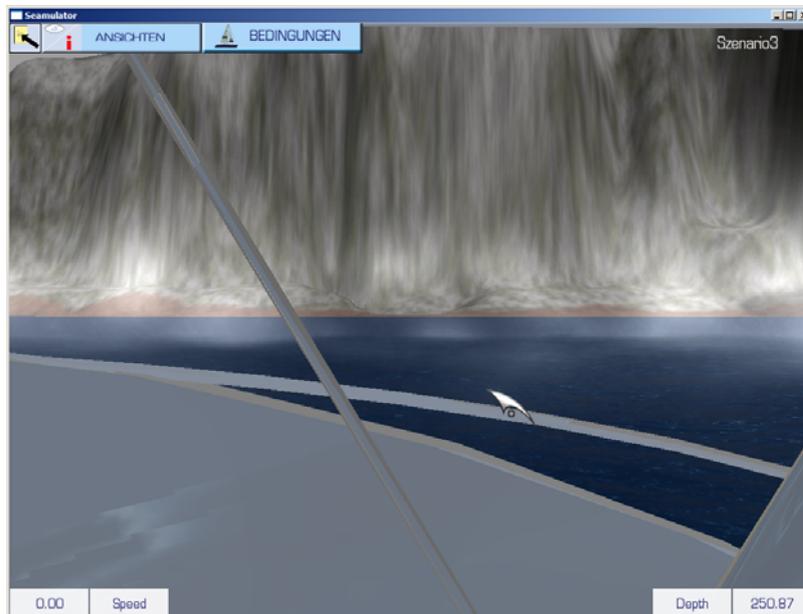
Screenshot



Spherical billboard effect without depth information of the terrain

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.

Glow

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

On the other hand, the Tone Mapping tool allows using 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.

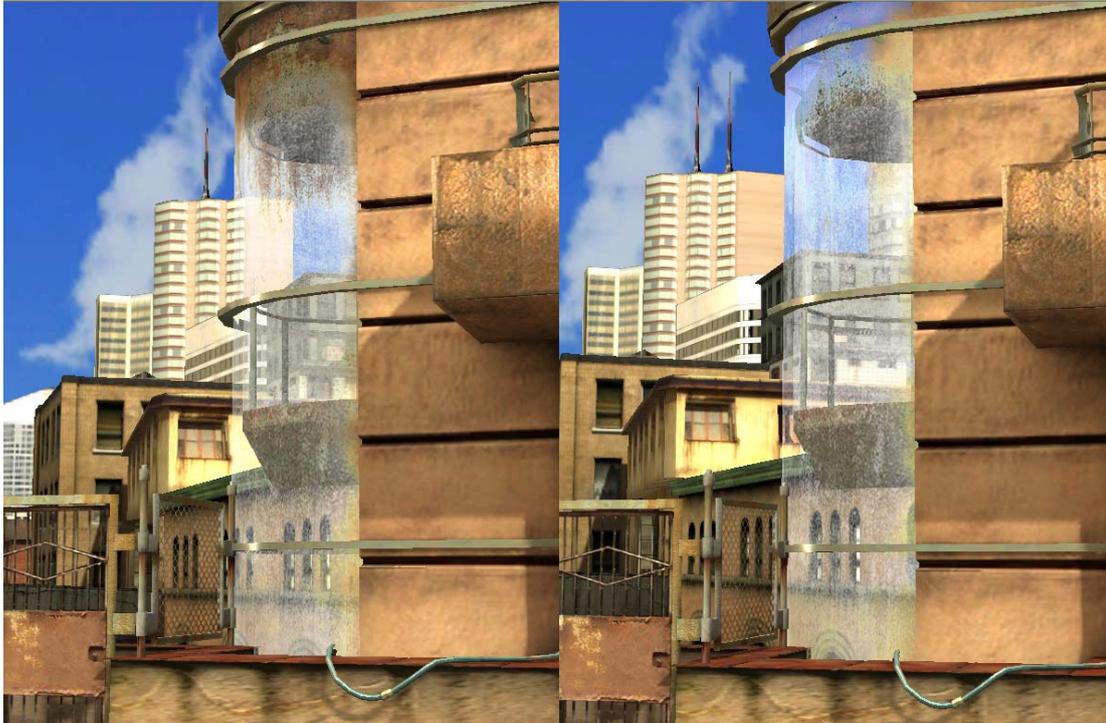
Pre-computed light paths tool

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, it can provide 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.

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 would be 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 would then reflect the environment and in cockpit view the control panel would reflect the ship hull.

Another issue was reflection of the terrain on the ship hull. Here the same problem occurred 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. Later a solution was found and the effects using depth information are now working properly.

Rain module:

Testing the rain 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. However Infowerk managed to convert the solution file and sent it back to the WP team.



There could be made some tests. The performance is very good providing a highly sophisticated and realistic demonstration of certain weather effects like rain or snow. The effects can also be scaled depending on which hardware the effect is executed. There are more detail rich approaches with the scene refracted in a rain drop and more performance friendly techniques like streak rendering of the rain drops. There can also be applied a force field since the particle effects are all GPU based.

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.

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.

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.



GAMETOOLS

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 it is possible to do the Glow effect (3 steps) only or the Glow + Tonemapping effect (3 + 4 steps).

The steps are:

- SceneToSceneAboveOne: 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*
- FinalComposition: 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.

Finally, the two WP groups of Visibility and Illumination managed to integrate their two techniques together in the OGRE. Using both techniques will be vital if it the integration of several techniques of the Illumination WP is needed. The frame rate gain of the Visibility plug-in complements the performance intensive effects of the Illumination plug-in ideally.

Actually many of the effects developed in the Illumination package would not be applicable without improving performance through the Visibility plug-in.

The following Screenshots are made by GEDAS and use several techniques of the GTP (see picture underwritings for more detail):

Hospital:



(Left: ICU, Right: Entrance)

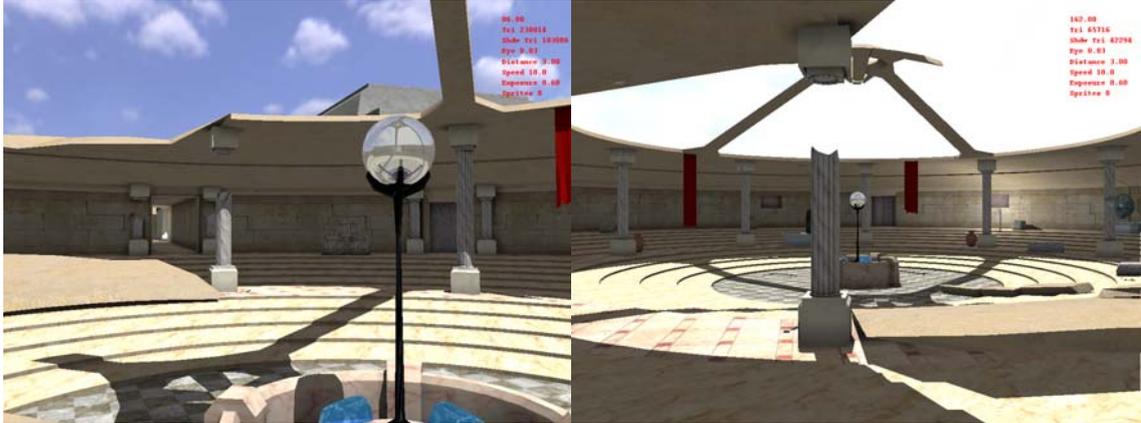


(Left: Laboratory, Right: ICU on the 1st floor)



(Left: Corridor, Right: Corridor on the 1st floor)

Amphitheatre:



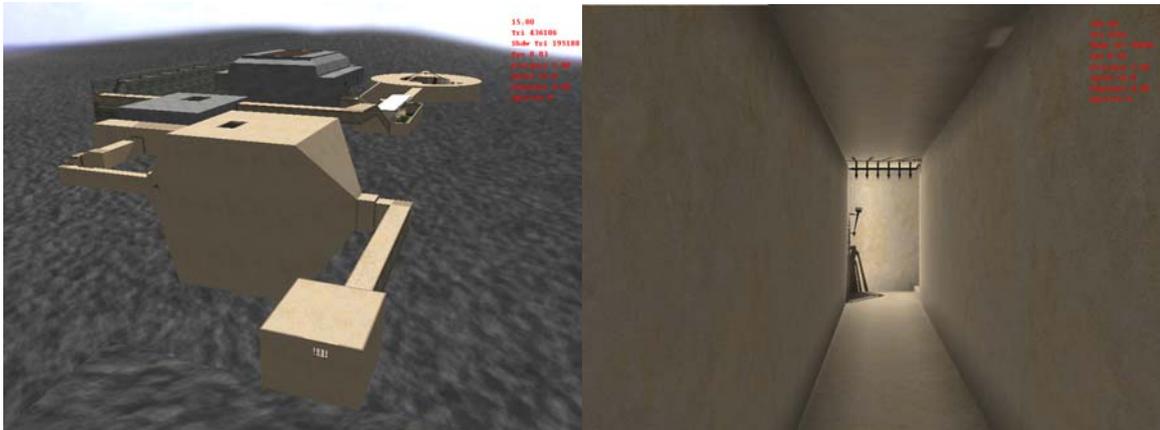
(Left & Right: Amphitheatre)



(Left & Right: Armory)



(Left: Armory entrance using obscurances, Right: Whole scene from extern viewpoint)



(Left: Whole scene from extern viewpoint, Right: Corridor using obscurances)



(Left: Entrance using obscurances, Right: Throne room without HDR)



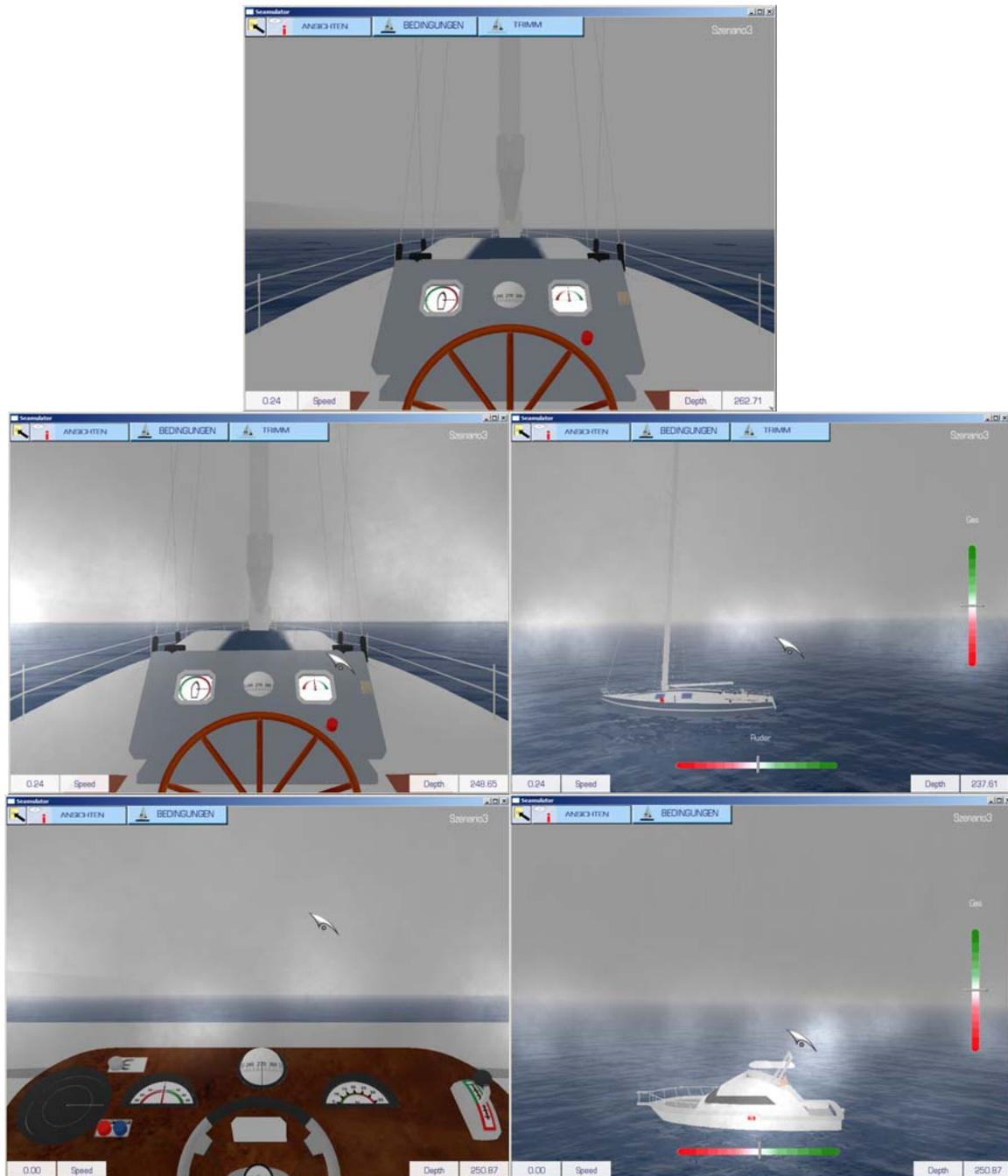
(Left: Throne room with HDR, Right: God rays)

After getting the Visibility and Illumination WP to work together, InfoWerk also moved on integrating the effects of the Illumination package.

Below some Screenshots are illustrating the gain in visual quality compared to the not GTP enhanced version.

Spherical Billboards:

Spherical Billboards are used to support the classic fog techniques (fixed-function) with a more 3-dimensional immersive fog effect.



(First picture: standard (fixed-function) fog, afterimages are using fixed-function supported by the spherical billboards particle effect)

Infowerk had to adapt the shader slightly to give the ordinary smoke particle effect a more “foggish” impression. The particles fade out at the near and fade in at farther distances.

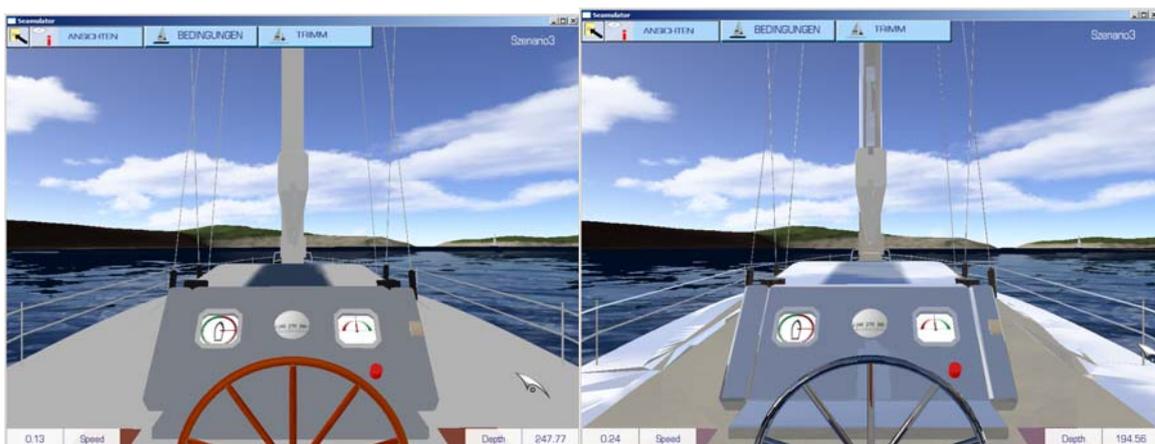
Glow:

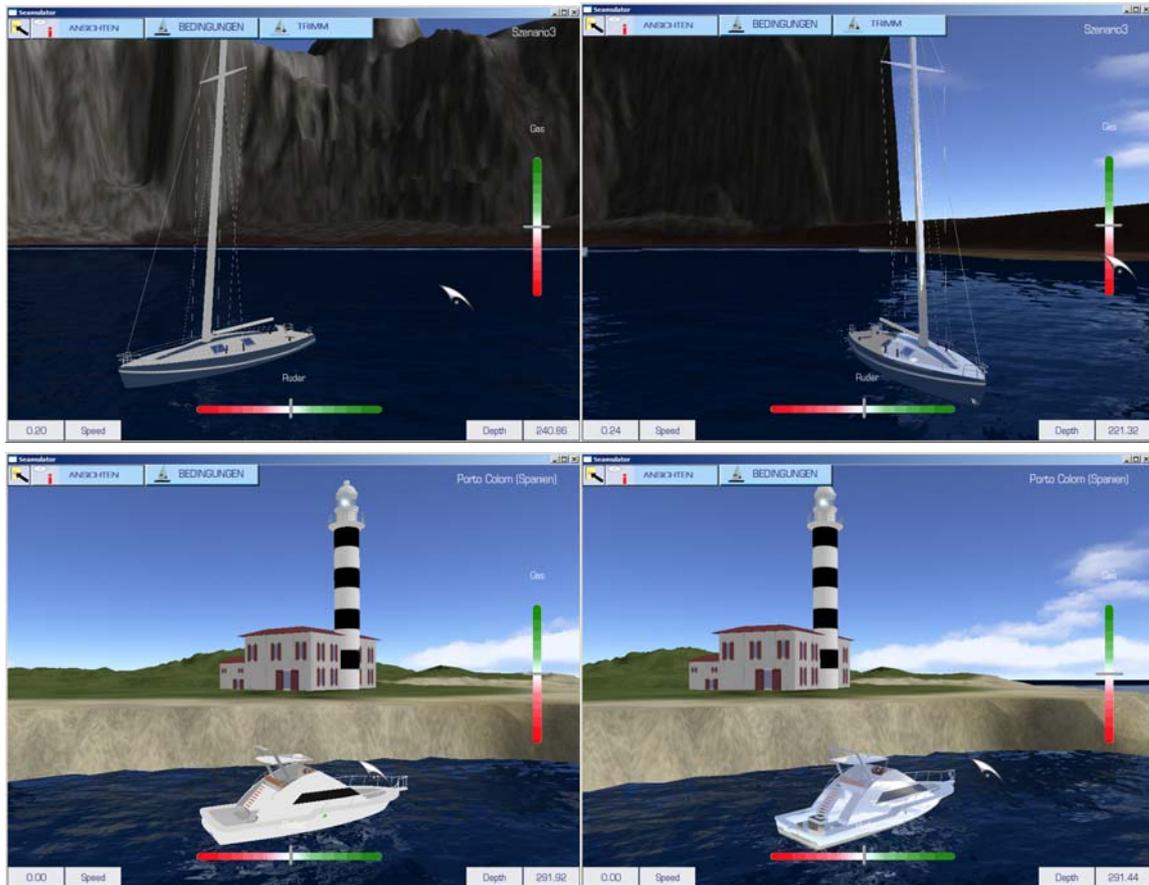


(Left: without Glow effect, Right: with Glow effect)

Environment Mapping:

The Environment Mapping technique is used for the ship bodies and certain instruments like the steering wheel of the sailing ship, which are made often of reflective materials.





(All on the left: without Environment Mapping, All on the right: with Environment Mapping)

Depth of Field:

With the use of this tool, we obtain images more similar to cinema or photography images. Moreover, due to the tool allows modify focal range, focal distance, and circle of confusion radius, is easy obtain the desired effect in each scene.

Nevertheless, this tool causes the typical intensity leakage problem present in most of the DOF algorithms.

This effect is produced when a blurred point is near of focused point. If we do a standard blur, it means, all tapped pixels has the same weight, the focused point affects to blurred point, and cause undesirable effects.



Image without Depth of Field



Images with different focal ranges

Glow/Tone mapping:

Using glows improves the brightness of the scene, as it affects to surrounding zones and allows burn the scene in bright zones. This is a double-edge feature, because without a good tweak of the glow filter parameters, the scene can be over burned, producing a undesirable effect.

On the other hand, the Tone Mapping tools allow using higher contrast levels in the environments. This is needed to obtain realistic daylight results, and allows soft changes between bright and dark zones, emulating the light adaptation of human eye. As glow tool, is important a previous work with the tone mapping parameters to obtain the desired results.



Image with and without glow



Image with and without tone mapping

Ray-trace effects:

In many types of materials (metals, polished surfaces...), the reflections are not an option if we wanted to reach a minimal degree of realism. This tool provides results close to those by ray tracing tools. Moreover, since it's an iterative algorithm, is possible adjust the complexity in base to the user's hardware. However, in most cases iterating only one time, the results are good enough.



Image without reflections



Reflections with standard system and ray-trace system